
MAPSERVER

MapServer Documentation

Release 5.4.2

The MapServer Team

January 13, 2010

CONTENTS

1	About	3
2	An Introduction to MapServer	5
2.1	MapServer Overview	5
2.2	Anatomy of a MapServer Application	6
2.3	Installation and Requirements	8
2.4	Introduction to the <i>Mapfile</i>	14
2.5	Making the Site Your Own	21
2.6	Enhancing your site	22
2.7	How do I get Help?	24
3	MapServer Tutorial	25
3.1	Tutorial Timeframe	25
3.2	Tutorial Data	25
3.3	Before Using the Tutorial	26
3.4	Windows, UNIX/Linux Issues	26
3.5	Other Resources	27
3.6	Section 1: Static Maps and the MapFile	27
3.7	Section 2: CGI variables and the User Interface	27
4	Installation	29
4.1	Compiling on Unix	29
4.2	Compiling on Win32	37
4.3	PHP MapScript Installation	44
4.4	.NET MapScript Compilation	50
4.5	IIS Setup for MapServer	55
4.6	Oracle Installation	58
5	Mapfile	63
5.1	CLASS	63
5.2	Expressions	66
5.3	FEATURE	71
5.4	FONTSET	71
5.5	INCLUDE	72
5.6	GRID	74
5.7	JOIN	75
5.8	LABEL	79
5.9	LAYER	82
5.10	LEGEND	89

5.11	MAP	89
5.12	OUTPUTFORMAT	92
5.13	PROJECTION	95
5.14	QUERYMAP	96
5.15	REFERENCE	97
5.16	SCALEBAR	97
5.17	STYLE	98
5.18	SYMBOL	100
5.19	Cartographic Symbol Construction with MapServer	101
5.20	Symbology Examples	123
5.21	Templating	126
5.22	Variable Substitution	134
5.23	WEB	135
5.24	Notes	136
6	MapScript	137
6.1	Introduction	137
6.2	SWIG MapScript API Reference	138
6.3	PHP MapScript	173
6.4	Python MapScript Appendix	215
6.5	Python MapScript Image Generation	217
6.6	Mapfile Manipulation	220
6.7	Querying	223
6.8	MapScript Variables	225
7	Data Input	233
7.1	Vector Data	233
7.2	Raster Data	289
8	Output Generation	301
8.1	AGG Rendering Specifics	301
8.2	AntiAliasing with MapServer	303
8.3	Dynamic Charting	306
8.4	Flash Output	311
8.5	HTML Legends with MapServer	316
8.6	HTML Imagemaps	325
8.7	PDF Output	328
8.8	SVG	333
8.9	Tile Mode	339
9	OGC Support and Configuration	345
9.1	WMS Server	345
9.2	WMS Client	364
9.3	WMS Time	371
9.4	Map Context	376
9.5	WFS Server	383
9.6	WFS Client	392
9.7	WFS Filter Encoding	395
9.8	SLD	402
9.9	WCS Server	412
9.10	WCS Use Cases	422
9.11	SOS Server	427
9.12	MapScript Wrappers for WxS Services	438
10	Optimization	447

10.1	Mapfile	447
10.2	Vector	449
10.3	Raster	451
10.4	FastCGI	452
10.5	Tile Indexes	455
11	Utilities	459
11.1	legend	459
11.2	msencrypt	459
11.3	scalebar	461
11.4	shp2img	461
11.5	shptree	463
11.6	shptreevis	464
11.7	sortshp	465
11.8	sym2img	467
11.9	tile4ms	468
11.10	Batch Scripting	472
11.11	File Management	472
12	CGI	475
12.1	MapServer CGI Introduction	475
12.2	mapserv	476
12.3	Map Context Files	476
12.4	MapServer CGI Controls	477
12.5	Run-time Substitution	481
12.6	A Simple CGI Wrapper Script	483
13	Community Activities	485
13.1	IRC	485
13.2	Mailing Lists	486
13.3	MapServer Wiki Pages	487
13.4	MapServer Service Providers	487
14	Development	489
14.1	Sponsors	489
14.2	Bug Submission	489
14.3	Subversion	490
14.4	Documentation Development Guide	491
14.5	Testing	497
14.6	Request for Comments	503
14.7	Mapfile Editing	665
14.8	External Links	667
15	Download	669
15.1	Source	669
15.2	Binaries	669
15.3	Demo Application	670
16	Glossary	671
17	Errors	675
17.1	drawEPP(): EPPL7 support is not available	675
17.2	loadMapInternal(): Given map extent is invalid	675
17.3	msSaveImageGD(): Unable to access file	676
17.4	msLoadMap(): Failed to open map file	677

17.5	msQueryByPoint: search returned no results	677
17.6	msLoadFontset(): Error opening fontset	677
17.7	msGetLabelSize(): Requested font not found	678
17.8	loadLayer(): Unknown identifier. Maximum number of classes reached	678
17.9	msReturnPage(): Web application error. Malformed template name	678
17.10	Unable to load dll (<i>MapScript</i>)	679
17.11	msProcessProjection(): Projection library error.major axis or radius = 0 not given	679
17.12	msProcessProjection(): no options found in 'init' file	680
17.13	msProcessProjection(): No such file or directory	680
17.14	msQueryByPoint: search returned no results	680
18	FAQ	683
18.1	Where is the MapServer log file?	683
18.2	What books are available about MapServer?	683
18.3	How do I compile MapServer for Windows?	683
18.4	What do MapServer version numbers mean?	683
18.5	Is MapServer Thread-safe?	683
18.6	What does STATUS mean in a LAYER?	684
18.7	How can I make my maps run faster?	685
18.8	What does Polyline mean in MapServer?	685
18.9	What is MapScript?	685
18.10	Does MapServer support reverse geocoding?	686
18.11	Does MapServer support geocoding?	686
18.12	How do I set line width in my maps?	686
18.13	Why do my JPEG input images look crappy via MapServer?	686
18.14	Which image format should I use?	687
18.15	Why doesn't PIL (Python Imaging Library) open my PNGs?	687
18.16	Why do my symbols look poor in JPEG output?	687
18.17	How do I add a copyright notice on the corner of my map?	688
18.18	How do I have a polygon that has both a fill and an outline with a width?	689
18.19	How can I create simple antialiased line features?	690
18.20	Which OGC Specifications does MapServer support?	690
18.21	Why does my requested WMS layer not align correctly?	691
18.22	When I do a GetCapabilities, why does my browser want to download mapserv.exe/mapserv?	692
18.23	Why do my WMS GetMap requests return exception using MapServer 5.0?	692
18.24	Where do I find my EPSG code?	693
18.25	How can I reproject my data using ogr2ogr?	693
19	License	695
20	Credits	697
	Bibliography	699
	Index	701


Note: The entire documentation is also available as a single PDF document 

Table 1: Quick Links

<i>An Introduction to MapServer</i>	<i>Installation</i>	<i>Mapfile</i>
<i>MapScript</i>	<i>Data Input</i>	<i>Output Generation</i>
<i>OGC Support and Configuration</i>	<i>Optimization</i>	<i>Utilities</i>
<i>Development</i>	<i>Glossary</i>	<i>Errors</i>
<i>Index</i>	<i>About</i>	<i>Community Activities</i>

ABOUT

MapServer is an [Open Source](#) geographic data rendering engine written in C. Beyond browsing GIS data, MapServer allows you create “geographic image maps”, that is, maps that can direct users to content. For example, the Minnesota DNR [Recreation Compass](#) provides users with more than 10,000 web pages, reports and maps via a single application. The same application serves as a “map engine” for other portions of the site, providing spatial context where needed.

MapServer was originally developed by the University of Minnesota (UMN) ForNet project in cooperation with NASA, and the Minnesota Department of Natural Resources (MNDNR). Later it was hosted by the TerraSIP project, a NASA sponsored project between the UMN and a consortium of land management interests.

MapServer is now a project of [OSGeo](#), and is maintained by a growing number of developers (nearing 20) from around the world. It is supported by a diverse group of organizations that fund enhancements and maintenance, and administered within OSGeo by the MapServer [Project Steering Committee](#) made up of developers and other contributors.

- Advanced cartographic output
 - Scale dependent feature drawing and application execution
 - Feature labeling including label collision mediation
 - Fully customizable, template driven output
 - TrueType fonts
 - Map element automation (scalebar, reference map, and legend)
 - Thematic mapping using logical- or regular expression-based classes
- Support for popular scripting and development environments
 - PHP, Python, Perl, Ruby, Java, and .NET
- Cross-platform support
 - Linux, Windows, Mac OS X, Solaris, and more
- Support of numerous [Open Geospatial Consortium](#) (OGC) standards
 - WMS (client/server), non-transactional WFS (client/server), WMC, WCS, Filter Encoding, SLD, GML, SOS, OM
- A multitude of raster and vector data formats
 - TIFF/GeoTIFF, EPPL7, and many others via [GDAL](#)
 - [ESRI shapefiles](#), [PostGIS](#), [ESRI ArcSDE](#), [Oracle Spatial](#), [MySQL](#) and many others via [OGR](#)
- Map projection support
 - On-the-fly map projection with 1000s of projections through the [Proj.4](#) library

AN INTRODUCTION TO MAPSERVER

Revision \$Revision: 8478 \$

Date \$Date: 2009-01-29 11:54:42 -0800 (Thu, 29 Jan 2009) \$

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author David Fawcett

Contact david.fawcett at moea.state.mn.us

Author Howard Butler

Contact hobu.inc at gmail.com

Contents

- An Introduction to MapServer
 - MapServer Overview
 - Anatomy of a MapServer Application
 - Installation and Requirements
 - Introduction to the *Mapfile*
 - Making the Site Your Own
 - Enhancing your site
 - How do I get Help?

2.1 MapServer Overview

MapServer is a popular Open Source project whose purpose is to display dynamic spatial maps over the Internet. Some of its major features include:

- support for display and querying of hundreds of raster, vector, and database formats
- ability to run on various operating systems (Windows, Linux, Mac OS X, etc.)
- support for popular scripting languages and development environments (PHP, Python, Perl, Ruby, Java, .NET)
- on-the-fly projections
- high quality rendering
- fully customizable application output

- many ready-to-use Open Source application environments

In its most basic form, MapServer is a *CGI* program that sits inactive on your Web server. When a request is sent to MapServer, it uses information passed in the request URL and the *Mapfile* to create an image of the requested map. The request may also return images for legends, scale bars, reference maps, and values passed as CGI variables.

See Also:

The *Glossary* contains an overview of many of the jargon terms in this document.

MapServer can be extended and customized through *MapScript* or *templating*. It can be built to support many different *vector* and *raster* input data formats, and it can generate a multitude of *output* formats. Most pre-compiled MapServer distributions contain most all of its features.

See Also:

Compiling on Unix and *Compiling on Win32*

Note: *MapScript* provides a scripting interface for MapServer for the construction of Web and stand-alone applications. MapScript can be used independently of CGI MapServer, and it is a loadable module that adds MapServer capability to your favorite scripting language. MapScript currently exists in *PHP*, Perl, *Python*, Ruby, Tcl, Java, and .NET flavors.

This guide will not explicitly discuss MapScript, check out the *MapScript Reference* for more information.

2.2 Anatomy of a MapServer Application

A simple MapServer application consists of:

- **Map File** - a structured text configuration file for your MapServer application. It defines the area of your map, tells the MapServer program where your data is and where to output images. It also defines your map layers, including their data source, projections, and symbology. It must have a .map extension or MapServer will not recognize it.

See Also:

MapServer Mapfile Reference

- **Geographic Data** - MapServer can utilize many geographic data source types. The default format is the ESRI shapefile. Many other data formats can be supported, this is discussed further below in Adding data to your site.

See Also:

Vector Input Reference and *Raster Input Reference*

- **HTML Pages** - the interface between the user and MapServer. They normally sit in Web root. In its simplest form, MapServer can be called to place a static map image on a html page. To make the map interactive, the image is placed in an html form on a page.

CGI programs are 'stateless', every request they get is new and they don't remember anything about the last time that they were hit by your application. For this reason, every time your application sends a request to MapServer, it needs to pass context information (what layers are on, where you are on the map, application mode, etc.) in hidden form variables or URL variables.

A simple MapServer *CGI* application may include two html pages:

- **Initialization File** - uses a form with hidden variables to send an initial query to the http server and MapServer. This form could be placed on another page or be replaced by passing the initialization information as variables in a URL.

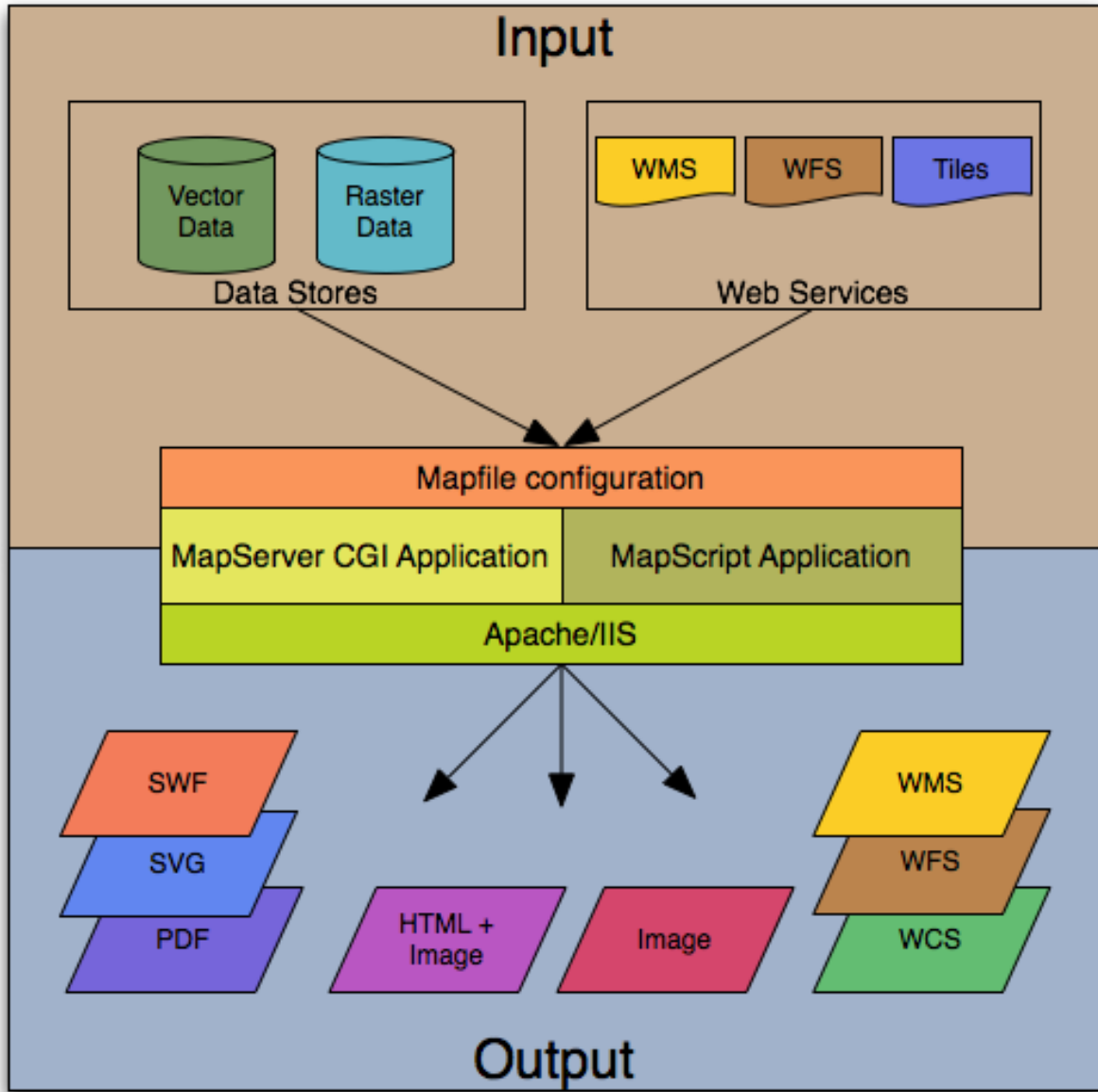


Figure 2.1: The basic architecture of MapServer applications.

- **Template File** - controls how the maps and legends output by MapServer will appear in the browser. By referencing MapServer CGI variables in the template html, you allow MapServer to populate them with values related to the current state of your application (e.g. map image name, reference image name, map extent, etc.) as it creates the html page for the browser to read. The template also determines how the user can interact with the MapServer application (browse, zoom, pan, query).

See Also:

Templating

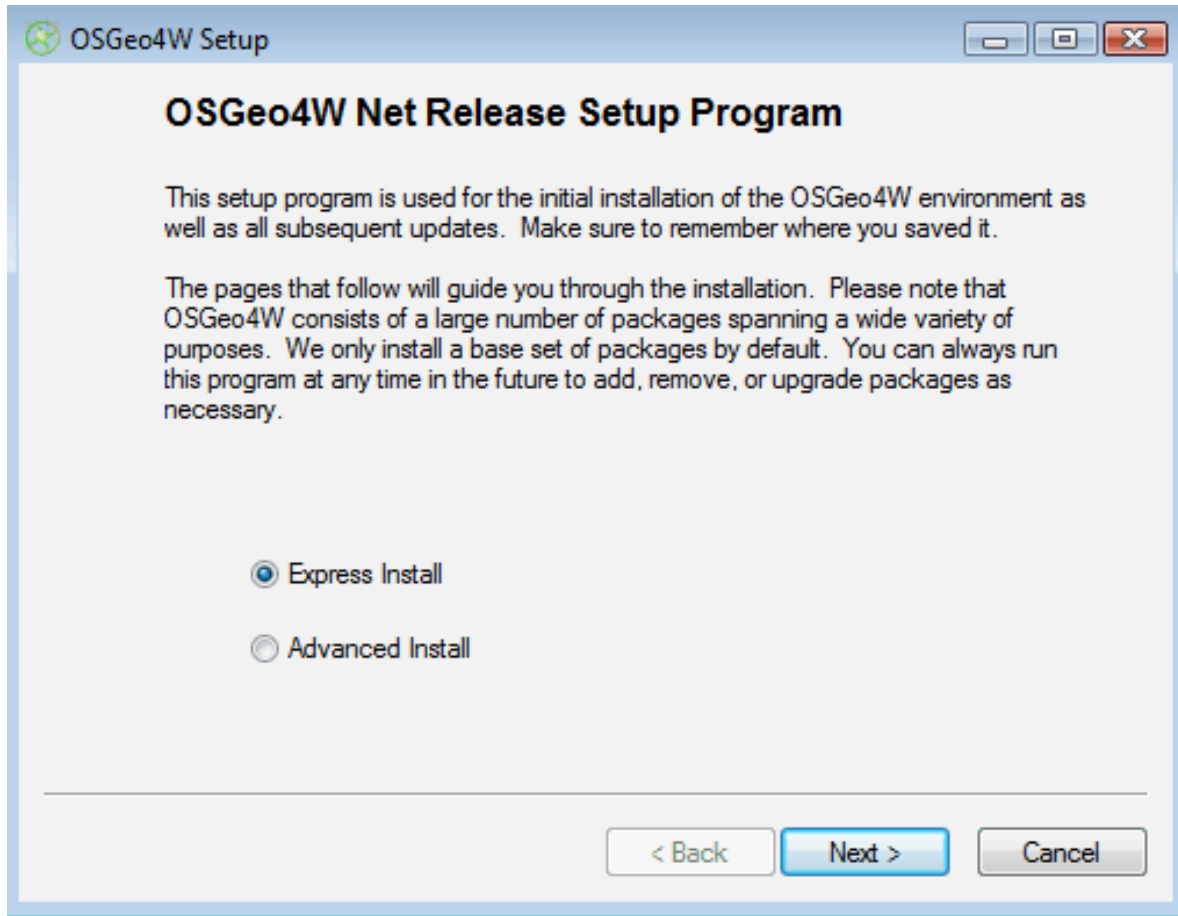
- **MapServer CGI** - The binary or executable file that receives requests and returns images, data, etc. It sits in the cgi-bin or scripts directory of the http server. The Web server user must have execute rights for the directory that it sits in, and for security reasons, it should not be in the web root. By default, this program is called *mapserv*
- **HTTP Server** - serves up the html pages when hit by the user's browser. You need a working HTTP (Web) server, such as [Apache](#) or Microsoft Internet Information Server, on the machine on which you are installing MapServer.

2.3 Installation and Requirements

2.3.1 Windows Installation

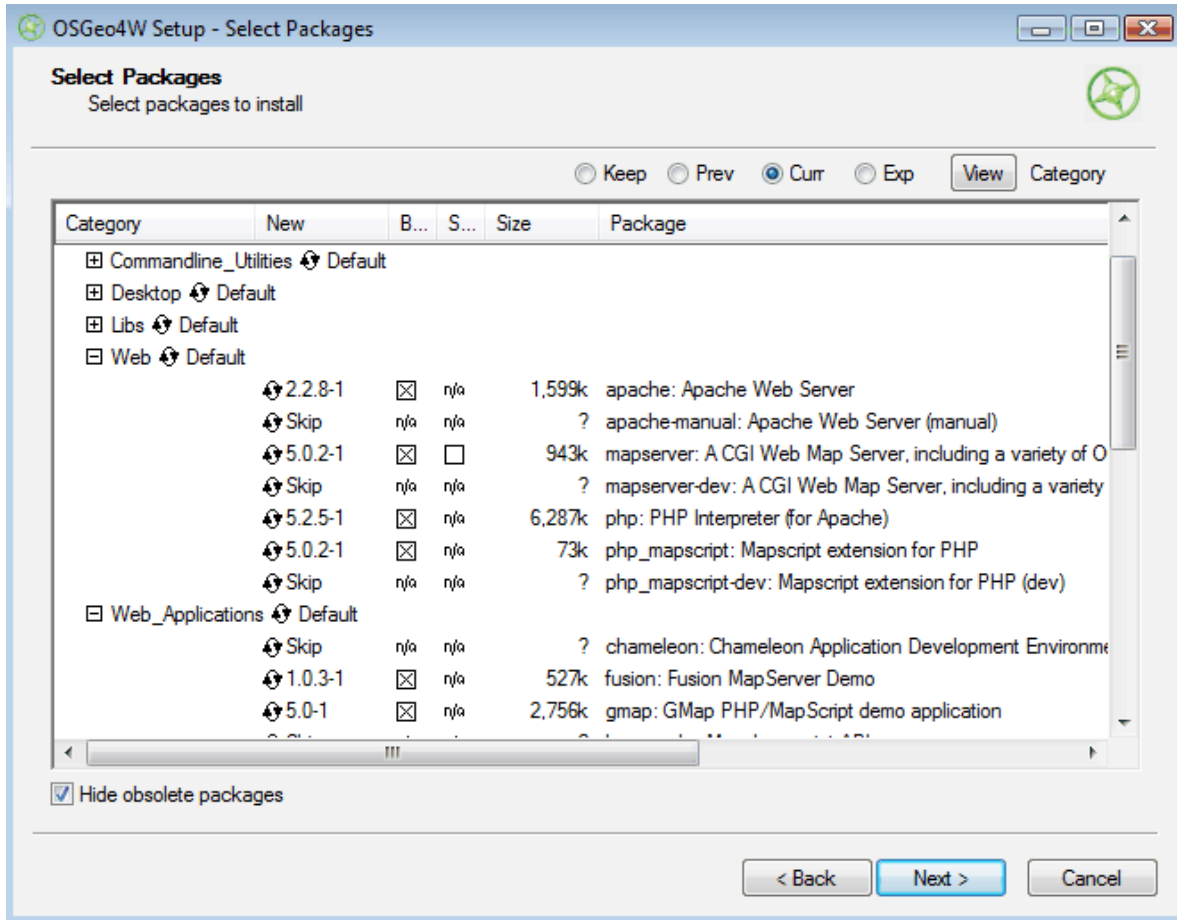
OSGeo4W is a new Windows installer that downloads and/or updates MapServer, add-on applications, and also other Open Source geospatial software. The following steps illustrate how to use OSGeo4W:

1. Download OSGeo4W <http://download.osgeo.org/osgeo4w/osgeo4w-setup.exe>
2. Execute (double-click) the .exe
3. Choose "Advanced" install type



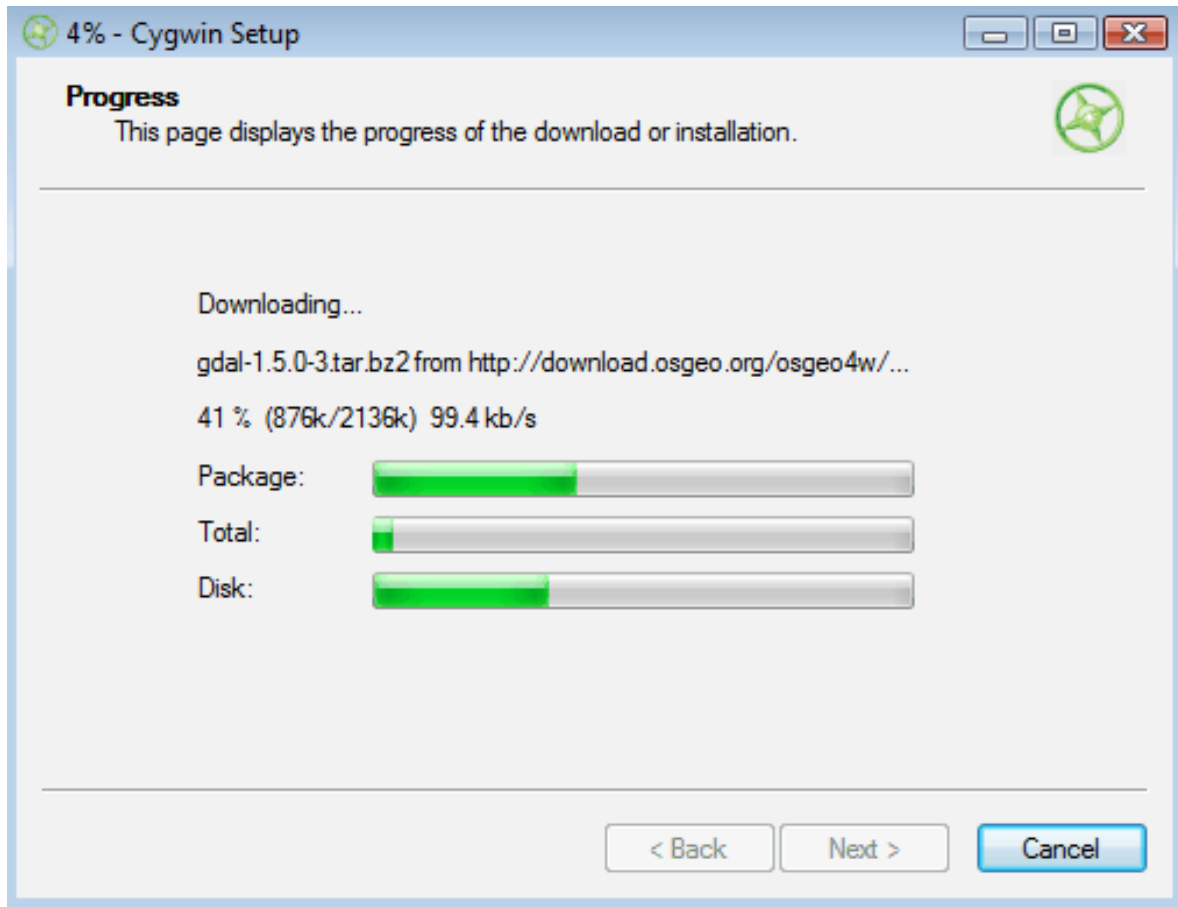
Note: Express contains options for higher-level packages such as MapServer, GRASS, and uDig. Advanced gives you full access to choosing commandline tools and applications for MapServer that are not included in the Express install

4. Select packages to install



Note: Click on the “Default” text beside the higher-level packages (such as Web) to install all of Web’s sub-packages, or click on the “Skip” text beside the sub-package (such as MapServer) to install that package and all of its dependencies.

5. Let the installer fetch the packages.



6. Run the `apache-install.bat` script to install the Apache Service.

Note: You must run this script under the “OSGeo4W Shell”. This is usually available as a shortcut on your desktop

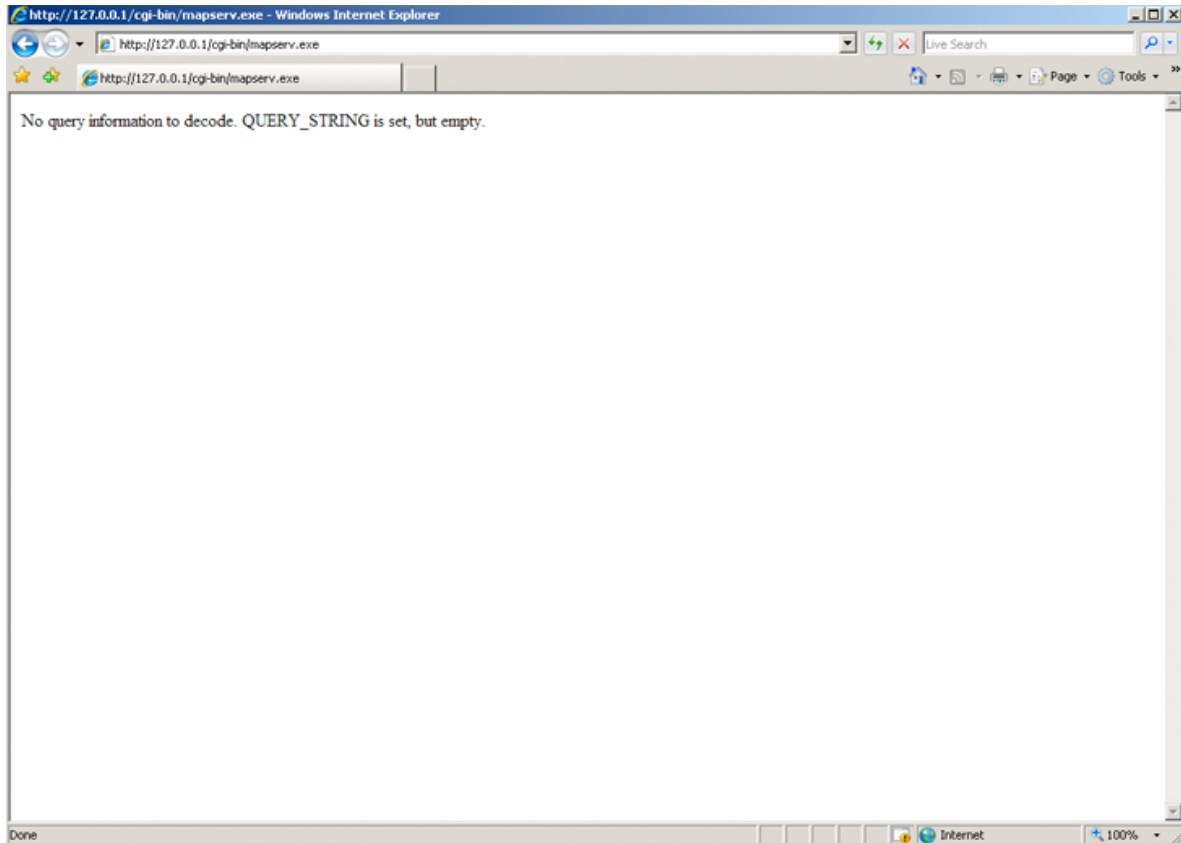
Note: A `apache-uninstall.bat` script is also available to remove the Apache service installation.

7. Start Apache from the OSGeo4W shell and navigate to `http://127.0.0.1`

```
apache-restart.bat
```



8. Verify that MapServer is working



2.3.2 Hardware Requirements

MapServer runs on Linux, Windows, Mac OS X, Solaris, and more. To compile or install some of the required programs, you may need administrative rights to the machine. People commonly ask questions about minimum hardware specifications for MapServer applications, but the answers are really specific to the individual application. For development and learning purposes, a very minimal machine will work fine. For deployment, you will want to investigate *Optimization* of everything from your data to server configuration.

2.3.3 Software Requirements

You need a working and properly configured HTTP (Web) server, such as [Apache](#) or Microsoft Internet Information Server, on the machine on which you are installing MapServer. OSGeo4W contains Apache already, but you can reconfigure things to use IIS if you need to. Alternatively, [MS4W](#) can be used to install MapServer on Windows.

If you are on a Windows machine, and you don't have a HTTP server installed, you may want to check out [MS4W](#), which will install a pre-configured HTTP server, MapServer, and more. The [FGS Linux Installer](#) provides similar functionality for several Linux distributions.

This introduction will assume you are using pre-compiled OSGeo4W Windows binaries to follow along. Obtaining MapServer or [Linux](#) or [Mac OS X](#) should be straightforward. Visit [Download](#) for installing pre-compiled MapServer builds on Mac OS X and Linux.

You will also need a Web browser, and a text editor (vi, emacs, notepad, homesite) to modify your html and *mapfiles*.

2.3.4 Skills

In addition to learning how the different components of a MapServer application work together and learning Map File syntax, building a basic application requires some conceptual understanding and proficiency in several skill areas.

You need to be able to create or at least modify [HTML](#) pages and understand how HTML forms work. Since the primary purpose of a MapServer application is to create maps, you will also need to understand the basics of geographic data and likely, map projections. As your applications get more complex, skills in SQL, DHTML/Javascript, Java, databases, expressions, compiling, and scripting may be very useful.

2.4 Introduction to the *Mapfile*

The `.map` file is the basic configuration file for data access and styling for MapServer. The file is an ASCII text file, and is made up of different objects. Each object has a variety of parameters available for it. All `.map` file (or `mapfile`) parameters are documented in the *mapfile reference*. A simple `mapfile` example displaying only one layer follows, as well as the map image output:

```
NAME "sample"
STATUS ON
SIZE 600 400
SYMBOLSET "../etc/symbols.txt"
EXTENT -180 -90 180 90
UNITS DD
SHAPEPATH "../data"
IMAGECOLOR 255 255 255
FONTSET "../etc/fonts.txt"

#
# Start of web interface definition
#
WEB
    IMAGEPATH "/ms4w/tmp/ms_tmp/"
    IMAGEURL "/ms_tmp/"
END

#
# Start of layer definitions
#
LAYER
    NAME 'global-raster'
    TYPE RASTER
    STATUS DEFAULT
    DATA bluemarble.gif
END
```

Note:

- Comments in a `mapfile` are specified with a '#' character
- MapServer parses `mapfiles` from top to bottom, therefore layers at the end of the `mapfile` will be drawn last (meaning they will be displayed on top of other layers)
- Using relative paths is always recommended
- Paths should be quoted (single or double quotes are accepted)

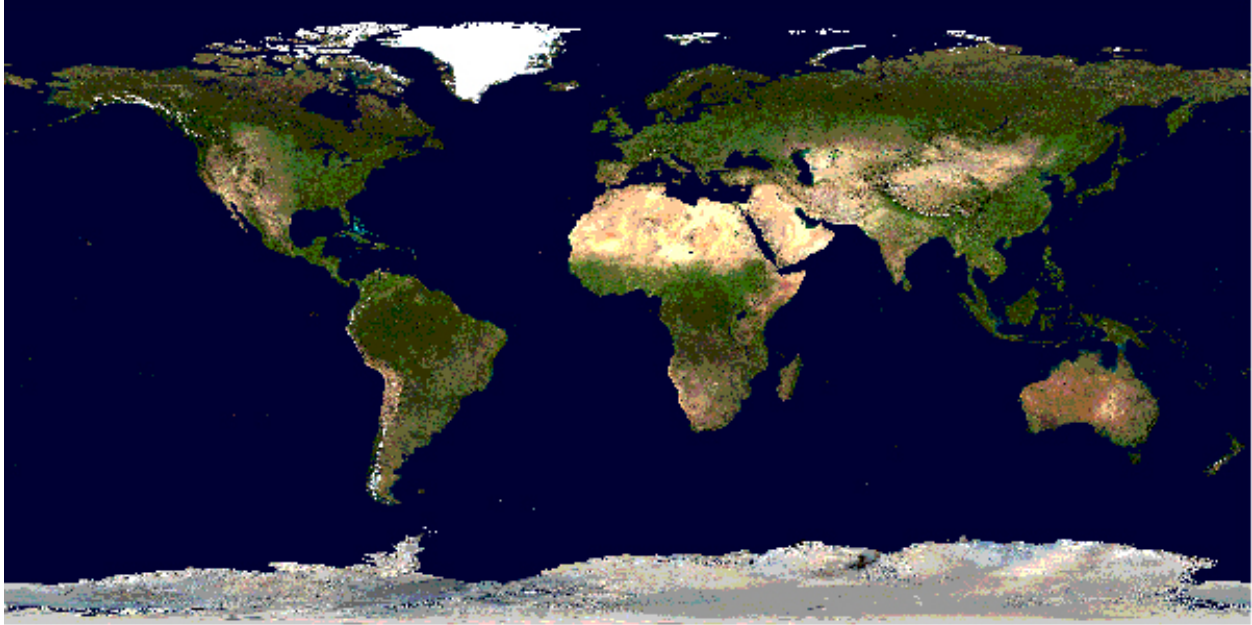


Figure 2.2: Rendered Bluemarble Image

2.4.1 *MAP* Object

```
MAP
  NAME          "sample"
  EXTENT        -180 -90 180 90 # Geographic
  SIZE          800 400
  IMAGECOLOR    128 128 255
END
```

- EXTENT is the output extent in the units of the output map
- SIZE is the width and height of the map image in pixels
- IMAGECOLOR is the default image background color

2.4.2 *LAYER* Object

- starting with MapServer 5.0, there is no limit to the number of layers in a mapfile
- DATA parameter is relative to the SHAPEPATH parameter the *MAP* object
- if no DATA extension is provided in the filename, MapServer will assume it is an ESRI shapefile (.shp)

Raster Layers

```
LAYER
  NAME bathymetry
  TYPE RASTER
  STATUS DEFAULT
  DATA bath-mapserver.tif
END
```

See Also:

Raster Data

Vector Layers

Vector layers of TYPE point, line, or polygon can be displayed. The following example shows how to display only lines from a TYPE polygon layer, using the OUTLINECOLOR parameter:

```
LAYER
  NAME "world_poly"
  DATA 'shapefile/countries_area.shp'
  STATUS ON
  TYPE POLYGON
  CLASS
    NAME 'The World'
    STYLE
      OUTLINECOLOR 0 0 0
    END
  END
END # layer
```

See Also:

Vector Data

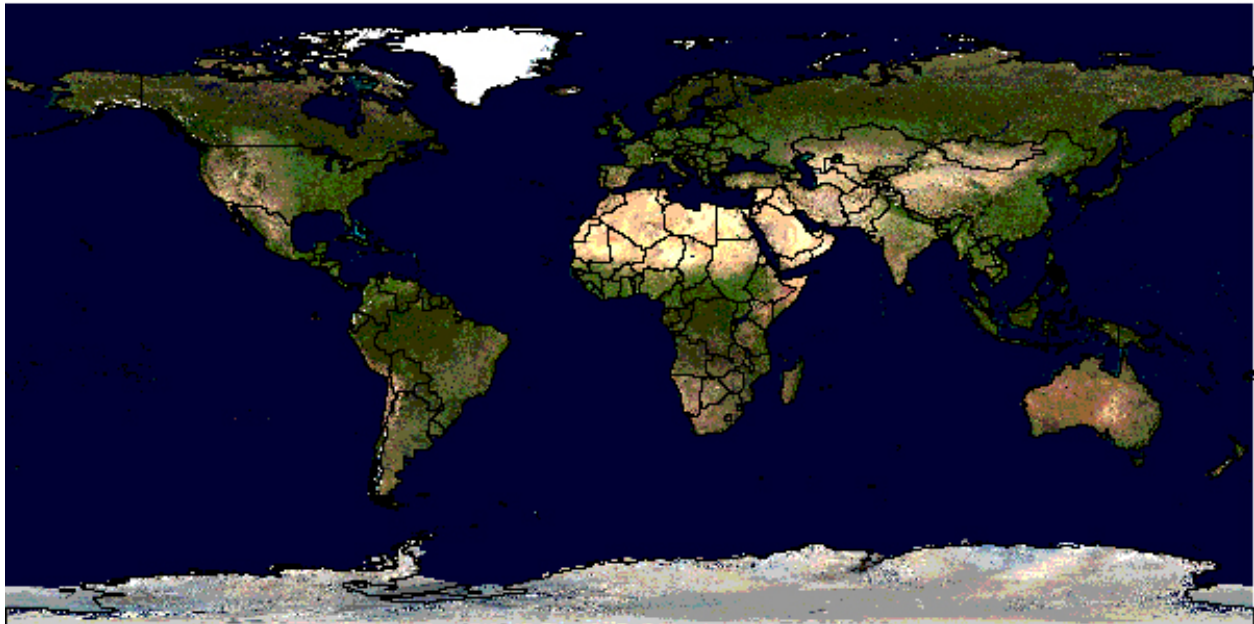


Figure 2.3: Rendered Bluemarble image with vector boundaries

2.4.3 CLASS and STYLE Objects

- typical styling information is stored within the *CLASS* and *STYLE* objects of a *LAYER*
- starting with MapServer 5.0, there is no limit to the number of classes or styles in a mapfile

- the following example shows how to display a road line with two colors by using overlaid STYLE objects

```

CLASS
NAME "Primary Roads"
STYLE
  SYMBOL "circle"
  COLOR 178 114 1
  SIZE 15
END #style1
STYLE
  SYMBOL "circle"
  COLOR 254 161 0
  SIZE 7
END #style2
END

```

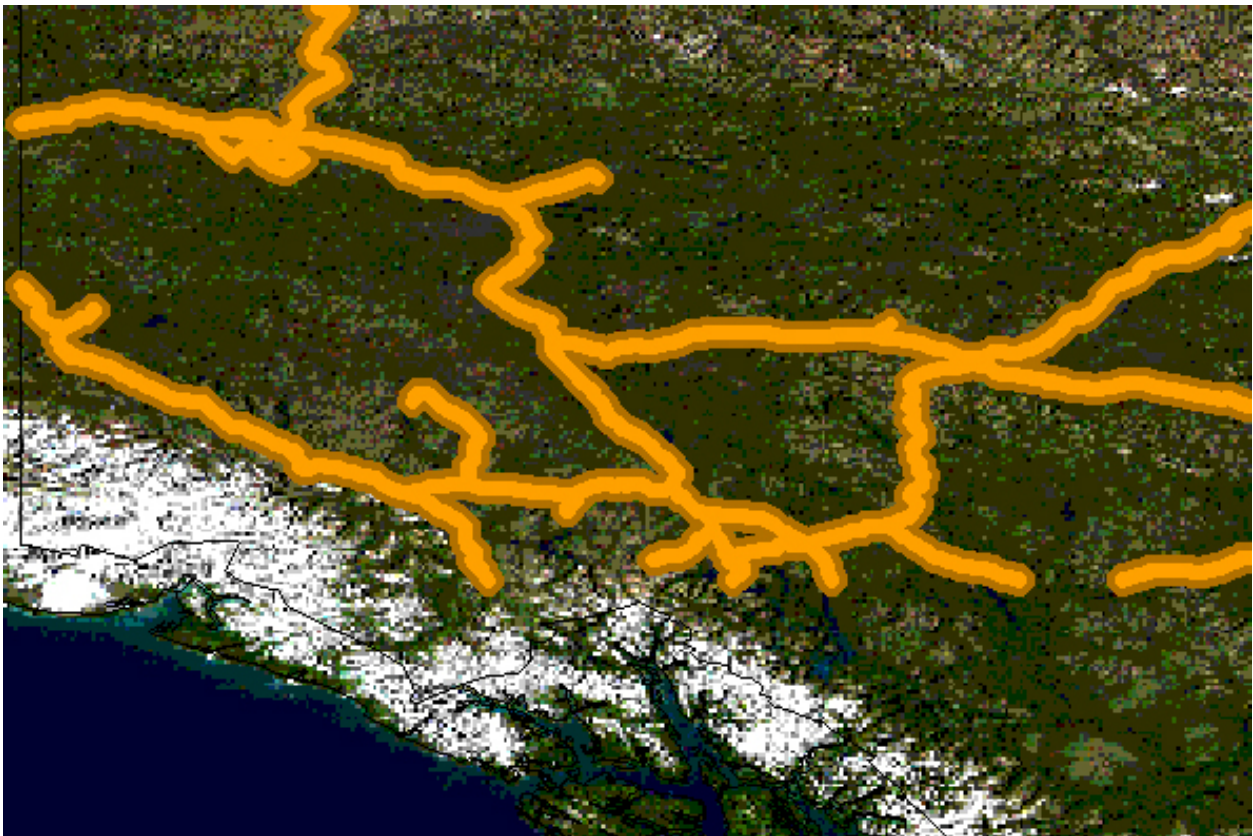


Figure 2.4: Rendered Bluemarble image with styled roads

2.4.4 SYMBOLS

- can be defined directly in the mapfile, or in a separate file
- the separate file method must use the SYMBOLSET parameter in the MAP object:

```

MAP
NAME "sample"
EXTENT -180 -90 180 90 # Geographic

```



```
SIZE          800 400
IMAGECOLOR    128 128 255
SYMBOLSET     "../etc/symbols.txt"
END
```

where symbols.txt might contain:

```
SYMBOL
  NAME "ski"
  TYPE PIXMAP
  IMAGE "ski.gif"
END
```

and the mapfile would contain:

```
LAYER
  ...
  CLASS
    NAME "Ski Area"
    STYLE
      SYMBOL "ski"
    END
  END
END # layer
```

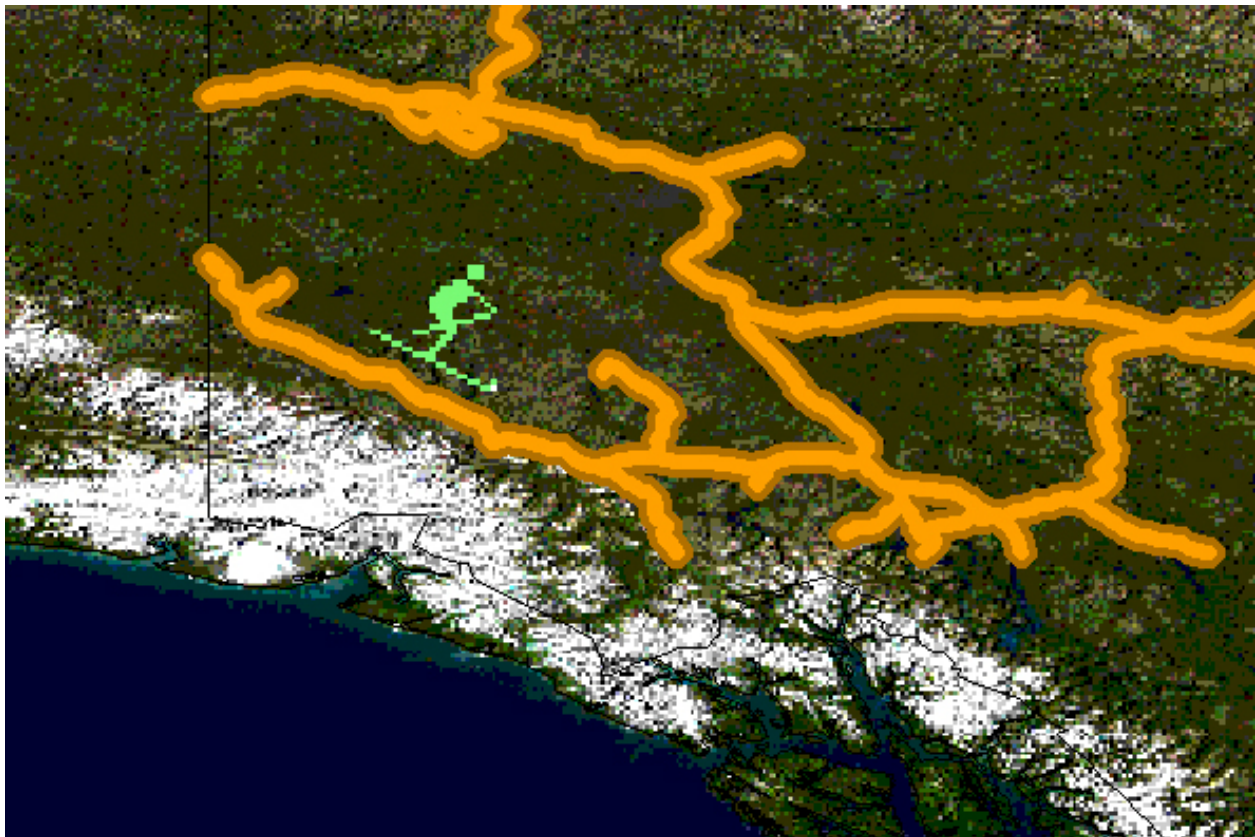


Figure 2.5: Rendered Bluemarble image with skier symbol

See Also:

Cartographic Symbol Construction with MapServer, Symbology Examples, and SYMBOL

2.4.5 LABEL

- defined within a *LAYER* object
- the LABELITEM parameters in the *LAYER* object can be used to label by a specific column in the data refer to a *FONTSET* file, that is set in the *MAP* object, that contains a reference to the available font names

An example *LABEL* object that references one of the above fonts might look like:

```
LABEL
  FONT "sans-bold"
  TYPE truetype
  SIZE 10
  POSITION LC
  PARTIALS FALSE
  COLOR 100 100 100
  OUTLINECOLOR 242 236 230
END # label
```

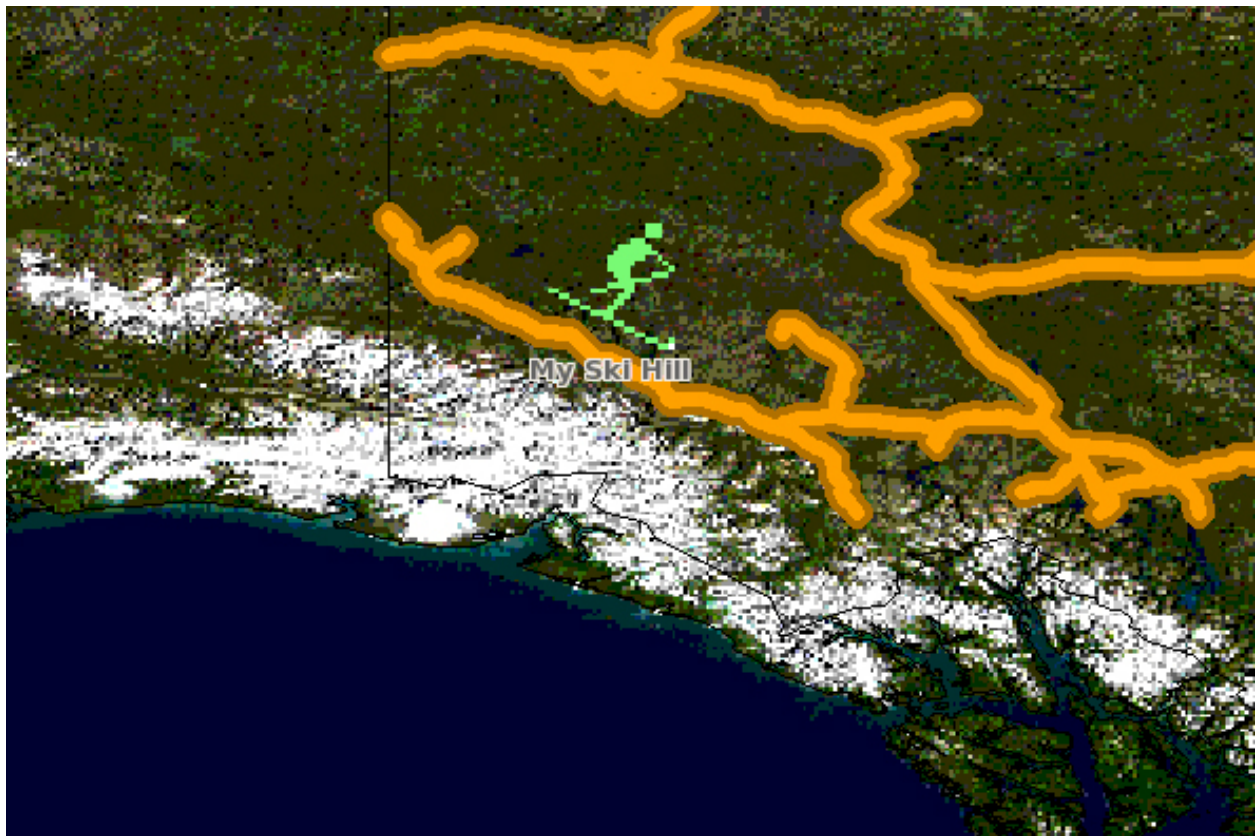


Figure 2.6: Rendered Bluemarble image with skier symbol and a label

See Also:

LABEL, *FONTSET*

2.4.6 CLASS Expressions

MapServer supports three types of *CLASS* expressions in a *LAYER*:

1. String comparisons

```
(EXPRESSION "africa")
```

2. Regular expressions

```
(EXPRESSION /^9|^10/)
```

3. Logical expressions

```
([POPULATION] > 50000 AND '[LANGUAGE]' eq 'FRENCH')
```

Note: Logical expressions should be avoided wherever possible as they are very costly in terms of drawing time.

See Also:

Expressions

2.4.7 INCLUDE

Added to MapServer 4.10, any part of the mapfile can now be stored in a separate file and added to the main mapfile using the *INCLUDE* parameter. The filename to be included can have any extension, and it is always relative to the main .map file. Here are some potential uses:

- *LAYER* s can be stored in files and included to any number of applications
- *STYLE* s can also be stored and included in multiple applications

The following is an example of using mapfile *includes* to include a layer definition in a separate file:

If 'shadedrelief.lay' contains:

```
LAYER
  NAME      'shadedrelief'
  STATUS    ON
  TYPE      RASTER
  DATA     'GLOBALeb3colshade.jpg'
END
```

therefore the main mapfile would contain:

```
MAP
  ...
  INCLUDE "shadedrelief.lay"
  ...
END
```

The following is an example of a mapfile where all *LAYER* s are in separate .lay files, and all other objects (*WEB*, *REFERENCE*, *SCALEBAR*, etc.) are stored in a ".ref" file:

```

NAME "base"
#
# include reference objects
#
INCLUDE "../templates/template.ref"
#
# Start of layer definitions
#
INCLUDE "../layers/usa/usa_outline.lay"
INCLUDE "../layers/canada/base/lm/provinces.lay"
INCLUDE "../layers/canada/base/lm/roads_atlas_of_canada_lm.lay"
INCLUDE "../layers/canada/base/lm/roads_atlas_of_canada_lm_shields.lay"
INCLUDE "../layers/canada/base/lm/populated_places.lay"
END # Map File

```

Warning: *Mapfiles* must end with the `.map` extension or MapServer will not recognize them. Include files can have any extension you want, however.

See Also:

INCLUDE

2.4.8 Get MapServer Running

```

MapServer version 5.2 OUTPUT=GIF OUTPUT=PNG
OUTPUT=JPEG OUTPUT=WBMP SUPPORTS=PROJ SUPPORTS=FREETYPE
SUPPORTS=WMS_SERVER INPUT=SHAPEFILE

```

You can also send a HTTP request directly to the MapServer CGI program without passing any configuration variables (e.g. <http://your.domain.name/cgi-bin/ms4/mapserv.exe>). If you receive the message, 'No query information to decode. QUERY_STRING not set.', your installation is working.

2.4.9 Get Demo Running

Download the [MapServer Demo](#). UnZip it and follow the directions in ReadMe.txt. You will need to move the demo files to their appropriate locations on your HTTP server, and modify the Map File and html pages to reflect the paths and URLs of your server. Next, point your browser to `init.html` and hit the 'initialize button'. If you get errors, verify that you have correctly modified the demo files.

2.5 Making the Site Your Own

Now that you have a working MapServer demo, you can use the demo to display your own data. Add new `LAYERs` to your Map file that refer to your own geographic data layers. (You will probably want to delete the existing layers or set their status to OFF.)

Unless you are adding layers that fall within the same geographic area as the demo, modify `MAP EXTENT` to match the extent of your data. To determine the extent of your data, you can use `ogrinfo`. If you have access to a GIS, you could use that as well. The `MAP EXTENT` needs to be in the units of your output projection.

If you add geographic data layers of different projections, you will need to modify your Map File to add a `PROJECTION` block to the `MAP` (output projection) and each of the `LAYER` (existing layer projection).

2.5.1 Adding Data to Your Site

MapServer supports several data input formats ‘natively’, and many more if it is compiled with the open source libraries *GDAL* and *OGR*.

2.5.2 Vector Data

Vector data includes features made up of points, lines, and polygons. MapServer supports ESRI shapefiles by default, but it can be compiled to support spatially enabled databases such as [PostgreSQL-PostGIS](#), [Geography Markup Language \(GML\)](#), [MapInfo](#), delimited text files, and more formats with *OGR*.

See the [Vector Data reference](#) for examples on how to add different geographic data sources to your MapServer project.

2.5.3 Raster Data

Raster data is image or grid data. By default, MapServer supports Tiff/GeoTiff, and EPPL7. With *GDAL*, it supports GRASS, Jpeg2000, ArcInfo Grids, and [more formats](#). If you do compile MapServer with *GDAL*, which includes tiff support, do not compile with native tiff support, as this will cause a conflict. More specific information can be found in the [Raster Data reference](#).

2.5.4 Projections

Because the earth is round and your monitor (or paper map) is flat, distortions will occur when you display geographic data in a two-dimensional image. Projections allow you to represent geographic data on a flat surface. In doing so, some of the original properties (e.g. area, direction, distance, scale or conformity) of the data will be distorted. Different projections excel at accurately portraying different properties. A good [primer](#) on map projections can be found at the University of Colorado.

With MapServer, if you keep all of your spatial data sets in the same projection (or unprojected Latitude and Longitude), you do not need to include any projection info in your Map File. In building your first MapServer application, this simplification is recommended.

On-the-fly projection can be accomplished when MapServer is compiled with *Proj.4* support. Instructions on how to enable Proj.4 support on Windows can be found on the [Wiki](#).

2.6 Enhancing your site

2.6.1 Adding Query Capability

There are two primary ways to query spatial data. Both methods return data through the use of templates and CGI variable replacement. A [QUERYMAP](#) can be used to map the results of the query.

To be queryable, each mapfile *LAYER* must have a *TEMPLATE* defined, or each *CLASS* within the *LAYER* must have a *TEMPLATE* defined. More information about the CGI variables used to define queries can be found in the [MapServer CGI Reference](#).

2.6.2 Attribute queries

The user selects features based on data associated with that feature. ‘Show me all of the lakes where depth is greater than 100 feet’, with ‘depth’ being a field in the shapefile .dbf or the spatial database. Attribute queries are accomplished

by passing query definition information to MapServer in the URL (or form post). Mode=itemquery returns a single result, and mode=itemnquery returns multiple result sets.

The request must also include a QLAYER, which identifies the layer to be queried, and a QSTRING which contains the query string. Optionally, QITEM, can be used in conjunction with QSTRING to define the field to be queried. Attribute queries only apply within the EXTENT set in the map file.

2.6.3 Spatial queries

The user selects features based on a click on the map or a user-defined selection box. Again the request is passed through a URL or form post. By setting mode=QUERY, a user click will return the one closest feature. In mode=NQUERY, all features found by a map click or user-defined selection box are returned. Additional query options can be found in the *CGI*.

2.6.4 Interfaces

See: OpenLayers <http://openlayers.org>

2.6.5 Data Optimization

Data organization is at least as important as hardware configuration in optimizing a MapServer application for performance. MapServer is quite efficient at what it does, but by reducing the amount of processing that it needs to do at the time of a user request, you can greatly increase performance. Here are a few rules:

- **Index Your data** - By creating spatial indexes for your shapefiles using *shptree*. Spatial indexes should also be created for spatially aware databases such as PostGIS and Oracle Spatial.
- **Tile Your Data** - Ideally, your data will be ‘sliced up’ into pieces about the size in which it will be displayed. There is unnecessary overhead to searching through a large shapefile or image of which you are only going to display a small area. By breaking the data up into tiles and creating a tile index, MapServer only needs to open up and search the data files of interest. Shapefile data can be broken into smaller tiles and then a tileindex shapefile can be created using the *tile4ms* utility. A *tileindex* shapefile for raster files can also be created.
- **Pre-Classify Your Data** - MapServer allows for the use of quite complex *EXPRESSIONs* to classify data. However, using logical and regular expressions is more resource intensive than string comparisons. To increase efficiency, you can divide your data into classes ahead of time, create a field to use as the CLASSITEM and populate it with a simple value that identifies the class, such as 1,2,3, or 4 for a four class data set. You can then do a simple string comparison for the class EXPRESSION.
- **Pre-Process Your Images** - Do resource intensive processing up front. See the *Raster Data reference* for more info.
- **Generalize for Overview** - create a more simple, generalized data layer to display at small scales, and then use scale-dependent layers utilizing *LAYER* MINSCALE and *LAYER* MAXSCALE to show more detailed data layers as the user zooms in. This same concept applies to images.

See Also:

Optimization

2.7 How do I get Help?

2.7.1 Documentation

- Official MapServer documentation lives here on this *site*.
- User contributed documentation exists on the MapServer [Wiki](#).

2.7.2 Users Mailing List

Register and post questions to the [MapServer Users](#) listserv. Questions to the list are usually answered quickly and often by the developers themselves. A few things to remember:

1. [Search the archives](#) for your answer first, people get tired of answering the same questions over and over.
2. Provide version and configuration information for your MapServer installation, and relevant snippets of your map and template files.
3. Always post your responses back to the whole list, as opposed to just the person who replied to your question.

2.7.3 IRC

MapServer users and developers can be found on Internet Relay Chat. The channel is [#mapserver](#) on [irc.freenode.net](#).

2.7.4 Gallery

See [examples](#) of existing MapServer applications.

2.7.5 Tutorial

Perry Nacionales built a great [Tutorial](#) on how to build a MapServer application. You are invited to extend the collection of examples if you see cases that are missing.

2.7.6 Test Suite

Download the [MapServer Test Suite](#) for a demonstration of some MapServer functionality.

2.7.7 Books

[Web Mapping Illustrated](#) , a new book by Tyler Mitchell that describes well and provides real-world examples for the use of Web mapping concepts, Open Source GIS software, MapServer, Web services, and PostGIS.

[Mapping Hacks](#) , by Schuyler Erle, Rich Gibson, and Jo Walsh, creatively demonstrates digital mapping tools and concepts. MapServer only appears in a handful of the 100 hacks, but many more are useful for concepts and inspiration.

[Beginning MapServer: Opensource GIS Development](#) , by Bill Kropla, is a new book focusing on MapServer. So new, I haven't seen it yet. According to the publisher, it covers installation and configuration, basic MapServer topics and features, incorporation of dynamic data, advanced topics, MapScript, and the creation of an actual application.

MAPSERVER TUTORIAL

Author Pericles S. Nacionales

Contact pnaciona at gmail.com

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8577 \$

Date \$Date: 2009-02-18 11:38:50 -0500 (Wed, 18 Feb 2009) \$

This tutorial was designed to give new users a quick (relatively speaking) introduction to the concepts behind MapServer. It is arranged into four sections with each section having one or more examples and increasing in complexity. Users can jump to any section at any time although it is recommended that absolute beginners work on the first three sections sequentially.

Section one focuses on basic MapServer configuration concepts such as layer and class ordering, using vector and raster data, projections and labeling. Section two provides examples on how to use HTML templates to create a simple interface for an interactive web mapping application. Section three introduces the use of HTML templates to provide a “query” interface. Finally, section four introduces some advanced user interface concepts.

3.1 Tutorial Timeframe

While some users can go through this tutorial in one day, those who work on each example in detail can probably expect to finish in one week.

3.2 Tutorial Data

The dataset used in this tutorial was taken from the U.S. Department of the Interior’s National Atlas of the United States. You can visit their web site at <http://www.nationalatlas.gov>. The dataset was clipped to the upper great lakes region (Minnesota, Michigan, and Wisconsin) to reduce storage size. Additional raster images were added courtesy of the TerraSIP project at the University of Minnesota. When using this tutorial, you are encouraged to use your own dataset.

Like MapServer itself, this tutorial is open and customizable to anyone. This was done in the hope that someone (or some folks) will help design and develop it further.

Download the data for this tutorial at http://demo.mapserver.org/tutorial/mstutorial_data.zip.

3.3 Before Using the Tutorial

There are some prerequisites to using this tutorial:

1. Users will need to have a web server installed and running on your computer. This web server has to have support for common gateway interface (CGI) programs.
2. Users should have a basic understanding of web servers and internet security. A poorly configured web server can easily be attacked by malicious people. At the very least your software installation will be corrupted and you'll lose hours of productivity, at worst your computer can be used to attack other computers on the internet.
3. It is recommended that users of this tutorial read the *Introduction to MapServer* before proceeding with this tutorial.
4. To use this tutorial, users will need to have a web server and a MapServer CGI program (mapserv or mapserv.exe) installed in their systems. MapServer source code is available for download [here](#). Documentation exists on how to compile and install MapServer:
 - for UNIX users, please read the *MapServer UNIX Compilation and Installation HOWTO*.
 - Windows users should read the *MapServer Win32 Compilation and Installation HOWTO*

In addition, Windows users can also download precompiled binaries from [here](#).

3.4 Windows, UNIX/Linux Issues

3.4.1 Paths

This tutorial was created in Linux/UNIX but should work with minimal changes on Windows platform. The main differences are the paths in the map files. Windows users need to specify the drive letter of the hard disk where their tutorial files reside. Here's an example:

A UNIX map file might include a parameter like this:

```
SHAPEPATH "/data/projects/tutorial/data"
```

In Windows, the same parameters might look like this:

```
SHAPEPATH "C:/data/projects/tutorial/data"
```

or:

```
SHAPEPATH "C:\data\projects\tutorial\data".
```

Notice that either slash or backslash works in Windows. The usual backslash may work well for you if you want to make a distinction between virtual (as in URLs or web addresses) and local paths in your map file. However, if you plan to move your application to UNIX at some point, you'll have the tedious task of switching all backslashes to slashes.

While we're on the subject of paths, keep in mind that paths in mapfiles are typically relative to the system's root directory: the slash ("/") in UNIX or some drive letter ("C:") in Windows. This is true except when specifically asked to enter a URL or when referencing a URL. When working with HTML template files, paths are relative to the web server's root directory. i.e., "tutorial/" is relative to "http://demo.mapserver.org/". Please read <http://www.alistapart.com/stories/slashforward/> for a few insights on URLs.

3.4.2 Executable

Another issue is that UNIX executable files don't require a .EXE or .COM extensions, but they do in Windows. If you are using Windows, append .exe to all instances of "/cgi-bin/mapserv" or "/cgi-bin/mapserv50" to make it "/cgi-bin/mapserv.exe" or "/cgi-bin/mapserv50.exe".

3.5 Other Resources

Other documentation exist to give you better understanding of the many customizations MapServer offer. Please visit the MapServer documentation page at <http://www.mapserver.org>. There you will find several HOWTO documents, from getting started to using MapScript, a scripting interface for MapServer.

[Back to Tutorial home](#) | [Proceed to Section 1](#)

3.6 Section 1: Static Maps and the MapFile

- **Take a shapefile. Any shapefile. We can use MapServer to display that shapefile on a web browser. Look...** –
Example 1.1 - A map with a single layer
 - **We can display the same shapefile repeatedly. We can display the polygon attributes on one LAYER and and the line attri**
Example 1.2 - A map with two layers
 - **And we can select which parts of the shapefile to display. We do this using the CLASS object...** –
Example 1.3 - Using classes to make a "useful" map
 - **We can also label our maps...** – *Example 1.4 - Labeling layers and label layers*
 - **Or add raster data such as satellite images, aerial photographs, or shaded reliefs...** – *Example 1.5 - Adding a raster layer*
 - **We can reproject our data from just about any projection to just about any... Yeah, check it out!** –
Example 1.6 - Projection/Reprojection
 - **And we can use layers from other map servers on the internet (as long as they are WMS servers)...** –
Example 1.7 - Adding a WMS layer
 - **MapServer can output to various formats such as PDF and GeoTIFF.** – *Example 1.8 - A different output format*
 - **MapServer not only generates static maps, it can also create interactive maps...** – *Example 1.9 - The difference between map and browse mode*
-

[Back to Tutorial home](#) | [Proceed to Section 2](#)

3.7 Section 2: CGI variables and the User Interface

So far we have only looked at the mapfile when creating maps. In creating web mapping applications, it is usually our intention to make maps that can be changed by the user (of the application) interactively. That is, a user should be able to change the content of (or the information in) the map. To accomplish this interactivity, we use the MapServer HTML templates.

3.7.1 HTML Templates

A MapServer HTML template is essentially an HTML file with a few MapServer specific tags. These tags are the MapServer CGI variables and are enclosed in square brackets “[]”. When the MapServer CGI program processes an application, it first parses the query string and the mapfile, and produces the necessary output. Some of this output will need to be written to the HTML template file which you would have to also specify in the mapfile using the web template keyword (or in a separate HTML initialization file). The CGI program will replace all the variables in the HTML template with the proper value before sending it back to the web browser. If you are to directly view an HTML template on a web browser, there won't be any maps rendered and you will instead get blank images and other junk.

Variables

MapServer provides several variables for web mapping: the “img” variable which you've seen in Example 1.9 is but one example. There are a few core CGI variables originally designed as part of the mapping interface but practically all the mapfile parameters can be defined as variables. The definitive reference to the CGI variables can be found at [here](#).

We can also define our own variables, which MapServer will pass along to our application. For example, we can create a variable called “root” to represent the root directory of this tutorial, the value for “root” will then be “/tutorial”. When the MapServer CGI program processes our HTML template, it will replace every instance of the “[root]” tag with “/tutorial”. You will see this in action for each of the following examples.

3.7.2 Examples

So, let's build an interactive interface for our application...

- Users of a web mapping application should be able to pan and zoom on the map: [Example 2.1 - Pan and Zoom Controls](#)
- They also should be able to turn on and off layers on a map: [Example 2.2 - Layer Control](#)
- A map should always include a scalebar. [Example 2.3 - Adding a Scalebar](#)
- If users are to navigate through the map, a reference map should be provided: [Example 2.4 - Adding a Reference Map](#)
- The map should include a legend. [Example 2.5- Adding a Legend](#)

[Back to Section 1 index](#)

[Begin tutorial](#)

INSTALLATION

4.1 Compiling on Unix

Author J.F. Doyon

Contact jdoyon at nrcan.gc.ca

Author Howard Butler

Contact hobu.inc at gmail.com

Revision \$Revision: 8463 \$

Date \$Date: 2009-01-28 07:37:36 -0800 (Wed, 28 Jan 2009) \$

Table of Contents

- Compiling on Unix
 - Introduction
 - Obtaining the necessary software
 - libgd
 - Anti-Grain Geometry Support
 - OGC Support
 - Spatial Warehousing
 - Compiling
 - Installation

4.1.1 Introduction

The University of Minnesota's MapServer is an open-source and freely available map rendering engine for the web. Due to its open-source nature, it can be compiled on a wide variety of platforms and operating systems. We will focus on how to obtain, compile and install MapServer on UNIX-like platforms.

You might also check the [MapServerCompilation](#) wiki page for additional information.

4.1.2 Obtaining the necessary software

You can obtain the MapServer source code as well as the demo package from the [Download](#) section.

You can also get the latest MapServer source code from [Subversion](#).

Required External Libraries

- **libpng**: libpng should be on your system by default. 1.2.12 is the current release with security patches, although versions all the way back to 1.2.7 should work.
- **freetype**: Version 2.x or above is required by GD.
- **GD**: libgd is used by MapServer for rendering images. Version 2.0.28 or greater required. Version 2.0.29 or later is required to use curved (following) labels, and version 2.0.34 is required for antialiasing (1 pixel wide lines/outlines).
- **zlib**: Zlib should be on your system by default. 1.2.1 is the current release with security patches.

Highly Recommended Libraries

- **libproj**: libproj provides projection support for MapServer. Version 4.4.6 or greater is required.
- **libcurl**: libcurl is the foundation of OGC (WFS/WMS/WCS) client and server support. Version 7.10 or greater is required
- **OGR**: OGR provides access to at least 18 different vector formats.
- **GDAL**: GDAL provides access to at least 42 different raster formats.
- **AGG**: AGG (Anti-Grain Geometry) is an optional dependency to enable high quality antialiased output for vector data. Currently versions 2.4 and 2.5 are identical featurewise, and only vary in their licence (2.4 is BSD, 2.5 is GPL)

Optional External Libraries

- **libtiff**: libtiff provides TIFF (Tagged Image File Format) reading support to MapServer.
- **libgeotiff**: libgeotiff provides support to read GeoTIFF files (TIFF files with geographic referencing).
- **libjpeg**: libjpeg allows MapServer to render images in JPEG format. A sufficient version should be installed by default on your system. Version 6b is the current version and dates back to 1998.
- **GEOS**: GEOS allows MapServer to do spatial predicate and algebra operations (within, touches, etc & union, difference, intersection). Requires version 4.10 or greater.
- **libxml**: libxml is required to use *OGC SOS* support in MapServer (versions 4.10 and greater).
- **SDE Client Library**: The client libraries for your platform should be part of the ArcSDE media kit. They are *not* publicly available for download.
- **Oracle Spatial OCI**: The client libraries for your platform are available for download from Oracle's website. Ideally, your client library matches the database you are querying from, but this is not a hard requirement.
- **libpq**: libpq is required to support the use of PostGIS geometries within the PostgreSQL database. Ideally, your client library matches the database you are querying from.
- **pdflib (lite)**: PDFlib Lite is the Open Source version of PDFlib that allows MapServer to produce PDF output. Version 4.0.3 or greater is required.
- **libming**: libming provides Macromedia Flash output to MapServer. Version 0.2a is required. Later versions are not known to work.

4.1.3 libgd

There are a number of issues that you should be aware of when using GD in combination with MapServer.

Minimum libgd versions

MapServer aggressively takes advantage of new features and bug fixes in the latest versions of libgd. The minimum required version to run MapServer is 2.0.29. Upgrading to at least 2.0.34 is advised as it includes an important bug fix for antialiased lines. Configure should detect which version of libgd you have installed, but you can quickly check yourself by issuing the following command:

```
gdlib-config --version
```

libiconv

If you intend to use international character sets, your version of libgd *must* be compiled against the GNU iconv libraries. If you are using a pre-packaged version, it is very likely that this is the case. To check for yourself, issue the following command and look for ‘-liconv’ in the output:

```
gdlib-config --libs
```

Pre-packaged/system libraries

If you intend to use your system’s libgd, ensure that you have the development package also installed so MapServer can find and use the appropriate headers.

MacOSX

A useful FAQ on for libgd on OSX is available at http://www.libgd.org/DOC_INSTALL_OSX

FreeType support

The GD you compile MapServer against **MUST** be compiled against the FreeType library in order to use TrueType fonts. MapServer no longer uses it’s own interface to FreeType, using it through GD instead.

When you run your “configure” script, look for the following output:

```
using GD ( -DUSE_GD_GIF -DUSE_GD_PNG -DUSE_GD_JPEG
           -DUSE_GD_WBMP -DUSE_GD_TTF -DGD_HAS_GDIMAGEGIFPTR) from system libs.
```

If your GD is built against FreeType, you will see either “-DUSE_GD_TTF” (Or “-DUSE_GD_FT” for Freetype 2.x) part. If it’s missing, you will need to recompile your GD to make sure you include FreeType support. See the GD documentation for more information.

Also note that the configure script looks for the FreeType library separately as well, generating output looking somewhat like this:

```
checking where FreeType is installed...
checking for FT_Init_FreeType in -lfreetype... yes
      using libfreetype -lfreetype from system libs.
```

Even though you have FreeType installed on your system *and* the configure script finds it, does *NOT* mean you will have TrueType font support. GD **MUST** be compiled against FreeType either way.

1px Anti-Aliasing and segfaults

Versions of libgd earlier than 2.0.34 contain a one very significant bug and will *always* cause a segfault if you attempt to do one pixel wide antialiasing. You can manually patch older gd's, or better yet upgrade to at least GD 2.0.34.

In gd.c, function `gdImageSetAAPixelColor()` change:

```
int dr, dg, db, p, r, g, b;
p = gdImageGetPixel(im, x, y);
```

to

```
int dr, dg, db, p, r, g, b;
if (!gdImageBoundsSafeMacro (im, x, y)) return;
p = gdImageGetPixel(im, x, y);
```

More detail about this patch (if you need any) was described by Steve Lime in a [post to mapserver-users](#).

Curved label support

ANGLE FOLLOW, a new feature that allows MapServer to draw curved labels about a linear feature like a road, requires libgd 2.0.29 and TrueType font support. Configure should autodetect if you have a sufficient libgd and TrueType support to be able to use this feature.

4.1.4 Anti-Grain Geometry Support

Since version 5.0 MapServer supports the AGG rendering backend. Download the 2.4 tarball from the [antigrain website](#) and just type `make` in the root directory. If you intend on using `mapscript`, you must beforehand tweak the `agg` makefile to add `-fPIC` to the compiler options.

4.1.5 OGC Support

MapServer provides support for many [OGC](#) specifications. At 4.2.3, it provides support for WMS (Web Mapping Service), SLD (Styled Layer Descriptor), WFS (Web Feature Service), and experimental support for WCS (Web Coverage Service).

WMS support

WMS Server

Support for this specification is automatically enabled when you include PROJ.4 support. (`--with-proj`) You can check this yourself by looking for the following in your “configure” output:

```
checking whether we should include WMS support...
  OGC WMS compatibility enabled (--DUSE_WMS).
```

If, for some reason you DON'T want WMS support, you can force it off by passing “`--without-wms`” to your configure script.

More information on using this feature is available in the WMS Server HOWTO available on the MapServer website.

WMS Client

Cascading is also supported. This allows mapserver to transparently fetch remote layers over WMS, basically acting like a client, and combine them with other layers to generate the final map.

In order to enable this feature, you will need to pass the “`-with-wmsclient`” option to the configure script. MapServer will automatically look for libcurl, which is also required.

To verify that the WMS Client feature is enabled, check the output from the configure script:

```
checking whether we should include WMS Client Connections support...
   OGC WMS Client Connections enabled (-DUSE_WMS_LYR).
```

Note that this feature is disabled by default, you have to specifically request it.

More information on using this feature is available in the WMS Client HOWTO available on the MapServer website.

WFS support

WFS Server

Support for this specification is enabled by passing the configure script the “`-with-wfs`” option. OGR and PROJ.4 support is required.

You can check this yourself by looking for the following in your “configure” output:

```
checking whether we should include WFS Server support...
   OGC WFS Server support enabled (-DUSE_WFS_SVR).
```

Note that this feature is disabled by default, you have to specifically request it.

More information on using this feature is available in the WFS Server HOWTO available on the MapServer website.

WFS Client

MapServer can also act as a WFS client. This effectively means that MapServer reads it’s data from a remote server’s WFS output and renders it into a map, just like it would when reading data from a shapefile.

In order to enable this feature, you will need to make sure you include OGR (Built with Xerces support) and PROJ.4 support, and pass the “`-with-wfsclient`” option to your configure script. MapServer will automatically look for libcurl, which is also required.

To verify that the WFS Client feature is enabled, check the output from the configure script:

```
checking whether we should include WFS Client Connections support...
   OGC WFS Client Connections enabled (-DUSE_WFS_LYR).
```

Note that this feature is disabled by default, you have to specifically request it.

More information on using this feature is available in the WFS Client HOWTO available on the MapServer website.

4.1.6 Spatial Warehousing

MapServer can use a wide variety of sources of data input. One of the solutions growing in popularity is to use spatially enabled databases to store data, and to use them directly to draw maps for the web.

Here you will find out how to enable mapserver to talk to one of these products. Please refer to the MapFile reference for more details on how to use these. This section only details how to compile MapServer for their use.

PostGIS

PostGIS adds 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 spatial database for geographic information systems (GIS), much like ESRI’s SDE or Oracle’s Spatial extension. PostGIS is included in many distributions’ packaging system, but you can also roll your own if needed.

MapServer can use PostGIS as a data source. In order to do so simply use “--with-postgis” when running your configure script.

```
--with-postgis=/usr/local/pgsql/bin/pg_config
```

ArcSDE

MapServer allows you to use SDE as a data source both for geometry and attributes. In order to achieve this, you must have the SDE client libraries at your disposition, and have them installed on the machine running MapServer.

In order to enable SDE support in MapServer, you have to compile it with two options specified:

```
--with-sde=/opt/sdeexe90  
--with-sde-version=90
```

Oracle Spatial

Oracle’s Spatial Warehousing cartridge is also supported by MapServer. In order to connect to it, you will need to compile MapServer against the Oracle libraries by passing the “--with-oraclespatial” argument to your configure script. You will very likely need an ORACLE_HOME environment variable set to have it configure things correctly.

```
--with-oraclespatial=/opt/oracle
```

4.1.7 Compiling

First prepare the ground by making sure all of your required and/or recommended libraries are installed before attempting to compile MapServer. This will make your life much less complicated ;). Here is the order that I usually use:

1. Compile GD. This often means acquiring libjpeg, libpng, zlib, and freetype before actually compiling the library. You shouldn’t have too much trouble finding binaries of the libraries that GD requires, and often, they will already be installed with your system. On unix, I’ve had very little luck finding pre-compiled binaries of the required GD library. See libgd section for notes about patching libgd if you plan to use antialiasing.
2. Compile GDAL/OGR. Describing how to compile GDAL/OGR is beyond the scope of this document. If you have requirements for lots of different formats, make sure to install those libraries first. I often find that building up a GDAL/OGR library often takes as long as compiling MapServer itself!

3. Compile Proj.4. Proj.4 is a straight-forward configure/make/make install library.
4. Compile libcurl. libcurl is a straight-forward configure/make/make install library.
5. Compile/install optional libraries. These might include SDE, PostGIS, Oracle Spatial, AGG, Ming, PDFlib, or MyGIS. Mix and match as you need them.
6. Unpack the MapServer tarball and cd into the mapserver directory:

```
[user@host user]$ tar -zxvf mapserver-X.Y.Z.tar.gz
```

7. Configure your environment using “configure”. I often place my configure command in its own file and changes its mode to be executable (+x) to save typing and have a record of how MapServer was configured.

```
./configure --with-sde=/usr/sde/sdeexe90 \
--with-sde-version=90 \
--with-ogr=/usr/local/bin/gdal-config \
--with-gdal=/usr/local/bin/gdal-config \
--with-httpd=/usr/sbin/httpd \
--with-wfsclient \
--with-wmsclient \
--enable-debug \
--with-curl-config=/usr/bin/curl-config \
--with-proj=/usr/local \
--with-tiff \
--with-gd=/usr/local/ \
--with-jpeg \
--with-freetype=/usr/ \
--with-oraclespatial=/usr/oracle \
--with-threads \
--with-wcs \
--with-postgis=/usr/local/database/bin/pg_config \
--with-libiconv=/usr \ # new in 4.8
--with-geos=/usr/local/bin/geos-config \ # new in 4.8
--with-libiconv=/usr \ # new in 4.8
--with-xml2-config=/usr/bin/xml2-config \ # new in 4.10
--with-sos \ # new in 4.10
--with-agg=/path/to/agg-2.4
```

8. Now that you have configured your build options and selected all the libraries you wish mapserver to use, you’re ready to compile the source code into an executable.

This is actually quite simple, just execute “make”:

```
[user@host mapserver]$ make
```

9. There is no *make install* step in the installation of MapServer. The output of the compilation of MapServer is a binary executable that you can use in a CGI execution environment.

To make sure all went well, look for the file called *mapserv*

```
[user@host mapserver]$ ls -al mapserv
-rwxr-xr-x  1 user  user    351177 Dec 21 11:38 mapserv
```

A simple test is to try and run it:

```
[user@host mapserver]$ ./mapserv
This script can only be used to decode form results and
should be initiated as a CGI process via a httpd server.
[user@host mapserver]$
```

The message above is perfectly normal, and means exactly what it says. If you get anything else, something went terribly wrong.

4.1.8 Installation

MapServer binary

The MapServer program itself consists of only one file, the “mapserv” binary executable. This is a CGI executable, meant to be called and run by your web server.

In this section, we will assume you are running Apache under its default directory structure in `/usr/local/apache`. You may need to have privileges to edit your `httpd.conf` (the main apache configuration file), or have someone (such as your webmaster) help you with the configuration details.

The main goal is to get the “mapserv” binary installed in a publicly accessible directory that is configured to run CGI programs and scripts.

The basic install

Under a default configuration, the CGI directory is “`/usr/local/apache/cgi-bin`” (RedHat users will use “`/home/httpd/cgi-bin`”). Placing the `mapserv` file in this directory makes it accessible by the following URL: “`http://yourhostname.com/cgi-bin/mapserv`”. When accessing this URL through your web client, you should expect the following output if all has worked well: “No query information to decode. QUERY_STRING is set, but empty.” If you get this message, you’re done installing MapServer.

Common problems

File permissions

The most common problem one is likely to encounter when attempting to install the binary are permissions issues:

- You do not have write permissions into your web server’s CGI Directory. Ask your webmaster to install the file for you.
- The web server gives you a “403 Permission denied” error. Make sure the user the web server runs as (usually “nobody”) has execute permission on the binary executable. Making the file world executable is perfectly fine and safe:

```
[user@host cgi-bin]$ chmod o+x mapserv
```

Apache errors

You may receive a few different type of errors as well if your web server configuration isn’t right:

- 500 Internal server error: This is a fairly generic error message. All it basically tells you is that the web server was unsuccessful in running the program. You will have to consult the web server's error log to find out more, and may need to enlist the help of your webmaster/system administrator.

Where to go once you've got it compiled

The *An Introduction to MapServer* document provides excellent coverage of getting started with MapServer.

4.2 Compiling on Win32

Author Pericles Nacionales

Contact pnaciona at gmail.com

Revision \$Revision: 8380 \$

Date \$Date: 2009-01-02 10:27:46 -0800 (Fri, 02 Jan 2009) \$

Table of Contents

- Compiling on Win32
 - Introduction
 - Compiling
 - Set up a Project Directory
 - Download MapServer Source Code and Supporting Libraries
 - The MapServer source code
 - Set Compilation Options
 - Compile the Libraries
 - Compile MapServer
 - Compiling MapServer with PostGIS support
 - Common Compiling Errors
 - Installation
 - Other Helpful Information
 - Acknowledgements

4.2.1 Introduction

This document provides a simple set of compilation procedures for MapServer on Win32 platforms.

If you've made it this far, chances are you already know about MapServer and are at least tempted to try compiling it for yourself. Pre-compiled binaries for MapServer are available from a variety of sources. Refer to *Windows*. Building MapServer for win32 platforms can be a daunting task, so if existing binaries are sufficient for your needs, it is strongly advised that they be used in preference to trying to build everything from source.

However, there can be a variety of reasons to want to build MapServer from source on win32. Reasons include the need to enable specific options, to build with alternate versions of support libraries (such as GDAL), the desire for MapScript support not part of the core builds, the need to debug and fix bugs or even to implement new features in MapServer. To make it easy for users and developers, I've made a list of steps to compile MapServer. Background information is provided in each step, along with examples. Each example is a continuation of the previous one and in the end will produce the MapServer DLL (libmap.dll), the CGI program (the mapserv.exe), and utility programs.

Warning: This document may refer to older library versions. You may want to try to use more recent library versions for your build.

4.2.2 Compiling

If you are new to Windows programming, please follow this document carefully. The compilation steps are fairly simple but I've added a few blurbs in each step to help you understand how MapServer compiles. For the more experienced programmers, perhaps reading the README.Win32 that accompanies the MapServer source code would be more useful. For those who are antsy, compiling MapServer involves download and unpacking the source codes, editing the make files, and invoking Microsoft's Visual C++ compiler from the command prompt. The resulting mapserv.exe is the CGI program that installs in the cgi-bin directory of your web server.

For those who are willing to take the time, the compilation steps follow.

4.2.3 Set up a Project Directory

Before you start to compile MapServer, I recommend creating a directory called "projects" where you can put the source code for MapServer and its supporting libraries. Since you will be working with DOS-style commands, you might as well get used to the Windows command prompt. For Windows 95/98 users the command processor would be called command.com. For Windows NT/2000/XP, it would be cmd.exe. So fire up the old command prompt and go to the drive where you want to create the project directory.

Here is an example of how to create a directory called projects on the C: drive:

```
C:\Users> mkdir C:\Projects
```

To go to that directory:

```
C:\Users> cd \Projects
C:\Projects>
```

From the projects directory, you can extract the source codes for MapServer and its libraries. Now you're ready to download the source codes.

4.2.4 Download MapServer Source Code and Supporting Libraries

After creating a project directory, download the MapServer source code and the codes for the supporting libraries and save the source code packages in the newly created "projects" directory. These source codes are usually packaged as ZIP, or as UNIX TAR and GZIP files. You'll need a software that can unzip these packages. 7-Zip is an example of software that can handle these files.

Cygwin is a free, open-source software package which is a port of these tools on Windows. You can use the gzip and tar utilities from this tool collection. Cygwin is available from <http://www.cygwin.com>.

In order to compile the MapServer CGI program, you must download a few required and optional libraries. At its simplest configuration, MapServer only requires the GD (to provide the image output) and REGEX (to provide regular expression support) libraries. This configuration allows the developer/data provider to use shapefiles as input and, depending on the version of GD library used, GIF or PNG images as output. Additional libraries are needed for input data in alternative formats. The libraries that work with MapServer are listed below.

4.2.5 The MapServer source code

The MapServer source code can be downloaded from the *download page*. If you'd like to get the current development version of the software, following the nightly snapshot link under the Interim Builds title. The absolute latest copy of the source code can be obtained from SVN; however, the SVN repository does not contain several important source files (maplexer.c, mapparser.c and mapparser.h) normally generated on unix, so if possible, using a nightly snapshot is substantially easier than working directly from *Subversion*.

Required Libraries

GD Library: MapServer uses the GD graphics library for rendering map images in GIF, PNG and JPEG format. These map images are displayed in web browser clients using the MapServer CGI. The current official version of GD is 2.0.33. The distributed makefiles are setup to use the prebuilt GD Win32 DLL binaries which include GD, libjpeg, libpng, libz, libgif and FreeType 2 all within one DLL. This package is generally listed as “Windows DLL .zip” and the latest version is normally available at <http://www.boutell.com/gd/http/gdwin32.zip>.

Regex: Regex is the regular expression library used by MapServer. It can be downloaded at <http://ftp.gnu.org/old-gnu/regex/regex-0.12.tar.gz>

Optional Libraries

JPEG library: This library is required by GD to render JPEG images, if building GD from source. You may download this library at <http://www.ijg.org/files/jpegsrc.v6b.tar.gz>

PNG library: This library is required by GD to render PNG images, if building GD from source. You may download this library at <http://sourceforge.net/projects/libpng/>

Zlib: This library is required by libpng to provide graphics compression support. It can be downloaded along with the PNG library, or at <http://www.zip.org/zlib.zip>.

FreeType 2: FreeType provides TrueType support in MapServer via GD. We only need to build FreeType separately if building GD from source. It can be downloaded at <http://gnuwin32.sourceforge.net/packages/freetype.htm>.

PROJ.4: Proj.4 provides on-the-fly projection support to MapServer. Users whose data are in different projection systems can use this library to reproject into a common projection. It is also required for WMS, WFS or WCS services.

GDAL/OGR: The GDAL/OGR library allows MapServer to read a variety of geospatial raster formats (GDAL) and vector formats (OGR). It can be downloaded at <http://www.gdal.org/>.

ArcSDE: ArcSDE is an ESRI proprietary spatial database engine. Most users will not have access to it but if you have ArcSDE license, you can use its libraries to give MapServer access to SDE databases.

EPPL7: This library allows MapServer to read EPPL7 datasets, as well as the older Erdas LAN/GIS files. This library is set as a default library in MapServer so there's no special source code to download.

Now that you have reviewed the libraries that provide support to MapServer, it is time to decide which ones to compile and use. We will work with the pre-built GD distributed on Boutell.com with PNG, GIF, JPEG, and FreeType “built in”. If you want to provide OGC Web Services (ie. WMS, WFS) or want to perform on the fly reprojection then the PROJ.4 library will be needed. If you need additional raster and vector data sources consider including GDAL/OGR support. GDAL is also required for WCS service.

Our example calls for the required libraries and on-the-fly projection support so we need to download GD, regex, and Proj.4 libraries. Go ahead and get those libraries.

4.2.6 Set Compilation Options

MapServer, like many of its support libraries, comes with a Visual C++ makefile called Makefile.vc. It includes the file nmake.opt which contains many of the site specific definitions. We will only need to edit the nmake.opt file to configure the build for our local site options, and support libraries. The Makefile.vc, and nmake.opt template file have been provided by Assefa Yewondwossen, and the DM Solutions folks.

As of MapServer 4.4, the default MapServer build options only include GD, and regex. MapServer is built using the /MD option (which means MSVCRT.DLL should be used), so if any support libraries are being built statically (rather than as DLLs) we need to use /MD when building them as well. By default modern PROJ.4 builds use /MD so we should be able to use the default PROJ.4 build without tweaking.

The example will compile with the GDWin32 pre-built DLL as well as regex-0.12, and PROJ.4. The PROJ.4 support will ensure we can enable MapServer OGC-WMS compatibility. Use notepad or another text editor to open the nmake.opt file and make the following changes.

Comments

Use the pound sign (#) to comment out the lines that you want to disable, or remove the pound sign to enable an option for NMAKE.

A. Enable PROJ.4 support, and update the path to the PROJ.4 directory. Uncomment the PROJ= line, and the PROJ_DIR= line as follows, and update the PROJ_DIR path to point to your PROJ build.

```
# Reprojecting.
# If you would like mapserver to be able to reproject data from one
# geographic projection to another, uncomment the following flag
# Proj.4 distribution (cartographic projection routines). PROJ.4 is
# also required for all OGC services (WMS, WFS, and WCS).
#
# For PROJ_DIR use full path to Proj.4 distribution
PROJ=-DUSE_PROJ -DUSE_PROJ_API_H
PROJ_DIR=c:\projects\proj-4.4.9
```

If you look down later in the file, you can see that once PROJ is enabled, MapServer will be linked with proj_i.lib, the PROJ.4 stub library, meaning that MapServer will be using the PROJ.DLL as opposed to statically linking in PROJ.4.

1. Uncomment the WMS option.

```
# Use this flag to compile with WMS Server support.
# To find out more about the OpenGIS Web Map Server Specification go to
# http://www.opengis.org/
WMS=-DUSE_WMS_SVR
```

1. Update to use GD. Here's what it should look like in our example.

```
GD_DIR=c:/projects/gdwin32
GD_LIB=$(GD_DIR)/bgd.lib
```

Note: As distributed the GDWin32 binary build does not include the bgd.lib stub library. It is necessary to run the **makemsvcmport.bat** script in the gdwin32 directory first.

D. Make sure the regex path is set correctly. In order for the “delete” command in the “nmake /f makefile.vc clean” target to work properly it is necessary to use backslashes in the REGEX_DIR definition.

```
# REGEX Library
#
# VC++ does not include the REGEX library... so we must provide our one.
# The following definitions will try to build GNU regex-0.12 located in the
# regex-0.12 sub-directory.
# If it was not included in the source distribution, then you can get it from:
#
#   ftp://ftp.gnu.org/pub/gnu/regex/regex-0.12.tar.gz
# Provide the full path to the REGEX project directory
# You do not need this library if you are compiling for PHP mapscript.
# In that case the PHP regex library will be used instead
!IFDEF PHP
REGEX_DIR=c:\projects\regex-0.12
!ENDIF
```

Your Makefile is now set.

4.2.7 Compile the Libraries

Before compiling MapServer, you must first compile its supporting libraries. How this is done varies for each library. For the PROJ.4 library a **nmake /f makefile.vc** command in the proj-4.4.9src directory should be sufficient. The regex-0.12 code is actually built by the MapServer build process, so you don't need to do anything there.

Compiling libcurl

Previously, curl libraries can be compiled using the following command:

```
nmake /f makefile.vc6 CFG=release
```

This creates a static library, libcurl.lib, to which you compile against. Versions newer than version 7.10.x should be compiled as dynamic library. This is accomplished using the command:

```
nmake /f makefile.vc6 CFG=release-dll
```

You will then need to edit MapServer's nmake.opt to replace the CURL_LIB variable with this line:

```
CURL_LIB = $(CURL_DIR)/lib/libcurl_imp.lib
```

4.2.8 Compile MapServer

Once you have compiled the supporting libraries successfully, you are ready to take the final compilation step. If you have not already done so, open a command prompt and set the VC++ environment variables by running the vcvars32.bat usually located in **C:\Program Files\Microsoft Visual Studio\VC98\bin\vcvars32.bat**.

```
C:\Users> cd \projects\mapserver
C:\Projects\mapserver&> C:\Program Files\Microsoft Visual Studio\VC98\Bin\vcvars32.bat "
C:\Projects\mapserver>
```

```
Setting environment for using Microsoft Visual C++ tool.
C:\Projects\mapserver>
```

Now issue the command: **nmake /f Makefile.vc** and wait for it to finish compiling. If it compiles successfully, you should get `mapserver.lib`, `libmap.dll`, `mapserv.exe`, and other `.EXE` files. That's it for the compilation process. If you run into problems, read section 4 about compiling errors. You can also ask for help from the helpful folks in the MapServer-dev e-mail list.

4.2.9 Compiling MapServer with PostGIS support

To compile PostGIS support into MapServer, here's what you need to do:

1. download the PostgreSQL 8.0.1 (or later) source from: <ftp://ftp.heanet.ie/pub/postgresql/source/>
2. I extracted them to `C:\projects\postgresql-8.0.1`
3. download the [Microsoft Platform SDK](#) otherwise you get link errors on `shfolder.lib`.
4. compile `libpq` under `C:\projects\postgresql-8.0.1\src\interfaces\libpq` using the `win32.mak` makefile
5. copy everything from `C:\projects\postgresql-8.0.1\src\interfaces\libpq\release` to `C:\projects\postgresql-8.0.1\src\interfaces\libpq` as the MapServer makefile will try to find it there
6. Define the following in the `nmake.opt` for MapServer: `POSTGIS =-DUSE_POSTGIS POSTGIS_DIR =c:/projects/postgresql-8.0.1/src`
7. `nmake /f makefile.vc`
8. don't forget to copy `libpq.dll` (from `C:\projects\postgresql-8.0.1\src\interfaces\libpq\release`) into a location where MapServer can find it.

4.2.10 Common Compiling Errors

Following are a few common errors you may encounter while trying to build MapServer.

- Visual C++ Tools Not Properly Initialized.

```
C:\projects\mapserver> nmake -f /makefile.vc
'nmake' is not recognized as an internal or external command,
operable program or batch file.
```

This occurs if you have not properly defined the path and other environment variables required to use MS VisualC++ from the command shell. Invoke the `VCVARS32.BAT` script, usually with the command **C:\Program Files\Microsoft Visual Studio\VC98\bin\vcvars32.bat** or something similar if visual studio was installed in an alternate location. To test if VC++ is available, just type "nmake" or "cl" in the command shell and ensure it is found.

- Regex Build Problems.

```
regex.obj : error LNK2001: unresolved external symbol _printchar
libmap.dll : fatal error LNK1120: 1 unresolved externals
NMAKE : fatal error U1077: 'link' : return code '0x460'
Stop.
```

This occurs if you use the stock `regex-0.12` we referenced. I work around this by commenting out the "extern" statement for the `printchar()` function, and replacing it with a stub implementation in `regex-0.12\regex.c`.

```
//extern void printchar ();
void printchar( int i ) {}
```

- GD Import Library Missing.


```
LINK : fatal error LNK1104: cannot open file 'c:/projects/gdwin32/bgd.lib'  
NMAKE : fatal error U1077: 'link' : return code '0x450'  
Stop.
```

If you are using the pre-built GD binaries, you still need to run the **makemsvcmport.bat** script in the gdwin32 directory to create a VC++ compatible stub library (bgd.lib).

4.2.11 Installation

The file we are most interested in is mapserv.exe. The other executable files are the MapServer utility programs.

See Also:

MapServer Utilities

to learn more about these utilities.

To test that the CGI program is working, type mapserv.exe at the command prompt. You should see the following message:

```
This script can only be used to decode form results and  
should be initiated as a CGI process via a httpd server.
```

You may instead get a popup indicating that a DLL (such as bgd.dll) is missing. You will need to copy all the required DLLs (ie. bgd.dll, and proj.dll) to the same directory as the mapserv.exe program.

Now type mapserv -v at the command prompt to get this message:

```
MapServer version 4.4.0-beta3 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP  
SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER INPUT=SHAPEFILE  
DEBUG=MSDEBUG
```

This tells us what data formats and other options are supported by mapserv.exe. Assuming you have your web server set up, copy mapserv.exe, libmap.dll, bgd.dll, proj.dll and any other required DLLs to the cgi-bin directory.

You are now ready to download the demo application and try out your own MapServer CGI program. If you wish, you can also create a directory to store the utility programs. I'd suggest making a subdirectory called "bin" under the directory "projects" and copy the executables to that subdirectory. You might find these programs useful as you develop MapServer applications.

4.2.12 Other Helpful Information

The MapServer Unix Compilation and Installation HOWTO has good descriptions of some MapServer compilation options and library issues. I will write more about those options and issues on the next revision of this HOWTO.

The README documents of each of the supporting libraries provide compilation instructions for Windows.

The MapServer User community has a collective knowledge of the nuances of MapServer compilation. Seek their advice wisely.

4.2.13 Acknowledgements

Thanks to Assefa Yewondwossen for providing the Makefile.vc. I would not have been able to write this HOWTO without that file.

Thanks to Bart van den Eijnden for the libcurl and PostGIS compilation info.

Thanks to the Steve Lime for developing MapServer and to the many developers who contribute time and effort in order to keep the MapServer project successful.

4.3 PHP MapScript Installation

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8459 \$

Date \$Date: 2009-01-27 10:13:56 -0800 (Tue, 27 Jan 2009) \$

Table of Contents

- PHP MapScript Installation
 - Introduction
 - Obtaining, Compiling, and Installing PHP and the PHP/MapScript Module
 - FAQ / Common Problems

4.3.1 Introduction

The PHP/MapScript module is a PHP dynamically loadable module that makes MapServer's MapScript functions and classes available in a PHP environment.

The original version of MapScript (in Perl) uses [SWIG](#), but since SWIG does not support the PHP language, the module has to be maintained separately and may not always be in sync with the Perl version.

The PHP module was developed by [DM Solutions Group](#) and is currently maintained by [Mapgears](#).

This document assumes that you are already familiar with certain aspects of your operating system:

- For Unix/Linux users, a familiarity with the build environment, notably *make*.
- For Windows users, some compilation skills if you don't have ready access to a pre-compiled installation and need to compile your own copy of MapServer with the PHP/MapScript module.

Which version of PHP is supported?

PHP MapScript was originally developed for PHP-3.0.14 but after MapServer 3.5 support for PHP3 has been dropped and as of the last update of this document, PHP 4.3.11 or more recent was required (PHP5 is well supported).

The best combinations of MapScript and PHP versions are:

- MapScript 4.10 with PHP 5.2.1 and up
- MapScript 4.10 with PHP 4.4.6 and up

How to Get More Information on the PHP/MapScript Module for MapServer

- For a list of all classes, properties, and methods available in the module see the [PHP MapScript](#) reference document.
- More information on the PHP/MapScript module can be found on the [PHP/MapScript page](#) on MapTools.org.

- The [MapServer Wiki](#) also has PHP/MapScript build and installation notes and some php code snippets.
- Questions regarding the module should be forwarded to the [MapServer mailing list](#).

4.3.2 Obtaining, Compiling, and Installing PHP and the PHP/MapScript Module

Download PHP and PHP/MapScript

- The PHP source or the Win32 binaries can be obtained from the [PHP web site](#).
- Once you have verified that PHP is installed and is running, you need to get the latest [MapServer source](#) and compile MapServer and the PHP module.

Setting Up PHP on Your Server

Unix

- Check if you have PHP already installed (several Linux distributions have it built in).
- If not, see the PHP manual's "[Installation on Unix systems](#)" section.

Windows

- [MS4W \(MapServer For Windows\)](#) is a package that contains Apache, PHP, and PHP/MapScript ready to use in a simple zipfile. Several Open Source applications are also available for use in MS4W.
- Windows users can follow steps in the [Installing Apache, PHP and MySQL on Windows tutorial](#) to install Apache and PHP manually on their system.
- Window users running PWS/IIS can follow [php.net's howto](#) for installing PHP for PWS/IIS 3, PWS 4 or newer, and IIS 4 or newer.

Note: When setting up PHP on Windows, make sure that PHP is configured as a CGI and not as an Apache module because `php_mapscript.dll` is not thread-safe and does not work as an Apache module (See the [Example Steps of a Full Windows Installation](#) section of this document).

Build/Install the PHP/MapScript Module

Building on a Linux Box

NOTE: For UNIX users, see the README.CONFIGURE file in the MapServer source, or see the [Compiling on Unix HowTo](#).

- The main MapServer configure script will automatically setup the main makefile to compile `php_mapscript.so` if you pass the `-with-php=DIR` argument to the configure script.
- Copy the `php_mapscript.so` library to your PHP extensions directory, and then use the `dl()` function to load the module at the beginning of your PHP scripts. See also the PHP function [extension_loaded\(\)](#) to check whether an extension is already loaded.
- The file [mapscript/php3/examples/phpinfo_mapscript.phtml](#) will test that the `php_mapscript` module is properly installed and can be loaded.
- If you get an error from PHP complaining that it cannot load the library, then make sure that you recompiled and reinstalled PHP with support for dynamic libraries. On RedHat 5.x and 6.x, this means adding "`-rdynamic`" to the `CLDFLAGS` in the main PHP3 Makefile after running `./configure`. Also make sure all directories in the path to the location of `php_mapscript.so` are at least `r-x` for the HTTPd user (usually 'nobody'), otherwise `dl()` may complain that it cannot find the file even if it's there.

Building on Windows

- For Windows users, it is recommended to look for a precompiled binary for your PHP version on the [MapServer download page](#) or on [MapTools.org](#).
- If for some reason you really need to compile your own Windows binary then see the README.WIN32 file in the MapServer source (good luck!).

Installing PHP/MapScript

Simply copy the file `php4_mapscript.dll` to your PHP4 extensions directory (`pathto/php/extensions`)

Using `phpinfo()`

To verify that PHP and PHP/MapScript were installed properly, create a `.php` file containing the following code and try to access it through your web server:

```
<HTML>
<BODY>

<?php
    if (PHP_OS == "WINNT" || PHP_OS == "WIN32")
    {
        dl("php_mapscript.dll");
    }
    else
    {
        dl("php_mapscript.so");
    }
    phpinfo();
?>

</BODY>
</HTML>
```

If PHP and PHP/MapScript were installed properly, several tables should be displayed on your page, and 'MapScript' should be listed in the 'Extensions' table.

Example Steps of a Full Windows Installation

Using MS4W (MapServer for Windows)

1. Download the latest [MS4W base package](#).
2. Extract the files in the archive to the root of one of your drives (e.g. `C:/` or `D:/`).
3. Double-click the file `/ms4w/apache-install.bat` to install and start the Apache Web server.
4. In a web browser goto <http://127.0.0.1>. You should see an MS4W opening page. You are now running PHP, PHP/MapScript, and Apache.
5. You can now optionally install other applications that are pre-configured for MS4W, which are located on the [MS4W download page](#).

Manual Installation Using Apache Server

1. Download the [Apache Web Server](#) and extract it to the root of a directory (eg. `D:/Apache`).
2. Download [PHP4](#) and extract it to your Apache folder (eg. `D:/Apache/PHP4`).

3. Create a temp directory to store MapServer created GIFs. NOTE: This directory is specified in the IMAGEPATH parameter of the WEB Object in the *Mapfile* reference. For this example we will call the temp directory "ms_tmp" (eg. E:/tmp/ms_tmp).
4. Locate the file *httpd.conf* in the conf directory of Apache, and open it in a text viewer (eg. TextPad, Emacs, Notepad).

In the *Alias* section of this file, add aliases to the ms_tmp folder and any other folder you require (for this example we will use the *msapps* folder):

```
Alias    /ms_tmp/      "path/to/ms_tmp/"
Alias    /msapps/     "path/to/msapps/"
```

In the *ScriptAlias* section of this file, add an alias for the PHP4 folder.

```
ScriptAlias    /cgi-php4/      "path/to/apache/php4/"
```

In the *AddType* section of this file, add a type for php4 files.

```
AddType application/x-httpd-php4    .php
```

In the *Action* section of this file, add an action for the php.exe file.

```
Action application/x-httpd-php4    "/cgi-php4/php.exe"
```

5. Copy the file *php4.ini-dist* located in your Apache/php4 directory and paste it into your WindowsNT folder (eg. c:/winnt), and then rename this file to *php.ini* in your WindowsNT folder.
6. If you want specific extensions loaded by default, open the *php.ini* file in a text viewer and uncomment the appropriate extension.
7. Place the file *php_mapscript.dll* into your Apache/php4/extensions folder.

Installation Using Microsoft's IIS

(please see the *IIS Setup for MapServer* document for uptodate steps)

1. Install IIS if required (see the [IIS 4.0 installation procedure](#)).
2. Install PHP and PHP/MapScript (see above).
3. Open the Internet Service Manager (eg. C:/WINNT/system32/inetsrv/inetmgr.exe).
4. Select the Default web site and create a virtual directory (right click, select New/Virtual directory). For this example we will call the directory *msapps*.
5. In the Alias field enter *msapps* and click Next.
6. Enter the path to the root of your application (eg. "c:/msapps") and click Next.
7. Set the directory permissions and click Finish.
8. Select the msapps virtual directory previously created and open the directory property sheets (by right clicking and selecting properties) and then click on the Virtual directory tab.
9. Click on the Configuration button and then click the App Mapping tab.
10. Click Add and in the Executable box type: *path/to/php4/php.exe %s %s*. You MUST have the *%s %s* on the end, PHP will not function properly if you fail to do this. In the Extension box, type the file name extension to be associated with your PHP scripts. Usual extensions needed to be associated are phtml and php. You must repeat this step for each extension.

11. Create a temp directory in Explorer to store MapServer created GIFs.

Note: This directory is specified in the IMAGEPATH parameter of the WEB Object in the *Mapfile*. For this example we will call the temp directory *ms_tmp* (eg. C:/tmp/ms_tmp).

12. Open the Internet Service Manager again.
13. Select the Default web site and create a virtual directory called *ms_tmp* (right click, select New/Virtual directory). Set the path to the ms_tmp directory (eg. C:/tmp/ms_tmp) . The directory permissions should at least be set to Read/Write Access.

4.3.3 FAQ / Common Problems

Questions Regarding Documentation

Q Is there any documentation available?

- A The main reference document is the *PHP MapScript reference*, which describes all of the current classes, properties and methods associated with the PHP/MapScript module.

To get a more complete description of each class and the meaning of their member variables, see the *MapScript reference* and the *MapFile reference*.

The [MapServer Wiki](#) also has PHP/MapScript build and installation notes and some php code snippets.

Q Where can I find sample scripts?

- A Some examples are included in directory *mapserver/mapsript/php3/examples/* in the MapServer source distribution. A good one to get started is *test_draw_map.phtml*: it's a very simple script that just draws a map, legend and scalebar in an HTML page.

A good intermediate example is the *PHP MapScript By Example guide* (note that this document was created for an earlier MapServer version but the code might be still useful).

The next example is the [GMap demo](#). You can download the whole source and data files from the [MapTools.org download page](#).

Questions About Installation

Q How can I tell that the module is properly installed on my server?

- A Create a file called *phpinfo.phtml* with the following contents:

```
<?php dl("php_mapscript.so");
      phpinfo();
?>
```

Make sure you replace the *php_mapscript.so* with the name under which you installed it, it could be *php_mapscript_46.so* on Unix, or *php_mapscript_46.dll* on Windows

You can then try the second test page *mapserver/mapsript/php3/examples/test_draw_map.phtml*. This page simply opens a MapServer .map file and inserts its map, legend, and scalebar in an HTML page. Modify the page to access one of your own MapServer .map files, and if you get the expected result, then everything is probably working fine.

Q I try to display my .phtml or .php page in my browser but the page is shown as it would it Notepad.

A The problem is that your PHP installation does not recognize “.phtml” as a PHP file extension. Assuming you’re using PHP4 under Apache then you need to add the following line with the other PHP-related AddType lines in the httpd.conf:

```
AddType application/x-httpd-php .phtml
```

For a more detailed explanation, see the *Example Steps of a Full Windows Installation* section of this document.

Q I installed the PROJ.4, GDAL, or one of the support libraries on my system, it is recognized by MapServer’s “configure” as a system lib but at runtime I get an error: “libproj.so.0: No such file or directory”.

A You are probably running a RedHat Linux system if this happened to you. This happens because the libraries install themselves under /usr/local/lib but this directory is not part of the runtime library path by default on your system.

(I’m still surprised that “configure” picked proj.4 as a system lib since it’s not in the system’s lib path...probably something magic in autoconf that we’ll have to look into)

There are a couple of possible solutions:

1. Add a “setenv LD_LIBRARY_PATH” to your httpd.conf to contain that directory
2. Edit /etc/ld.so.conf to add /usr/local/lib, and then run “/sbin/ldconfig”. This will permanently add /usr/local/lib to your system’s runtime lib path.
3. Configure MapServer with the following options:

```
--with-proj=/usr/local --enable-runpath
```

and the /usr/local/lib directory will be hardcoded in the exe and .so files

I (Daniel Morissette) personally prefer option #2 because it is permanent and applies to everything running on your system.

Q Does PHP/MapScript have to be setup as a CGI? If so, why?

A Yes, please see the [PHP/MapScript CGI page](#) in the MapServer Wiki for details.

Q I have compiled PHP as a CGI and when PHP tries to load the php_mapscript.so, I get an “undefined symbol: _register_list_destructors” error. What’s wrong?

A Your PHP CGI executable is probably not linked to support loading shared libraries. The MapServer configure script must have given you a message about a flag to add to the PHP Makefile to enable shared libs.

Edit the main PHP Makefile and add “-rdynamic” to the LDFLAGS at the top of the Makefile, then relink your PHP executable.

Note: The actual parameter to add to LDFLAGS may vary depending on the system you’re running on. On Linux it is “-rdynamic”, and on *BSD it is “-export-dynamic”.

Q What are the best combinations of MapScript and PHP versions?

A The best combinations are:

- MapScript 4.10 with PHP 5.2.1 and up
- MapScript 4.10 with PHP 4.4.6 and up

Q I am dynamically loading gd.so and php_mapscript.so and running into problems, why?

A The source of the problems could be a mismatch of GD versions. The PHP GD module compiles its own version of libgd, and if the GD library is loaded before the mapscript library, mapscript will use the php-specific version. Wherever possible you should use a gd.so built with the same GD as PHPMapScript. A workaround is to load the php_mapscript module before the GD module.

4.4 .NET MapScript Compilation

Author Tamas Szekeres

Contact szekerest at gmail.com

Revision \$Revision: 8386 \$

Date \$Date: 2009-01-04 14:03:29 -0800 (Sun, 04 Jan 2009) \$

4.4.1 Compilation

Before compiling C# MapScript you should compile MapServer with the options for your requirements. For more information about the compilation of MapServer please see *Win32 Compilation and Installation Guide*. It is highly recommended to minimize the library dependency of your application, so when compiling MapServer enable only the features really needed. To compile the C# binding SWIG 1.3.31 or later is required.

Warning: This document may refer to older library versions. You may want to try to use more recent library versions for your build.

Win32 compilation targeting the MS.NET framework 1.1

You should compile MapServer, MapScript and all of the subsequent libraries using Visual Studio 2003. Download and uncompress the latest SWIGWIN package that contains the precompiled swig.exe Open the Visual Studio .NET 2003 Command Prompt and step into the /mapscript/csharp directory. Edit makefile.vc and set the SWIG variable to the location of your swig.exe

Use

```
nmake -f makefile.vc
```

to compile mapscript.dll and mapscript_csharp.dll.

Win32 compilation targeting the MS.NET framework 2.0

You should compile MapServer, MapScript and all of the subsequent libraries using Visual Studio 2005. Download and uncompress the latest SWIGWIN package that contains the precompiled swig.exe. Open the Visual Studio 2005 Command Prompt and step into the /mapscript/csharp directory. Edit makefile.vc and set the SWIG variable to the location of your swig.exe.

Use

```
nmake -f makefile.vc
```

to compile mapscript.dll and mapscript_csharp.dll.

Win32 compilation targeting the MONO framework

Before the compilation you should download and install the recent mono Win32 setup package (eg. mono-1.1.13.2-gtksharp-2.8.1-win32-1.exe). Edit makefile.vc and set the CSC variable to the location of your mcs.exe. Alternatively you can define

```
MONO = YES
```

in your nmake.opt file.

You should use the same compiler for compiling MapScript as the compiler has been used for the MapServer compilation. To compile MapScript open the Command Prompt supplied with your compiler and use

```
nmake -f makefile.vc
```

to compile mapscript.dll and mapscript_csharp.dll.

Alternative compilation methods on Windows

Beginning from MapServer 4.8.3 you can invoke the C# compilation from the MapServer directory by uncommenting DOT_NET in nmake.opt

```
# ~~~~~
# .NET/C# MapScript
# -----
# .NET will of course only work with MSVC 7.0 and 7.1. Also note that
# you will definitely want USE_THREAD defined.
# ~~~~~
#DOT_NET = YES
```

and invoking the compilation by

```
nmake -f makefile.vc csharp
```

You can also use

```
nmake -f makefile.vc install
```

for making the compilation and copying the targets into a common output directory.

Testing the compilation

For testing the compilation and the runtime environment you can use

```
nmake -f makefile.vc test
```

within the csharp directory for starting the sample applications compiled previously. Before making the test the location of the corresponding libraries should be included in the system PATH.

Linux compilation targeting the MONO framework

Before the compilation you should download and install the recent mono Linux package. Some distributions have pre-compiled binaries to install, but for using the latest version you might want to compile and install it from the source. Download and uncompress the latest SWIG release. You should probably compile it from the source if pre-compiled binaries are not available for your platform.

Before compiling MapScript, MapServer should be configured and compiled. Beginning from MapServer 4.8.2 during configuration the mapsript/csharp/Makefile will be created according to the configuration options. Edit this file and set the SWIG and CSC for the corresponding executable paths if the files could not be accessed by default. To compile at a console step into the /mapsript/csharp directory use.

```
make
```

to compile libmapsript.so and mapsript_csharp.dll.

For testing the compilation and the runtime environment you can use

```
make test
```

for starting the sample applications compiled previously.

OSX compilation targeting the MONO framework

Beginning from 4.10.0 the csharp/Makefile supports the OSX builds. Before making the build the recent MONO package should be installed on the system.

Before compiling MapScript, MapServer should be configured and compiled. Beginning from MapServer 4.8.2 during configuration the mapsript/csharp/Makefile will be created according to the configuration options. Edit this file and set the SWIG and CSC for the corresponding executable paths if the files could not be accessed by default. To compile at a console step into the /mapsript/csharp directory use

```
make
```

to compile libmapsript.dylib and mapsript_csharp.dll.

For testing the compilation and the runtime environment you can use

```
make test
```

for starting the sample applications compiled previously.

To run the applications mapsript_csharp.dll.config is needed along with the mapsript_csharp.dll file. This file is created during the make process

4.4.2 Installation

The files required for your application should be manually installed. It is highly recommended to copy the files into the same folder as the executable resides.

4.4.3 Known issues

Visual Studio 2005 requires a manifest file to load the CRT native assembly wrapper

If you have compiled MapServer for using the CRT libraries and you are using the MS.NET framework 2.0 as the execution runtime you should supply a proper manifest file along with your executable, like:

```
<?xml version="1.0" encoding="utf-8"?>
<assembly xsi:schemaLocation="urn:schemas-microsoft-com:asm.v1 assembly.adaptive.xsd" manifestVersion="1.0" >
<assemblyIdentity name="drawmap.exe" version="1.0.0.0" type="win32" />
<dependency>
<dependentAssembly asmv2:dependencyType="install" asmv2:codebase="Microsoft.VC80.CRT.manifest" asmv2:processorArchitecture="x86" asmv2:publicKeyToken="1fc8b3b9a1e18e3b" />
<assemblyIdentity name="Microsoft.VC80.CRT" version="8.0.50608.0" publicKeyToken="1fc8b3b9a1e18e3b" processorArchitecture="x86" />
<hash xmlns="urn:schemas-microsoft-com:asm.v2">
<dsig:Transforms>
<dsig:Transform Algorithm="urn:schemas-microsoft-com:HashTransforms.Identity" />
</dsig:Transforms>
<dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<dsig:DigestValue>UMOlhUBGeKRrrg9DaaPNgyhRjyM=</dsig:DigestValue>
</hash>
</dependentAssembly>
</dependency>
</assembly>
```

This will inform the CLR that your exe depends on the CRT and the proper assembly wrapper is to be used. If you are using the IDE the manifest file could be pregenerated by adding a reference to Microsoft.VC80.CRT.manifest within the /Microsoft Visual Studio 8/VC/redist/x86/Microsoft.VC80.CRT directory.

Manifests for the dll-s must be embedded as a resource

According to the windows makefile the MapScript compilation target (mapscript.dll) is linked with the /MD option. In this case the VS2005 linker will generate a manifest file containing the unmanaged assembly dependency. The sample contents of the manifest file are:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<assembly xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>
<dependency>
<dependentAssembly>
<assemblyIdentity type='win32' name='Microsoft.VC80.CRT' version='8.0.50608.0' processorArchitecture='x86' />
</dependentAssembly>
</dependency>
</assembly>
```

Like previously mentioned if you are creating a windows application the common language runtime will search for a manifest file for the application. The name of the manifest file should be the same as the executable append and end with the .manifest extension. However if the host process is not controlled by you (like web mapping applications using aspnet_wp.exe as the host process) you will not be certain if the host process (.exe) will have a manifest containing a reference to the CRT wrapper. In this case you may have to embed the manifest into the dll as a resource using the mt tool like:

```
mt /manifest mapscript.dll.manifest /outputresource:mapscript.dll;#2
```

the common language runtime will search for the embedded resource and load the CRT assembly properly.

Normally it is enough to load the CRT with the root dll (mapscript.dll), but it is not harmful embedding the manifest into the dependent libraries as well.

Issue with regex and Visual Studio 2005

When compiling with Microsoft Visual Studio 2005 variable name collision may occur between regex.c and crtdefs.h. For more details see:

<http://trac.osgeo.org/mapserver/ticket/1651>

C# MapScript library name mapping with MONO

Using the MapScript interface created by the SWIG interface generator the communication between the C# wrapper classes (mapscript_csharp.dll) and the C code (mapscript.dll) takes place using platform invoke like:

```
[DllImport("mapscript", EntryPoint="CSharp_new_mapObj")]  
public static extern IntPtr new_mapObj(string jarg1);
```

The DllImport declaration contains the library name, however to transform the library name into a file name is platform dependent. On Windows the library name is simply appended with the .dll extension (mapscript.dll). On the Unix systems the library file name normally starts with the lib prefix and appended with the .so extension (libmapscript.so).

Mapping of the library name may be manually controlled using a dll.config file. This simply maps the library file the DllImport is looking for to its unix equivalent. The file normally contains the following information (mapscript_csharp.dll.config):

```
<configuration>  
<dllmap dll="mapscript" target="libmapscript.so" />  
</configuration>
```

and with the OSX builds:

```
<configuration>  
<dllmap dll="mapscript" target="libmapscript.dylib" />  
</configuration>
```

The file should be placed along with the corresponding mapscript_csharp.dll file, and created by default during the make process. For more information see:

<http://trac.osgeo.org/mapserver/ticket/1596> http://www.mono-project.com/Interop_with_Native_Libraries

Localization issues with MONO/Linux

According to <http://trac.osgeo.org/mapserver/ticket/1762> MapServer may not operate equally well on different locale settings. Especially when the decimal separator is other than “.” inside the locale of the process may cause parse errors when the mapfile contains float numbers. Since the MONO process takes over the locale settings of the environment it is worth considering to set the default locale to “C” of the host process, like:

```
LC_ALL=C mono ./drawmap.exe ../../tests/test.map test_csharp.png
```

4.4.4 Most frequent errors

This chapter will summarize the most frequent problems the user can run into. The issues were collected mainly from the -users list and the IRC.

Unable to load dll (MapScript)

You can get this problem on Windows and in most cases it can be dedicated to a missing or an unloadable shared library. The error message talks about mapsript.dll but surely one or more of the dll-s are missing that libmap.dll depends on. So firstly you might want to check for the dependencies of your libmap.dll in your application directory. You can use the Visual Studio Dependency Walker to accomplish this task. You can also use a file monitoring tool (like SysInternal's filemon) to detect the dll-s that could not be loaded. I propose to store all of the dll-s required by your application in the application folder. If you can run the drawmap C# sample application with your mapfile your compilation might be correct and all of the dlls are available.

You may find that the MapScript C# interface behaves differently for the desktop and the ASP.NET applications. Although you can run the drawmap sample correctly you may encounter the dll loading problem with the ASP.NET applications. When creating an ASP.NET project your application folder will be 'Inetpubwwwroot[YourApp]bin' by default. The host process of the application will aspnet_wp.exe or w3wp.exe depending on your system. The application will run under a different security context than the interactive user (under the context of the ASPNET user by default). When placing the dll-s outside of your application directory you should consider that the PATH environment variable may differ between the interactive and the ASPNET user and/or you may not have enough permission to access a dll outside of your application folder.

4.4.5 Bug reports

If you find a problem dedicated to the MapScript C# interface feel free to file a bug report to the [Issue Tracker](#).

4.5 IIS Setup for MapServer

Author Debbie Paquirek

Last Updated 2005/12/12

Table of Contents

- IIS Setup for MapServer
 - Base configuration
 - Php.ini file
 - Internet Services Manager
 - Under the tree for your new website - add virtual directories for
 - Test PHP
 - Mapfiles for IIS
 - Configuration files:

Some help on how to set up MapServer/Chameleon/PhpPgAdmin on Microsoft IIS (v5.0). Contains note on changes to the php.ini file and necessary changes to the MapServer mapfiles. Please contribute or make changes as required.

4.5.1 Base configuration

- Windows 2000
- IIS 5.0
- MS4W 1.2.1
- Chameleon 2.2
- PHP 4.3.11
- MapServer 4.7
- PhpPgAdmin 3.5.4 (if using postgresql/postgis)
- Postgres 8.0.3 (if using postgresql/postgis)
- Postgis 1.0.3 (if using postgresql/postgis)

This setup assumes that MS4W was unzipped to form c:\ms4w\ directory.

4.5.2 Php.ini file

- session.save_path (absolute path to your tmp directory)
- extension_dir (relative path to your php/extensions directory)
- cgi.force_redirect = 0
- enable the pg_sql extension (php_pgsql.dll) (for Postgresql)

4.5.3 Internet Services Manager

Under your website tree, create a new website (e.g. msprojects). View the properties for the new website.

Web Site Tab • set the IP address and under the Advanced tab put the complete Host Header name (e.g.msprojects.gc.ca).

Home Directory Tab • content should come from: A directory located on this computer.

- Local Path: c:\ms4w\Apache\htdocs
- Read access + whatever else you need
- Execute Permissions: Scripts only
- Configuration button - App Mappings (Add extensions .php and .phtml, Executable is c:\ms4w\Apache\cgi-bin\php.exe,select All verbs, Script Engine, and check that file exists

Documents Tab • Add index.phtml and index.html

- **Directory Security Tab** – Anonymous access amd authentication control
 - Select Anonymous access and the edit button should indicate the IUSR_account

Server Extensions Tab • Enable authoring is selected and client scripting says Javascript

4.5.4 Under the tree for your new website - add virtual directories for

cgi-bin Under Properties, virtual directory tab Local Path should point to `c:\ms4w\apache\cgi-bin`. Select Read. Execute Permissions should say “scripts and executables”

ms_tmp Under Properties, virtual directory tab Local Path should point to `c:\ms4w\tmp\ms_tmp`. Select Read, Write. Execute Permissions should say “scripts only”. This is where temporary images are written to so in the File system Security tab (use windows explorer), the `c:\ms4w\tmp\ms_tmp` directory should have permissions set for the Internet Guest Account (Read and execute, Read, Write, List Folder Contents).

tmp Under Properties, virtual directory tab Local Path should point to `c:\ms4w\tmp`. Select Read, Write. Execute Permissions should say “scripts only”. This is where chameleon writes sessions to so in the File system Security tab (use windows explorer), the `c:\ms4w\tmp` directory should have permissions set for the Internet Guest Account (Read and execute, Read, Write, List Folder Contents).

chameleon Under Properties, virtual directory tab Local Path should point to `C:\ms4w\apps\chameleon\htdocs`. Select Read. Execute Permissions should say “scripts only”. Under the Chameleon tree, you can add virtual directories for admin (`c:\ms4w\apps\chameleon\admin\htdocs`), samples (`c:\ms4w\apps\chameleon\samples\htdocs`), cwc2 (`c:\ms4w\apps\chameleon\cwc2\htdocs`)

phpPgAdmin If using postgresql/postgis, under Properties, virtual directory tab Local Path should point to `C:\ms4w\Apache\htdocs\phpPgAdmin`. Select Read, Write. Execute Permissions should say “scripts and executables”. Under Documents - add `index.php`.

Note: We had to unzip the phppgadmin package into this directory in order to get phppgadmin to show us the login page at <http://yourserver/phppgadmin/index.php>. You might want additional security on this directory.

gmap Good for testing purposes. Remember to change your mapfiles as discussed in Mapfiles for IIS below. Under Properties, virtual directory tab Local Path should point to `C:\ms4w\apps\gmap\htdocs`. Select Read. Execute Permissions should say “scripts only”.

4.5.5 Test PHP

In a command line window, navigate to `c:\ms4w\apache\cgi-bin` and run `php -i`. This should return the output that the `phpinfo()` function returns. I got an error about how it couldn't find `ntwdblib.dll`. I found this in `c:\ms4w\apache\php\dlls` and I copied it to the `cgi-bin` directory.

4.5.6 Mapfiles for IIS

- Add a config line to the MAP level of the mapfile

```
CONFIG PROJ_LIB "c:\ms4w\proj\nad"
```

- change the `IMAGEPATH` to be an absolute path to your `tmp/ms_tmp` folder

```
IMAGEPATH "c:\ms4w\tmp\ms_tmp"
```

4.5.7 Configuration files:

For Chameleon

```
C:\ms4w\apps\chameleon\config\chameleon.xml
C:\ms4w\apps\chameleon\config\cwc2.xml
```

For phppgadmin: (if using postgresql/postgis)

C:\ms4w\apps\phpPgAdmin\conf\config.inc.php

4.6 Oracle Installation

Author Till Adams

Last Updated 2007/02/16

Table of Contents

- Oracle Installation
 - Preface
 - System Assumptions
 - Compile MapServer
 - Set Environment Variables

4.6.1 Preface

This document explains the whole configuration needed to get the connect between MapServer *CGI* and an Oracle database server on a linux (Ubuntu) box. The aim of this document is just to put a lot of googled knowledge in ONE place. Hopefully it will preserve many of people spending analog amount of time than I did!

This manual was written, because I spent several days googling around to get my UMN having access to an oracle database. I'm NOT an oracle expert, so the aim of this document is just to put a lot of googled knowledge in ONE place. Hopefully it will preserve many of people spending analog amount of time than I did! (Or: If you have the choice: Try *PostGIS* ;-))

Before we start, some basic knowledge, I didn't know before:

- MapServer can access oracle spatial as well as geodata from any oracle locator installation! Oracle locator comes with every oracle instance, there is no need for an extra license.
- There is no need for further installation of any packages beside oracle/oracle OCI

4.6.2 System Assumptions

We assume that Oracle is already installed, there is a database and there is some geodata in the database. The following paths should be known by the reader:

- ORACLE_HOME
- ORACLE_SID
- ORACLE_BASE
- LD_LIBRARY_PATH

We also assume that you have installed **apache2** (our version was 2.0.49) and you are used to work with Linux/UNIX systems. We also think you are able to handle the editor *vi/vim*.

We ensure that the Oracle user who later accesses the database has write-access to the oracle_home directory.

We also assume, that you already have setup the tnsnames.ora file. It should look like that:


```

MY_ORACLE =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP) (HOST = host) (PORT = 1521))
    (CONNECT_DATA =
      (SERVICE_NAME = your_name)
    )
  )

```

It is important that you know the NAME of the datasource, in this example this is “MY_ORACLE” and will be used further on. Done that, you’re fine using `User/Password@MY_ORACLE` in your mapfile to connect to the oracle database. But first we have to do some more stuff.

4.6.3 Compile MapServer

Compile as normal compilation and set this flag:

```
--with-oraclespatial=/path/to/oracle/home/
```

If MapServer configure and make runs well, try

```
./mapserv -v
```

This should at least give this output:

```
INPUT=ORACLESPATIAL
```

If you got that, you’re fine from the MapServer point of view.

4.6.4 Set Environment Variables

It is important to set all environment variables correctly. There are one the one hand system-wide environment variables to be set, on the other hand there should be set some for the cgi-directory in your Apache configuration.

System Variables

On Ubuntu (and on many other systems) there is the file “/etc/profile” which sets environment variables for all users on the system (you may also dedicate user-specific environment variables by editing the users “.profile” file in their home directory, but usually the oracle database users are not users of the system with their own home)

Set the following variables:

```

$ cd /etc

$ echo export ORACLE_HOME=/path/to/oracle/home >> /etc/profile

# ** (e.g. ORACLE_HOME=/app/oracle/ora10g)

$ echo export ORACLE_BASE=/path/to/oracle >> /etc/profile

# ** (e.g. ORACLE_HOME=/app/oracle)

$ echo export ORACLE_SID=MY_ORACLE >> /etc/profile

```

```
$ echo export LD_LIBRARY_PATH=path/to/oracle/home/lib >> /etc/profile
# ** (e.g. ORACLE_HOME=/app/oracle/ora10g/lib)
```

The command comes silent, so there is no system output if you didn't mistype anything!

Setting the Apache Environment

Sometimes it is confusing WHERE to set WHAT in the splitted apache2.conf-files. In the folder "/etc/apache2/sites_available" you find your sites-file. If you did not do sth. Special e.g. installing virtual hosts, the file is named "default". In this file, the apache cgi-directory is defined. Our file looks like this:

```
ScriptAlias /cgi-bin/ /var/www/cgi-bin/
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
    Order allow,deny
    Allow from all
</Directory></p>
```

In this file, the local apache environment variables must be set. We did it within a location-block like this:

```
<Location "/cgi-bin/">
    SetEnv ORACLE_HOME "/path/to/oracle/home"
</Location></p>
```

Where /cgi-bin/ in the opening location block refers to the script alias /cgi-bin/ and the TNS_ADMIN directory point to the location of the tnsnames.ora file.

Then restart apache:

```
$ /etc/init.d/apache2 force-reload
```

Create mapfile

Before we start creating our mapfile ensure that you have a your access data (User/Password) and that you know the Oracle SRID, which could be different from the proj-*EPSG*!

The data access parameters:

- CONNECTIONTYPE oraclespatial
- CONNECTION 'user/password@MY_ORACLE'
- DATA 'GEOM FROM MY_LAYER USING SRID 82032'

[...]

Where:

- GEOM is the name of the geometry column
- MY_LAYER the name of the table
- 82032 is equivalent to the EPSG code 31468 (German projection system)

Testing & Error handling

So you are fine now. Load the mapfile in your application and try it. If everything goes well: Great, if not, possibly this ugly error-emssage occurs (this one cmae by querying MapServer through the WMS interface as a GetMap-request):

```
<ServiceExceptionReport version="1.0.1">
  <ServiceException>
    msDrawMap(): Image handling error. Failed to draw layer named 'test1'.
    msOracleSpatialLayerOpen(): OracleSpatial error. Cannot create OCI Handlers.
    Connection failure. Check the connection string. Error: .
  </ServiceException>
</ServiceExceptionReport>
```

This points us towards, that there might be a problem with the connection to the database. First of all, let's check, if the mapfile is all right. Therefore we use the MapServer utility program *shp2img*.

Let's assume you are in the directory, where you compiled MapServer and run shp2img:

```
$ cd /var/src/mapserver_version/

$ shp2img -m /path/to/mapfile/mapfile.map -i png -o /path/to/output/output.png
```

The output of the command should look like this:

```
[Fri Feb  2 14:32:17 2007].522395 msDrawMap(): Layer 0 (test1), 0.074s
[Fri Feb  2 14:32:17 2007].522578 msDrawMap(): Drawing Label Cache, 0.000s
[Fri Feb  2 14:32:17 2007].522635 msDrawMap() total time: 0.075s
```

If not, this possibly points you towards any error in your mapfile or in the way to access the data directly. In this case, take a look at *Oracle Spatial*. If there is a problem with your oracle connect, the same message as above (MsDrawMap() ...) occurs. Check your mapfile syntax and/or the environment settings for Oracle.

For Debian/Ubuntu it's worth also checking the file "/etc/environment" and test-wise to add the system variables comparable to System Variables

If the output is OK, you may have a look at the generated image (output.png). Then your problem reduces to the access of apache to oracle home directory. Carefully check your apache configuration. Please note, that the apache.config file differs in several linux-distributions. For this paper we talk about Ubuntu, which should be the same as Debian.

MAPFILE

Author Steve Lime

Contact steve.lime at dnr.state.mn.us

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Jean-François Doyon

Contact jdoyon at ccrs.nrcan.gc.ca

The Mapfile is the heart of MapServer. It defines the relationships between objects, points MapServer to where data are located and defines how things are to be drawn.

There are some important concepts that you must understand before you can reliably use mapfiles to configure MapServer. First is the concept of a *LAYER*. A layer is the combination of data plus styling. Data, in the form of attributes plus geometry, are given styling using *CLASS* and *STYLE* directives.

See Also:

An Introduction to MapServer for “An Introduction to the Mapfile”

5.1 CLASS

BACKGROUND_COLOR [r] [g] [b] Color to use for non-transparent symbols.

COLOR [r] [g] [b] Color to use for drawing features.

DEBUG [on|off] Enables debugging of the class object. Verbose output is generated and sent to the standard error output (STDERR) or the MapServer logfile if one is set using the LOG parameter in the *WEB* object.

See Also:

MS RFC 28: Redesign of LOG/DEBUG output mechanisms

EXPRESSION [string] Four types of expressions are now supported to define class membership. String comparisons, regular expressions, simple logical expressions, and string functions. If no expression is given, then all features are said to belong to this class.

- String comparisons are case sensitive and are the fastest to evaluate. No special delimiters are necessary although string must be quoted if they contain special characters. (As a matter of good habit, it is recommended you quote all strings).
- Regular expressions function just like previous versions of MapServer. However, you must now delimit a regular expression using `/regex/`. No quotes should be used.

- Logical expressions allow you to build fairly complex tests based on one or more attributes and therefore are only available with shapefiles. Logical expressions are delimited by parentheses “(expression)”. Attribute names are delimited by square brackets “[ATTRIBUTE]”. These names are case sensitive and must match the items in the shapefile. For example: EXPRESSION ([POPULATION] > 50000 AND [LANGUAGE] eq 'FRENCH') ... The following logical operators are supported: =,>,<,<=,>=,or,and,lt,gt,ge,le,eq,ne. As you might expect this level of complexity is slower to process.
- One string function exists: length(). This obviously computes the length of a string. An example follows:

```
EXPRESSION (length(' [NAME_E]') < 8)
```

String comparisons and regular expressions work from the classitem defined at the layer level. You may mix expression types within the different classes of a layer.

GROUP [string] Allows for grouping of classes. It is only used when a CLASSGROUP at the *LAYER* level is set. If the CLASSGROUP parameter is set, only classes that have the same group name would be considered at rendering time. An example of a layer with grouped classes might contain:

```
LAYER
...
CLASSGROUP "group1"
...
CLASS
  NAME "name1"
  GROUP "group1"
  ...
END
CLASS
  NAME "name2"
  GROUP "group2"
  ...
END
CLASS
  NAME "name3"
  GROUP "group1"
  ...
END
...
END # layer
```

KEYIMAGE [filename] Full filename of the legend image for the *CLASS*. This image is used when building a legend (or requesting a legend icon via MapScript or the *CGI application*).

LABEL Signals the start of a *LABEL* object.

MAXSCALEDENOM [double] Maximum scale at which this *CLASS* is drawn. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated MAXSCALE parameter.

See Also:

Map Scale

MAXSCALE [double] - deprecated Since MapServer 5.0 the proper parameter to use is MAXSCALEDENOM instead. The deprecated MAXSCALE is the maximum scale at which this CLASS is drawn. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000.

MAXSIZE [integer] Maximum size in pixels to draw a symbol. Default is 50.

MINSCALEDENOM [double] Minimum scale at which this *CLASS* is drawn. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated **MINSCALE** parameter.

See Also:

Map Scale

MINSCALE [double] - deprecated Since MapServer 5.0 the proper parameter to use is **MINSCALEDENOM** instead. The deprecated **MINSCALE** is the minimum scale at which this *CLASS* is drawn. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000.

MINSIZE [integer] Minimum size in pixels to draw a symbol. Default is 0.

NAME [string] Name to use in legends for this class. If not set class won't show up in legend.

OUTLINECOLOR [r] [g] [b] Color to use for outlining polygons and certain marker symbols. Line symbols do not support outline colors.

SIZE [integer] Height, in pixels, of the symbol/pattern to be used. Only useful with scalable symbols. For vector (and ellipse) symbol types the default size is based on the range of Y values in the **POINTS** defining the symbol. For pixmaps, the default is the vertical size of the image. Default size is 1 for TTF symbols.

STATUS [on|off] Sets the current display status of the class. Default turns the class on.

STYLE Signals the start of a *STYLE* object. A class can contain multiple styles.

SYMBOL [integer|string|filename] The symbol name or number to use for all features if attribute tables are not used. The number is the index of the symbol in the symbol file, starting at 1, the 5th symbol in the file is therefore symbol number 5. You can also give your symbols names using the **NAME** keyword in the symbol definition file, and use those to refer to them. Default is 0, which results in a single pixel, single width line, or solid polygon fill, depending on layer type.

You can also specify a gif or png filename. The path is relative to the location of the mapfile.

TEMPLATE [filename] Template file or URL to use in presenting query results to the user. See *Templating* for more info.

TEXT [string] Static text to label features in this class with. This overrides values obtained from the **LABELTIEM**. The string may be given as an expression delimited using the ()'s. This allows you to concatenate multiple attributes into a single label. For example: ([FIRSTNAME],[LASTNAME]).

You can also “stack” 2 symbols to achieve interesting effects. You define the second symbol, which effectively sits “on top” of the symbol normally defined above. See *Cartographic Symbol Construction with MapServer* for more information.

The following parameters allow you to define the symbol, and they are equivalent to their non-overlay counterparts:

- OVERLAYBACKGROUNDCOLOR
- OVERLAYCOLOR
- OVERLAYOUTLINECOLOR
- OVERLAYSIZE
- OVERLAYMINSIZE
- OVERLAYMAXSIZE
- OVERLAYSYMBOL

5.2 Expressions

Author Dirk Tilger

Contact dirk at MIRIUP.DE

Author Umberto Nicoletti

Contact umberto.nicoletti at gmail.com

Revision \$Revision: 8295 \$

Date \$Date: 2008-12-26 21:08:04 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2007/07/09

Contents

- Expressions
 - Introduction
 - Expression Types
 - “MapServer expressions”

5.2.1 Introduction

As of version 4.6.1, expressions are used in two places. They’re used to filter layers for specific records in the dataset and they’re used in *CLASS EXPRESSIONs* to specify to which items this *CLASS* does apply to.

5.2.2 Expression Types

Expression are used to match attribute values with certain logical checks. There are three different types of expressions you can use with MapServer:

- String comparisons: A single attribute is compared with a string value.
- Regular expressions: A single attribute is matched with a regular expression.
- Logical “MapServer expressions”: One or more attributes are compared using logical expressions.

String comparison

String comparison means as the name suggests that attribute values are checked if they are equal to some value. String comparison are the simplest form of MapServer expressions and the fastest option. To use a string comparison for filtering a *LAYER*, both *FILTERITEM* and *FILTER* must be set. *FILTERITEM* is set to the attribute name. *FILTER* is set to the value for comparison. The same rule applies to *CLASSITEM* and *EXPRESSION* in the *CLASS* object.

Example for a simple string comparison filter

```
FILTER "2005"  
FILTERITEM "year"
```

would match all records that have the attribute “year” set to “2005”. The rendered map would appear as if the dataset would only contain those items that have the “year” set to “2005”.

Similarly, a classification for the items matched above would be done by setting the *CLASSITEM* in the layer and the *EXPRESSION* in the class


```
LAYER
  NAME "example"
  CLASSITEM "year"
  ...
  CLASS
    NAME "year-2005"
    EXPRESSION "2005"
    ...
  END
END
```

For a reason explained later on the values for both CLASSITEM and FILTERITEM should start neither with an `'` nor with a `(` character.

Regular expression comparison

Regular expressions are a standard text pattern matching mechanism from the UNIX world. The functionality of regular expression matching is provided by the operating system on UNIX systems and therefore slightly operating system dependent. However their minimum set of features are those defined by the POSIX standard. The documentation of the particular regular expression library is usually in the “regex” manual page (“man regex”).

Regular expression with MapServer work similarly to string comparison, but allow more complex operation. They are slower than pure string comparisons, but might be still faster than logical expression. As with the string comparison use regular expressions, a FILTERITEM or a CLASSITEM has to defined, respectively.

A regular expression typically consists of characters with special meanings and characters that are interpreted as they are. Alphanumeric characters (A-Z, a-z and 0-9) are taken as they are. Characters with special means are:

- `.` will match a single character
- `[` and `]` are used for grouping. For example `[A-Z]` would match the characters A,B,C,...,X,Y,Z.
- `{`, `}`, and `*` are used to specify how often something should match.
- `^` matches the beginning, `$` matches the end of the value.
- The backslash `\` is used to take away the special meaning. For example `$` would match the dollar sign.

The following *LAYER* configuration would have all records rendered on the map that have “hotel” in the attribute named “placename”

```
LAYER
  NAME "regexp-example"
  FILTERITEM "placename"
  FILTER /hotel/
  ...
END
```

Note: The regular expression is case-sensitive, thus records having “Hotel” in them would not have matched.

Example: Match records that have a value for the current century (as of 2005 ;) in the attribute “year”

```
FILTERITEM "year"
FILTER /^20[0-9][0-9]/
```

Example: Match all the records that are either purely numerical or empty

```
FILTER /^[0-9]*$/
```

Note: If you experience frequently segmentation faults when working with MapServer and regular expressions, it might be that your current working environment is linked against more than one regular expression library. This can happen when MapServer is linked with components that bring their own copy, like the Apache httpd or PHP. In these cases the author has made best experiences with making all those components using the regular expression library of the operating system (i.e. the one in libc). That involved editing the build files of some of the components, however.

5.2.3 “MapServer expressions”

MapServer expressions are the most complex and depending how they are written can become quite slow. They can match any of the attributes and thus allow filtering and classification depending on more than one attribute. Besides pure logical operations there are also expressions allow also certain arithmetic, string- and time operations.

To be able to use a MapServer expression for a FILTER or EXPRESSION value, the expression has to be finally of a logical value.

Logical expressions

Syntactically, a logical expression is everything encapsulated in round brackets. Logical expressions take logical values as their input and return logical values. A logical expression is either ‘true’ or ‘false’.

- `((...) AND (...)) ((...) && (...)) ...` will become true when both of the two logical expressions in the innermost brackets are true.
- `((...) OR (...)) ((...) || (...)) ...` will become true when at least one of the two logical expressions in the innermost brackets is true.
- `NOT (...) ! (...) ...` will become true, when the logical expression in the brackets becomes false.

String operations that result in a logical value

Syntactically, a sting is something encapsulated in double-quotes.

- `(“String1” eq “String2”) (“String1” == “String2”) (“String1” = “String2”) ...` will become true when both strings are equal. This operation is identical to the MapServer string comparison described earlier.
- `(“String1” != “String2”) (“String1” ne “String2”) ...` will become true when both strings are not equal.
- `(“String1” < “String2”) (“String1” lt “String2”) ...` will become true when “String1” is lexicographically smaller than “String2”
- `(“String1” > “String2”) (“String1” gt “String2”) ...` will become true when “String1” is lexicographically larger than “String2”.
- `(“String1” <= “String2”) (“String1” le “String2”) ...` will become true when “String1” is not lexicographically larger than “String2”
- `(“String1” >= “String2”) (“String1” ge “String2”) ...` will become true when “String1” is not lexicographically smaller than “String2”.
- `(“String1” IN “token1,token2,...,tokenN”) ...` will become true when “String1” is in equal one of the given tokens.

Note:

The separator for those tokens is the comma. That means that you must not add unnecessary white space to the list and that you cannot compare to tokens that have a comma in it.

- `(“String1” =~ /regex/) ...` will become true, when “String1” matches the regular expression “regex”. This operation is identical to the regular expression matching described earlier.

String operations that return string values

There is only one operation for strings that returns a string value:

- “String1” + “String2 ... will return “String1String2”, thus the two string concatenated to each other.

Arithmetic expressions returning logical values

The basic element for the arithmetic operation is the number. There are some purely arithmetic operations that are returning numbers as their value. They will be covered in the next section.

- $(n1 \text{ eq } n2) (n1 \text{ == } n2) (n1 = n2) \dots$ will become true when both numbers are equal.
- $(n1 \text{ != } n2) (n1 \text{ ne } n2) \dots$ will become true when both numbers are not equal.
- $(n1 < n2) (n1 \text{ lt } n2) \dots$ will become true when $n1$ is smaller than $n2$
- $(n1 > n2) (n1 \text{ gt } n2) \dots$ will become true when $n1$ is larger than $n2$.
- $(n1 \leq n2) (n1 \text{ le } n2) \dots$ will become true when $n1$ is smaller or equal $n2$
- $(n1 \geq n2) (n1 \text{ ge } n2) \dots$ will become true when $n1$ is larger or equal $n2$.
- $(n1 \text{ IN } \text{“number1,number2,...,numberN”}) \dots$ will become true when $n1$ is equal to one of the given numbers.

Arithmetic expression returning a number

As stated in the previous section, MapServer can do purely numerical operations with numbers.

- $n1 + n2 \dots$ will become the sum of $n1$ and $n2$
- $n1 - n2 \dots$ will become $n2$ subtracted from $n1$
- $n1 * n2 \dots$ will become $n1$ multiplied with $n2$
- $n1 / n2 > \dots$ will become $n1$ divided by $n2$
- $-n1 \dots$ will become $n1$ with negated sign
- $n1 ^ n2 \dots$ will become $n1$ by a power of $n2$
- $\text{length}(\text{“String1”}) \dots$ will become the number of characters of “String1”

Note: When the numerical operations above are used like logical operations, the following rule applies: values equal to zero will be taken as ‘false’ and everything else will be ‘true’. That means the expression

:: $(6 + 5) \dots$

would evaluate as true, but

$(5 - 5) \dots$

...

would evaluate as false.

Temporal expressions

MapServer uses an internal time type to do comparison. To convert a keys value into this time type it will check the list below from the top down if the specified time matches and if so, it will do the conversion.

- YYYY-MM-DDTHH:MM:SSZ (‘Z’ and ‘T’ being the characters itself)

- YYYY-MM-DDTHH:MM:SS ('T' being the character itself)
- YYYY-MM-DD HH:MM:SS
- YYYY-MM-DDTHH:MM ('T' being the character itself)
- YYYY-MM-DD HH:MM
- YYYY-MM-DDTHH ('T' being the character itself)
- YYYY-MM-DD HH
- YYYY-MM-DD
- YYYY-MM
- YYYY
- THH:MM:SSZ ('Z' and 'T' being the characters itself)
- THH:MM:SS

For temporal values obtained this way, the following operations are supported:

- $(n1 \text{ eq } n2) (n1 == n2) (n1 = n2) \dots$ will become true when both times are equal.
- $(t1 \neq t2) (t1 \text{ ne } t2) \dots$ will become true when both times are not equal.
- $(t1 < t2) (t1 \text{ lt } t2) \dots$ will become true when t1 is earlier than t2
- $(t1 > t2) (t1 \text{ gt } t2) \dots$ will become true when t1 is later than t2.
- $(t1 \leq t2) (t1 \text{ le } t2) \dots$ will become true when t1 is earlier or same t2
- $(n1 \geq n2) (n1 \text{ ge } n2) \dots$ will become true when t1 is later or same t2.

How the attributes are referenced

To make a meaningful use of the expressions above, we need to get the attribute values into the expressions. That is done by enclosing the attribute key into square brackets, like this: [KEY]. Then before the expression is evaluated every occurrence of “[KEY]” will be replaced by the value for attribute “KEY”.

Example: how a simple string comparison would be evaluated. The filter is set to:

```
FILTER ( "[BUILDING_NAME]" == "National Museum" )
```

There is a attribute “BUILDING_NAME” and its value is “National Government Building”. Thus the expression actually evaluated is...

```
"National Government Building" == "National Museum" )
```

...and as such should be false.

Some layers do not really come with metadata. For raster layers for example special attributes have been defined that can be used for classification:

- [PIXEL] ... will become the pixel value as number
- [RED], [GREEN], [BLUE] ... will become the color value for the red, green and blue component in the pixel value, respectively.

Quotes escaping in strings

Note: Quotes escaping is not supported in MapServer versions lower than 5.0.

Starting with MapServer 5.0, if your dataset contains double-quotes, you can use a C-like escape sequence in the expression string. For example if your key “NAME” has the value ‘National “hero” statue’ you could write the FILTER expression as follows:

```
FILTER ( "[NAME]" == "National \"hero\" statue" )
...
```

to escape a single quote use the following sequence instead:

```
FILTER ( "[NAME]" == "National \'hero\' statue" )
```

5.3 FEATURE

POINTS A set of xy pairs terminated with an END, for example:

```
POINTS 1 1 50 50 1 50 1 1 END
```

Note: POLYGON/POLYLINE layers POINTS must start and end with the same point (i.e. close the feature).

ITEMS Comma separated list of the feature attributes:

```
ITEMS "value1;value2;value3"
```

Note: Specifying the same number of items is recommended for each features of the same layer. The item names should be specified as a PROCESSING option of the layer.

TEXT [string] String to use for labeling this feature.

WKT [string] A geometry expressed in OpenGIS Well Known Text geometry format. This feature is only supported if MapServer is built with OGR or GEOS support.

```
WKT "POLYGON((500 500, 3500 500, 3500 2500, 500 2500, 500 500))"
WKT "POINT(2000 2500)"
```

Note: Inline features should be defined as their own layers in the mapfile. If another CONNECTIONTYPE is specified in the same layer, MapServer will always use the inline features to draw the layer and ignore the other CONNECTIONTYPES.

5.4 FONTSET

Author Kari Guerts

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8295 \$

Date \$Date: 2008-12-26 21:08:04 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2008/10/08

Contents

- FONTSET
 - Format

5.4.1 Format

The format is very simple. Each line contains 2 items: an alias and the name/path of the font separated by white space. The alias is simply the name you refer to the font as in your *Mapfile* (eg. times-bold). The name is the actual name of the TrueType file. If not full path then it is interpreted as relative to the location of the fontset. Here's the fontset I use (the font.list file and all .ttf files are stored in the same sub-directory).

Note: Aliases are case sensitive. Excellent reference information about the TrueType format and online font resources is available from the [FreeType](#).

arial	arial.ttf
arial-bold	arialbd.ttf
arial-italic	ariali.ttf
arial-bold-italic	arialbi.ttf
arial_black	ariblk.ttf
comic_sans	comic.ttf
comic_sans-bold	comicbd.ttf
courier	cour.ttf
courier-bold	courbd.ttf
courier-italic	couri.ttf
courier-bold-italic	courbi.ttf
georgia	georgia.ttf
georgia-bold	georgiab.ttf
georgia-italic	georgiai.ttf
georgia-bold-italic	georgiaz.ttf
impact	impact.ttf
monotype.com	monotype.ttf
recreation_symbols	recreate.ttf
times	times.ttf
times-bold	timesbd.ttf
times-italic	timesi.ttf
times-bold-italic	timesbi.ttf
trebuchet_ms	trebuc.ttf
trebuchet_ms-bold	trebucbd.ttf
trebuchet_ms-italic	trebucit.ttf
trebuchet_ms-bold-italic	trebucbi.ttf
verdana	verdana.ttf
verdana-bold	verdanab.ttf
verdana-italic	verdanai.ttf
verdana-bold-italic	verdanaz.ttf

5.5 INCLUDE

When this directive is encountered parsing switches to the included file immediately. As a result the included file can be comprised of any valid mapfile syntax. For example:

```
INCLUDE 'myLayer.map'
```

Performance does not seem to be seriously impacted with limited use, however in high performance instances you may want to use includes in a pre-processing step to build a production mapfile. The C pre-processor can also be used (albeit with a different syntax) and is far more powerful.

5.5.1 Notes

- Supported in versions 4.10 and higher.
- The name of the file to be included **MUST be quoted** (single or double quotes).
- Includes may be nested, up to 5 deep.
- File location can be given as a full path to the file, or (in MapServer >= 4.10.1) as a path relative to the mapfile.
- Debugging can be problematic because:
 1. the file an error occurs in does not get output to the user
 2. the line number counter is not reset for each file. Here is one possible error that is thrown when the include file cannot be found:

```
msyylex(): Unable to access file. Error opening included file "parks_include.map"
```

5.5.2 Example

```
MAP
  NAME "include_mapfile"
  EXTENT 0 0 500 500
  SIZE 250 250

  INCLUDE "test_include_symbols.map"
  INCLUDE "test_include_layer.map"
END
```

where test_include_symbols.map contains:

```
SYMBOL
  NAME 'square'
  TYPE VECTOR
  FILLED TRUE
  POINTS 0 0 0 1 1 1 1 0 0 0 END
END
```

and test_include_layer.map contains:

```
LAYER
  TYPE POINT
  STATUS DEFAULT
  FEATURE
    POINTS 10 10 40 20 300 300 400 10 10 400 END
  END
  CLASS
    NAME 'Church'
    COLOR 0 0 0
    SYMBOL 'square'
    SIZE 7
```

```
STYLE
  SYMBOL "square"
  SIZE 5
  COLOR 255 255 255
END
STYLE
  SYMBOL "square"
  SIZE 3
  COLOR 0 0 255
END
END
END
```

5.6 GRID

LABELFORMAT [DD|DDMM|DDMMSS|C format string] Format of the label. “DD” for degrees, “DDMM” for degrees minutes, and “DDMMSS” for degrees, minutes, seconds. A C-style formatting string is also allowed, such as “%g°” to show decimal degrees with a degree symbol. The default is decimal display of whatever SRS you’re rendering the GRID with.

MINARCS [double] The minimum number of arcs to draw. Increase this parameter to get more lines. Optional.

MAXARCS [double] The maximum number of arcs to draw. Decrease this parameter to get fewer lines. Optional.

MININTERVAL [double] The minimum number of intervals to try to use. The distance between the grid lines, in the units of the grid’s coordinate system. Optional.

MAXINTERVAL [double] The maximum number of intervals to try to use. The distance between the grid lines, in the units of the grid’s coordinate system. Optional.

MINSUBDIVIDE [double] The minimum number of segments to use when rendering an arc. If the lines should be very curved, use this to smooth the lines by adding more segments. Optional.

MAXSUBDIVIDE [double] The maximum number of segments to use when rendering an arc. If the graticule should be very straight, use this to minimize the number of points for faster rendering. Optional, default 256.

The following is an example of a GRID object in use:

```
LAYER
  NAME "grid"
  METADATA
    "DESCRIPTION" "Grid"
  END
  TYPE LINE
  STATUS ON
  CLASS
    NAME "Graticule"
    COLOR 0 0 0
    LABEL
      COLOR 255 0 0
      FONT "fritgat"
      TYPE truetype
      SIZE 8
      POSITION AUTO
      PARTIALS FALSE
      BUFFER 5
      OUTLINECOLOR 255 255 255
```



```

    END
END
PROJECTION
    "init=epsg:4326"
END
GRID
    LABELFORMAT "DDMM"
# LABELFORMAT '%g°' # dec degrees with symbol
    MAXARCS 10
    MAXINTERVAL 10
    MAXSUBDIVIDE 2
# LABELFORMAT '%7.0f m' # nice if a projected SRS used
# MININTERVAL 20000
# MAXSUBDIVIDE 2
END
END # Layer

```

5.7 JOIN

5.7.1 Description

Joins are defined within a *LAYER* object. It is important to understand that JOINS are *ONLY* available once a query has been processed. You cannot use joins to affect the look of a map. The primary purpose is to enable lookup tables for coded data (e.g. 1 => Forest) but there are other possible uses.

5.7.2 Supported Formats

- DBF/XBase files
- CSV (comma delimited text file)
- PostgreSQL and PostGIS tables
- MySQL tables

5.7.3 Mapfile Parameters:

CONNECTION [**string**] Parameters required for the join table's database connection (not required for DBF or CSV joins). The following is an example for *PostgreSQL*:

```
CONNECTION "host=127.0.0.1 port=5432 user=postgres password=postgres dbname=somename"
```

CONNECTIONTYPE [**string**] Type of connection (not required for DBF or CSV joins). The following is an example for PostgreSQL:

```
CONNECTIONTYPE ogr
```

FROM [**item**] Join item in the dataset. This is case sensitive.

NAME [**string**] Unique name for this join. Required.

TABLE [filename|tablename] For file-based joins this is the name of XBase or comma delimited file (relative to the location of the mapfile) to join TO. For PostgreSQL and MySQL support this is the name of the PostgreSQL/MySQL table to join TO.

TEMPLATE [filename] Template to use with one-to-many joins. The template is processed once for each record and can only contain substitutions for items in the joined table. Refer to the column in the joined table in your template like [joinname_columnname], where joinname is the NAME specified for the JOIN object.

TO [item] Join item in the table to be joined. This is case sensitive.

TYPE [ONE-TO-ONE|ONE-TO-MANY] The type of join. Default is one-to-one.

5.7.4 Example 1: Join from SHP file to DBF file

Mapfile Layer

```
LAYER
  NAME "prov_bound"
  TYPE POLYGON
  STATUS DEFAULT
  DATA "prov.shp"
  CLASS
    NAME "Province"
    STYLE
      OUTLINECOLOR 120 120 120
      COLOR 255 255 0
    END
  END
  TEMPLATE "../htdocs/cgi-query-templates/prov.html"
  HEADER "../htdocs/cgi-query-templates/prov-header.html"
  FOOTER "../htdocs/cgi-query-templates/footer.html"
  JOIN
    NAME "test"
    TABLE "../data/lookup.dbf"
    FROM "ID"
    TO "IDENT"
    TYPE ONE-TO-ONE
  END
END # layer
```

Ogrinfo

```
>ogrinfo lookup.dbf lookup -summary
INFO: Open of 'lookup.dbf'
using driver 'ESRI Shapefile' successful.
```

```
Layer name: lookup
Geometry: None
Feature Count: 12
Layer SRS WKT:
(unknown)
IDENT: Integer (2.0)
VAL: Integer (2.0)
```

```
>ogrinfo prov.shp prov -summary
INFO: Open of `prov.shp'
using driver `ESRI Shapefile' successful.
```

```
Layer name: prov
Geometry: Polygon
Feature Count: 12
Extent: (-2340603.750000, -719746.062500) - (3009430.500000, 3836605.250000)
Layer SRS WKT:
(unknown)
NAME: String (30.0)
ID: Integer (2.0)
```

Template

```
<tr bgcolor="#EFEFEF"><td align="left">[NAME]</td><td align="left">[test_VAL]</td></tr>
```

5.7.5 Example 2: Join from SHP file to PostgreSQL table

Mapfile Layer

```
LAYER
  NAME "prov_bound"
  TYPE POLYGON
  STATUS DEFAULT
  DATA "prov.shp"
  CLASS
    NAME "Province"
    STYLE
      OUTLINECOLOR 120 120 120
      COLOR 255 255 0
    END
  END
  TOLERANCE 20
  TEMPLATE "../htdocs/cgi-query-templates/prov.html"
  HEADER "../htdocs/cgi-query-templates/prov-header.html"
  FOOTER "../htdocs/cgi-query-templates/footer.html"
  JOIN
    NAME "test"
    CONNECTION "host=127.0.0.1 port=5432 user=postgres password=postgres dbname=join"
    CONNECTIONTYPE ogr
    TABLE "lookup"
    FROM "ID"
    TO "ident"
    TYPE ONE-TO-ONE
  END
END # layer
```

Ogrinfo

```
>ogrinfo -ro PG:"host=127.0.0.1 port=5432 user=postgres password=postgres dbname=join" lookup -summary
INFO: Open of 'PG:host=127.0.0.1 port=5432 user=postgres password=postgres dbname=join'
using driver 'PostgreSQL' successful.
```

```
Layer name: lookup
Geometry: Unknown (any)
Feature Count: 12
Layer SRS WKT:
(unknown)
ident: Integer (0.0)
val: Integer (0.0)
```

Template

```
<tr bgcolor="#E0E0E0"><td align="left">[NAME]</td><td align="left">[test_val]</td></tr>
```

Note: When testing with MapServer 4.10.0 on Windows this postgresql join caused a mapserv.exe crash. However when testing this with a MapServer build > 4.10.0 the crash did not occur.

5.7.6 Example 3: Join from SHP file to CSV file

Mapfile Layer

```
LAYER
  NAME "prov_bound"
  TYPE POLYGON
  STATUS DEFAULT
  DATA "prov.shp"
  CLASS
    NAME "Province"
    STYLE
      OUTLINECOLOR 120 120 120
      COLOR 255 255 0
    END
  END
  TOLERANCE 20
  TEMPLATE "../htdocs/cgi-query-templates/prov.html"
  HEADER "../htdocs/cgi-query-templates/prov-header.html"
  FOOTER "../htdocs/cgi-query-templates/footer.html"
  JOIN
    NAME "test"
    TABLE "../data/lookup.csv"
    FROM "ID"
    TO "IDENT"
    TYPE ONE-TO-ONE
  END
END # layer
```

CSV File Structure

```
"IDENT", "VAL"
1,12
2,11
3,10
4,9
5,8
6,7
7,6
8,5
9,4
10,3
11,2
12,1
```

Ogrinfo

```
>ogrinfo lookup.csv lookup -summary
INFO: Open of 'lookup.csv'
using driver 'CSV' successful.
```

```
Layer name: lookup
Geometry: None
Feature Count: 12
Layer SRS WKT:
(unknown)
IDENT: String (0.0)
VAL: String (0.0)
```

Template

```
<tr bgcolor="#EFEFEF"><td align="left">[NAME]</td><td align="left">[test_VAL]</td></tr>
```

5.8 LABEL

ALIGN [~~left~~~~center~~~~right~~] Since version 5.4 . Specifies text alignment for multiline labels (see WRAP) Note that the alignment algorithm is far from precise, so don't expect fabulous results (especially for *right* alignment) if you're not using a fixed width font.

ANGLE [~~double~~~~auto~~~~follow~~~~attribute~~] • Angle, given in degrees, to draw the label.

- AUTO allows MapServer to compute the angle. Valid for LINE layers only.
- FOLLOW was introduced in version 4.10 and tells MapServer to compute a curved label for appropriate linear features (see *MS RFC 11: Support for Curved Labels* for specifics).
- [*Attribute*] was introduced in version 5.0, to specify the item name in the attribute table to use for angle values. The hard brackets [] are required. For example, if your shapefile's DBF has a field named "MYANGLE" that holds angle values for each record, your LABEL object might contain:

```

LABEL
  COLOR 150 150 150
  OUTLINECOLOR 255 255 255
  FONT s a n s
  TYPE truetype
  SIZE 6
  ANGLE [ M Y ANGLE ]
  POSITION AUTO
  PARTIALS FALSE
END

```

The associated RFC document for this feature is *MS RFC 19: Style & Label attribute binding*.

ANTIALIAS [**true**/**false**] Should text be antialiased? Note that this requires more available colors, decreases drawing performance, and results in slightly larger output images.

BACKGROUNDCOLOR [**r**] [**g**] [**b**] Color to draw a background rectangle (i.e. billboard). Off by default.

BACKGROUNDSHADOWCOLOR [**r**] [**g**] [**b**] Color to draw a background rectangle (i.e. billboard) shadow. Off by default.

BACKGROUNDSHADOWSIZE [**x**][**y**] How far should the background rectangle be offset? Default is 1.

BUFFER [**integer**] Padding, in pixels, around labels. Useful for maintaining spacing around text to enhance readability. Available only for cached labels. Default is 0.

COLOR [**r**] [**g**] [**b**] | [**attribute**] • Color to draw text with.

- [*Attribute*] was introduced in version 5.0, to specify the item name in the attribute table to use for color values. The hard brackets [] are required. For example, if your shapefile's DBF has a field named "MY-COLOR" that holds color values for each record, your LABEL object might contain:

```

LABEL
  COLOR [ M Y COLOR ]
  OUTLINECOLOR 255 255 255
  FONT s a n s
  TYPE truetype
  SIZE 6
  POSITION AUTO
  PARTIALS FALSE
END

```

The associated RFC document for this feature is *MS RFC 19: Style & Label attribute binding*.

ENCODING [**string**] Supported encoding format to be used for labels. If the format is not supported, the label will not be drawn. Requires the iconv library (present on most systems). The library is always detected if present on the system, but if not the label will not be drawn.

Required for displaying international characters in MapServer. More information can be found at: <http://www.foss4g.org/FOSS4G/MAPSERVER/mpsnf-i18n-en.html>.

FONT [**name**] Font alias (as defined in the FONTSET) to use for labeling.

FORCE [**true**/**false**] Forces labels for a particular class on, regardless of collisions. Available only for cached labels. Default is false.

MAXLENGTH [**integer**] Introduced in mapserver 5.4, this keyword interacts with the WRAP keyword so that line breaks only occur after the defined number of characters

Table 5.1: Interaction with WRAP keyword

	maxlength = 0	maxlength > 0	maxlength < 0
wrap = 'char'	always wrap at the WRAP character	newline at the first WRAP character after MAXLENGTH characters	hard wrap (always break at exactly MAXLENGTH characters)
no wrap	no processing	skip label if it contains more than MAXLENGTH characters	hard wrap (always break at exactly MAXLENGTH characters)

The associated RFC document for this feature is [MS RFC 40: Support Label Text Transformations](#).

MAXSIZE [integer] Maximum font size to use when scaling text (pixels). Default is 256.

MINDISTANCE [integer] Minimum distance between duplicate labels. Given in pixels.

MINFEATURESIZE [integer|auto] Minimum size a feature must be to be labeled. Given in pixels. For line data the overall length of the displayed line is used, for polygons features the smallest dimension of the bounding box is used. “Auto” keyword tells MapServer to only label features that are larger than their corresponding label. Available for cached labels only.

MINSIZE [integer] Minimum font size to use when scaling text (pixels). Default is 4.

OFFSET [x][y] Offset values for labels, relative to the lower left hand corner of the label and the label point. Given in pixels. In the case of rotated text specify the values as if all labels are horizontal and any rotation will be compensated for.

OUTLINECOLOR [r] [g] [b] | [attribute] • Color to draw a one pixel outline around the text.

- [Attribute] was introduced in version 5.0, to specify the item name in the attribute table to use for color values. The hard brackets [] are required. For example, if your shapefile’s DBF has a field named “MY-OUTCOLOR” that holds color values for each record, your LABEL object might contain:

```

LABEL
  COLOR 150 150 150
  OUTLINECOLOR [ M Y O U T COLOR ]
  FONT s a n s
  TYPE truetype
  SIZE 6
  POSITION AUTO
  PARTIALS FALSE
END

```

The associated RFC document for this feature is [MS RFC 19: Style & Label attribute binding](#).

OUTLINEWIDTH [integer] Width of the outline if OUTLINECOLOR has been set. Defaults to 1. Currently only the AGG renderer supports values greater than 1, and renders these as a ‘halo’ effect: recommended values are 3 or 5.

PARTIALS [true|false] Can text run off the edge of the map? Default is true.

POSITION [ulluclur|cllclcr|llllcllr|auto] Position of the label relative to the labeling point (layers only). First letter is “Y” position, second letter is “X” position. “Auto” tells MapServer to calculate a label position that will not interfere with other labels. With points, MapServer selects from the 8 outer positions (i.e. excluding cc). With polygons, MapServer selects from cc (added in MapServer 5.4), uc, lc, cl and cr as possible positions. With lines, it only uses lc or uc, until it finds a position that doesn’t collide with labels that have already been drawn. If all positions cause a conflict, then the label is not drawn (Unless the label’s FORCE a parameter is set to “true”). “Auto” placement is only available with cached labels.

PRIORITY [integer][item_name] The priority parameter (added in v5.0) takes an integer value between 1 (lowest) and 10 (highest). The default value is 1. It is also possible to bind the priority to an attribute (item_name) using square brackets around the [item_name]. e.g. “PRIORITY [someattribute]”

Labels are stored in the label cache and rendered in order of priority, with the highest priority levels rendered first. Specifying an out of range PRIORITY value inside a map file will result in a parsing error. An out of range value set via MapScript or coming from a shape attribute will be clamped to the min/max values at rendering time. There is no expected impact on performance for using label priorities.

SHADOWCOLOR [r] [g] [b] Color of drop shadow.

SHADOWSIZE [x][y] Shadow offset in pixels.

SIZE [integer][**tiny**||**small**||**medium**||**large**||**giant**][**attribute**] • Text size. Use “integer” to give the size in pixels of your TrueType font based label, or any of the other 5 listed keywords to bitmap fonts.

- [Attribute] was introduced in version 5.0, to specify the item name in the attribute table to use for size values. The hard brackets [] are required. For example, if your shapefile’s DBF has a field named “MYSIZE” that holds size values for each record, your LABEL object might contain:

```

LABEL
  COLOR 150 150 150
  OUTLINECOLOR 255 255 255
  FONT s a n s
  TYPE truetype
  SIZE [ M Y SIZE ]
  POSITION AUTO
  PARTIALS FALSE
END

```

The associated RFC document for this feature is *MS RFC 19: Style & Label attribute binding*.

TYPE [bitmap|truetype] Type of font to use. Generally bitmap fonts are faster to draw than TrueType fonts. However, TrueType fonts are scalable and available in a variety of faces. Be sure to set the FONT parameter if you select TrueType.

WRAP [character] Character that represents an end-of-line condition in label text, thus resulting in a multi-line label. Interacts with MAXLENGTH for conditional line wrapping after a given number of characters

5.9 LAYER

CLASS Signals the start of a *CLASS* object.

Inside a layer, only a single class will be used for the rendering of a feature. Each feature is tested against each class in the order in which they are defined in the mapfile. The first class that matches the its min/max scale constraints and its *EXPRESSION* check for the current feature will be used for rendering.

CLASSITEM [attribute] Item name in attribute table to use for class lookups.

CLASSGROUP [string] Specify the class’s group that would be considered at rendering time. The *CLASS* object’s GROUP parameter must be used in combination with CLASSGROUP.

CONNECTION [string] Database connection string to retrieve remote data.

An SDE connection string consists of a hostname, instance name, database name, username and password separated by commas.

A PostGIS connection string is basically a regular PostgreSQL connection string, it takes the form of “user=nobody password=***** dbname=dbname host=localhost port=5432”

An Oracle connection string: user/pass[@db]

See Also:

See *Vector Data* for specific connection information for various data sources.

CONNECTIONTYPE [**locallsdelogr|postgis|oraclespatial|wms**] Type of connection. Default is local. See additional documentation for any other type.

See Also:

See *Vector Data* for specific connection information for various data sources.

DATA [**filename**][**sde parameters**][**postgis table/column**][**oracle table/column**] Full filename of the spatial data to process. No file extension is necessary for shapefiles. Can be specified relative to the SHAPEPATH option from the Map Object.

If this is an SDE layer, the parameter should include the name of the layer as well as the geometry column, i.e. “mylayer,shape,myversion”.

If this is a PostGIS layer, the parameter should be in the form of “<columnname> from <tablename>”, where “columnname” is the name of the column containing the geometry objects and “tablename” is the name of the table from which the geometry data will be read.

For Oracle, use “shape FROM table” or “shape FROM (SELECT statement)” or even more complex Oracle compliant queries! Note that there are important performance impacts when using spatial subqueries however. Try using MapServer’s FILTER whenever possible instead. You can also see the SQL submitted by forcing an error, for instance by submitting a DATA parameter you know won’t work, using for example a bad column name.

See Also:

See *Vector Data* for specific connection information for various data sources.

DEBUG [**off|on|0|1|2|3|4|5**] Enables debugging of a layer in the current map.

Debugging with MapServer versions >= 5.0:

Verbose output is generated and sent to the standard error output (STDERR) or the MapServer errorfile if one is set using the “MS_ERRORFILE” environment variable. You can set the environment variable by using the CONFIG parameter at the MAP level of the mapfile, such as:

```
CONFIG "MS_ERRORFILE" "/ms4w/tmp/ms_error.txt"
```

You can also set the environment variable in Apache by adding the following to your httpd.conf:

```
SetEnv MS_ERRORFILE "/ms4w/tmp/ms_error.txt"
```

Once the environment variable is set, the DEBUG mapfile parameter can be used to control the level of debugging output. Here is a description of the possible DEBUG values:

- **DEBUG 0 or OFF** - only msSetError() calls are logged to MS_ERRORFILE. No msDebug() output at all. This is the default and corresponds to the original behavior of MS_ERRORFILE in MapServer 4.x
- **DEBUG 1 or ON** - includes all output from DEBUG 0 plus msDebug() warnings about common pitfalls, failed assertions or non-fatal error situations (e.g. missing or invalid values for some parameters, missing shapefiles in tileindex, timeout error from remote WMS/WFS servers, etc.)
- **DEBUG 2** - includes all output from DEBUG 1 plus notices and timing information useful for tuning mapfiles and applications
- **DEBUG 3** - all of DEBUG 2 plus some debug output useful in troubleshooting problems such as WMS connection URLs being called, database connection calls, etc. This is the recommended level for debugging mapfiles.
- **DEBUG 4** - DEBUG 3 plus even more details...
- **DEBUG 5** - DEBUG 4 plus any msDebug() output that might be more useful to the developers than to the users.

You can also set the debug level by using the “MS_DEBUGLEVEL” environment variable.

The DEBUG setting can also be specified for the entire map, by setting the DEBUG parameter in the *MAP* object.

For more details on this debugging mechanism, please see *MS RFC 28: Redesign of LOG/DEBUG output mechanisms*.

Debugging with MapServer versions < 5:

Verbose output is generated and sent to the standard error output (STDERR) or the MapServer logfile if one is set using the LOG parameter in the WEB object. Apache users will see timing details for drawing in Apache’s error_log file. Requires MapServer to be built with the DEBUG=MSDEBUG option (–with-debug configure option).

DUMP [truelfalse] Switch to allow MapServer to return data in GML format. Useful when used with WMS GetFeatureInfo operations. “false” by default.

EXTENT [minx] [miny] [maxx] [maxy] The spatial extent of the data. In most cases you will not need to specify this, but it can be used to avoid the speed cost of having MapServer compute the extents of the data. An application can also possibly use this value to override the extents of the map.

FEATURE Signals the start of a *FEATURE* object.

FILTER [string] This parameter allows for data specific attribute filtering that is done at the same time spatial filtering is done, but before any CLASS expressions are evaluated. For OGR and shapefiles the string is simply a mapserver regular expression. For spatial databases the string is a SQL WHERE clause that is valid with respect to the underlying database.

For example: FILTER “type=’road’ and size <2”

FILTERITEM [attribute] Item to use with simple FILTER expressions. OGR and shapefiles only.

FOOTER [filename] Template to use *after* a layer’s set of results have been sent. Multiresult query modes only.

GRID Signals the start of a *GRID* object.

GROUP [name] Name of a group that this layer belongs to. The group name can then be reference as a regular layer name in the template files, allowing to do things like turning on and off a group of layers at once.

HEADER [filename] Template to use *before* a layer’s set of results have been sent. Multiresult query modes only.

JOIN Signals the start of a *JOIN* object.

LABELANGLEITEM [attribute] (As of MapServer 5.0 this parameter is no longer available. Please see the *LABEL* object’s ANGLE parameter) For MapServer versions < 5.0, this is the item name in attribute table to use for class annotation angles. Values should be in degrees. Deprecated since version 5.0.

LABELCACHE [onloff] Specifies whether labels should be drawn as the features for this layer are drawn, or whether they should be cached and drawn after all layers have been drawn. Default is on. Label overlap removal, auto placement etc... are only available when the label cache is active.

LABELITEM [attribute] Item name in attribute table to use for class annotation (i.e. labeling).

LABELMAXSCALEDENOM [double] Maximum scale at which this LAYER is labeled. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated LABELMAXSCALE parameter.

See Also:

Map Scale

LABELMAXSCALE [double] - deprecated Since MapServer 5.0 the proper parameter to use is LABELMAXSCALEDENOM instead. The deprecated LABELMAXSCALE is the maximum scale at which this LAYER is

labeled. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Deprecated since version 5.0.

LABELMINSCALE **[double]** - **deprecated** Minimum scale at which this LAYER is labeled. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated LABELMINSCALE parameter.

See Also:

Map Scale

LABELMINSCALE **[double]** - **deprecated** Since MapServer 5.0 the proper parameter to use is LABELMINSCALEDENOM instead. The deprecated LABELMINSCALE is the minimum scale at which this LAYER is labeled. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Deprecated since version 5.0.

LABELREQUIRES **[expression]** Sets context for labeling this layer, for example:

```
LABELREQUIRES "![orthoquads]"
```

means that this layer would NOT be labeled if a layer named "orthoquads" is on. The expression consists of a boolean expression based on the status of other layers, each [layer name] substring is replaced by a 0 or a 1 depending on that layer's STATUS and then evaluated as normal. Logical operators AND and OR can be used.

LABELSIZEITEM **[attribute]** (As of MapServer 5.0 this parameter is no longer available. Please see the *LABEL* object's SIZE parameter) For MapServer versions < 5.0, this is the item name in attribute table to use for class annotation sizes. Values should be in pixels. Deprecated since version 5.0.

MAXFEATURES **[integer]** Specifies the number of features that should be drawn for this layer in the CURRENT window. Has some interesting uses with annotation and with sorted data (i.e. lakes by area).

MAXSCALE **[double]** - **deprecated** Maximum scale at which this LAYER is drawn. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated MAXSCALE parameter.

See Also:

Map Scale

MAXSCALE **[double]** - **deprecated** Since MapServer 5.0 the proper parameter to use is MAXSCALEDENOM instead. The deprecated MAXSCALE is the maximum scale at which this LAYER is drawn. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Deprecated since version 5.0.

METADATA This keyword allows for arbitrary data to be stored as name value pairs. This is used with *OGC WMS* to define things such as layer title. It can also allow more flexibility in creating templates, as anything you put in here will be accessible via template tags.

Example:

```
METADATA
  title "My layer title"
  author "Me!"
END
```

MINSCALE **[double]** Minimum scale at which this LAYER is drawn. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated MINSCALE parameter.

See Also:

Map Scale

MINSCALE [double] - deprecated Since MapServer 5.0 the proper parameter to use is `MINSCALEDENOM` instead. The deprecated `MINSCALE` is the minimum scale at which this `LAYER` is drawn. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Deprecated since version 5.0.

NAME [string] Short name for this layer. Limit is 20 characters. This name is the link between the mapfile and web interfaces that refer to this name. They must be identical. The name should be unique, unless one layer replaces another at different scales. Use the `GROUP` option to associate layers with each other.

OFFSITE [r] [g] [b] Sets the color index to treat as transparent for raster layers.

OPACITY [integer|alpha] Sets the opacity level (or the inability to see through the layer) of all classed pixels for a given layer. The value can either be an integer in the range (0-100) or the named symbol "ALPHA". A value of 100 is opaque and 0 is fully transparent. Implemented in MapServer 5.0, to replace the deprecated `TRANSPARENCY` parameter.

The "ALPHA" symbol directs the MapServer rendering code to honor the indexed or alpha transparency of pixmap symbols used to style a layer. This is only needed in the case of RGB output formats, and should be used only when necessary as it is expensive to render transparent pixmap symbols onto an RGB map image.

POSTLABELCACHE [true|false] Tells MapServer to render this layer after all labels in the cache have been drawn. Useful for adding neatlines and similar elements. Default is false.

PROCESSING [string] Passes a processing directive to be used with this layer. The supported processing directives vary by layer type, and the underlying driver that processes them.

- **Attributes Directive** - The `ITEMS` processing option allows to specify the name of attributes for inline layers or specify the subset of the attributes to be used by the layer, such as:

```
PROCESSING "ITEMS=itemname1,itemname2,itemname3"
```

- **Connection Pooling Directive** - This is where you can enable connection pooling for certain layer types. Connection pooling will allow MapServer to share the handle to an open database or layer connection throughout a single map draw process. Additionally, if you have `FastCGI` enabled, the connection handle will stay open indefinitely, or according to the options specified in the *FastCGI* configuration. *Oracle Spatial*, *ArcSDE*, *OGR* and *PostGIS/PostgreSQL* currently support this approach.

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

- **OGR Styles Directive** - This directive can be used for obtaining label styles through MapScript. For more information see the *MapServer's OGR document*.

```
PROCESSING "GETSHAPE_STYLE_ITEMS=all"
```

- **Raster Directives** - All raster processing options are described in *Raster Data*. Here we see the `SCALE` and `BANDS` directives used to autoscale raster data and alter the band mapping.

```
PROCESSING "SCALE=AUTO"
```

```
PROCESSING "BANDS=3, 2, 1"
```

PROJECTION Signals the start of a *PROJECTION* object.

REQUIRES [expression] Sets context for displaying this layer (see `LABELREQUIRES`).

SIZEUNITS [pixels|feet|inches|kilometers|meters|miles] Sets the unit of *CLASS* object `SIZE` values (default is pixels). Useful for simulating buffering.

STATUS [on|off|default] Sets the current status of the layer. Often modified by MapServer itself. Default turns the layer on permanently.

Note: In *CGI* mode, layers with STATUS DEFAULT cannot be turned off using normal mechanisms. It is recommended to set layers to STATUS DEFAULT while debugging a problem, but set them back to ON/OFF in normal use.

Note: For *WMS*, layers in the server mapfile with STATUS DEFAULT are always sent to the client.

STYLEITEM [attribute] Item to use for feature specific styling. This is *very* experimental and OGR only at the moment.

SYMBOLSCALEDENOM [double] The scale at which symbols and/or text appear full size. This allows for dynamic scaling of objects based on the scale of the map. If not set then this layer will always appear at the same size. Scaling only takes place within the limits of MINSIZE and MAXSIZE as described above. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated SYMBOLSCALE parameter.

See Also:

Map Scale

SYMBOLSCALE [double] - deprecated Since MapServer 5.0 the proper parameter to use is SYMBOLSCALEDENOM instead. The deprecated SYMBOLSCALE is the scale at which symbols and/or text appear full size. This allows for dynamic scaling of objects based on the scale of the map. If not set then this layer will always appear at the same size. Scaling only takes place within the limits of MINSIZE and MAXSIZE as described above. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Deprecated since version 5.0.

TEMPLATE [file|url] Used as a global alternative to CLASS TEMPLATE.

TILEINDEX [filename|layername] Name of the tileindex file or layer. A tileindex is similar to an ArcInfo library index. The tileindex contains polygon features for each tile. The item that contains the location of the tiled data is given using the TILEITEM parameter. When a file is used as the tileindex for shapefile or raster layers, the tileindex should be a shapefile. For CONNECTIONTYPE OGR layers, any OGR supported datasource can be a tileindex. Normally the location should contain the path to the tile file relative to the shapepath, not relative to the tileindex itself. If the DATA parameter contains a value then it is added to the end of the location. When a tileindex layer is used, it works similarly to directly referring to a file, but any supported feature source can be used (ie. postgres, oracle).

Note: All files in the tileindex should have the same coordinate system, and for vector files the same set of attributes in the same order.

TILEITEM [attribute] Item that contains the location of an individual tile, default is "location".

TOLERANCE [double] Sensitivity for point based queries (i.e. via mouse and/or map coordinates). Given in TOLERANCEUNITS. If the layer is a POINT or a LINE, the default is 3. For all other layer types, the default is 0. To restrict polygon searches so that the point must occur in the polygon set the tolerance to zero.

TOLERANCEUNITS [pixels|feet|inches|kilometers|meters|miles|dd] Units of the TOLERANCE value. Default is pixels.

TRANSPARENCY [integer|alpha] - deprecated Since MapServer 5.0 the proper parameter to use is OPACITY. The deprecated TRANSPARENCY parameter sets the transparency level of all classed pixels for a given layer. The value can either be an integer in the range (0-100) or the named symbol "ALPHA". Although this parameter is named "transparency", the integer values actually parameterize layer opacity. A value of 100 is opaque and 0 is fully transparent.

The "ALPHA" symbol directs the MapServer rendering code to honor the indexed or alpha transparency of pixmap symbols used to style a layer. This is only needed in the case of RGB output formats, and should be

used only when necessary as it is expensive to render transparent pixmap symbols onto an RGB map image. Deprecated since version 5.0.

See Also:

OPACITY

TRANSFORM [**true**|**false** |**ulluclurllcclllrllllcllr**] Tells MapServer whether or not a particular layer needs to be transformed from some coordinate system to image coordinates. Default is true. This allows you to create shapefiles in image/graphics coordinates and therefore have features that will always be displayed in the same location on every map. Ideal for placing logos or text in maps. Remember that the graphics coordinate system has an origin in the upper left hand corner of the image, contrary to most map coordinate systems.

Version 4.10 introduces the ability to define features with coordinates given in pixels (or percentages, see UNITS), most often inline features, relative to something other than the UL corner of an image. That is what 'TRANSFORM FALSE' means. By setting an alternative origin it allows you to anchor something like a copy-right statement to another portion of the image in a way that is independent of image size.

TYPE [**point**|**line**|**polygon**|**circle**|**annotation**|**raster**|**query**|**chart**] Specifies how the data should be drawn. Need not be the same as the shapefile type. For example, a polygon shapefile may be drawn as a point layer, but a point shapefile may not be drawn as a polygon layer. Common sense rules. Annotation means that a label point will be calculated for the features, but the feature itself will not be drawn although a marker symbol can be optionally drawn. this allows for advanced labeling like numbered highway shields. Points are labeled at that point. Polygons are labeled first using a centroid, and if that doesn't fall in the polygon a scanline approach is used to guarantee the label falls within the feature. Lines are labeled at the middle of the longest arc in the visible portion of the line. Query only means the layer can be queried but not drawn.

In order to differentiate between POLYGONS and POLYLINES (which do not exist as a type), simply respectively use or omit the COLOR keyword when classifying. If you use it, it's a polygon with a fill color, otherwise it's a polyline with only an OUTLINECOLOR.

A circle must be defined by a a minimum bounding rectangle. That is, two points that define the smallest square that can contain it. These two points are the two opposite corners of said box.

The following is an example using inline points to draw a circle:

```
LAYER
  NAME 'inline_circles'
  TYPE CIRCLE
  STATUS ON
  FEATURE
    POINTS
      74.01 -53.8
      110.7 -22.16
    END
  END
  CLASS
    STYLE
      COLOR 0 0 255
    END
  END
END
```

See Also:

For CHART layers, see the *Dynamic Charting* HowTo.

UNITS [**feet**|**inches**|**kilometers**|**meters**|**miles**|**dd**|**pixels**|**percentages**] Units of the layer. *Percentages* was added in MapServer 4.10 and is mostly geared for inline features.

5.10 LEGEND

The size of the legend image is NOT known prior to creation so be careful not to hard-code width and height in the tag in the template file.

IMAGECOLOR [**r**] [**g**] [**b**] Color to initialize the legend with (i.e. the background).

INTERLACE [**on|off**] Default is [on]. This keyword is now deprecated in favor of using the **FORMATOPTION** “INTERLACE=ON” line in the *OUTPUTFORMAT* declaration. Deprecated since version 4.6.

LABEL Signals the start of a *LABEL* object

OUTLINECOLOR [**r**] [**g**] [**b**] Color to use for outlining symbol key boxes.

POSITION [**ulluclurllllcllr**] Where to place an embedded legend in the map. Default is lr.

KEYSIZE [**x**][**y**] Size of symbol key boxes in pixels. Default is 20 by 10.

KEYSPACING [**x**][**y**] Spacing between symbol key boxes ([y]) and labels ([x]) in pixels. Default is 5 by 5.

POSTLABELCACHE [**true|false**] Tells MapServer to render this legend after all labels in the cache have been drawn. Useful for adding neatlines and similar elements. Default is false.

STATUS [**on|off|embed**] Is the legend image to be created.

TEMPLATE [**filename**] HTML legend template file.

See Also:

HTML Legends with MapServer

TRANSPARENT [**on|off**] Should the background color for the legend be transparent. This flag is now deprecated in favor of declaring transparency within *OUTPUTFORMAT* declarations. Default is off. Deprecated since version 4.6.

5.11 MAP

ANGLE [**double**] Angle, given in degrees, to rotate the map. Default is 0. The rendered map will rotate in a clockwise direction. The following are important notes:

- Requires a *PROJECTION* object specified at the MAP level and for each *LAYER* object (even if all layers are in the same projection).
- Requires *MapScript* (*SWIG*, *PHP MapScript*). Does not work with *CGI* mode.
- If using the *LABEL* object’s **ANGLE** or the *LAYER* object’s **LABELANGLEITEM** parameters as well, these parameters are relative to the map’s orientation (i.e. they are computed after the MAP object’s **ANGLE**). For example, if you have specified an **ANGLE** for the map of 45, and then have a layer **LABELANGLEITEM** value of 45, the resulting label will not appear rotated (because the resulting map is rotated clockwise 45 degrees and the label is rotated counter-clockwise 45 degrees).
- More information can be found on the MapRotation [Wiki Page](#).

CONFIG [**key**] [**value**] This can be used to specify several values at run-time, for both MapServer and GDAL/OGR libraries. Developers: values will be passed on to `CPLSetConfigOption()`. Details on GDAL/OGR options are found in their associated driver documentation pages (*GDAL/OGR*). The following options are available specifically for MapServer:

CGI_CONTEXT_URL [**value**] The **CONFIG** parameter can be used to enable loading a map context from a URL. See the *Map Context HowTo* for more info.

MS_ENCRYPTION_KEY [filename] The CONFIG parameter can be used to specify an encryption key that is used with MapServer's *msencrypt utility*.

MS_ERRORFILE [filename] The CONFIG parameter can be used to write MapServer errors to a file (as of MapServer 5.0). A full path (absolute reference) is required, including the filename. For more on this see the DEBUG parameter below.

MS_NONSQUARE [yes|no] The CONFIG parameter can be used to allow non-square WMS requests.

MS_PROJ_LIB [path] The CONFIG parameter can be used to define the location of your EPSG files. For more info see the 'PROJ_LIB' parameter below.

ON_MISSING_DATA [FAIL|LOG|IGNORE] The CONFIG parameter can be used to tell MapServer how to handle missing data in tile indexes (as of MapServer 5.3-dev, r8015). Previous MapServer versions required a compile-time switch ("IGNORE_MISSING_DATA"), but this is no longer required.

FAIL This will cause MapServer to throw an error and exit (to crash, in other words) on a missing file in a tile index. This is the default.

```
CONFIG "ON_MISSING_DATA" "FAIL"
```

LOG This will cause MapServer to log the error message for a missing file in a tile index, and continue with the map creation. Note: DEBUG parameter and CONFIG "MS_ERRORFILE" need to be set for logging to occur, so please see the DEBUG parameter below for more information.

```
CONFIG "ON_MISSING_DATA" "LOG"
```

IGNORE This will cause MapServer to not report or log any errors for missing files, and map creation will occur normally.

```
CONFIG "ON_MISSING_DATA" "IGNORE"
```

PROJ_LIB [path] The CONFIG parameter can be used to define the location of your EPSG files for the *Proj.4* library. Setting the [key] to PROJ_LIB and the [value] to the location of your EPSG files will force PROJ.4 to use this value. Using CONFIG allows you to avoid setting environment variables to point to your PROJ_LIB directory. Here are some examples:

1. Unix

```
CONFIG "PROJ_LIB" "/usr/local/share/proj/"
```

2. Windows

```
CONFIG "PROJ_LIB" "C:/somedir/proj/nad/"
```

DATAPATTERN [regular expression] This defines a regular expression to be applied to requests to change DATA parameters via URL requests (i.e. map_layername_data=...). If a pattern doesn't exist then web users can't monkey with support files via URLs. This allows you to isolate one application from another if you desire, with the default operation being very conservative. See also TEMPLATEPATTERN.

DEBUG [off|on|0|1|2|3|4|5] Enables debugging of all of the layers in the current map.

Debugging with MapServer versions >= 5.0:

Verbose output is generated and sent to the standard error output (STDERR) or the MapServer errorfile if one is set using the "MS_ERRORFILE" environment variable. You can set the environment variable by using the CONFIG parameter

at the MAP level of the mapfile, such as:


```
CONFIG "MS_ERRORFILE" "/ms4w/tmp/ms_error.txt"
```

You can also set the environment variable in Apache by adding the following to your httpd.conf:

```
SetEnv MS_ERRORFILE "/ms4w/tmp/ms_error.txt"
```

Once the environment variable is set, the `DEBUG` mapfile parameter can be used to control the level of debugging output. Here is a description of the possible `DEBUG` values:

- **DEBUG 0 or OFF** - only `msSetError()` calls are logged to `MS_ERRORFILE`. No `msDebug()` output at all. This is the default and corresponds to the original behavior of `MS_ERRORFILE` in MapServer 4.x
- **DEBUG 1 or ON** - includes all output from `DEBUG 0` plus `msDebug()` warnings about common pitfalls, failed assertions or non-fatal error situations (e.g. missing or invalid values for some parameters, missing shapefiles in `tileindex`, timeout error from remote WMS/WFS servers, etc.)
- **DEBUG 2** - includes all output from `DEBUG 1` plus notices and timing information useful for tuning mapfiles and applications
- **DEBUG 3** - all of `DEBUG 2` plus some debug output useful in troubleshooting problems such as WMS connection URLs being called, database connection calls, etc. This is the recommended level for debugging mapfiles.
- **DEBUG 4** - `DEBUG 3` plus even more details...
- **DEBUG 5** - `DEBUG 4` plus any `msDebug()` output that might be more useful to the developers than to the users.

You can also set the debug level by using the “`MS_DEBUGLEVEL`” environment variable.

The `DEBUG` setting can also be specified for a layer, by setting the `DEBUG` parameter in the `LAYER` object.

For more details on this debugging mechanism, please see *MS RFC 28: Redesign of LOG/DEBUG output mechanisms*.

Debugging with MapServer versions < 5:

Verbose output is generated and sent to the standard error output (STDERR) or the MapServer logfile if one is set using the `LOG` parameter in the `WEB` object. Apache users will see timing details for drawing in Apache’s `error_log` file. Requires MapServer to be built with the `DEBUG=MSDEBUG` option (–with-debug configure option).

EXTENT [minx] [miny] [maxx] [maxy] The spatial extent of the map to be created. In most cases you will need to specify this, although MapServer can sometimes (expensively) calculate one if it is not specified.

FONTSET [filename] Filename of fontset file to use. Can be a path relative to the mapfile, or a full path.

IMAGECOLOR [r] [g] [b] Color to initialize the map with (i.e. background color). When transparency is enabled (`TRANSPARENT ON`) for the typical case of 8-bit pseudocolored map generation, this color will be marked as transparent in the output file palette. Any other map components drawn in this color will also be transparent, so for map generation with transparency it is best to use an otherwise unused color as the background color.

IMAGEQUALITY [int] *Deprecated* Use `FORMATOPTION “QUALITY=n”` in the `OUTPUTFORMAT` declaration to specify compression quality for JPEG output. *Deprecated since version 4.6.*

IMAGETYPE [gif|png|jpeg|wbmp|gtiff|swfluserdefined] Output format to generate. See details in the `OUTPUTFORMAT` section for available formats. The name here must match the ‘`NAME`’ of a user defined or internally generated `OUTPUTFORMAT` section.

INTERLACE [on|off] *Deprecated* Use `FORMATOPTION “INTERLACE=ON”` in the `OUTPUTFORMAT` declaration to specify if the output images should be interlaced. *Deprecated since version 4.6.*

LAYER Signals the start of a *LAYER* object.

LEGEND Signals the start of a *LEGEND* object.

MAXSIZE [integer] Sets the maximum size of the map image. This will override the default value. For example, setting this to 2048 means that you can have up to 2048 pixels in both dimensions (i.e. max of 2048x2048).

NAME [name] Prefix attached to map, scalebar and legend GIF filenames created using this mapfile. It should be kept short.

PROJECTION Signals the start of a *PROJECTION* object.

QUERYMAP Signals the start of a *QUERYMAP* object.

REFERENCE Signals the start of a *REFERENCE* MAP object.

RESOLUTION [int] Sets the pixels per inch for output, only affects scale computations and nothing else, default is 72.

SCALEDENOM [double] Computed scale of the map. Set most often by the application. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated *SCALE* parameter.

See Also:

Map Scale

SCALE [double] - deprecated Since MapServer 5.0 the proper parameter to use is *SCALEDENOM* instead. The deprecated *SCALE* is the computed scale of the map. Set most often by the application. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Deprecated since version 5.0.

SCALEBAR Signals the start of a *SCALEBAR* object.

SHAPEPATH [filename] Path to the directory holding the shapefiles or tiles. There can be further subdirectories under *SHAPEPATH*.

SIZE [x][y] Size in pixels of the output image (i.e. the map).

STATUS [onloff] Is the map active? Sometimes you may wish to turn this off to use only the reference map or scale bar.

SYMBOLSET [filename] Filename of the symbolset to use. Can be a path relative to the mapfile, or a full path.

SYMBOL Signals the start of a *SYMBOL* object.

TEMPLATEPATTERN [regular expression] This defines a regular expression to be applied to requests to change *TEMPLATE* parameters via URL requests (i.e. `map_layername_template=...`). If a pattern doesn't exist then web users can't monkey with support files via URLs. This allows you to isolate one application from another if you desire, with the default operation being very conservative. See also *DATAPATTERN*.

TRANSPARENT [onloff] Use *FORMATOPTION* "TRANSPARENT=ON" in the *OUTPUTFORMAT* declaration to specify if the output images should be transparent. Deprecated since version 4.6.

UNITS [feet|inches|kilometers|meters|miles|dd] Units of the map coordinates. Used for scalebar and scale computations.

WEB Signals the start of a *WEB* object.

5.12 OUTPUTFORMAT

A map file may have zero, one or more *OUTPUTFORMAT* object declarations, defining available output formats supported including formats like PNG, GIF, JPEG, GeoTIFF and Flash (SWF).

If OUTPUTFORMAT sections declarations are not found in the map file, the following implicit declarations will be made. Only those for which support is compiled in will actually be available. The GeoTIFF depends on building with GDAL support, and the Flash (SWF) depends on compiling with support for the MING library.

```

OUTPUTFORMAT
  NAME aggpng24
  DRIVER AGG/PNG
  MIMETYPE "image/png"
  IMAGEMODE RGB
  EXTENSION "png"
END
OUTPUTFORMAT
  NAME gif
  DRIVER "GD/GIF"
  MIMETYPE "image/gif"
  IMAGEMODE PC256
  EXTENSION "gif"
END
OUTPUTFORMAT
  NAME png
  DRIVER "GD/PNG"
  MIMETYPE "image/png"
  IMAGEMODE PC256
  EXTENSION "png"
END
OUTPUTFORMAT
  NAME jpeg
  DRIVER "GD/JPEG"
  MIMETYPE "image/jpeg"
  IMAGEMODE RGB
  EXTENSION "jpg"
END
OUTPUTFORMAT
  NAME wbmp
  DRIVER "GD/WBMP"
  MIMETYPE "image/wbmp"
  IMAGEMODE PC256
  EXTENSION "wbmp"
END
OUTPUTFORMAT
  NAME swf
  DRIVER "SWF"
  MIMETYPE "application/x-shockwave-flash"
  EXTENSION "swf"
  IMAGEMODE PC256
  FORMATOPTION "OUTPUT_MOVIE=SINGLE"
END
OUTPUTFORMAT
  NAME GTiff
  DRIVER "GDAL/GTiff"
  MIMETYPE "image/tiff"
  IMAGEMODE RGB
  EXTENSION "tif"
END

```

NAME [name] The name to use in the IMAGETYPE keyword of the map file to select this output format.(optional)

DRIVER [name] The name of the driver to use to generate this output format. Some driver names include the defini-

tion of the format if the driver supports multiple formats. For AGG, the possible driver names are “AGG/PNG” and “AGG/JPEG”. For GD the possible driver names are “GD/Gif”, “GD/PNG”, “GD/WBMP” and “GD/JPEG”. For flash the driver is just called “SWF”. For output through GDAL the GDAL shortname for the format is appended, such as “GDAL/GTiff”. Note that PNG, JPEG and GIF output can be generated with either GDAL or GD (GD is generally more efficient).(mandatory)

IMAGEMODE [PC256/RGB/RGBA/INT16/FLOAT32] Selects the imaging mode in which the output is generated. Does matter for non-raster formats like Flash. Not all formats support all combinations. For instance GD/GIF supports only PC256. (optional)

- PC256: Produced a pseudocolored result with up to 256 colors in the palette (traditional MapServer mode)
- RGB: Render in 24bit Red/Green/Blue mode. Supports all colors but does not support transparency.
- RGBA: Render in 32bit Red/Green/Blue/Alpha mode. Supports all colors, and alpha based transparency. All features are rendered against an initially transparent background.
- BYTE: Render raw 8bit pixel values (no presentation). Only works for RASTER layers (through GDAL) and WMS layers currently.
- INT16: Render raw 16bit signed pixel values (no presentation). Only works for RASTER layers (through GDAL) and WMS layers currently.
- FLOAT32: Render raw 32bit floating point pixel values (no presentation). Only works for RASTER layers (through GDAL) and WMS layers currently.

MIMETYPE [type] Provide the mime type to be used when returning results over the web. (optional)

EXTENSION [type] Provide the extension to use when creating files of this type. (optional)

TRANSPARENT [ON/OFF] Indicates whether transparency should be enabled for this format. Note that transparency does not work for IMAGEMODE RGB output. Not all formats support transparency (optional). When transparency is enabled for the typical case of 8-bit pseudocolored map generation, the IMAGECOLOR color will be marked as transparent in the output file palette. Any other map components drawn in this color will also be transparent, so for map generation with transparency it is best to use an otherwise unused color as the background color.

FORMATOPTION [option] Provides a driver or format specific option. Zero or more FORMATOPTION statement may be present within a OUTPUTFORMAT declaration. (optional)

- GD/JPEG: The “QUALITY=n” option may be used to set the quality of jpeg produced (value from 0-100).
- GD/PNG: The “INTERLACE=[ON/OFF]” option may be used to turn interlacing on or off.
- GD/GIF: The “INTERLACE=[ON/OFF]” option may be used to turn interlacing on or off.
- GDAL/GTiff: Supports the TILED=YES, BLOCKXSIZE=n, BLOCKYSIZE=n, INTERLEAVE=[PIXEL/BAND] and COMPRESS=[NONE,PACKBITS,JPEG,LZW,DEFLATE] format specific options.
- GDAL/*: All FORMATOPTIONs are passed onto the GDAL create function. Options supported by GDAL are described in the detailed documentation for each GDAL format
- GD/PNG and AGG/PNG both support quantizing from 24/32 bits to 8bits, in order to reduce the final image size (and therefore save bandwidth) (see also <http://trac.osgeo.org/mapserver/ticket/2436#comment:4> for strategies when applying these options):
 - “QUANTIZE_FORCE=on” used to reduce an RGB or RGBA image into an 8bit (or less) paletted images. The colors used in the palette are selected to best fit the actual colors in the RGB image (RGBA quantization was introduced in mapserver 5.2, and requires the `-enable-experimental-png` flag to be used when configuring)

- “QUANTIZE_COLORS=256” used to specify the number of colors to be used when applying quantization. Maximum value is 256. Specifying anything between 17 and 255 is probably a waste of quality as each pixel is still encoded with a full byte. Specifying a value under 16 will produce tiny images, but severely degraded.
- “QUANTIZE_NEW=on” used to force using a slower but higher quality quantization algorithm for RGB images. The default behavior is to use a fast but crude quantization algorithm, that usually results in duplicate entries in the palette, and visible artifacts when tiling.
- “PALETTE_FORCE=on” is used to reduce image depth with a predefined palette. This option is incompatible with the previous quantization options.
- “PALETTE=/path/to/palette.txt” is used to define the absolute path where palette colors can be found. This file must contain 256 entries of r,g,b triplets for RGB imagemodes, or r,g,b,a quadruplets for RGBA imagemodes. The expected format is one triplet (or quadruplet) per line, each value separated by commas, and each triplet/quadruplet on a single line. If less than 256 triplets are found in the file, 256-n remaining colors will be computed by quantization. If you have enabled the RGBA_PNG support (the `-with-experimental-png` option) and want to use transparency with a palette, it is important to have these two colors in the palette file: 0,0,0 and 255,255,255,255.

5.13 PROJECTION

To set up projections you must define two projection objects: one for the output image (in the *MAP* object) and one for each layer (in the *LAYER* objects) to be projected. MapServer relies on the *Proj.4* library for projections. Projection objects therefore consist of a series of PROJ.4 keywords, which are either specified within the object directly or referred to in an *EPSG* file. An EPSG file is a lookup file containing projection parameters, and is part of the PROJ.4 library.

The following two examples both define the same projection (UTM zone 15, NAD83), but use 2 different methods:

Example 1: Inline Projection Parameters

```
PROJECTION
  "proj=utm"
  "ellps=GRS80"
  "datum=NAD83"
  "zone=15"
  "units=m"
  "north"
  "no_defs"
END
```

Example 2: EPSG Projection Use

```
PROJECTION
  "init=epsg:26915"
END
```

Note: This refers to an EPSG lookup file that contains a ‘26915’ code with the full projection parameters. “epsg” in this instance is case-sensitive because it is referring to a file name. If your file system is case-sensitive, this must be lower case, or MapServer (Proj.4 actually) will complain about not being able to find this file.

Note: See <http://spatialreference.org/ref/epsg/26915> for more information on this coordinate system.

The next two examples both display how to possibly define unprojected lat/longs (“geographic”):

Example 3: Inline Projection Parameters

```
PROJECTION
  "proj=latlong"
  "ellps=WGS84"
  "datum=WGS84"
END
```

Example 4: epsg Projection Use

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

5.13.1 Important Notes

- If all of your data in the mapfile is in the same projection, you DO NOT have to specify any projection objects. MapServer will assume that all of the data is in the same projection.
- Think of the *MAP*-level projection object as your output projection. The *EXTENT* and *UNITS* values at the *MAP*-level must be in the output projection units. Also, if you have layers in other projections (other than the *MAP*-level projection) then you must define *PROJECTION* objects for those layers, to tell MapServer what projections they are in.
- If you specify a *MAP*-level projection, and then only one other *LAYER* projection object, MapServer will assume that all of the other layers are in the specified *MAP*-level projection.
- Always refer to the EPSG file in lowercase, because it is a lowercase filename and on Linux/Unix systems this parameter is case sensitive.

5.13.2 For More Information

- If you get projection errors, refer to the *Errors* to check if your exact error has been discussed.
- Search the MapServer-users [email list archives](#), odds are that someone has faced your exact issue before.
- See the *PROJ.4* user guides for complete descriptions of supported projections and coordinate systems.
- Refer to the [Cartographical Map Projections](#) page for background information on projections.

5.14 QUERYMAP

COLOR [r] [g] [b] Color in which features are highlighted. Default is yellow.

SIZE [x][y] Size of the map in pixels. Defaults to the size defined in the map object.

STATUS [on|off] Is the query map to be drawn?

STYLE [normal|hilitelselected] Sets how selected features are to be handled. Layers not queried are drawn as usual.

- Normal: Draws all features according to the settings for that layer.
- Hilite: Draws selected features using *COLOR*. Non-selected features are drawn normally.
- Selected: draws only the selected features normally.

5.15 REFERENCE

Three types of reference maps are supported. The most common would be one showing the extent of a map in an interactive interface. It is also possible to request reference maps as part of a query. Point queries will generate an image with a marker (see below) placed at the query point. Region based queries will depict the extent of the area of interest. Finally, feature based queries will display the selection feature(s) used.

COLOR [*r*] [*g*] [*b*] Color in which the reference box is drawn. Set any component to -1 for no fill. Default is red.

EXTENT [*minx*][*miny*][*maxx*][*maxy*] The spatial extent of the base reference image.

IMAGE [*filename*] Full filename of the base reference image. Must be a GIF image.

MARKER [*integer*|*string*] Defines a symbol (from the symbol file) to use when the box becomes too small (see `MINBOXSIZE` and `MAXBOXSIZE` below). Uses a crosshair by default.

MARKERSIZE [*integer*] Defines the size of the symbol to use instead of a box (see `MARKER` above).

MINBOXSIZE [*integer*] If box is smaller than `MINBOXSIZE` (use box width or height) then use the symbol defined by `MARKER` and `MARKERSIZE`.

MAXBOXSIZE [*integer*] If box is greater than `MAXBOXSIZE` (use box width or height) then draw nothing (Often the whole map gets covered when zoomed way out and it's perfectly obvious where you are).

OUTLINECOLOR [*r*] [*g*] [*b*] Color to use for outlining the reference box. Set any component to -1 for no outline.

SIZE [*x*][*y*] Size, in pixels, of the base reference image.

STATUS [*on*|*off*] Is the reference map to be created? Default is off.

5.16 SCALEBAR

Scalebars currently do not make use of TrueType fonts. The size of the scalebar image is NOT known prior to rendering, so be careful not to hard-code width and height in the `` tag in the template file. Future versions will make the image size available.

ALIGN [*left*|*center*|*right*] Defines how the scalebar is aligned within the scalebar image. Default is center. Available in versions 5.2 and higher. New in version 5.2.

BACKGROUNDCOLOR [*r*] [*g*] [*b*] Color to use for scalebar background, not the image background.

COLOR [*r*] [*g*] [*b*] Color to use for drawing all features if attribute tables are not used.

IMAGECOLOR [*r*] [*g*] [*b*] Color to initialize the scalebar with (i.e. background).

INTERLACE [*true*|*false*] Should output images be interlaced? Default is `[on]`. This keyword is now deprecated in favour of using the `FORMATOPTION "INTERLACE=ON"` line in the `OUTPUTFORMAT` declaration. Deprecated since version 4.6.

INTERVALS [*integer*] Number of intervals to break the scalebar into. Default is 4.

LABEL Signals the start of a `LABEL` object

OUTLINECOLOR [*r*] [*g*] [*b*] Color to use for outlining individual intervals. Set any component to -1 for no outline which is the default.

POSITION [*u*][*l*][*u*][*l*][*u*][*l*][*l*][*l*][*l*][*l*][*r*] Where to place an embedded scalebar in the image. Default is `lr`.

POSTLABELCACHE [*true*|*false*] For use with embedded scalebars only. Tells the MapServer to embed the scalebar after all labels in the cache have been drawn. Default is false.

SIZE [*x*][*y*] Size in pixels of the scalebar. Labeling is not taken into account.

STATUS [**on|off|embed**] Is the scalebar image to be created, and if so should it be embedded into the image? Default is off. (Please note that embedding scalebars require that you define a markerset. In essence the scalebar becomes a custom marker that is handled just like any other annotation.)

STYLE [**integer**] Chooses the scalebar style. Valid styles are 0 and 1.

TRANSPARENT [**on|off**] Should the background color for the scalebar be transparent. This flag is now deprecated in favor of declaring transparency within *OUTPUTFORMAT* declarations. Default is off. Deprecated since version 4.6.

UNITS [**feet|inches|kilometers|meters|miles**] Output scalebar units, default is miles. Used in conjunction with the map's units to develop the actual graphic. Note that decimal degrees are not valid scalebar units.

5.17 STYLE

Style holds parameters for symbolization. Multiple styles may be applied within a class.

This object is new in 4.0 and is intended to separate logic from looks. The final intent is to have named styles (Not yet supported) that will be re-usable through the mapfile. This is the new, preferred way of defining the appearance of an object, notably a class.

ANGLE [**double|attribute|AUTO**]

- Angle, given in degrees, to draw the line work. Default is 0. For symbols of Type HATCH, this is the angle of the hatched lines. For its use with hatched lines, see Example#8 in the *symbology examples*.
- [*Attribute*] was introduced in version 5.0, to specify the item name in the attribute table to use for angle values. The hard brackets [] are required. For example, if your shapefile's DBF has a field named "MYANGLE" that holds angle values for each record, your STYLE object for hatched lines might contain:

```
STYLE
  SYMBOL 'hatch-test'
  COLOR 255 0 0
  ANGLE [MYANGLE]
  SIZE 4
  WIDTH 3
END
```

The associated RFC document for this feature is *MS RFC 19: Style & Label attribute binding*.

- The AUTO keyword was added in version 5.4, and currently only applies when coupled with the GEOM-TRANSFORM keyword.

ANGLEITEM [**string**]

- this parameter was removed in MapServer 5.0. You should use the ANGLE [*attribute*] parameter instead.
- For MapServer versions <5, this is the attribute/field that stores the angle to be used in rendering. Angle is given in degrees with 0 meaning no rotation.

ANTIALIAS [**true|false**] Should TrueType fonts and Cartoline symbols be antialiased.

BACKGROUNDCOLOR [**r**] [**g**] [**b**] Color to use for non-transparent symbols.

COLOR [**r**] [**g**] [**b**] | [**attribute**]

- Color to use for drawing features.
- [*Attribute*] was introduced in version 5.0, to specify the item name in the attribute table to use for color values. The hard brackets [] are required. For example, if your *shapefile*'s DBF has a field named "MYCOLOR" that holds color values for each record, your STYLE object for might contain:


```

STYLE
  COLOR [M Y COLOR]
  OUTLINECOLOR 150 150 150
END

```

The associated RFC document for this feature is *MS RFC 19: Style & Label attribute binding*.

GEOMTRANSFORM [start|end|vertices|bbox]

- Used to indicate that the current feature will be transformed before the actual style is applied. Only applies to versions from 5.4
- “bbox”: uses the current style for rendering the bounding box of the underlying geometry
- “start” and “end”: uses the current style to render a marker on the first or last vertex of the current geometry. When used with ANGLE AUTO, this can be used to render arrowheads or tails on line segments.
- “vertices”: uses the current style for rendering a marker on the intermediate vertices of the underlying geometry. When used with ANGLE AUTO, the marker is oriented by the half angle formed by the two adjacent line segments.

MAXSIZE [integer] Maximum size in pixels to draw a symbol. Default is 50.

MINSIZE [integer] Minimum size in pixels to draw a symbol. Default is 0.

MINWIDTH [integer] Minimum width in pixels to draw the line work.

OFFSET [x][y] Offset values for shadows, hollow symbols, etc ...

OPACITY [integer] Opacity to draw the current style (applies to 5.2+, *AGG Rendering Specifics* only, does not apply to pixmap symbols)

OUTLINECOLOR [r] [g] [b] | [attribute]

- Color to use for outlining polygons and certain marker symbols. Line symbols do not support outline colors.
- [Attribute] was introduced in version 5.0, to specify the item name in the attribute table to use for color values. The hard brackets [] are required. For example, if your shapefile’s DBF has a field named “MYOUTCOLOR” that holds color values for each record, your STYLE object for might contain:

```

STYLE
  COLOR 255 0 0
  OUTLINECOLOR [M Y O U T COLOR]
END

```

The associated RFC document for this feature is *MS RFC 19: Style & Label attribute binding*.

SIZE [integer|attribute]

- Height, in layer SIZEUNITS, of the symbol/pattern to be used. Only useful with scalable symbols. Default is 1. For symbols of Type HATCH, the SIZE is the distance between hatched lines. For its use with hatched lines, see Example#8 in the *symbology examples*.
- [Attribute] was introduced in version 5.0, to specify the item name in the attribute table to use for size values. The hard brackets [] are required. For example, if your shapefile’s DBF has a field named “MYSIZE” that holds size values for each record, your STYLE object for hatched lines might contain:

```

STYLE
  SYMBOL 'hatch-test'
  COLOR 255 0 0
  ANGLE 45
  SIZE [M Y SIZE]

```

WIDTH 3
END

The associated RFC document for this feature is *MS RFC 19: Style & Label attribute binding*.

- Starting from version 5.4, the value can also be a fractional value (and not only integer). Note that currently only the *AGG* renderer can correctly render fractional widths.

SIZEITEM [string]

- this parameter was removed in MapServer 5.0. You should use the *SIZE* [attribute] parameter instead.
- For MapServer versions <5, this is the attribute/field that stores the size to be used in rendering. Value is given in pixels.

SYMBOL [integer|string|filename] The symbol name or number to use for all features if attribute tables are not used. The number is the index of the symbol in the symbol file, starting at 1, the 5th symbol in the file is therefore symbol number 5. You can also give your symbols names using the *NAME* keyword in the symbol definition file, and use those to refer to them. Default is 0, which results in a single pixel, single width line, or solid polygon fill, depending on layer type.

You can also specify a gif or png filename. The path is relative to the location of the mapfile.

WIDTH [integer|attribute]

- Width refers to the thickness of line work drawn, in layer *SIZEUNITS*. Default is 1.
- For symbols of Type *HATCH*, the *WIDTH* is how thick the hatched lines are. For its use with hatched lines, see Example#8 in the *symbolology examples*.
- Attribute binding was added in version 5.2
- Starting from version 5.4, the value can also be a fractional value (and not only integer). Note that currently only the *AGG* renderer can correctly render fractional widths.

5.18 SYMBOL

- Symbol definitions can be included within the main MapFile or, more commonly, in a separate file. Symbol definitions in a separate file are designated using the *SYMBOLSET* keyword, as part of the *MAP Object*. This recommended setup is ideal for re-using symbol definitions across multiple MapServer applications.
- There are 3 main types of symbols in MapServer: Markers, Shadesets, and Lines.
- Symbol 0 is always the degenerate case for a particular class of symbol. For points, symbol 0 is a single pixel, for shading (i.e. filled polygons) symbol 0 is a solid fill, and for lines, symbol 0 is a single pixel wide line.
- Symbol definitions contain no color information, colors are set within *CLASS* objects.
- For MapServer versions < 5 there is a maximum of 64 symbols per file. This can be changed by editing *mapsymbol.h* and changing the value of *MS_MAXSYMBOLS* at the top of the file. As of MapServer 5.0 there is no symbol limit.
- More information can be found in the *Construction of Cartographic Symbols* document.

ANTI_ALIAS [true|false] Should TrueType fonts be antialiased.

CHARACTER [char] Character used to reference a particular TrueType font character. You'll need to figure out the mapping from the keyboard character to font character.

FILLED [true|false] Sets the symbol to be filled with a user defined color (See the *CLASS* object). For marker symbols, if *OUTLINECOLOR* was specified then the symbol is outlined with it.

FONT [string] Name of TrueType font to use as defined in the *FONTSET*.

GAP [int] Given in pixels. This defines a distance between symbols for TrueType lines. As of MapServer 5.0 this also applies to Pixmap symbols.

When drawing the symbol along a line segment, a negative GAP will add 180 degrees to the angle. The TrueType and Pixmap symbols are always oriented along the line. A GAP of 0 (the default value) will cause MapServer to use the symbol as a brush to draw the line.

IMAGE [string] Image (GIF or PNG) to use as a marker or brush for type PIXMAP symbols.

NAME [string] Alias for this font to be used in *CLASS* objects

LINECAP [butt|round|square|triangle] Sets the line cap type for the cartoline symbol. Default is butt. Works with the CARTOLINE symbol only!

LINEJOIN [round|miter|bevel] Sets the line join type for the cartoline symbol. Default is "none" - lines will not be joined. Works with the CARTOLINE symbol only!

LINEJOINMAXSIZE [int] Sets the max length of the miter line join type. The value represents a coefficient which multiplies a current symbol size. Default is 3. Works with the CARTOLINE symbol only!

PATTERN [num on] [num off] [num on] ... END Defines a dash style or pattern. Implemented in MapServer 5.0, to replace the deprecated STYLE parameter.

POINTS [x y] [x y] ... END Signifies the start of the definition of points that make up a vector symbol or that define the x and y radius of an ellipse symbol. The end of this section is signified with the keyword END. Coordinates are given in pixels and define the default size of the symbol before any scaling. You can create non-contiguous paths by inserting negative coordinates at the appropriate place. For ellipse symbols you provide a single point that defines the x and y radius of an ellipse. Circles are created when x and y are equal.

Note: If a class using this symbol doesn't contain an explicit size, then the default symbol size will be based on the range of "y" values in the point coordinates. e.g. if the y coordinates of the points in the symbol range from 0 to 5, then the default size for this symbol will be assumed to be 5.

STYLE [num on] [num off] [num on] ... END -deprecated Since MapServer 5.0 the proper parameter to use is PATTERN instead. The deprecated STYLE parameter defines a dash style or pattern. Deprecated since version 5.0.

TRANSPARENT [color index] Sets a transparent color for the input GIF image for pixmap symbols, or determines whether all shade symbols should have a transparent background. For shade symbols it may be desirable to have background features "show through" a transparent hatching pattern, creating a more complex map. By default a symbol's background is the same as the parent image (i.e. color 0). This is user configurable.

TYPE [vector|ellipse|pixmap|truetype|simple|cartoline|hatch] • *vector*: a simple drawing is used to define the shape of the symbol.

- *ellipse*: radius values in the x and y directions define an ellipse.
- *pixmap*: a user supplied GIF image will be used as the symbol.
- *truetype*: TrueType font to use as defined in the FONTSET.
- *cartoline*: allows for different designs of line ends (mitered, rounded, beveled). More information can be found in the *Cartographic Symbols document*.
- *hatch*: produces hatched lines throughout the shape.

5.19 Cartographic Symbol Construction with MapServer

Author Peter Freimuth

Contact pf at mapmedia.de

Author Arnulf Christl

Contact arnulf.christl at wheregroup.com

Revision \$Revision: 8484 \$

Date \$Date: 2009-01-31 07:38:27 -0800 (Sat, 31 Jan 2009) \$

Table of Contents

- Cartographic Symbol Construction with MapServer
 - Abstract
 - Introduction
 - Using Cartographic Symbols in UMN MapServer
 - Examples
 - Current Problems / Open Issues

5.19.1 Abstract

This Document refers to the syntax of MAP and symbolfiles for MapServer 4.6. It is based upon the results of a project carried out at the University of Hannover, Institute of Landscape and Nature Conservation. It was initiated by Mr. Dipl. Ing. Roland Hachmann. Parts have been taken from a study carried through by Karsten Hoffmann, student of Geography and Cartography at the FU Berlin. In the context of a hands-on training in the company GraS GmbH Mr. Hoffman mainly dealt with the development of symbols. ([Download study report in German](#)) His degree dissertation will also concern this subject.

At the end of this document you will find a link to a paper discussing further development of the UMN MapServer software regarding the cartographic rendering capabilities. We will try to get around to also translating it to English asap. We welcome everybody who is interested in this subject to participate in the discussion. As a result of this discussion we want to put together a list with change requests and new requirements for UMN MapServer.

5.19.2 Introduction

Cartographic characters can be distinguished as point, line and area symbols. These symbols may vary depending on their special attributes (variables). Bertin (1974) created a clear and logical symbol scheme in which symbols can be varied referring to graphical variables. The following variables are used within MapServer: FORM, SIZE, PATTERN, COLOR and LIGHTNESS. Point and area symbols as well as text fonts (tff) can additionally be displayed with a 1-pixel wide frame which we call OUTLINE.

Multiple Rendering and Overlay

Complex cartographic effects can be achieved by rendering the same vector data with different attributes, sizes and colors on top of each other. This is an easy workaround to creating complex signatures but obviously it will also reflect on the performance of the application. Every rendering process of the same geometries will take up additional processor time.

As an example consider displaying a highway with a black border line, two yellow lanes and a red center line. You can achieve this by rendering the same highway geometry three times on top of each other. The lowest (in drawing order) is rendered as a broad black line with a width of 9 pixel. The second level lines are rendered in yellow color with a width of 7 pixel. The topmost layer is displayed as the red center line with a width of 3 pixel in size. That way each yellow colored lane will have a width of $(7-1)/2 = 3$ pixel.

This can be a solution for many kinds of cartographic questions, a combination of different geometry types is also possible. First define a polygon layer as TYPE LINE. It will frame the polygons with a line signature. On top of this include the same geometry a second time but now as TYPE POLYGON with a symbol filling the polygon. This way half of the underlying outline is covered by the polygon with the fill symbol. The clipping effect renders an asymmetric line symbol. To present the outline without clipping just reorder the layers and put the outline presentation on top of the layer with the fill symbol.

Each step growing more complex, yet another way to construct asymmetric line signatures for framed faces is to tamper with the original geometries by buffering or clipping the original geometry such that the new objects lie inside the original polygons or grow over the borders. A new line symbol for these geometries can use the OFFSET parameter with the y-value set to -99. This is where current development stops short this functions has not been developed further. (Anybody need this feature?). Buffer areas may provide ideal geometry for ambitious cartography, this should be possible as soon as the GEOS library is implemented in MapServer. Until then using PostGIS helps to solve a lot of problems.

Symbol Scaling

There are two basically different ways to handle the display size of symbols and cartographic elements in a map at different scales. The size of cartographic objects is either set in screen pixel or in real world units.

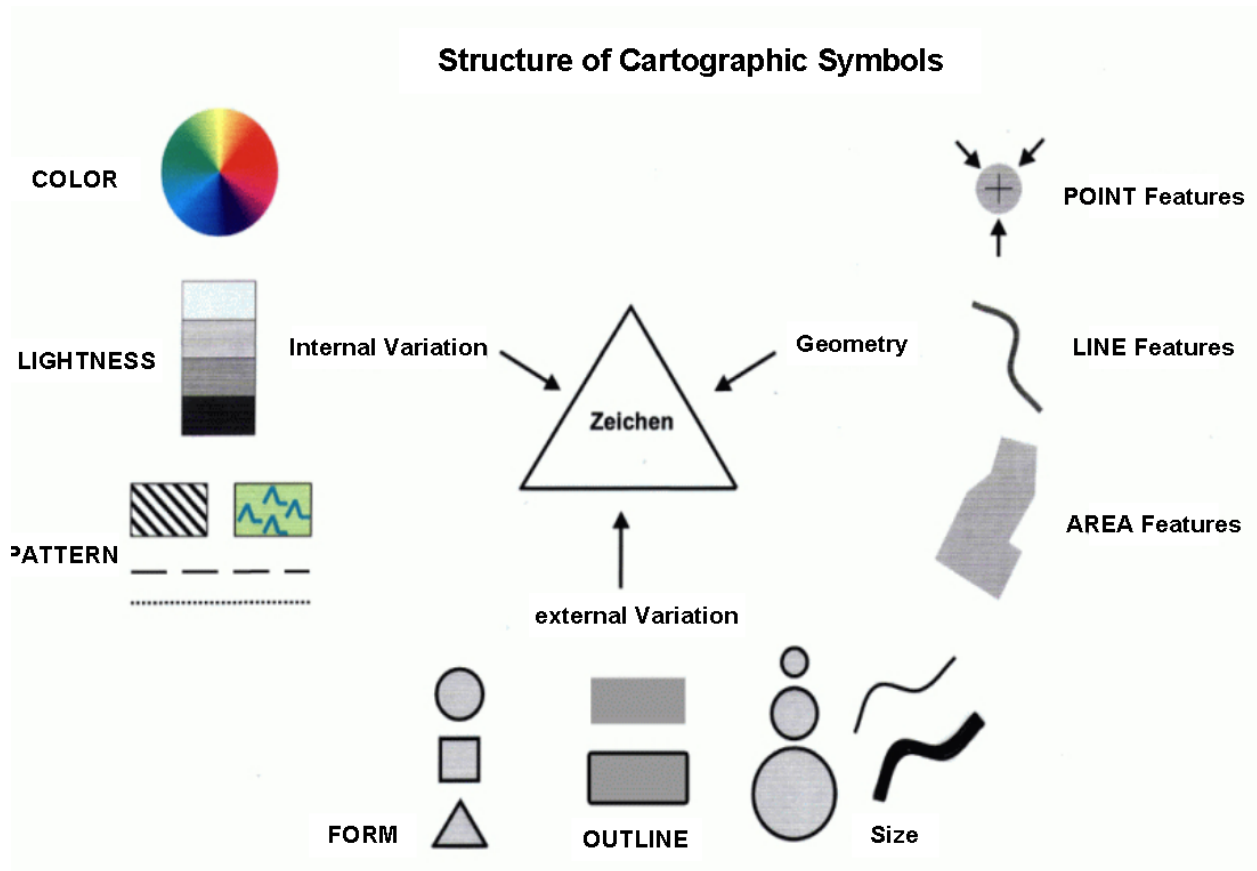
- If the size is set in real world units (i.e. meters), then cartographic objects are scalable, they will shrink and grow together with the scale at which the map is displayed.
- Object sizes in screen pixel will be displayed in the same size no matter at what scale the map is displayed.

UMN MapServer implements the screen pixel size type for displaying cartographic elements. Additionally the size of cartographic elements can be tied to defined scales, such that the symbols “grow” and “shrink” with the scale. The involved parameters include SYMBOLSCALE, MINSIZE and MAXSIZE settings in the MAP file.

All symbol and style definitions referenced in the following section are contained in the archive at the end of this document.

The following figure shows the theoretical structure of cartographic symbols, which are also used in UMN MapServer:

Figure1: Structure of Cartographic Symbols⁶



In a MapServer application symbol parameters are taken from the MAP and symbol file as follows:

MAP file:

- The parameter **TYPE** defines the type of geometry (point, line or polygon) for each layer. The symbols are accordingly rendered as single points, along a line or area symbols.
- The definition of the color, brightness (through the color parameter), size and outline is set inside the **STYLE**-section of a **CLASS**-section by the parameters **COLOR**, **SIZE** and **OUTLINECOLOR**.
- Combining several basic elements to achieve a complex signature with several **STYLES** inside one layer (observe the display order).

Symbol file:

- Definition of form and pattern with **TYPE**, **POINTS**, **IMAGE**, **FILLED**, **STYLE** (meaning fill pattern) and **GAP**.

The following figure shows the interaction of these elements and explains the configuration in the **MAP** and the **SYMBOL** file sections necessary for rendering cartographic symbols in the map:

Figure2: Schema showing interaction of MAP and SYMBOL file

LAYER-Sektion der Map-Datei:

```

LAYER
NAME Hochschulen
TYPE POINT
STATUS ON
DATA hochschule
CLASS
STYLE
SYMBOL "quadrat"
SIZE 16
COLOR 255 0 0
OUTLINECOLOR 0 0 0
END
END

```

Bestimmung der Geometrie (Punkte, Linien oder Flächen), hier: Punkte

Angabe der Datei hochschule.shp mit den Vektordaten

Beginn der STYLE-Sektion

Angabe des Zeichens aus der Symboldatei, welches auf die Punkte gezeichnet werden soll

Größe der Signatur in Pixel

Farbe in RGB-Werten mit der das Zeichen gefüllt wird, hier: Rot

Definition einer 1-Pixel breiten Linie, die das Zeichen umrandet, hier: Schwarz

So sieht die Signatur dann letztendlich aus. Sie wird dann in der Karte auf den Punkten, deren Koordinaten in der Vektordatei stehen, gezeichnet.

Zeichen aus der Symboldatei:

```

SYMBOL
NAME "quadrat"
TYPE VECTOR
POINTS
0 0
0 1
1 1
1 0
0 0
END
FILLED TRUE
END

```

Angabe der Art des Zeichens bestehend aus Vektoren, einer Ellipse, einem Rasterbild oder einem TrueTypeFont, hier: Vektor

Angabe der Punkte in x- und y-Koordinaten aus denen die Vektoren bestehen

das Zeichen wird ausgefüllt

so könnte das Zeichen dann aussehen:



5.19.3 Using Cartographic Symbols in UMN MapServer

The basic graphic elements in a symbolfile are made up of vectors, truetype fonts, raster images or cartolines and are defined by the TYPE-parameter. The following section explains all four types of basic graphic elements and how they can be combined to create complex cartographic symbols.

Scaling of Symbols

The parameter SYMBOLSCALE in the LAYER section specifies the scale number at which the symbol or text label is displayed in exactly the pixel size defined by the SIZE parameter. Observe that the SIZE parameter is tightly connected to the SYMBOLSCALE parameter. The parameters MAXSIZE and MINSIZE inside the STYLE block limit the rendering of symbols to the maximum and minimum scale specified here. Both the minimum reasonable display size for symbols and the minimum of space required to display a symbol within a polygon should be observed.

Symbols of TYPE PIXMAP will not be scaled if used as line and polygon fills! In addition PIXMAPs will not display nicely when rescaled due to roundings caused by the integer arithmetics of the GD. The basic elements of a composite cartographic symbol may change their relative positions to each other, especially at very small scales (large number). They can also slightly change their form when they are displayed as tiny little images.

Furthermore the gaps of line symbols (the pattern itself) will not be scaled. The gaps stay fixed whereas the width is scaled. This can result in strange effects seeming to change the original pattern.

There is no possibility to separately define the display intervals (gaps) with MINSIZE and MAXSCALE in the STYLE-section (anybody need this?), so this tuning has to be solved at the LAYER level. TO do this create several layers with the same geometries for different scale levels and specify the appropriate sizes accordingly.

Always observe that cartographic symbols depend a lot on the scale! So be careful with the interaction of content, symbols and scale when creating projects. All three parameters heavily interact and have to be coordinated to produce a good map.

Symbols of TYPE VECTOR and ELLIPSE

The TYPE VECTOR defines the form of a symbol by setting x- and y-values in a local cartesian coordinate system with the origin at the upper left corner. Each symbol is parenthesized by the tags POINTS and END. The maximum number of points can be increased by changing the parameter MS_MAXVECTORPOINTS in the file mapsymbols.h. The current default is 100. Note that by setting the end point equal to the start point you obtain a closed form (polygon). To use the coordinates -99 -99 to break a line, the following point will not be connected by a line with the preceding one.

When creating symbols of the TYPE VECTOR you should observe some style guidelines. Avoid downtilted lines in area symbols, as they will lead to heavy aliasing effects. Furthermore you should not go below a useful minimum size, which is relevant for all types of symbols. Keep in mind that the points in the local coordinate system are rendered as pixel images, thus only integer values make any sense. Every symbol of the TYPE VECTOR is first rendered as a pixel image and then added to the geometry. This is the basic principle of the GD graphic kernel. PIXMAP symbols may be used directly for drawing.

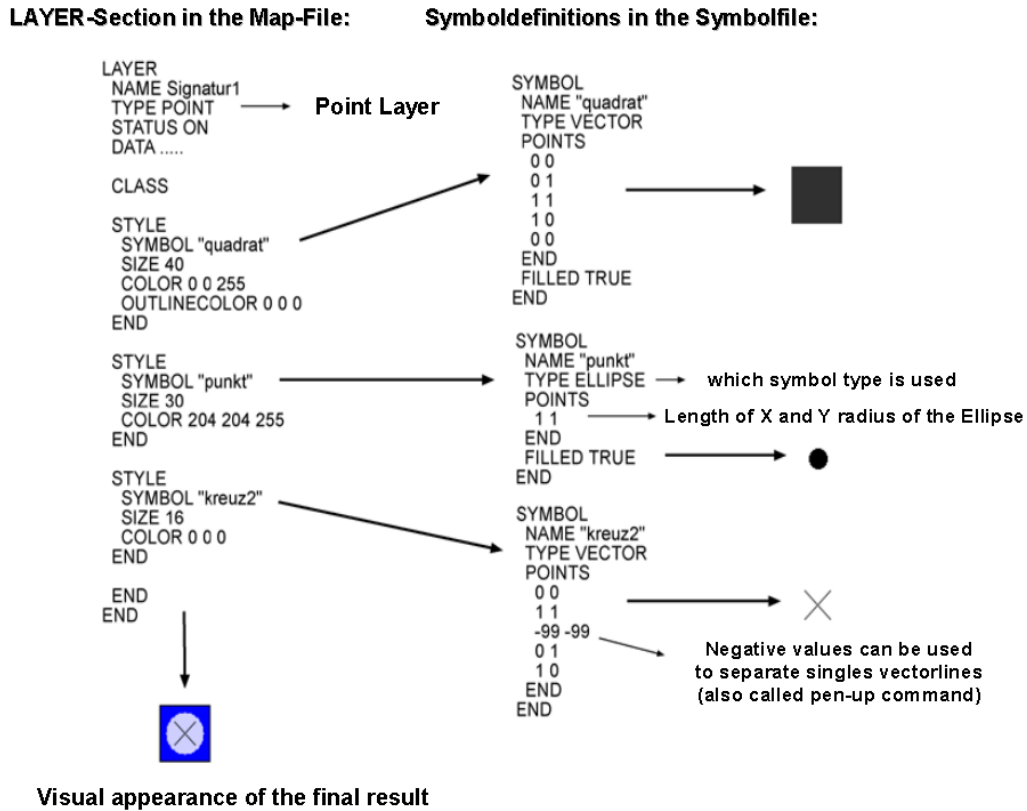
To create circles and ellipses use the TYPE ELLIPSE. These forms are created by setting the x and y values as the radius of the circle or ellipse inside POINTS and END.

Construction of Point Symbols

The following figure shows how to combine several basic elements to create a new point symbol. The combination is achieved by adding several STYLEs within one layer. Each basic element references one item of the SYMBOL file, which are centered and overlaid when rendered.

Notice that the SIZE parameter in the STYLE section only refers to the width of the symbol (size in the y-direction). An edgewise rectangle will thus display smaller than a lying rectangle, although both have the same SIZE parameter. When combining several point elements on top of each other, they will not always be centered correctly. We have not found a regular rule yet. We can only recommend to set an even-numbered SIZE for combined elements. Combining elements with even and odd numbered SIZE parameters seem to have larger irregularities.

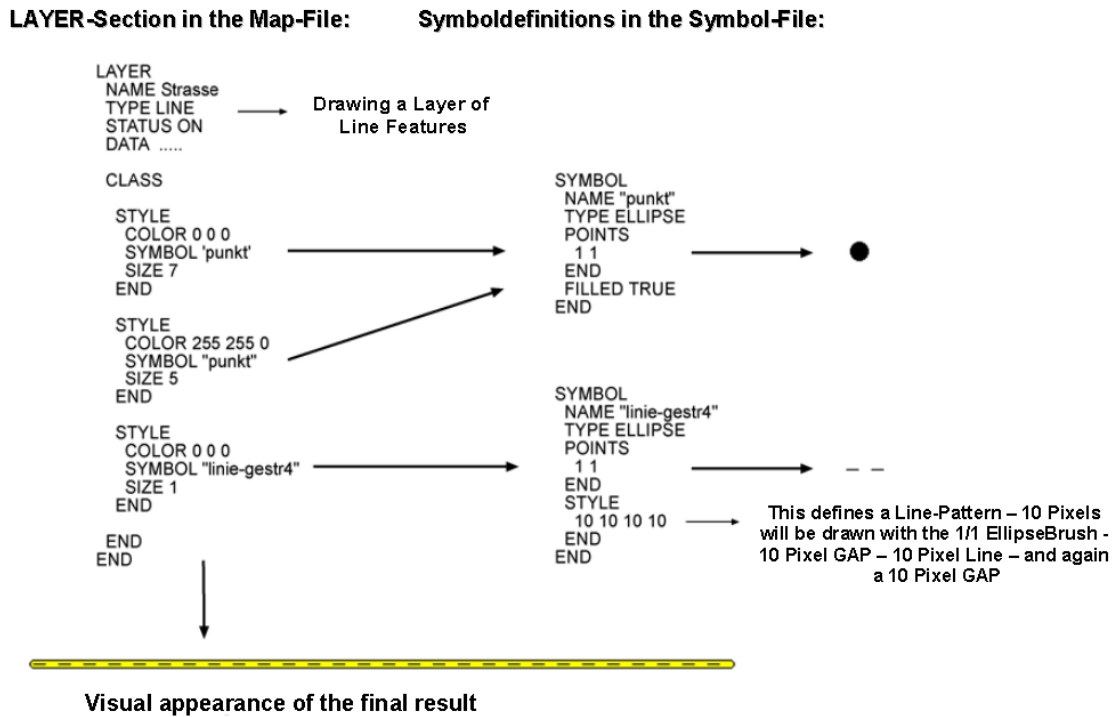
Figure3: Construction of Point Symbols



Construction of Line Symbols

For displaying line geometries most often a simple point symbol (filled circle / SYMBOL 0) is used. This point is painted for each pixel along the line, giving a continuous line with rounded ends. To create line patterns use the STYLE section of the SYMBOL file (do not confuse this with the STYLE-section of the CLASS object). Here you state how many pixel of the section shall be displayed and how many are left blank. This pattern will be repeated as many times as that pattern will fit into the element. The following figure shows this effect. Unfortunately up to now no OFFSET (start gap) can be defined to create asymmetric patterns. (anybody need this?)

Figure4: Construction of Line Symbols



When using the point character for rendering all line features will have rounded ends. This can be a desired effect or not, it gets more obvious the larger the width of line is set. Alternatively a rectangle can be used to generate a line. It can be enhanced with a STYLE-parameter to create line patterns (see below).

Table 1. Creating a Symbol to Display Railways

CLASS-section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE COLOR 102 102 102 SYMBOL 'point' SIZE 4 END STYLE COLOR 255 255 255 SYMBOL 'rectangle-train' SIZE 2 END END </pre>	<pre> SYMBOL NAME 'point' TYPE ELLIPSE POINTS 1 1 END FILLED TRUE END SYMBOL NAME 'rectangle-train' TYPE VECTOR POINTS 0 0 0 0.6 1 0.6 1 0 0 0 END FILLED TRUE STYLE 8 12 8 12 END END </pre>

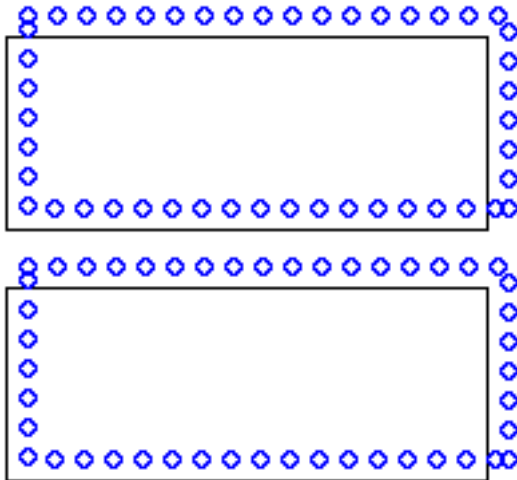
The STYLE parameter can be used for elements of the SYMBOL file with the TYPE VECTOR, ELLIPSE, CARTO-LINE and PIXMAP. It will define the number of intervals in which the symbol is rendered. This can be done using the GAP parameter with TRUETYPE symbols (see below).

When combining of several symbols on a line, they will be positioned on the baseline which is defined by the geometry of the object in pixel coordinates. Again we face the problem of centering. In most cases MapServer correctly centers symbols. The combination of a line displayed in 6 pixel width and overlaid with 4 pixel width results in a line symbol with a 1 pixel border. If the cartographic symbol is to contain a centered line with a width of 1 pixel, then the SIZE parameters have to be reconfigured for example to 7 and a 5 pixel. As a rule of thumb only combine even numbered or odd numbered SIZE parameters (see above).

In the STYLE section of the MAP file an OFFSET parameter can be set to shift symbols or characters in the x and y direction. Unfortunately the displacement is set relative to the map border, not the inclination of the line geometry (with the exception: OFFSET n -99). Therefore the iterated characters are all shifted in the same direction, independent of which direction the line takes (see Example 2). To generate asymmetrical line symbols apply -99 for the y value of the OFFSET. Then the x-value defines the distance from a parallel to the original geometry, for which the selected symbol is used (what a pity this wasn't documented anywhere!).

Table 2: Use of the OFFSET parameter with line signatures

CLASS-section from the Mapfile	Character from the Symbolfile	CLASS-Section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE SIZE 1 COLOR 0 0 0 END STYLE SYMBOL "circle" SIZE 7 COLOR 0 0 255 OFFSET 8 -8 END END </pre>	<pre> SYMBOL NAME "circle" TYPE ELLIPSE POINTS 1 1 END STYLE 1 10 1 10 END END </pre>	<pre> CLASS STYLE SIZE 1 COLOR 0 0 0 ANTIALIAS END STYLE SYMBOL "circle" SIZE 12 COLOR 0 0 255 OFFSET -8 -99 ANTIALIAS TRUE END END </pre>	<pre> SYMBOL NAME "circle" TYPE ELLIPSE POINTS 1 1 END STYLE 1 10 1 10 END END </pre>



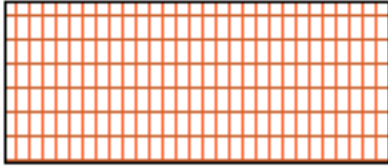
Area Symbols

Areas (polygons) can be filled with elements of the SYMBOL file to create e.g. hatches and graticules. These are by default rendered without spacing one after the other in x and y direction and fill out the whole polygon. Simple line hatches (e.g. horizontal, vertical and diagonal) can be created by filling the polygon with a line symbol from the SYMBOL file (see example 5).

The SIZE parameter in the STYLE section for line hatches only specifies the distance between the lines and not their width. Thus in these hatches all lines will always have a width of 1 pixel. Unfortunately there is no additional parameter to define the line width (Anybody need this feature?). Notice that the SIZE parameter is interpreted by MapServer differently for horizontal and vertical lines. Vertical lines with a SIZE parameter of 8 pixel result in a distance of 8 pixel between the lines. Horizontal lines with the same SIZE parameter are instead rendered with a much smaller gap (see example 5). For creating cross hatches composed of vertical and horizontal lines the best method is to use a simple cross from the SYMBOL file. Polygons can also be filled with other POINT elements to obtain special patterns (e.g. with circles or triangles).

Table 3: Construction of a cross hatch with different line distances

CLASS-Section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE SYMBOL "line-vertical" SIZE 8 COLOR 255 102 51 OUTLINECOLOR 0 0 0 END STYLE SYMBOL "line-horizontal" SIZE 8 COLOR 204 102 51 OUTLINECOLOR 0 0 0 END END </pre>	<pre> SYMBOL NAME "line-vertical" TYPE vector POINTS 0.5 0 0.5 1 END END SYMBOL NAME "line-horizontal" TYPE vector POINTS 0 0.5 1 0.5 END END </pre>



Signatures of TYPE PIXMAP

Symbols of the TYPE PIXMAP are simply small raster images. The name of the raster images are specified in the SYMBOL file with the parameter IMAGE. MapServer supports the raster formats GIF and PNG. Observe the color depth of the images and avoid using 24 bit PNG symbols displayed in 8 bit mode as this may cause unexpected color leaps. When using raster images the color cannot be modified in the SMBOL file subsequently. But you can specify a color with the TRANSPARENT parameter which will not be displayed - i.e. it will be transparent. As a result all underlying objects and colors are visible.

The SIZE parameter defines the size of PIXMAP symbols when used as point symbols. Observe that the pixel structure will show when the SIZE grows too large. Prevent this from happening by setting a MAXSIZE parameter. When using PIXMAPS as line symbols or as fill symbols for polygons they will not be resized but are displayed in the original size. So the use of PIXMAP STYLE objects for signatures with a high need of scaling is rather limited.

PIXMAP symbols are always rendered respective to the map border and will not follow the inclination of a line or polygon outline. Only truetype font symbols can follow an inclined line geometry, although with some defects (see below).

To create more complex area symbols, e.g. with defined distances between single characters or hatches with broad lines, raster images are probably the most suitable objects. Depending on the desired pattern you have to generate the raster image with high precision using a graphic editor. To obtain a regular allocation of symbols with defined spacing you could use the raster image as shown in Figure1.

Figure5: Raster image for a regular polygon symbol fill

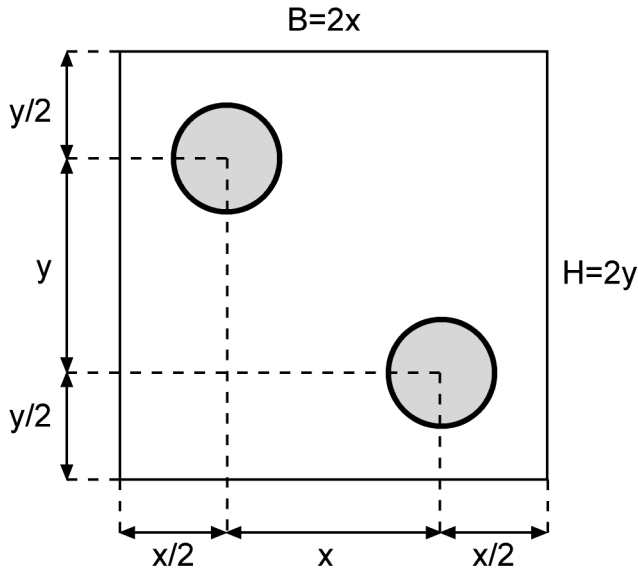
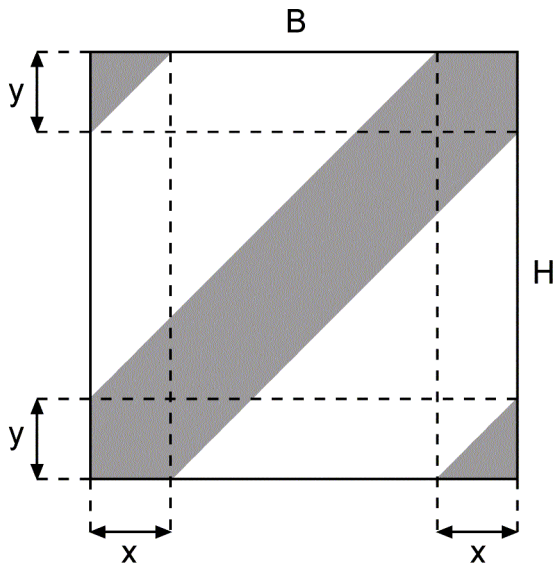


Figure6: Raster image for a hatched fill



Instead of using circles you can use different characters. B defines the width and H the height of the raster image. For a regular arrangement of symbols in a 45 degree angle $B = H$. For symbols, which are regularly arranged in parallel and without offset between each other one centered symbol with same x and y distances to the imageborder is enough. A regular hatch with wider lines can be created by using the raster image in fig. 2. To create a 45 degree hatch use:


$B = H$ and $x = y$

When using the MapServer legend observe that each raster PIXMAP is displayed only once in the original size in the middle of the legend box.

See the example below of some PIXMAP symbols, which can be used as area symbols with transparency. The raster images were created using FreeHand, finished with Photoshop and exported to PNG with special attention regarding the color palette. Observe that you *have* to specify a COLOR in the STYLE section of the CLASS to display raster

PIXMAPS although it has no influence on the output and color of the image.

Table 4: Construction of a horizontally arranged area symbol

CLASS-Section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE COLOR 255 255 0 END STYLE SYMBOL "in_the_star" COLOR 0 0 0 OUTLINECOLOR 0 0 0 END END </pre>	<pre> SYMBOL NAME "in_the_star" TYPE PIXMAP IMAGE "stern.png" TRANSPARENT 8 END </pre> 

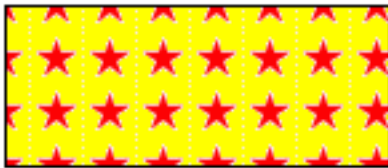



Table 5: Construction of a diagonally arranged area symbol

CLASS-Section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE SYMBOL "in_point1" COLOR 0 0 0 OUTLINECOLOR 0 0 0 END END </pre>	<pre> SYMBOL NAME "in_point1" TYPE PIXMAP IMAGE "flaechel_1.png" TRANSPARENT 13 END </pre> 

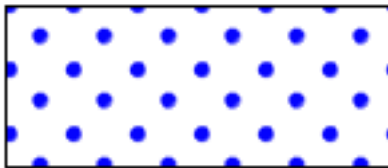



Table 6: Construction of a face hatch

CLASS-Section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE COLOR 255 255 0 END STYLE SYMBOL "in_hatch" COLOR 0 0 0 OUTLINECOLOR 0 0 0 END END </pre>	<pre> SYMBOL NAME "in_hatch" TYPE PIXMAP IMAGE "schraffur.png" TRANSPARENT 2 END </pre> 



Signatures of TYPE CARTOLINE

The TYPE Cartoline is not really an independent SYMBOL type or basic element but it is useful for creating line signatures. The advantage over the above types is the antialiasing of lines with any width. Conventional lines can only use antialiasing with a width of 1 pixel. This line type supports the ANTIALIAS parameter in the STYLE block of the CLASS definition just like the truetype fonts and symbols. Here you can also define line patterns and offsets. The design of the line ends can be controlled with the LINECAP parameter. The LINECAP value BUTT stops the signature exactly at the end of the top line. Author of this features is Tomas Krecmer ([tokr\(a\)tmapy.cz](mailto:tokr(a)tmapy.cz)). If you have questions concerning this type of signature or suggestions for improvement please contact the author.

Dashed line of TYPE CARTOLINE

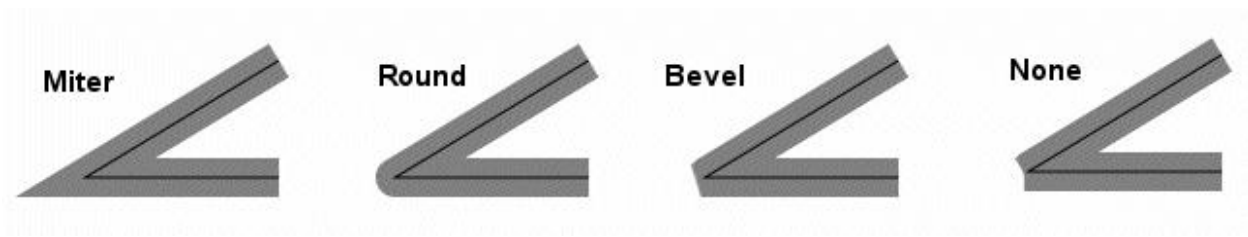
```

SYMBOL
  NAME "cartoline"
  TYPE cartoline
  LINECAP round #[butt/round/square/triangle]
  LINEJOIN miter #[round/miter/bevel]
  LINEJOINMAXSIZE 3
  STYLE
    40 17 1 17 1 17 1 17
  END
END

```

LINEJOIN

The different values for the parameter LINEJOIN have the following visual effects (see the following figure). Default is 'not set' but do not specify the string 'not set', just omit the parameter LINEJOIN.

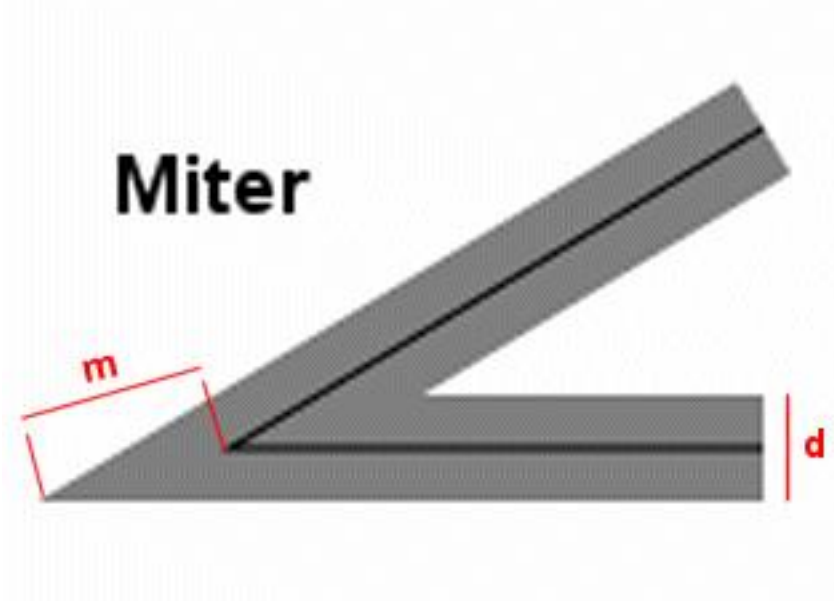


Miter follow line borders until they intersect, fill the resulting area.

Round see above

Bevel see above

None Lines will not be connected but drawn with LINECAPP 'butt' at the respective ends.



LINEJOINMAXSIZE

Specify the maximum length of m (only relevant for LINEJOIN type MITER). The value is a multiplication factor (default 3). The actual max length is calculated as follows:

```
m - current join size
d - symbol size
m_max = d * LINEJOINMAXSIZE
```

If current $m > m_max$ then the connection length will be set to m_max .

Signatures of TYPE TRUETYPE

Finally you can use symbols from truetype fonts. The symbol settings are also defined in the SYMBOL file. Specify the ASCII number of the symbol to be used in the CHARACTER parameter. Add the ALIAS name of the font file to the parameter FONT as defined in the file fonts.list. Remember to set the FONTSET parameter in the MAP file to link the correct fonts.list file. With the parameter ANTIALIAS you define whether to apply antialiasing to the symbols or characters. It is recommended to do this especially with more complex symbols and whenever they don't fit

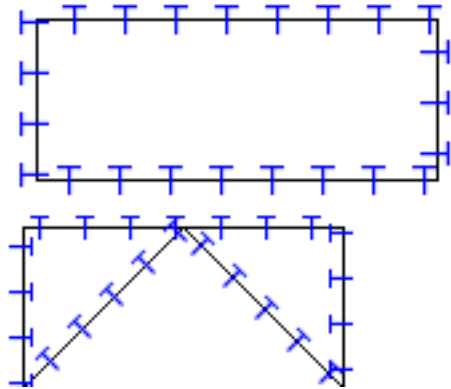
well into the raster matrix or show a visible pixel structure. Set a POSITION [ulluclur|cllclcr|llllcllr] relative to the geometric origin of the geometry. In the STYLE section of the LAYER object it is possible to define colors for true type symbols (as with signatures of the TYPE VECTOR).

When using truetype chracters to render complex line symbols, you can define the distance in pixel until the symbol is repeated by setting the GAP parmeter in the SYMBOL file. A more complicated pattern like the STYLE parameters for VECTOR, ELLIPSE or PIXMAP type symbols cannot be used not (Anybody need this feature?). The OFFSET parameter is currently not implemented yet. Truetype symbols follow the inclination of the accompanying line.

When using asymmetrical symbols they unfortunately do not always follow the outside or the inside, left or right of a line but change. If possible symbols will always be displayed upside (which makes a lot of sense for string characters, what true type fonts basically are). On vertical lines symbols are rendered to the right or left side depending on the drawing direction of the line. If the line is drawn from bottom to top, the truetype symbol is displayed to the left, if the line is drawn from top to bottom, the symbol is displayed to the right (see table below). In left picture in the table the line is drawn clockwise and in the right picture drawn counterclockwise.

Table 7: TrueType-character used on lines

CLASS-Section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE SYMBOL "T" SIZE 12 COLOR 0 0 255 END END </pre>	<pre> SYMBOL NAME "T" TYPE TRUETYPE FONT "arial" CHARACTER "&#84" ANTIALIAS TRUE GAP 10 END </pre>



To find out the character number of a symbol use one of the following options:

- Use the software FontMap (Shareware, with free trial version for download, thanks Till!)
- Use the MS Windows truetype map
- Trial and Error :-)

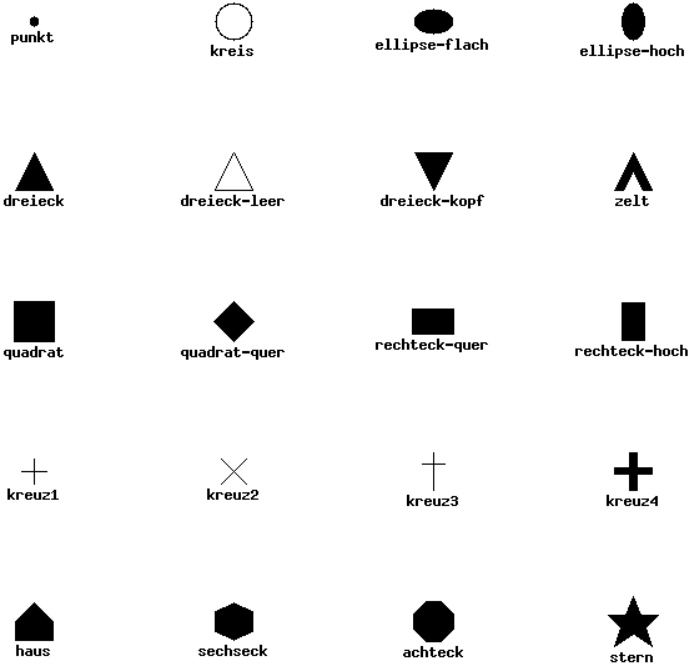
Please note that the numbering of the so-called “symbol fonts” starts at 61440! So if you want to use character T, you have to use 61440 + 84 = . (ain’t that a pain!!)

5.19.4 Examples

Find some examples to show different possibilities of the UMN MapServer for cartographical symbols of the vector based mapobjects (this is just a selection!):

Basic Symbols

Graphic Primitives for Point-Symbolizers located in the defined Symbolfile symbols.sym



Symboldefinitions from TrueTypeFont-Files



Graphic Primitives for Line-Symbolizers located in the defined Symbolfile symbols.sym

linie-gestr1

linie-gestr2

linie-gestr3

linie-gestr4

.....
linie-gepunkt1

.....
linie-gepunkt2

.....
linie-gepunkt3

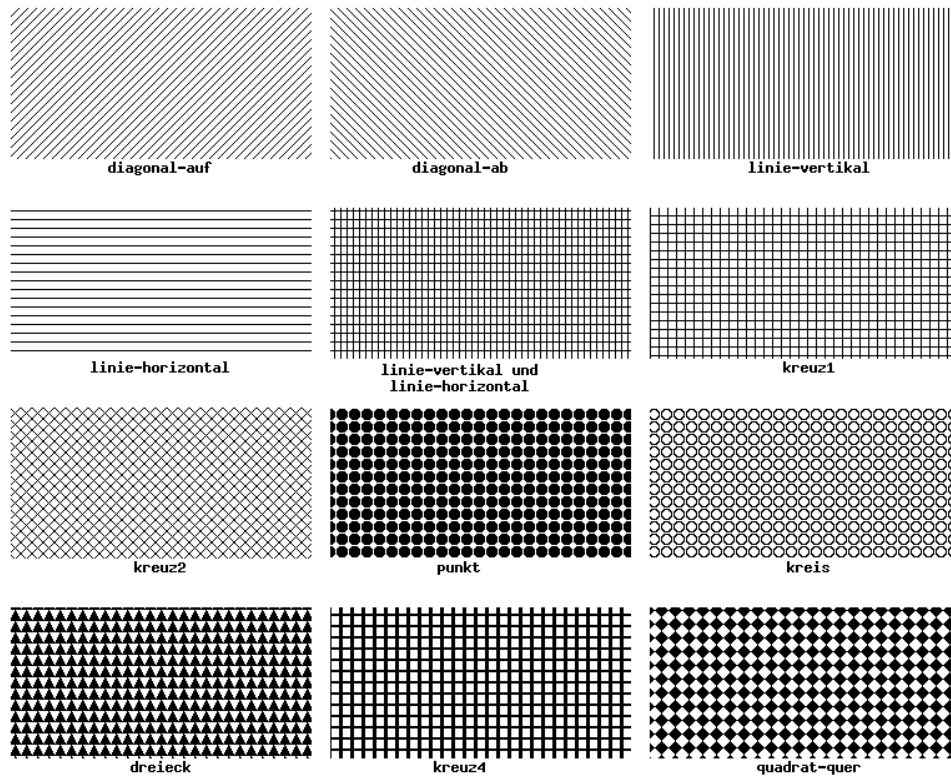
grenze1 (based on circle symbol of type ELLIPSE)

grenze2 (based on rectangle symbol of type VECTOR)

rechteck-quer-st

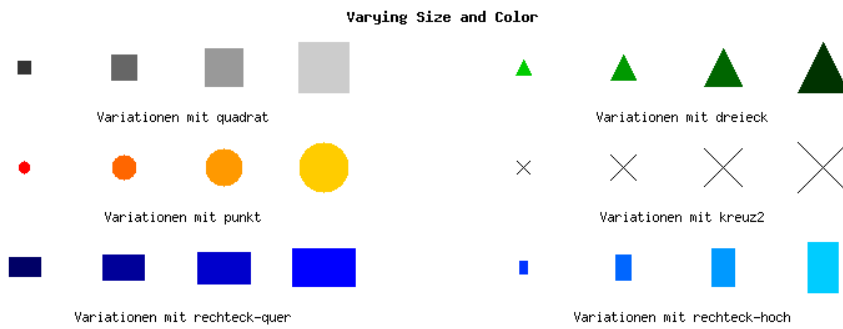
rechteck-bahn

Graphical Primitives for Polygon-Symbolizers located in the defined Symbolfile symbols.sym

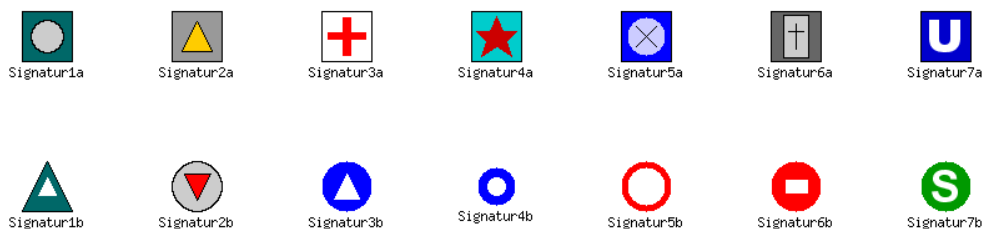


Complex Symbols

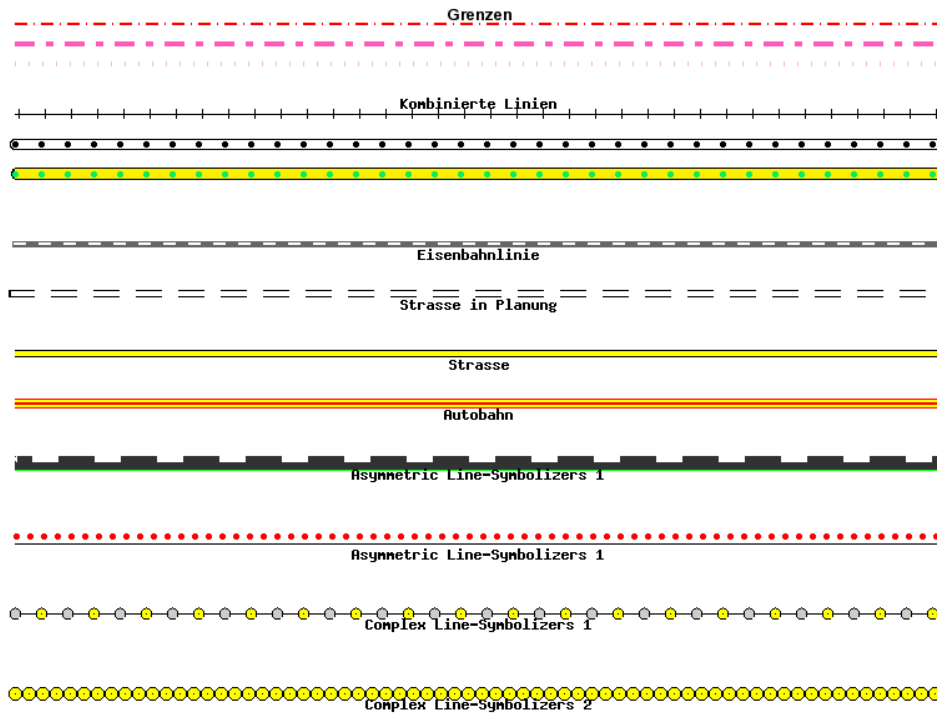
Examples of Point-Symbolizers varying some graphical Attributes



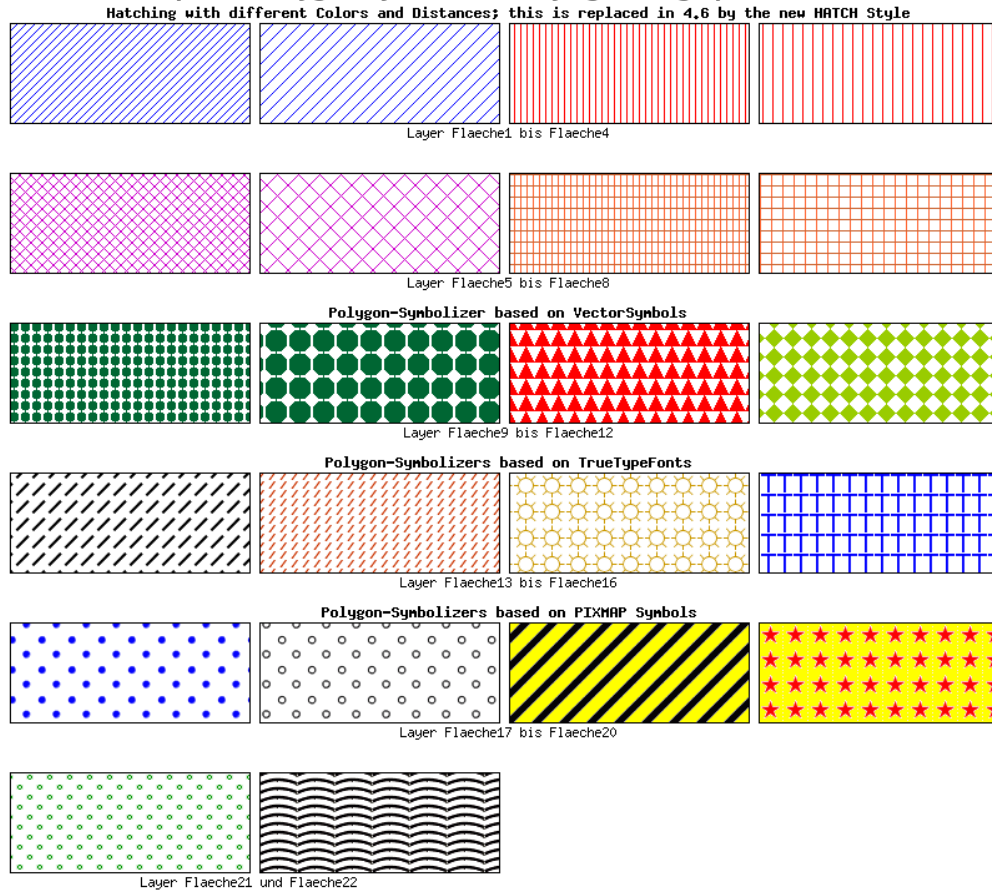
Examples for combinations of several Basetypes



Examples of combined Line-Symbolizers varying some graphical Attributes



Examples of Polygon-Symbolizers varying some graphical Attributes



5.19.5 Current Problems / Open Issues

The current version of MapServer renders open symbols (e.g. a cross) and empty symbols of the TYPE VECTOR incorrectly by filling them. This bug has been removed during our code examination and should be eliminated with a future release. Additionally the OUTLINECOLOR-parameter can not be used to create a 1 pixel outline for symbols and characters on a line. This can be achieved in a more flexible way as mentioned above. Another problem concerns the use of line patterns (e.g. dashed line) on features with many points and nooks. The line pattern will then often be displayed irregularly (see Table9).

Table 8: Use of a simple cross on a line

CLASS-Section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE SIZE 1 COLOR 0 0 0 END STYLE SYMBOL "cross2" SIZE 8 COLOR 204 153 0 OFFSET 1 -7 END END </pre>	<pre> SYMBOL NAME "cross2" TYPE VECTOR POINTS 0 0 1 1 -99 -99 0 1 1 0 END STYLE 1 15 1 15 END END </pre>



Table9: Irregularities with line patterns

CLASS-Section from the Mapfile	Character from the Symbolfile
<pre> CLASS STYLE SYMBOL "border2" SIZE 2 COLOR 255 0 0 END END </pre>	<pre> SYMBOL NAME "border2" TYPE VECTOR POINTS 0 0 1 0 1 0.8 0 0.8 0 0 END STYLE 10 6 1 6 END FILLED TRUE END </pre>



All symbols in this document were created with MAP files and SYMBOL files, which can be [downloaded](#). If you want to use this MAP file please note, that your MapServer must at least be able to handle with 50 symbols. Otherwise you get an error while loading the SYMBOL files.

I hope that this document will help you to present your data in a cartographically nice manner with the UMN MapServer and explains some basics and possibilities of the concept of the UMN MapServer as well as some weakness. It would be great to put together a cartographic symbols library for the profit of everyone. This especially concerns truetype-fonts, which were developed for some projects and contain some typical signatures for cartographic needs.

You can also view the [discussion paper](#) for the improvement of the MapServer Graphic-Kernel (sorry as yet German

only).

5.20 Symbology Examples

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8295 \$

Date \$Date: 2008-12-26 21:08:04 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2008/07/16

Table of Contents

- Symbology Examples
 - Example 1. Dashed Line
 - Example 2. TrueType font marker symbol
 - Example 3. Vector triangle marker symbol
 - Example 4. Non-contiguous vector marker symbol (Cross)
 - Example 5. Circle vector symbol
 - Example 6. Downward diagonal fill
 - Example 7. Dashed Cartoline symbol
 - Example 8. Using the Symbol Type HATCH (new in 4.6)

The following example creates a dashed line with 10 pixels on, 5 off, 5 on, 10 off ...

5.20.1 Example 1. Dashed Line

```
SYMBOL
NAME 'dashed1'
TYPE ELLIPSE
POINTS 1 1 END
FILLED true
PATTERN 10 5 5 10 END
END
```

The next example symbol is a star, used to represent the national capital, hence the name. The font name is defined in the *FONTSET* file. The code number “114” varies, you can use MS Windows’ character map to figure it out, or guesstimate.

5.20.2 Example 2. TrueType font marker symbol

```
SYMBOL
NAME "natcap"
TYPE TRUETYPE
FONT g e o
FILLED true
ANTIALIAS true
CHARACTER "&#114;"
END
```

The next example is fairly straight forward. Note that to have 3 sides you need 4 points, hence the first and last points are identical.

5.20.3 Example 3. Vector triangle marker symbol

```
SYMBOL
  NAME "triangle"
  TYPE vector
  POINTS
    0 4
    2 0
    4 4
    0 4
  END
END
```

The next example draws a cross, that is 2 lines (vectors) that are not connected end-to-end (Like the triangle in the previous example). The negative values separate the two.

5.20.4 Example 4. Non-contiguous vector marker symbol (Cross)

```
SYMBOL
  NAME "cross"
  TYPE vector
  POINTS
    2 0
    2 4
    -99 -99
    0 2
    4 2
  END
END
```

The next example creates a simple filled circle. Using non-equal values for the point will give you an actual ellipse.

5.20.5 Example 5. Circle vector symbol

```
SYMBOL
  NAME "circle"
  TYPE ellipse
  FILLED true
  POINTS
    1 1
  END
END
```

5.20.6 Example 6. Downward diagonal fill

```
SYMBOL
  NAME "downwarddiagonalfill"
  TYPE vector
```

```

TRANSPARENT 0
POINTS
  0 1
  1 0
END
END

```

The next example creates a dashed line with 10 pixels on, 5 off,... The line will have butt caps and short miter joins. For layers with a scaled symbol (SYMBOLSCALE, MINSIZE, MAXSIZE, ...) the PATTERN will be resized to maintain symbol ratios.

5.20.7 Example 7. Dashed Cartoline symbol

```

SYMBOL
  NAME "cartoline"
  TYPE cartoline
  LINECAP butt
  LINEJOIN miter
  LINEJOINMAXSIZE 1
  PATTERN 10 5 END
END

```

5.20.8 Example 8. Using the Symbol Type HATCH (new in 4.6)

As of MapServer 4.6, you can now use the symbol type HATCH to produce hatched lines. The following will display hatched lines at a 45 degree angle, 10 pixels apart, and 3 pixels wide.

Symbol definition:

```

SYMBOL
  NAME 'hatch-test'
  TYPE HATCH
END

```

Layer definition:

```

LAYER
  ...
  CLASS
    ...
    STYLE
      SYMBOL 'hatch-test'
      COLOR 255 0 0
      ANGLE 45
      SIZE 10
      WIDTH 3
    END
  END
END

```

Other parameters available for HATCH are: ANGLEITEM, SIZEITEM, MINWIDTH, and MAXWIDTH.

5.21 Templating

Author Frank Koormann

Contact frank.koormann at intevation.de

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 9647 \$

Date \$Date: 2010-01-01 07:12:59 -0800 (Fri, 01 Jan 2010) \$

Last Updated 2008/07/18

Table of Contents

- Templating
 - Introduction
 - Format
 - Example Template

5.21.1 Introduction

Templates are used:

- to define the look of a MapServer CGI application interface and
- to present the results of a query.

They guide the presentation of results, either a query or a map, to the user. Templates are almost always HTML files although they can also be a URL (e.g.. [http://www.somewhere.com/\[ATTRIBUTE\]/info.html](http://www.somewhere.com/[ATTRIBUTE]/info.html)). URL templates can only be used with simple QUERY or ITEMQUERY results so many substitutions defined below are not available for them. Simple pan/zoom interfaces use a single template file while complicated queries often require many templates. Templates often use JavaScript to enhance the basic interface.

Notes

- Templates *must* contain the magic string ‘mapserver template’ in the first line of the template. Often this takes the form of an HTML, javascript or XML comment. This line is *not* written to the client. The magic string is not case sensitive.
- All *CGI parameters* can be referenced in template substitutions, MapServer specific parameters as well as user defined ones. In principle parameters are handed through by the MapServer 1:1. This feature is essential for implementing MapServer applications.

The reference below only lists special template substitution strings which are needed to obtain information modified by the MapServer, e.g. a new scale, query results, etc.

- Template substitution strings are case sensitive.
- Attribute item substitutions must be the same case as the item names in the dbase file.
- ArcView and ArcInfo generally produce dbase files with item names that are all uppercase. Appropriate URL encoding (i.e. ‘ ‘ to ‘+’) is applied when templates are URLs.
- Some substitutions are also available in escaped form (i.e. URL encoded).

As an example this is needed when generating links within a template. This might pass the current mapextent to a new MapServer call. `[mapext]` is substituted by a space delimited set of lower left and upper right coordinates. This would break the URL. `[mapext_esc]` is substituted by a proper encoded set.

5.21.2 Format

Templates are simply HTML files or URL strings that contains special characters that are replaced by `mapserv` each time the template is processed. The simple substitution allows information such as active layers or the spatial extent to be passed from the user to `mapserv` and back again. Most often the new values are dumped into form variables that will be passed on again. The list of special characters and form variables is given below. HTML templates can include just about anything including JavaScript and Java calls.

In HTML files, the attribute values can be inside quotes(“”). Writing attribute values inside quotes allows you to set special characters in value that you couldn’t use normally (ie:],=,” and space). To write a single quote in a attribute value, just use two quotes (“”).

General

[version] The MapServer version number.

[id] Unique session id. The id can be passed in via a form but is more commonly generated by the software. In that case the id is a concatenation of UNIX time (or NT equivalent) and the process id. Unless you’re getting more requests in a second than the system has process ids the id can be considered unique. ;->

[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]`.

Also available as escaped version.

[web_meta data key],[web_meta data key_esc] Web object meta data access (e.g `[web_projection]`)

Also available as escaped version.

[errmsg], [errmsg_esc] Current error stack output. Various error messages are delimited by semi-colons.

Also available as escaped version.

File Reference

[img] Path (relative to document root) of the new image, just the image name if `IMAGE_URL` is not set in the mapfile.

In a map interface template, `[img]` is substituted with the path to the map image. In a query results template, it is substituted with the path to the querymap image (if a `QUERYMAP` object is defined in the *Mapfile*).

[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.

Since version 3.5.1 a new HTML Legend template is provided by MapServer. If a template is defined in the *Mapfile* the `[legend]` string is replaced by the processed legend as. See the *HTML Legends with MapServer* for details.

[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).

Also available as escaped version.

[mapwidth], **[mapheight]** Current image width or height.

[scaledenom] 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.

[scale] - **deprecated** Since MapServer 5.0 the proper parameter to use is [scaledenom] instead. The deprecated [scale] is the 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 an 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).

Also available as escaped version. (mapext_esc is deprecated in MapServer 5.2. You should use the “escape=” argument instead)

The default template [mapext] returns coordinates in the format of: mixx miny maxx maxy

Available arguments:

- **escape=** Escape the coordinates returned. Default is “none”. Valid values are:
 - “**url**” Use URL escape codes to encode the coordinates returned.
 - “**none**” Do not escape.
- **expand=** Expand the bounds of the extents by a specific value. Specified in map coordinates. For example, [mapext] might return:

```
123456 123456 567890 567890
```

and [mapext expand=1000] would therefore return:

```
122456 122456 568890 568890
```

- **format=** Format of the coordinates. Default is “\$minx \$miny \$maxx \$maxy”. For example, to add commas to the coordinates you would use:

```
[mapext format="$minx,$miny,$maxx,$maxy"]
```

- **precision=** The number of decimal places to output for coordinates (default is 0).

[minx], [miny], [maxx], [maxy] Minimum / maximum X or Y coordinate of new map extent.

[dx], [dy] The differences of minimum / maximum X or Y coordinate of new map extent.

Useful for creating cachable extents (i.e. 0 0 dx dy) with legends and scalebars

[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. Useful for spatial query building.

Also available as escaped version. (rawext_esc is deprecated in MapServer 5.2. You should use the “escape=” argument instead)

The default template [rawext] returns coordinates in the format of: mixx miny maxx maxy

Available arguments:

- **escape=** Escape the coordinates returned. Default is “none”. Valid values are:
 - “**url**” Use URL escape codes to encode the coordinates returned.
 - “**none**” Do not escape.
- **expand=** Expand the bounds of the extents by a specific value. Specified in map coordinates. For example, [rawext] might return:

```
123456 123456 567890 567890
```

and [rawext expand=1000] would therefore return:

```
122456 122456 568890 568890
```

- **format=** Format of the coordinates. Default is “\$minx \$miny \$maxx \$maxy”. For example, to add commas to the coordinates you would use:

```
[rawext format="$minx,$miny,$maxx,$maxy"]
```

- **precision=** The number of decimal places to output for coordinates (default is 0).

[rawminx], [rawminy], [rawmaxx], [rawmaxy] Minimum / maximum X or Y coordinate of a raw map/search extent.

The following substitutions are only available if the MapServer was compiled with PROJ support and a *PROJECTION* is defined in the *Mapfile*.

[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.

Also available as escaped version. (mapext_latlon_esc is deprecated in MapServer 5.2. You should use the “escape=” argument instead)

The default template [mapext_latlon] returns coordinates in the format of: mixx miny maxx maxy

Available arguments:

- **escape=** Escape the coordinates returned. Default is “none”. Valid values are:
 - “**url**” Use URL escape codes to encode the coordinates returned.
 - “**none**” Do not escape.
- **expand=** Expand the bounds of the extents by a specific value. Specified in map coordinates. For example, [mapext_latlon] might return:

```
123456 123456 567890 567890
```

and [mapext_latlon expand=1000] would therefore return:

```
122456 122456 568890 568890
```

- **format=** Format of the coordinates. Default is “\$minx \$miny \$maxx \$maxy”. For example, to add commas to the coordinates you would use:

```
[mapext_latlon format="$minx,$miny,$maxx,$maxy"]
```

- **precision=** The number of decimal places to output for coordinates (default is 0).

[minlon], [minlat], [maxlon] [maxlat] Minimum / maximum longitude or latitude value of mapextent. Available only when projection enabled.

[refext], [refext_esc] Reference map extent (separated by spaces).

This template has been added with version 4.6 on behalf of an enhancement request. See the thread in the [MapServer ticket#1102](#) for potential use cases.

Also available as escaped version. (refext_esc is deprecated in MapServer 5.2. You should use the “escape=” argument instead)

The default template [refext] returns coordinates in the format of: mixx miny maxx maxy

Available arguments:

- **escape=** Escape the coordinates returned. Default is “none”. Valid values are:
 - “**url**” Use URL escape codes to encode the coordinates returned.
 - “**none**” Do not escape.
- **expand=**

Expand the bounds of the extents by a specific value. Specified in map coordinates. For example, [refext] might return:

```
123456 123456 567890 567890
```

and [refext expand=1000] would therefore return:

```
122456 122456 568890 568890
```

- **format=** Format of the coordinates. Default is “\$minx \$miny \$maxx \$maxy”. For example, to add commas to the coordinates you would use:

```
[refwext format="$minx,$miny,$maxx,$maxy"]
```

- **precision=** The number of decimal places to output for coordinates (default is 0).

Layer

[layers] | [layers_esc] All active layers space delimited. Used for a “POST” request.

Also available as escaped version.

[toggle_layers] | **[toggle_layers_esc]** List of all layers that can be toggled, i.e. all layers defined in the *Mapfile* which status is currently not default.

Also available as escaped version.

[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 *Mapfile*. 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

The following substitutions are only available when the template is processed as a result of a query.

[shpext], **[shpext_esc]** Extent of current shape plus a 5 percent buffer. Available only when processing query results.

The default template [shpext] returns coordinates in the format of: mixx miny maxx maxy

Available arguments:

- **escape=** Escape the coordinates returned. Default is “none”. Valid values are:
 - “url” Use URL escape codes to encode the coordinates returned.
 - “none” Do not escape.
- **expand=** Expand the bounds of the extents by a specific value. Specified in map coordinates. For example, [shpext] might return:

```
123456 123456 567890 567890
```

and [shpext expand=1000] would therefore return:

```
122456 122456 568890 568890
```

- **format=** Format of the coordinates. Default is “\$minx \$miny \$maxx \$maxy”. For example, to add commas to the coordinates you would use:

```
[shpext format="$minx,$miny,$maxx,$maxy"]
```

- **precision=** The number of decimal places to output for coordinates (default is 0).

[shpminx], **[shpminy]**, **[shpmaxx]**, **[shpmaxy]** Minimum / maximum X or Y coordinate 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 coordinate 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.

[shpclass] Classindex value of the current shape. Available only when processing query results.

[shpxy formatting options] The list of shape coordinates, with list formatting options, especially useful for SVG.

The default template [shpxy] returns a comma separated list of space delimited of coordinates (i.e. x1 y1, x2 y2, x3 y3).

Available only when processing query results.

Available attributes (h = header, f=footer, s=separator):

- **cs=** Coordinate separator, default is comma (“,”).
- **xh=, xf=** Characters to put before and after the x coordinates (defaults are xh=”” and xf=””).
- **yh= yf=** Characters to put before and after the y coordinates (defaults are no characters).
- **ph=, pf=, ps=** Characters to put before and after and separators between feature parts (e.g. holes, defaults are no characters).
- **sh=, sf=** Characters to put before and after a feature (defaults are no characters)
- **precision=** The number of decimal places to output for coordinates (default is 0).
- **proj=** The output projection definition for the coordinates (default is none), a special value of “image” will convert to image coordinates.

As a simple example:

```
[shpxy xh="(" yf=")"] will result in: (x1 y1), (x2 y2), (x3 y3)
```

[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.

By default the attributes are encoded especially for HTML representation. In addition the escaped version (for use in URLs) as well as the raw data is available.

[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 *Mapfile* has to be given, second the tables fields can be accessed similar to the layers attribute data. Available only when processing query results.

By default the attributes are encoded especially for HTML representation. In addition the escaped version (for use in URLs) as well as the raw data is available.

[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 *Mapfile*.

[metadata_meta data key], [metadata_meta data key_esc] Queried layer meta data access (e.g [meta-data_projection])

Also available as escaped version.

For query modes that allow for multiple result sets, the following string substitutions are available. For FEATURESELECT and FEATURENSELECT modes the totals are adjusted so as not to include the selection layer. The selection layer results ARE available for display to the user.

- [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.

5.21.3 Example Template

A small example to give an idea how to work with templates. Note that it covers MapServer specific templates (e.g. the [map], [mapext]) and user defined templates (e.g. [htmlroot] or [program]) used to store application settings.

```

1  <!-- MapServer Template -->
2  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3      "http://www.w3.org/TR/html4/transitional.dtd">
4  <html>
5      <head>
6          <title>MapServer Template Sample</title>
7      </head>
8
9      <body>
10         MapServer Template Sample<br>
11
12         <!-- The central form the application is based on. -->
13         <form method="GET" action="[program]">
14
15             <!-- CGI MapServer applications are server stateless in principle,
16                  all information must be "stored" in the client. This includes
17                  some basic settings as below.
18                  The example is based on the pan and zoom test suite:
19                  http://maps.dnr.state.mn.us/mapserver_demos/tests36/          -->
20             <input type="hidden" name="map" value="[map]">
21             <input type="hidden" name="imgext" value="[mapext]">
22             <input type="hidden" name="imgxy" value="149.5 199.5">
23             <input type="hidden" name="program" value="[program]">
24             <input type="hidden" name="htmlroot" value="[htmlroot]">
25             <input type="hidden" name="map_web" value="[map_web]">
26
27             <!-- A table for minimal page formatting. -->
28             <table border=0 cellpadding=5>
29                 <tr>
30                     <!-- First column: Map and scale bar -->
31                     <td align=center>
32                         <!-- The map -->
33                         <input type="image" name="img" src="[img]"

```

```

34         style="border:0;width:300;height:400">
35         <br>
36         <!-- The scale bar-->
37         
38     </td>
39
40     <!-- Second column: Zoom direction, Legend and Reference -->
41     <td valign=top>
42         <!-- Zoom direction -->
43         <b>Map Controls</b><br>
44         Set your zoom option:<br>
45         <select name="zoom" size="1">
46             <option value="2" [ z o o m _ 2 _ s e l e c t ] > Zoom in 2 times
47             <option value="1" [ z o o m _ 1 _ s e l e c t ] > Recenter Map
48             <option value="-2" [ z o o m _ - 2 _ s e l e c t ] > Zoom out 2 times
49         </select>
50         <br>
51
52         <!-- Legend -->
53         <b>Legend</b><br>
54         <br><br><br><br>
55
56         <!-- Reference map -->
57         <input type="image" name="ref" src="[ref]"
58             style="border:0;width:150;height:150">
59     </td>
60 </tr>
61 </table>
62
63 </form>
64
65 </body>
66 </html>

```

5.22 Variable Substitution

Syntax: ‘%’ + variable name + ‘%’

Example 1. Connecting securely to a Spatial Database

You want to map some sensitive data held in a PostGIS database. The username and password to be used for the database connection are held in 2 cookies previously set by a separate authentication mechanism, “uid” and “passwd”.

```
CONNECTION "user=%uid% password=%passwd% dbname=postgis"
```

Example 2. Handling temporary files

You have a user based discovery application that generates shapefiles and stores them in a user’s home directory on the server. The “username” comes from a cookie, the “filename” comes from a request parameter.

```
DATA "/home/%username%/tempshp/%filename%"
```

This feature is only available in the CGI version of MapServer through a mapfile pre-processor. If you are using MapScript, you will have to code the substitution logic into your application yourself (By writing your own pre-processor).

5.23 WEB

EMPTY [url] URL to forward users to if a query fails. If not defined the value for ERROR is used.

ERROR [url] URL to forward users to if an error occurs. Ugly old MapServer error messages will appear if this is not defined

FOOTER [filename] Template to use AFTER anything else is sent. Multiresult query modes only.

HEADER [filename] Template to use BEFORE everything else has been sent. Multiresult query modes only.

IMAGEPATH [path] Path to the temporary directory for writing temporary files and images. Must be writable by the user the web server is running as. Must end with a / or depending on your platform.

IMAGEURL [path] Base URL for IMAGEPATH. This is the URL that will take the web browser to IMAGEPATH to get the images.

LOG [filename] File to log MapServer activity in. Must be writable by the user the web server is running as.

MAXSCALEDENOM [double] Maximum scale at which this interface is valid. When a user requests a map at a bigger scale, MapServer automatically returns the map at this scale. This effectively prevents user from zooming too far out. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated MAXSCALE parameter. Deprecated since version 5.0.

MAXSCALE [double] - deprecated Since MapServer 5.0 the proper parameter to use is MAXSCALEDENOM instead. The deprecated MAXSCALE is the maximum scale at which this interface is valid. When a user requests a map at a bigger scale, MapServer automatically returns the map at this scale. This effectively prevents user from zooming too far out. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000.

See Also:

Map scale

MAXTEMPLATE [file|url] Template to be used if above the maximum scale for the app, useful for nesting apps.

METADATA This keyword allows for arbitrary data to be stored as name value pairs. This is used with OGC WMS to define things such as layer title. It can also allow more flexibility in creating templates, as anything you put in here will be accessible via template tags. Example:

```
METADATA
  title "My layer title"
  author "Me!"
END
```

MINSCALEDENOM [double] Minimum scale at which this interface is valid. When a user requests a map at a smaller scale, MapServer automatically returns the map at this scale. This effectively prevents the user from zooming in too far. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated MINSSCALE parameter.

See Also:

Map scale

MINSSCALE [double] - deprecated Since MapServer 5.0 the proper parameter to use is MINSCALEDENOM instead. The deprecated MINSSCALE is the minimum scale at which this interface is valid. When a user requests a map at a smaller scale, MapServer automatically returns the map at this scale. This effectively prevents the user from zooming in too far. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Deprecated since version 5.0.

MINTEMPLATE Template to be used if above the minimum scale for the app, useful for nesting apps.

OUTPUTFORMAT [mime-type] Format of the query output. Default is “text/html”. This is experimental, the use of the *OUTPUTFORMAT* object is recommended instead.

TEMPLATE [filenameurl] Template file or URL to use in presenting the results to the user in an interactive mode (i.e. map generates map and so on ...)

5.24 Notes

- The Mapfile 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.
- For MapServer versions < 5, there was a default maximum of 200 layers per mapfile (there is no layer limit with MapServer >= 5). This can be changed by editing the map.h file to change the value of MS_MAXLAYERS to the desired number and recompiling. Here are other important default limits when using a MapServer version < 5:
 - MAXCLASSES 250 (set in map.h)
 - MAXSTYLES 5 (set in map.h)
 - MAXSYMBOLS 64 (set in mapsymbol.h)

MapServer versions >= 5 have no limits for classes, styles, symbols, or layers.

- File paths may be given as absolute paths, or as paths relative to the location of the mapfile. In addition, data files may be specified relative to the SHAPEPATH.
- The mapfile has a hierarchical structure, with the MAP object being the “root”. All other objects fall under this one.
- Comments are designated with a #.
- Attributes are named using the following syntax: [ATTRIBUTENAME] ... Note that the name of the attribute included between the square brackets *IS CASE SENSITIVE*. Generally ESRI generated shapefiles have their attributes (.dbf column names) all in upper-case for instance, and for PostGIS, *ALWAYS* use lower-case.
- MapServer Regular Expressions are used through the operating system’s C Library. For information on how to use and write Regular Expressions on your system, you should read the documentation provided with your C Library. On Linux, this is Glibc, and you can read “man 7 regex” ... This man page is also available on most UNIX’s. Since these RegEx’s are POSIX compliant, they should be the same on Windows as well, so windows users can try searching the web for “man 7 regex” since man pages are available all over the web.

MAPSCRIPT

Release 5.4.2

Date January 13, 2010

6.1 Introduction

This is language agnostic documentation for the MapScript interface to MapServer generated by SWIG. This document is intended for developers and to serve as a reference for writers of more extensive, language specific documentation located at *Mapfile*

6.1.1 Appendices

Language-specific extensions are described in the following appendices

Python Appendix

6.1.2 Documentation Elements

Classes will be documented in alphabetical order in the manner outlined below. Attributes and methods will be formatted as definition lists with the attribute or method as item, the type or return type as classifier, and a concise description. To make the document as agnostic as possible, we refer to the following types: int, float, and string. There are yet no mapscript methods that return arrays or sequences or accept array or sequence arguments.

We will use the SWIG term *immutable* to indicate that an attribute's value is read-only.

6.1.3 fooObj

A paragraph or two about class fooObj.

fooObj Attributes

attribute [type [access]] Concise description of the attribute.

Attribute name are completely lower case. Multiple words are packed together like *outlinecolor*.

Note that because of the way that mapscript is generated many confusing, meaningless, and even dangerous attributes are creeping into objects. See `outputFormatObj.refcount` for example. Until we get a grip on the structure members we are exposing to SWIG this problem will continue to grow.

fooObj Methods

method(type mandatory_parameter [, type optional_parameter=default]) [type] Description of the method including elaboration on the method arguments, the method's actions, and returned values. Optional parameters and their default values are enclosed in brackets.

Class method names are camel case with a leading lower case character like *getExpressionString*.

6.1.4 Additional Documentation

There's no point in duplicating the MapServer Mapfile Reference, which remains the primary reference for mapscript class attributes.

6.2 SWIG MapScript API Reference

Author Sean Gillies

Author Steve Lime

Contact steve.lime at dnr.state.mn.us

Author Frank Warmerdam

Contact warmerdam at pobox.com

Author Umberto Nicoletti

Contact umberto.nicoletti at gmail.com

Author Tamas Szekeres

Contact szekerest at gmail.com

Author Daniel Morissette

Contact dmorissette at mapgears.com

Revision \$Revision: 8391 \$

Date \$Date: 2009-01-06 10:27:50 -0800 (Tue, 06 Jan 2009) \$

Contents

- SWIG MapScript API Reference
 - Introduction
 - * Appendices
 - * Documentation Elements
 - * fooObj
 - * Additional Documentation
 - MapScript Functions
 - MapScript Classes
 - * classObj
 - * colorObj
 - * errorObj
 - * fontSetObj
 - * hashTableObj
 - * imageObj
 - * intarray
 - * labelCacheMemberObj
 - * labelCacheObj
 - * labelObj
 - * layerObj
 - * legendObj
 - * lineObj
 - * mapObj
 - * markerCacheMemberObj
 - * outputFormatObj
 - * OWSRequest
 - * pointObj
 - * projectionObj
 - * rectObj
 - * referenceMapObj
 - * resultCacheMemberObj
 - * resultCacheObj
 - * scalebarObj
 - * shapefileObj
 - * shapeObj
 - * styleObj
 - * symbolObj
 - * symbolSetObj
 - * webObj

6.2.1 Introduction

This is language agnostic documentation for the mapscript interface to MapServer generated by SWIG. This document is intended for developers and to serve as a reference for writers of more extensive, language specific documentation in DocBook format for the MDP.

Appendices

Language-specific extensions are described in the following appendices

Python MapScript Appendix

Documentation Elements

Classes will be documented in alphabetical order in the manner outlined below. Attributes and methods will be formatted as definition lists with the attribute or method as item, the type or return type as classifier, and a concise description. To make the document as agnostic as possible, we refer to the following types: int, float, and string. There are yet no mapscript methods that return arrays or sequences or accept array or sequence arguments.

We will use the SWIG term *immutable* to indicate that an attribute's value is read-only.

fooObj

A paragraph or two about class fooObj.

fooObj Attributes

attribute [type [access]] Concise description of the attribute.

Attribute name are completely lower case. Multiple words are packed together like *outlinecolor*.

Note that because of the way that mapscript is generated many confusing, meaningless, and even dangerous attributes are creeping into objects. See `outputFormatObj.refcount` for example. Until we get a grip on the structure members we are exposing to SWIG this problem will continue to grow.

fooObj Methods

method(type mandatory_parameter [, type optional_parameter=default]) [type] Description of the method including elaboration on the method arguments, the method's actions, and returned values. Optional parameters and their default values are enclosed in brackets.

Class method names are camel case with a leading lower case character like *getExpressionString*.

Additional Documentation

There's no point in duplicating the MapServer Mapfile Reference, which remains the primary reference for mapscript class attributes.

6.2.2 MapScript Functions

msCleanup() [void] `msCleanup()` attempts to recover all dynamically allocated resources allocated by MapServer code and dependent libraries. It is used primarily for final cleanup in scripts that need to do memory leak testing to get rid of "noise" one-time allocations. It should not normally be used by production code.

msGetVersion() [string] Returns a string containing MapServer version information, and details on what optional components are built in. The same report as produced by "`mapserv -v`".

msGetVersionInt() [int] Returns the MapServer version number (x.y.z) as an integer ($x*10000 + y*100 + z$). (New in v5.0) e.g. V5.4.3 would return 50403.

msResetErrorList() [void] Clears the current error stack.

msIO_installStdoutToBuffer() [void] Installs a mapserver IO handler directing future stdout output to a memory buffer.

msIO_installStdinFromBuffer() [void] Installs a mapserver IO handler directing future stdin reading (ie. post request capture) to come from a buffer.

msIO_resetHandlers() [void] Resets the default stdin and stdout handlers in place of “buffer” based handlers.

msIO_getStdoutBufferString() [string] Fetch the current stdout buffer contents as a string. This method does not clear the buffer.

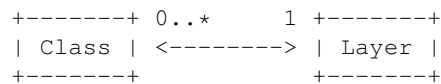
msIO_getStdoutBufferBytes() [binary data] Fetch the current stdout buffer contents as a binary buffer. The exact form of this buffer will vary by mapscript language (eg. string in Python, byte[] array in Java and C#, unhandled in perl)

msIO_stripStdoutBufferContentType() [string] Strip the Content-type header off the stdout buffer if it has one, and if a content type is found it is return (otherwise NULL/None/etc).

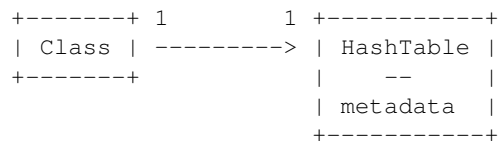
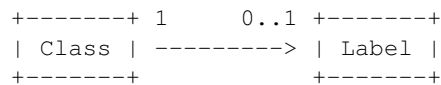
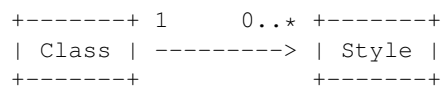
6.2.3 MapScript Classes

classObj

An instance of classObj is associated with with one instance of layerObj.



The other important associations for classObj are with styleObj, labelObj, and hashTableObj.



Multiple class styles are now supported in 4.1. See the styleObj section for details on use of multiple class styles.

classObj Attributes

debug [int] MS_TRUE or MS_FALSE

keyimage [string] **TODO** Not sure what this attribute is for

label [labelObj immutable] Definition of class labeling

layer [layerObj immutable] Reference to the parent layer

maxscale [float] The maximum scale at which class is drawn

metadata [hashTableObj immutable] class metadata hash table.

minscale [float] The minimum scale at which class is drawn

name [string] Unique within a layer

numstyles [int] Number of styles for class. In the future, probably the 4.4 release, this attribute will be made *immutable*.

status [int] MS_ON or MS_OFF. Draw features of this class or do not.

template [string] Template for queries

title [string] Text used for legend labeling

type [int] The layer type of its parent layer

classObj Methods

new classObj([layerObj parent_layer=NULL]) [classObj] Create a new child classObj instance at the tail (highest index) of the class array of the *parent_layer*. A class can be created outside the context of a parent layer by omitting the single constructor argument.

clone() [classObj] Return an independent copy of the class without a parent layer.

createLegendIcon(mapObj map, layerObj layer, int width, int height) [imageObj] Draw and return a new legend icon.

drawLegendIcon(mapObj map, layerObj layer, int width, int height, imageObj image, int dstx, int dsty) [int] Draw the legend icon onto *image* at *dstx*, *dsty*. Returns MS_SUCCESS or MS_FAILURE.

getExpressionString() [string] Return a string representation of the expression enclosed in the quote characters appropriate to the expression type.

getFirstMetaDataKey() [string] Returns the first key in the metadata hash table. With getNextMetaDataKey(), provides an opaque iterator over keys.

getMetaData(string key) [string] Return the value of the classObj metadata at *key*.

getNextMetaDataKey(string lastkey) [string] Returns the next key in the metadata hash table or NULL if *lastkey* is the last valid key. If *lastkey* is NULL, returns the first key of the metadata hash table.

Note: getFirstMetaDataKey(), getMetaData(), and getNextMetaDataKey() are deprecated and will be removed in a future version. Replaced by direct metadata access, see hashTableObj.

getStyle(int index) [styleObj] Return a reference to the styleObj at *index* in the styles array.

See the styleObj section for more details on multiple class styles.

getTextString() [string] Return a string representation of the text enclosed in the quote characters appropriate to the text expression type (logical or simple string).

insertStyle(styleObj style [, int index=-1]) [int] Insert a **copy** of *style* into the styles array at index *index*. Default is -1, or the end of the array. Returns the index at which the style was inserted.

moveStyleDown(int index) [int] Swap the styleObj at *index* with the styleObj *index* + 1.

moveStyleUp(int index) [int] Swap the styleObj at *index* with the styleObj *index* - 1.

removeStyle(int index) [styleObj] Remove the styleObj at *index* from the styles array and return a copy.

setExpression(string expression) [int] Set expression string where *expression* is a MapServer regular, logical or string expression. Returns MS_SUCCESS or MS_FAILURE.

setMetaData(string key, string value) [int] Insert *value* into the classObj metadata at *key*. Returns MS_SUCCESS or MS_FAILURE.

Note: `setMetaData()` is deprecated and will be removed in a future version. Replaced by direct metadata access, see `hashTableObj`.

setText(string text) [int] Set text string where *text* is a MapServer text expression. Returns `MS_SUCCESS` or `MS_FAILURE`.

Note: Older versions of MapScript (pre-4.8) featured the an undocumented `setText()` method that required a `layerObj` be passed as the first argument. That argument was completely bogus and has been removed.

colorObj

Since the 4.0 release, MapServer colors are instances of `colorObj`. A `colorObj` may be a lone object or an attribute of other objects and have no other associations.

colorObj Attributes

blue [int] Blue component of color in range [0-255]

green [int] Green component of color in range [0-255]

red [int] Red component of color in range [0-255]

pen [int] Don't mess with this unless you know what you are doing!

Note: Because of the issue with *pen*, setting colors by individual components is unreliable. Best practice is to use `setRGB()`, `setHex()`, or assign to a new instance of `colorObj`.

colorObj Methods

new colorObj([int red=0, int green=0, int blue=0, int pens=-4]) [colorObj] Create a new instance. The color arguments are optional.

setRGB(int red, int green, int blue) [int] Set all three RGB components. Returns `MS_SUCCESS` or `MS_FAILURE`.

setHex(string hexcolor) [int] Set the color to values specified in case-independent hexadecimal notation. Calling `setHex('#ffffff')` assigns values of 255 to each color component. Returns `MS_SUCCESS` or `MS_FAILURE`.

toHex() [string] Complement to `setHex`, returning a hexadecimal representation of the color components.

errorObj

This class allows inspection of the MapServer error stack. Only needed for the Perl module as the other language modules expose the error stack through exceptions.

errorObj Attributes

code [int] MapServer error code such as `MS_IMGERR` (1).

message [string] Context-dependent error message.

routine [string] MapServer function in which the error was set.

errorObj Methods

next [errorObj] Returns the next error in the stack or NULL if the end has been reached.

fontSetObj

A fontSetObj is always a 'fontset' attribute of a mapObj.

fontSetObj Attributes

filename [string immutable] Path to the fontset file on disk.

fonts [hashTableObj immutable] Mapping of fonts.

numfonts [int immutable] Number of fonts in set.

fontSetObj Methods

None

hashTableObj

A hashTableObj is a very simple mapping of case-insensitive string keys to single string values. Map, Layer, and Class *metadata* have always been hash tables and now these are exposed directly. This is a limited hash that can contain no more than 41 values.

hashTableObj Attributes

numitems [int immutable] Number of hash items.

hashTableObj Methods

clear() [void] Empties the table of all items.

get(string key [, string default=NULL]) [string] Returns the value of the item by its *key*, or *default* if the key does not exist.

nextKey([string key=NULL]) [string] Returns the name of the next key or NULL if there is no valid next key. If the input *key* is NULL, returns the first key.

remove(string key) [int] Removes the hash item by its *key*. Returns MS_SUCCESS or MS_FAILURE.

set(string key, string value) [int] Sets a hash item. Returns MS_SUCCESS or MS_FAILURE.

imageObj

An image object is a wrapper for GD and GDAL images.

imageObj Attributes

format [outputFormatObj immutable] Image format.

height [int immutable] Image height in pixels.

imagepath [string immutable] If image is drawn by mapObj.draw(), this is the mapObj's web.imagepath.

imageurl [string immutable] If image is drawn by mapObj.draw(), this is the mapObj's web.imageurl.

renderer [int] MS_RENDER_WITH_GD, MS_RENDER_WITH_SWF, MS_RENDER_WITH_RAWDATA, MS_RENDER_WITH_PDF, or MS_RENDER_WITH_IMAGEMAP. Don't mess with this!

size [int immutable] To access this attribute use the getSize method.

Note: the getSize method is inefficient as it does a call to getBytes and then computes the size of the byte array. The bytearray is then immediately discarded. In most cases it is more efficient to call getBytes directly.

width [int immutable] Image width in pixels.

imageObj Methods

new imageObj(int width, int height [, outputFormatObj format=NULL [, string filename=NULL]])

[imageObj] Create new instance of imageObj. If *filename* is specified, an imageObj is created from the file and any specified *width*, *height*, and *format* parameters will be overridden by values of the image in *filename*. Otherwise, if *format* is specified an imageObj is created using that format. See the *format* attribute above for details. If *filename* is not specified, then *width* and *height* should be specified.

getBytes() [binary data] Returns the image contents as a binary buffer. The exact form of this buffer will vary by mapscript language (eg. string in Python, byte[] array in Java and C#, unhandled in perl)

getSize() [int] Returns the size of the binary buffer representing the image buffer.

Note: the getSize method is inefficient as it does a call to getBytes and then computes the size of the byte array. The byte array is then immediately discarded. In most cases it is more efficient to call getBytes directly.

save(string filename [, mapObj parent_map=NULL]) [int] Save image to *filename*. The optional *parent_map* parameter must be specified if saving GeoTIFF images.

write([FILE file=NULL]) [int] Write image data to an open file descriptor or, by default, to *stdout*. Returns MS_SUCCESS or MS_FAILURE.

Note: This method is current enabled for Python and C# only. C# supports writing onto a Stream object. User-contributed typemaps are needed for Perl, Ruby, and Java.

Note: The free() method of imageObj has been deprecated. In MapServer revisions 4+ all instances of imageObj will be properly disposed of by the interpreter's garbage collector. If the application can't wait for garbage collection, then the instance can simply be deleted or undef'd.

intarray

An intarray is a utility class generated by SWIG useful for manipulating map layer drawing order. See mapObj::getLayersDrawingOrder for discussion of mapscript use and see http://www.swig.org/Doc1.3/Library.html#Library_nn5 for a complete reference.

intarray Attributes

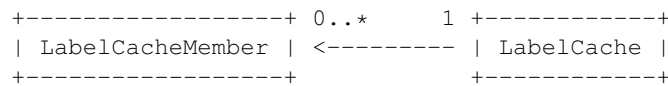
None

intarray Methods

new intarray(int numitems) [intarray] Returns a new instance of the specified length.

labelCacheMemberObj

An individual feature label. The labelCacheMemberObj class is associated with labelCacheObj.



labelCacheMemberObj Attributes

classindex [int immutable] Index of the class of the labeled feature.

featuresize [float immutable] **TODO**

label [labelObj immutable] Copied from the class of the labeled feature.

layerindex [int immutable] The index of the layer of the labeled feature.

numstyles [int immutable] Number of styles as for the class of the labeled feature.

point [pointObj immutable] Label point.

poly [shapeObj immutable] Label bounding box.

shapeindex [int immutable] Index within shapefile of the labeled feature.

status [int immutable] Has the label been drawn or not?

styles [styleObj immutable] **TODO** this should be protected from SWIG.

text [string immutable] Label text.

tileindex [int immutable] Tileindex of the layer of the labeled feature.

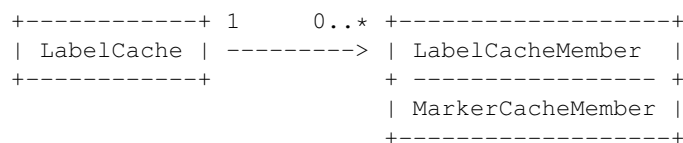
labelCacheMemberObj Methods

None.

Note: No real scripting control over labeling currently, but there may be some interesting new possibilities if users have control over labeling text, position, and status.

labelCacheObj

Set of a map's cached labels. Has no other existence other than as a 'labelcache' attribute of a mapObj. Associated with labelCacheMemberObj and markerCacheMemberObj.



labelCacheObj Attributes

cachesize [int immutable] **TODO**

markercachesize [int immutable] **TODO**

numlabels [int immutable] Number of label members.

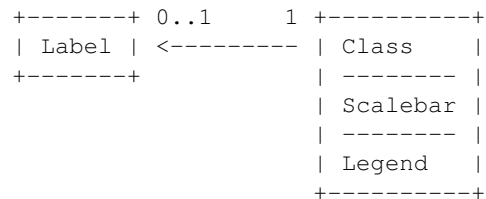
nummarkers [int immutable] Number of marker members.

labelCacheObj Methods

freeCache() [void] Free the labelcache.

labelObj

A labelObj is associated with a classObj, a scalebarObj, or a legendObj.



labelObj Attributes

angle [float] **TODO**

antialias [int] MS_TRUE or MS_FALSE

autoangle [int] MS_TRUE or MS_FALSE

autofollow [int] MS_TRUE or MS_FALSE. Tells mapserver to compute a curved label for appropriate linear features (see *MS RFC 11: Support for Curved Labels* for specifics).

autominfeaturesize: int MS_TRUE or MS_FALSE

backgroundcolor [colorObj] Color of background rectangle or billboard.

backgroundshadowcolor [colorObj] Color of background rectangle or billboard shadow.

backgroundshadowsize [int] Horizontal offset of drop shadow in pixels.

backgroundshadowsizey [int] Vertical offset of drop shadow in pixels.

buffer [int] Maybe this should've been named 'padding' since that's what it is: padding in pixels around a label.

color [colorObj] Foreground color.

encoding [string] Supported encoding format to be used for labels. If the format is not supported, the label will not be drawn. Requires the iconv library (present on most systems). The library is always detected if present on the system, but if not the label will not be drawn. Required for displaying international characters in MapServer. More information can be found at: <http://www.foss4g.org/FOSS4G/MAPSERVER/mpsnf-i18n-en.html>.

font [string] Name of TrueType font.

force [int] MS_TRUE or MS_FALSE.

- maxsize** [int] Maximum height in pixels for scaled labels. See `symbolscale` attribute of `layerObj`.
- mindistance** [int] Minimum distance in pixels between duplicate labels.
- minfeaturesize** [int] Features of this size or greater will be labeled.
- minsize** [int] Minimum height in pixels.
- offsetx** [int] Horizontal offset of label.
- offsety** [int] Vertical offset of label.
- outlinecolor** [colorObj] Color of one point outline.
- partials** [int] `MS_TRUE` (default) or `MS_FALSE`. Whether or not labels can flow past the map edges.
- position** [int] `MS_UL`, `MS_UC`, `MS_UR`, `MS_CL`, `MS_CC`, `MS_CR`, `MS_LL`, `MS_LC`, `MS_LR`, or `MS_AUTO`.
- shadowcolor** [colorObj] Color of drop shadow.
- shadowsize** [int] Horizontal offset of drop shadow in pixels.
- shadowsizey** [int] Vertical offset of drop shadow in pixels.
- size** [int] Annotation height in pixels.
- type** [int] `MS_BITMAP` or `MS_TRUETYPE`.
- wrap** [string] Character on which legend text will be broken to make multi-line legends.

labelObj Methods

None

layerObj

A `layerObj` is associated with `mapObj`. In the most recent revision, an instance of `layerObj` can exist outside of a `mapObj`.

```
+-----+ 0..*  0..1 +-----+
| Layer | <-----> | Map |
+-----+                +-----+
```

The other important association for `layerObj` is with `classObj`

```
+-----+ 1      0..* +-----+
| Layer | <-----> | Class |
+-----+                +-----+
```

and `hashTableObj`

```
+-----+ 1          1 +-----+
| Layer | -----> | HashTable |
+-----+          |   --   |
                  | metadata |
                  +-----+
```

layerObj Attributes

- bandsitem** [string] The attribute from the index file used to select the source raster band(s) to be used. Normally NULL for default bands processing.
- classitem** [string] The attribute used to classify layer data.
- connection** [string] Layer connection or DSN.
- connectiontype** [int] See MS_CONNECTION_TYPE in mapserver.h for possible values. When setting the connection type setConnectionType() should be used in order to initialize the layer vtable properly.
- data** [string] Layer data definition, values depend upon connectiontype.
- debug** [int] Enable debugging of layer. MS_ON or MS_OFF (default).
- dump** [int] Switch to allow mapserver to return data in GML format. MS_TRUE or MS_FALSE. Default is MS_FALSE.
- extent** [rectObj] optional limiting extent for layer features.
- filteritem** [string] Attribute defining filter.
- footer** [string] **TODO**
- group** [string] Name of a group of layers.
- header** [string] **TODO**
- index** [int immutable] Index of layer within parent map's layers array.
- labelangleitem** [string] Attribute defining label angle.
- labelcache** [int] MS_ON or MS_OFF. Default is MS_ON.
- labelitem** [string] Attribute defining feature label text.
- labelmaxscale** [float] Maximum scale at which layer will be labeled.
- labelminscale** [float] Minimum scale at which layer will be labeled.
- labelrequires** [string] Logical expression.
- labelsizeitem** [string] Attribute defining label size.
- map** [mapObj immutable] Reference to parent map.
- maxfeatures** [int] Maximum number of layer features that will be drawn. For shapefile data this means the first N features where N = maxfeatures.
- maxscale** [float] Maximum scale at which layer will be drawn.
- metadata** [hashTableObj immutable] Layer metadata.
- minscale** [float] Minimum scale at which layer will be drawn.
- name** [string] Unique identifier for layer.
- numclasses** [int immutable] Number of layer classes.
- numitems** [int immutable] Number of layer feature attributes (items).
- numjoins** [int immutable] Number of layer joins.
- numprocessing** [int immutable] Number of raster processing directives.
- offsite** [colorObj] transparent pixel value for raster layers.
- postlabelcache** [int] MS_TRUE or MS_FALSE. Default is MS_FALSE.

requires [string] Logical expression.

sizeunits [int] Units of class size values. MS_INCHES, MS_FEET, MS_MILES, MS_METERS, MS_KILOMETERS, MS_DD or MS_PIXELS

status [int] MS_ON, MS_OFF, or MS_DEFAULT.

styleitem [string] Attribute defining styles.

symbolscale [float] Scale at which symbols are default size.

template [string] Template file. Note that for historical reasons, the query attribute must be non-NULL for a layer to be queryable.

tileindex [string] Layer index file for tiling support.

tileitem [string] Attribute defining tile paths.

tolerance [float] Search buffer for point and line queries.

toleranceunits [int] MS_INCHES, MS_FEET, MS_MILES, MS_METERS, MS_KILOMETERS, MS_DD or MS_PIXELS

transform [int] Whether or not layer data is to be transformed to image units. MS_TRUE or MS_FALSE. Default is MS_TRUE. Case of MS_FALSE is for data that are in image coordinates such as annotation points.

transparency [int] Layer opacity percentage in range [0, 100]. The special value of MS_GD_ALPHA (1000) indicates that the alpha transparency of pixmap symbols should be honored, and should be used only for layers that use RGBA pixmap symbols.

type [int] See MS_LAYER_TYPE in mapserver.h.

units [int] Units of the layer. See MS_UNITS in mapserver.h.

layerObj Methods

new layerObj([mapObj parent_map=NULL]) [layerObj] Create a new layerObj in *parent_map*. The layer index of the new layerObj will be equal to the *parent_map* numlayers - 1. The *parent_map* arg is now optional and Layers can exist outside of a Map.

addFeature(shapeObj shape) [int] Add a new inline feature on a layer. Returns -1 on error. **TODO:** Is this similar to inline features in a mapfile? Does it work for any kind of layer or connection type?

addProcessing(string directive) [void] Adds a new processing directive line to a layer, similar to the PROCESSING directive in a map file. Processing directives supported are specific to the layer type and underlying renderer.

applySLD(string sld, string stylelayer) [int] Apply the SLD document to the layer object. The matching between the sld document and the layer will be done using the layer's name. If a namedlayer argument is passed (argument is optional), the NamedLayer in the sld that matches it will be used to style the layer. See SLD HOWTO for more information on the SLD support.

applySLDURL(string sld, string stylelayer) [int] Apply the SLD document pointed by the URL to the layer object. The matching between the sld document and the layer will be done using the layer's name. If a namedlayer argument is passed (argument is optional), the NamedLayer in the sld that matches it will be used to style the layer. See SLD HOWTO for more information on the SLD support.

clearProcessing() [int] Clears the layer's raster processing directives. Returns the subsequent number of directives, which will equal MS_SUCCESS if the directives have been cleared.

clone() [layerObj] Return an independent copy of the layer with no parent map.

close() [void] Close the underlying layer.

Note: `demote()` is removed in MapServer 4.4

draw(mapObj map, imageObj image) [int] Renders this layer into the target image, adding labels to the cache if required. Returns `MS_SUCCESS` or `MS_FAILURE`. **TODO:** Does the map need to be the map on which the layer is defined? I suspect so.

drawQuery(mapObj map, imageObj image) : Draw query map for a single layer into the target image. Returns `MS_SUCCESS` or `MS_FAILURE`.

executeWFSGetFeature(layer) [string] Executes a GetFeature request on a WFS layer and returns the name of the temporary GML file created. Returns an empty string on error.

generateSLD() [void] Returns an SLD XML string based on all the classes found in the layers.

getClass(int i) [classObj] Fetch the requested class object. Returns NULL if the class index is out of the legal range. The `numclasses` field contains the number of classes available, and the first class is index 0.

getExtent() [rectObj] Fetches the extents of the data in the layer. This normally requires a full read pass through the features of the layer and does not work for raster layers.

getFeature(int shapeindex [, int tileindex=-1]) [shapeObj] Return the layer feature at *shapeindex* and *tileindex*.

getFilterString() [string] Returns the current filter expression.

getFirstMetaDataKey() [string] Returns the first key in the metadata hash table. With `getNextMetaDataKey()`, provides an opaque iterator over keys.

getItem(int i) [string] Returns the requested item. Items are attribute fields, and this method returns the item name (field name). The `numitems` field contains the number of items available, and the first item is index zero.

getMetaData(string key) [string] Return the value at *key* from the metadata hash table.

getNextMetaDataKey(string lastkey) [string] Returns the next key in the metadata hash table or NULL if *lastkey* is the last valid key. If *lastkey* is NULL, returns the first key of the metadata hash table.

Note: `getFirstMetaDataKey()`, `getMetaData()`, and `getNextMetaDataKey()` are deprecated and will be removed in a future version. Replaced by direct metadata access, see `hashTableObj`.

getNumFeatures() [int] Returns the number of inline features in a layer. **TODO:** is this really only online features or will it return the number of non-inline features on a regular layer?

getNumResults() [int] Returns the number of entries in the query result cache for this layer.

getProcessing(int index) [string] Return the raster processing directive at *index*.

getProjection() [string] Returns the PROJ.4 definition of the layer's projection.

getResult(int i) [resultCacheMemberObj] Fetches the requested query result cache entry, or NULL if the index is outside the range of available results. This method would normally only be used after issuing a query operation.

Note: `getNumResults()` and `getResult()` are deprecated in MapServer 4.4. Users should instead use the new querying API described in `querying-HOWTO.txt`. `layerObj::getResults()` is the entry point for the new API.

getResults() [resultCacheObj] Returns a reference to layer's result cache. Should be NULL prior to any query, or after a failed query or query with no results.

getShape(shapeObj shape, int tileindex, int shapeindex) [int] Get a shape from layer data.

Note: `getShape()` is deprecated. Users should adopt `getFeature()` for new applications.

getWMSFeatureInfoURL(mapObj map, int click_x, int click_y, int feature_count, string info_format)

[string] Return a WMS GetFeatureInfo URL (works only for WMS layers) `clickX`, `clickY` is the location of to query in pixel coordinates with (0,0) at the top left of the image. `featureCount` is the number of results to return. `infoFormat` is the format the format in which the result should be requested. Depends on remote server's

capabilities. MapServer WMS servers support only “MIME” (and should support “GML.1” soon). Returns “” and outputs a warning if layer is not a WMS layer or if it is not queriable.

insertClass(classObj class [, int index=-1]) [int] Insert a *copy* of the class into the layer at the requested *index*. Default index of -1 means insertion at the end of the array of classes. Returns the index at which the class was inserted.

isVisible() [int] Returns MS_TRUE or MS_FALSE after considering the layer status, minscale, and maxscale within the context of the parent map.

moveClassDown(int class) [int] The class specified by the class index will be moved up into the array of layers. Returns MS_SUCCESS or MS_FAILURE. ex. moveClassDown(1) will have the effect of moving class 1 down to position 2, and the class at position 2 will be moved to position 1.

moveClassUp(int class) [int] The class specified by the class index will be moved up into the array of layers. Returns MS_SUCCESS or MS_FAILURE. ex. moveClassUp(1) will have the effect of moving class 1 up to position 0, and the class at position 0 will be moved to position 1.

nextShape() [shapeObj] Called after msWhichShapes has been called to actually retrieve shapes within a given area returns a shape object or MS_FALSE

example of usage :

```
mapObj map = new mapObj("d:/msapps/gmap-ms40/htdocs/gmap75.map");
layerObj layer = map.getLayerByName('road');
int status = layer.open();
status = layer.whichShapes(map.extent);
shapeObj shape;
while ((shape = layer.nextShape()) != null)
{
    ...
}
layer.close();
```

open() [void] Opens the underlying layer. This is required before operations like getFeature() will work, but is not required before a draw or query call.

Note: promote() is eliminated in MapServer 4.4.

queryByAttributes(mapObj map, string qitem, string qstring, int mode) [int] Query layer for shapes that intersect current map extents. qitem is the item (attribute) on which the query is performed, and qstring is the expression to match. The query is performed on all the shapes that are part of a CLASS that contains a TEMPLATE value or that match any class in a layer that contains a LAYER TEMPLATE value.

Note that the layer’s FILTER/FILTERITEM are ignored by this function. Mode is MS_SINGLE or MS_MULTIPLE depending on number of results you want. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened.

queryByFeatures(mapObj map, int slayer) [int] Perform a query set based on a previous set of results from another layer. At present the results MUST be based on a polygon layer. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened

queryByIndex(mapObj map, int shapeindex, int tileindex [, int bAddToQuery=MS_FALSE]) [int] Pop a query result member into the layer’s result cache. By default clobbers existing cache. Returns MS_SUCCESS or MS_FAILURE.

queryByPoint(mapObj map, pointObj point, int mode, float buffer) [int] Query layer at point location specified in georeferenced map coordinates (i.e. not pixels). The query is performed on all the shapes that are part of a CLASS that contains a TEMPLATE value or that match any class in a layer that contains a LAYER TEMPLATE value. Mode is MS_SINGLE or MS_MULTIPLE depending on number of results you want. Passing buffer <=0 defaults to tolerances set in the map file (in pixels) but you can use a constant buffer (specified in ground units)

instead. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened.

queryByRect(mapObj map, rectObj rect) [int] Query layer using a rectangle specified in georeferenced map coordinates (i.e. not pixels). The query is performed on all the shapes that are part of a CLASS that contains a TEMPLATE value or that match any class in a layer that contains a LAYER TEMPLATE value. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened.

queryByShape(mapObj map, shapeObj shape) [int] Query layer based on a single shape, the shape has to be a polygon at this point. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened

removeClass(int index) [classObj] Removes the class indicated and returns a copy, or NULL in the case of a failure. Note that subsequent classes will be renumbered by this operation. The numclasses field contains the number of classes available.

removeMetaData(string key) [int] Delete the metadata hash at *key*. Returns MS_SUCCESS or MS_FAILURE.

Note: removeMetaData() is deprecated and will be removed in a future version. Replaced by direct metadata access, see hashTableObj.

setConnectionType(int connectiontype, string library_str) [int] Changes the connectiontype of the layer and recreates the vtable according to the new connection type. This method should be used instead of setting the connectiontype parameter directly. In case when the layer.connectiontype = MS_PLUGIN the library_str parameter should also be specified so as to select the library to load by mapserver. For the other connection types this parameter is not used.

setExtent(float minx, float miny, float maxx, float maxy) [int] Sets the extent of a layer. Returns MS_SUCCESS or MS_FAILURE.

setFilter(string filter) [int] Sets a filter expression similarly to the FILTER expression in a map file. Returns MS_SUCCESS on success or MS_FAILURE if the expression fails to parse.

setMetaData(string key, string value) [int] Assign *value* to the metadata hash at *key*. Return MS_SUCCESS or MS_FAILURE.

Note: setMetaData() is deprecated and will be removed in a future version. Replaced by direct metadata access, see hashTableObj.

setProcessingKey(string key, string value) [void] Adds or replaces a processing directive of the form “key=value”. Unlike the addProcessing() call, this will replace an existing processing directive for the given key value. Processing directives supported are specific to the layer type and underlying renderer.

setProjection(string proj4) [int] Set the layer projection using a PROJ.4 format projection definition (ie. “+proj=utm +zone=11 +datum=WGS84” or “init=EPSG:26911”). Returns MS_SUCCESS or MS_FAILURE.

setWKTProjection(string wkt) [int] Set the layer projection using OpenGIS Well Known Text format. Returns MS_SUCCESS or MS_FAILURE.

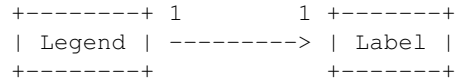
int whichShapes(rectObj rect) [int] Performs a spatial, and optionally an attribute based feature search. The function basically prepares things so that candidate features can be accessed by query or drawing functions (eg using nextShape function). Returns MS_SUCCESS or MS_FAILURE.

legendObj

legendObj is associated with mapObj

```
+-----+ 0..1      1 +-----+
| Legend | <-----> | Map |
+-----+             +-----+
```

and with labelObj.



legendObj Attributes

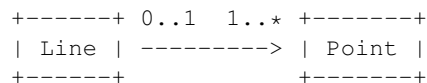
- height** [int] Legend height.
- imagecolor** [colorObj] Legend background color.
- keysize** [int] Width in pixels of legend keys.
- keysizey** [int] Pixels.
- keyspacingx** [int] Horizontal padding around keys in pixels.
- keyspacingy** [int] Vertical padding.
- label** [labelObj immutable] legend label.
- map** [mapObj immutable] Reference to parent mapObj.
- outlinecolor** [colorObj] key outline color.
- position** [int] MS_UL, MS_UC, MS_UR, MS_LL, MS_LC, or MS_LR.
- postlabelcache** [int] MS_TRUE or MS_FALSE.
- status** [int] MS_ON, MS_OFF, or MS_EMBED.
- template** [string] Path to template file.
- width** [int] Label width.

legendObj Methods

None

lineObj

A lineObj is composed of one or more pointObj instances.



lineObj Attributes

- numpoints** [int immutable] Number of points in the line.

lineObj Methods

add(pointObj point) [int] Add *point* to the line. Returns MS_SUCCESS or MS_FAILURE.

get(int index) [pointObj] Return reference to point at *index*.

project(projectionObj proj_in, projectionObj proj_out) [int] Transform line in place from *proj_in* to *proj_out*. Returns MS_SUCCESS or MS_FAILURE.

set(int index, pointObj point) [int] Set the point at *index* to *point*. Returns MS_SUCCESS or MS_FAILURE.

mapObj

A mapObj is primarily associated with instances of layerObj.

```
+-----+ 0..1  0..* +-----+
| Map | <-----> | Layer |
+-----+          +-----+
```

Secondary associations are with legendObj, scalebarObj, referenceMapObj,

```
+-----+ 1      0..1 +-----+
| Map | -----> | Legend   |
+-----+          | -----+
                   | Scalebar |
                   | -----+
                   | ReferenceMap |
                   +-----+
```

outputFormatObj.

```
+-----+ 1      1..* +-----+
| Map | -----> | OutputFormat |
+-----+          +-----+
```

mapObj Attributes

cellsize [float] Pixel size in map units.

configoptions [hashObj immutable] A hash table of configuration options from CONFIG keywords in the .map. Direct access to config options is discouraged. Use the setConfigOption() and getConfigOption() methods instead.

datapattern [string] **TODO** not sure this is meaningful for mapscript.

debug [int] MS_TRUE or MS_FALSE.

extent [rectObj] Map's spatial extent.

fontset [fontSetObj immutable] The map's defined fonts.

height [int] Map's output image height in pixels.

Note: direct setting of *height* is deprecated in MapServer version 4.4. Users should set width and height simultaneously using setSize().

imagecolor [colorObj] Initial map background color.

imagequality [int] JPEG image quality.

Note: map imagequality is deprecated in MapServer 4.4 and should instead be managed through map outputformats.

imagetype [string immutable] Name of the current output format.

interlace [int] Output image interlacing.

Note: map interlace is deprecated in MapServer 4.4 and should instead be managed through map outputformats.

labelcache [labelCacheObj immutable] Map's labelcache.

legend [legendObj immutable] Reference to map's legend.

mappath [string] Filesystem path of the map's mapfile.

maxsize [int] **TODO** ?

name [string] Unique identifier.

numlayers [int immutable] Number of map layers.

numoutputformats [int] Number of output formats.

outputformat [outputFormatObj] The currently selected output format.

Note: Map outputformat should not be modified directly. Use the selectOutputFormat() method to select named formats.

outputformatlist [outputFormatObj[]] Array of the available output formats.

Note: Currently only available for C#. A proper typemaps should be implemented for the other languages.

querymap [queryMapObj immutable] **TODO** should this be exposed to mapsript?

reference [referenceMapObj immutable] Reference to reference map.

resolution [float] Nominal DPI resolution. Default is 72.

scale [float] The nominal map scale. A value of 25000 means 1:25000 scale.

scalebar [scalebarObj immutable] Reference to the scale bar.

shapepath [string] Base filesystem path to layer data.

status [int] MS_OFF, MS_ON, or MS_DEFAULT.

symbolset [symbolSetObj immutable] The map's set of symbols.

templatepattern [string] **TODO** not sure this is meaningful for mapsript.

transparent [int] MS_TRUE or MS_FALSE.

Note: map transparent is deprecated in MapServer 4.4 and should instead be managed through map outputformats.

units [int] MS_DD, MS_METERS, etc.

web [webObj immutable] Reference to map's web definitions.

width [int] Map's output image width in pixels.

Note: direct setting of *width* is deprecated in MapServer version 4.4. Users should set width and height simultaneously using setSize().

mapObj Methods

new mapObj([string filename=""]) [mapObj] Create a new instance of mapObj. Note that the filename is now optional.

appendOutputFormat(outputFormatObj format) [int] Attach *format* to the map's output format list. Returns the updated number of output formats.

applyConfigOptions() [void] Apply the defined configuration options set by setConfigOption().

applySLD(string sldxml) [int] Parse the SLD XML string *sldxml* and apply to map layers. Returns MS_SUCCESS or MS_FAILURE.

applySLDURL(string sldurl) [int] Fetch SLD XML from the URL *sldurl* and apply to map layers. Returns MS_SUCCESS or MS_FAILURE.

clone() [mapObj] Returns a independent copy of the map, less any caches.

Note: In the Java module this method is named 'cloneMap'.

draw() [imageObj] Draw the map, processing layers according to their defined order and status. Return an imageObj.

drawLabelCache(imageObj image) [int] Draw map's label cache on *image*. Returns MS_SUCCESS or MS_FAILURE.

drawLegend() [imageObj] Draw map legend, returning an imageObj.

drawQuery() [imageObj] Draw query map, returning an imageObj.

drawReferenceMap() [imageObj] Draw reference map, returning an imageObj.

drawScalebar() [imageObj] Draw scale bar, returning an imageObj.

embedLegend(imageObj image) [int] Embed map's legend in *image*. Returns MS_SUCCESS or MS_FAILURE.

embedScalebar(imageObj image) [int] Embed map's scalebar in *image*. Returns MS_SUCCESS or MS_FAILURE.

freeQuery([int qlayer=-1]) [void] Clear layer query result caches. Default is -1, or *all* layers.

generateSLD() [string] Return SLD XML as a string for map layers.

getConfigOption(string key) [string] Fetches the value of the requested configuration key if set. Returns NULL if the key is not set.

getFirstMetaDataKey() [string] Returns the first key in the web.metadata hash table. With getNextMetaDataKey(), provides an opaque iterator over keys.

getLayer(int index) [layerObj] Returns a reference to the layer at *index*.

getLayerByName(string name) [layerObj] Returns a reference to the named layer.

getLayersDrawingOrder() [int*] Returns an array of layer indexes in drawing order.

Note: Unless the proper typemap is implemented for the module's language a user is more likely to get back an unuseable SWIG pointer to the integer array.

getMetaData(string key) [string] Return the value at *key* from the web.metadata hash table.

getNextMetaDataKey(string lastkey) [string] Returns the next key in the web.metadata hash table or NULL if *lastkey* is the last valid key. If *lastkey* is NULL, returns the first key of the metadata hash table.

getNumSymbols() [int] Return the number of symbols in map.

getOutputFormatByName(string imagetype) [outputFormatObj] Return the output format corresponding to driver name *imagetype* or to format name *imagetype*. This works exactly the same as the IMAGETYPE directive in a mapfile, is case insensitive and allows an output format to be found either by driver (like 'GD/PNG') or name (like 'PNG24').

getProjection() [string] Returns the PROJ.4 definition of the map's projection.

getSymbolByName(string name) [int] Return the index of the named symbol in the map's symbolset.

Note: This method is poorly named and too indirect. It is preferable to use the `getSymbolByName` method of `symbolSetObj`, which really does return a `symbolObj` reference, or use the `index` method of `symbolSetObj` to get a symbol's index number.

insertLayer(layerObj layer [, int nIndex=-1]) [int] Insert a copy of *layer* into the Map at index *nIndex*. The default value of *nIndex* is -1, which means the last possible index. Returns the index of the new Layer, or -1 in the case of a failure.

loadMapContext(string filename [, int useUniqueNames=MS_FALSE]) [int] Load an OGC map context file to define extents and layers of a map.

loadOWSParameters(OWSRequest request [, string version='1.1.1']) [int] Load OWS request parameters (BBOX, LAYERS, &c.) into map. Returns MS_SUCCESS or MS_FAILURE.

loadQuery(string filename) [int] Load a saved query. Returns MS_SUCCESS or MS_FAILURE.

moveLayerDown(int layerindex) [int] Move the layer at *layerindex* down in the drawing order array, meaning that it is drawn later. Returns MS_SUCCESS or MS_FAILURE.

moveLayerUp(int layerindex) [int] Move the layer at *layerindex* up in the drawing order array, meaning that it is drawn earlier. Returns MS_SUCCESS or MS_FAILURE.

nextLabel() [labelCacheMemberObj] Return the next label from the map's labelcache, allowing iteration over labels.

Note: `nextLabel()` is deprecated and will be removed in a future version. Replaced by `getLabel()`.

getLabel(int labelindex) [labelCacheMemberObj] Return label at specified index from the map's labelcache.

OWSDispatch(OWSRequest req) [int] Processes and executes the passed OpenGIS Web Services request on the map. Returns MS_DONE (2) if there is no valid OWS request in the req object, MS_SUCCESS (0) if an OWS request was successfully processed and MS_FAILURE (1) if an OWS request was not successfully processed. OWS requests include WMS, WFS, WCS and SOS requests supported by MapServer. Results of a dispatched request are written to stdout and can be captured using the msIO services (ie. `msIO_installStdoutToBuffer()` and `msIO_getStdoutBufferString()`)

prepareImage() [imageObj] Returns an imageObj initialized to map extents and outputformat.

prepareQuery() [void] **TODO** this function only calculates the scale or am I missing something?

processLegendTemplate(string names[], string values[], int numitems) [string] Process MapServer legend template and return HTML.

processQueryTemplate(string names[], string values[], int numitems) [string] Process MapServer query template and return HTML.

processTemplate(int generateimages, string names[], string values[], int numitems) [string] Process MapServer template and return HTML.

Note: None of the three template processing methods will be useable unless the proper typemaps are implemented in the module for the target language. Currently the typemaps are not implemented.

queryByFeatures(int layerindex) [int] Query map layers, result sets contain features that intersect or are contained within the features in the result set of the MS_LAYER_POLYGON type layer at *layerindex*. Returns MS_SUCCESS or MS_FAILURE.

queryByPoint(pointObj point, int mode, float buffer) [int] Query map layers, result sets contain one or more features, depending on *mode*, that intersect *point* within a tolerance *buffer*. Returns MS_SUCCESS or MS_FAILURE.

queryByRect(rectObj rect) [int] Query map layers, result sets contain features that intersect or are contained within *rect*. Returns MS_SUCCESS or MS_FAILURE.

queryByShape(shapeObj shape) [int] Query map layers, result sets contain features that intersect or are contained within *shape*. Returns MS_SUCCESS or MS_FAILURE.

- removeLayer(int index)** [int] Remove the layer at *index*.
- removeMetaData(string key)** [int] Delete the web.metadata hash at *key*. Returns MS_SUCCESS or MS_FAILURE.
- removeOutputFormat(string name)** [int] Removes the format named *name* from the map's output format list. Returns MS_SUCCESS or MS_FAILURE.
- save(string filename)** [int] Save map to disk as a new map file. Returns MS_SUCCESS or MS_FAILURE.
- saveMapContext(string filename)** [int] Save map definition to disk as OGC-compliant XML. Returns MS_SUCCESS or MS_FAILURE.
- saveQuery(string filename)** [int] Save query to disk. Returns MS_SUCCESS or MS_FAILURE.
- saveQueryAsGML(string filename)** [int] Save query to disk. Returns MS_SUCCESS or MS_FAILURE.
- selectOutputFormat(string imagetype)** [void] Set the map's active output format to the internal format named *imagetype*. Built-in formats are "PNG", "PNG24", "JPEG", "GIF", "GTIFF".
- setConfigOption(string key, string value)** [void] Set the indicated key configuration option to the indicated value. Equivalent to including a CONFIG keyword in a map file.
- setExtent(float minx, float miny, float maxx, float maxy)** [int] Set the map extent, returns MS_SUCCESS or MS_FAILURE.
- offsetExtent(float x, float y)** [int] Offset the map extent based on the given distances in map coordinates, returns MS_SUCCESS or MS_FAILURE.
- scaleExtent(float zoomfactor, float minscaledenom, float maxscaledenom)** [int] Scale the map extent using the zoomfactor and ensure the extent within the minscaledenom and maxscaledenom domain. If minscaledenom and/or maxscaledenom is 0 then the parameter is not taken into account. returns MS_SUCCESS or MS_FAILURE.
- setCenter(pointObj center)** [int] Set the map center to the given map point, returns MS_SUCCESS or MS_FAILURE.
- setFontSet(string filename)** [int] Load fonts defined in *filename* into map fontset. The existing fontset is cleared. Returns MS_SUCCESS or MS_FAILURE.
- setImageType(string name)** [void] Sets map outputformat to the named format.
- Note:** setImageType() remains in the module but it's use is deprecated in favor of selectOutputFormat().
- setLayersDrawingOrder(int layerindexes[])** [int] Set map layer drawing order.
- Note:** Unless the proper typemap is implemented for the module's language users will not be able to pass arrays or lists to this method and it will be unusable.
- setMetaData(string key, string value)** [int] Assign *value* to the web.metadata hash at *key*. Return MS_SUCCESS or MS_FAILURE.
- setOutputFormat(outputFormatObj format)** [void] Sets map outputformat.
- setProjection(string proj4)** [int] Set map projection from PROJ.4 definition string *proj4*.
- setRotation(float rotation_angle)** [int] Set map rotation angle. The map view rectangle (specified in EXTENTS) will be rotated by the indicated angle in the counter- clockwise direction. Note that this implies the rendered map will be rotated by the angle in the clockwise direction. Returns MS_SUCCESS or MS_FAILURE.
- setSize(int width, int height)** [int] Set map's image width and height together and carry out the necessary subsequent geotransform computation. Returns MS_SUCCESS or MS_FAILURE.
- setSymbolSet(string filename)** [int] Load symbols defined in *filename* into map symbolset. The existing symbolset is cleared. Returns MS_SUCCESS or MS_FAILURE.

setWKTProjection(string wkt) [int] Sets map projection from OGC definition *wkt*.

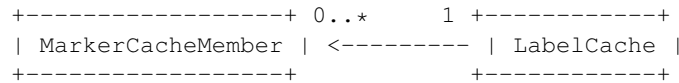
zoomPoint(int zoomfactor, pointObj imgpoint, int width, int height, rectObj extent, rectObj maxextent)
 [int] Zoom by *zoomfactor* to *imgpoint* in pixel units within the image of *height* and *width* dimensions and georeferenced *extent*. Zooming can be constrained to a maximum *maxextent*. Returns MS_SUCCESS or MS_FAILURE.

zoomRectangle(rectObj imgrect, int width, int height, rectObj extent, rectObj maxextent) [int] Zoom to a pixel coordinate rectangle in the image of *width* and *height* dimensions and georeferencing *extent*. Zooming can be constrained to a maximum *maxextent*. Returns MS_SUCCESS or MS_FAILURE.

zoomScale(float scale, pointObj imgpoint, int width, int height, rectObj extent, rectObj maxextent) [int] Like the previous methods, but zooms to the point at a specified scale.

markerCacheMemberObj

An individual marker. The markerCacheMemberObj class is associated with labelCacheObj.



markerCacheMemberObj Attributes

id [int immutable] Id of the marker.

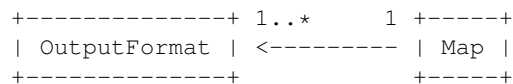
poly [shapeObj immutable] Marker bounding box.

markerCacheMemberObj Methods

None.

outputFormatObj

An outputFormatObj is associated with a mapObj



and can also be an attribute of an imageObj.

outputFormatObj Attributes

bands [int] The number of bands in the raster. Only used for the “raw” modes, MS_IMAGEMODE_BYTE, MS_IMAGEMODE_INT16, and MS_IMAGEMODE_FLOAT32. Normally set via the BAND_COUNT format option ... this field should be considered read-only.

driver [string] A string such as ‘GD/PNG’ or ‘GDAL/GTiff’.

extension [string] Format file extension such as ‘png’.

imagemode [int] MS_IMAGEMODE_PC256, MS_IMAGEMODE_RGB, MS_IMAGEMODE_RGBA, MS_IMAGEMODE_INT16, MS_IMAGEMODE_FLOAT32, MS_IMAGEMODE_BYTE, or MS_IMAGEMODE_NULL.

mimetype [string] Format mimetype such as 'image/png'.

name [string] A unique identifier.

renderer [int] MS_RENDER_WITH_GD, MS_RENDER_WITH_SWF, MS_RENDER_WITH_RAWDATA, MS_RENDER_WITH_PDF, or MS_RENDER_WITH_IMAGEMAP. Normally set internally based on the driver and some other setting in the constructor.

transparent [int] MS_ON or MS_OFF.

outputFormatObj Methods

new outputFormatObj(string driver [, string name=driver]) [outputFormatObj] Create new instance. If *name* is not provided, the value of *driver* is used as a name.

getOption(string key [, string value=""]) [string] Return the format option at *key* or *value* if *key* is not a valid hash index.

setExtension(string extension) [void] Set file extension for output format such as 'png' or 'jpg'. Method could probably be deprecated since the extension attribute is mutable.

setMimetype(string mimetype) [void] Set mimetype for output format such as 'image/png' or 'image/jpeg'. Method could probably be deprecated since the mimetype attribute is mutable.

setOption(string key, string value) [void] Set the format option at *key* to *value*. Format options are mostly driver specific.

validate() [int] Checks some internal consistency issues, and returns MS_TRUE if things are OK and MS_FALSE if there are problems. Some problems are fixed up internally. May produce debug output if issues encountered.

OWSRequest

Not associated with other mapscript classes. Serves as a message intermediary between an application and MapServer's OWS capabilities. Using it permits creation of lightweight WMS services:

```
wms_map = mapscript.mapObj('wms.map')
wms_request = mapscript.OWSRequest()

# Convert application request parameters (req.args)
for param, value in req.args.items():
    wms_request.setParam(param, value)

# Map loads parameters from OWSRequest, adjusting its SRS, extents,
# active layers accordingly
wms_map.loadWMSRequest('1.1.0', wms_request)

# Render the Map
img = wms_map.draw()
```

OWSRequest Attributes

NumParams [int immutable] Number of request parameters. Eventually should be changed to numparams lowercase like other attributes.

postrequest [string] **TODO**

type [int] MS_GET_REQUEST or MS_POST_REQUEST.

OWSRequest Methods

new OWSRequest() [OWSRequest] Create a new instance.

setParameter(string name, string value) [void] Set a request parameter. For example

```
request.setParameter('REQUEST', 'GetMap')
request.setParameter('BBOX', '-107.0,40.0,-106.0,41.0')
```

Note: MapServer's OWSRequest supports only single valued parameters.

getName(int index) [string] Return the name of the parameter at *index* in the request's array of parameter names.

getValue(int index) [string] Return the value of the parameter at *index* in the request's array of parameter values.

getValueByName(string name) [string] Return the value associated with the parameter *name*.

loadParams() [int] Initializes the OWSRequest object from the cgi environment variables REQUEST_METHOD, QUERY_STRING and HTTP_COOKIE. Returns the number of name/value pairs collected. Warning: most errors will result in a process exit!

pointObj

A pointObj instance may be associated with a lineObj.

```
+-----+ 1..*  0..1 +-----+
| Point | <-----| Line |
+-----+          +-----+
```

pointObj Attributes

m [float] Measure. Meaningful only for measured shapefiles. Given value -2e38 if not otherwise assigned to indicate "nodata".

x [float] Easting

y [float] Northing

z [float] Elevation

pointObj Methods

new pointObj([float x=0.0, float y=0.0, float z=0.0, float m=-2e38]) [pointObj] Create new instance. Easting, northing, and measure arguments are optional.

distanceToPoint(pointObj point) [float] Returns the distance to *point*.

distanceToSegment(pointObj point1, pointObj point2) [float] Returns the minimum distance to a hypothetical line segment connecting *point1* and *point2*.

distanceToShape(shapeObj shape) [float] Returns the minimum distance to *shape*.

draw(mapObj map, layerObj layer, imageObj image, int classindex, string text) [int] Draw the point using the styles defined by the *classindex* class of *layer* and labeled with string *text*. Returns MS_SUCCESS or MS_FAILURE.

project(projectionObj proj_in, projectionObj proj_out) [int] Reproject point from *proj_in* to *proj_out*. Transformation is done in place. Returns MS_SUCCESS or MS_FAILURE.

setXY(float x, float y [, float m=2e-38]) [int] Set spatial coordinate and, optionally, measure values simultaneously. The measure will be set only if the value of *m* is greater than the ESRI measure no-data value of 1e-38. Returns MS_SUCCESS or MS_FAILURE.

setXYZ(float x, float y, float z [, float m=-2e38]) [int] Set spatial coordinate and, optionally, measure values simultaneously. The measure will be set only if the value of *m* is greater than the ESRI measure no-data value of -1e38. Returns MS_SUCCESS or MS_FAILURE.

setXYZM(float x, float y, float z, float m) [int] Set spatial coordinate and, optionally, measure values simultaneously. The measure will be set only if the value of *m* is greater than the ESRI measure no-data value of -1e38. Returns MS_SUCCESS or MS_FAILURE.

toString() [string] Return a string formatted like

```
{ 'x': %f , 'y': %f, 'z': %f }
```

with the coordinate values substituted appropriately. Python users can get the same effect via the pointObj `__str__` method

```
>>> p = mapscript.pointObj(1, 1)
>>> str(p)
{ 'x': 1.000000 , 'y': 1.000000, 'z': 1.000000 }
```

toShape() [shapeObj] Convenience method to quickly turn a point into a shapeObj.

projectionObj

This class is not really fully implemented yet. MapServer's Maps and Layers have Projection attributes, and these are C projectionObj structures, but are not directly exposed by the mapscript module. Currently we have to do some round-a-bout logic like this

```
point.project( projectionObj( mapobj.getProjection() ),
              projectionObj( layer.getProjection() ) )
```

to project a point from map to layer reference system.

projectionObj Attributes

numargs [int immutable] Number of PROJ.4 arguments.

projectionObj Methods

new projectionObj(string proj4) [projectionObj] Create new instance of projectionObj. Input parameter *proj4* is a PROJ.4 definition string such as “init=EPSG:4269”.

getUnits() [int] Returns the units of a projection object. Returns -1 on error.

rectObj

A rectObj may be a lone object or an attribute of another object and has no other associations.

rectObj Attributes

maxx [float] Maximum easting

maxy [float] Maximum northing

minx [float] Minimum easting

miny [float] Minimum northing

rectObj Methods

new rectObj([float minx=-1.0, float miny=-1.0, float maxx=-1.0, float maxy=-1.0, int imageunits=MS_FALSE]) [rectObj] Create new instance. The four easting and northing arguments are optional and default to -1.0. Note the new optional fifth argument which allows creation of rectangles in image (pixel/line) units which are also tested for validity.

draw(mapObj map, layerObj layer, imageObj img, int classindex, string text) [int] Draw rectangle into *img* using style defined by the *classindex* class of *layer*. The rectangle is labeled with the string *text*. Returns MS_SUCCESS or MS_FAILURE.

getCenter() [pointObj] Return the center point of the rectagle.

project(projectionObj proj_in, projectionObj proj_out) [int] Reproject rectangle from *proj_in* to *proj_out*. Transformation is done in place. Returns MS_SUCCESS or MS_FAILURE.

toPolygon() [shapeObj] Convert to a polygon of five vertices.

toString() [string] Return a string formatted like

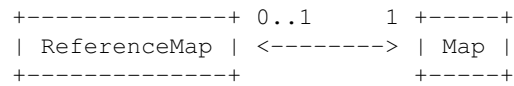
```
{ 'minx': %f , 'miny': %f , 'maxx': %f , 'maxy': %f }
```

with the bounding values substituted appropriately. Python users can get the same effect via the rectObj `__str__` method

```
>>> r = mapscript.rectObj(0, 0, 1, 1)
>>> str(r)
{ 'minx': 0 , 'miny': 0 , 'maxx': 1 , 'maxy': 1 }
```

referenceMapObj

A referenceMapObj is associated with mapObj.



referenceMapObj Attributes

color [colorObj] Color of reference box.

extent [rectObj] Spatial extent of reference in units of parent map.

height [int] Height of reference map in pixels.

image [string] Filename of reference map image.

map [mapObj immutable] Reference to parent mapObj.

marker [int] Index of a symbol in the map symbol set to use for marker.

markername [string] Name of a symbol.

markersize [int] Size of marker.

maxboxsize [int] Pixels.

minboxsize [int] Pixels.

outlinecolor [colorObj] Outline color of reference box.

status [int] MS_ON or MS_OFF.

width [int] In pixels.

referenceMapObj Methods

None

resultCacheMemberObj

Has no associations with other MapScript classes and has no methods. By using several indexes, a resultCacheMemberObj refers to a single layer feature.

resultCacheMemberObj Attributes

classindex [int immutable] The index of the layer class into which the feature has been classified.

shapeindex [int immutable] Index of the feature within the layer.

tileindex [int immutable] Meaningful for tiled layers only, index of the shapefile data tile.

resultCacheObj

See querying-HOWTO.txt for extra guidance in using the new 4.4 query API.

resultCacheObj Attributes

bounds [rectObj immutable] Bounding box of query results.

numresults [int immutable] Length of result set.

resultCacheObj Methods

getResult(int i) [resultCacheObj] Returns the result at index *i*, like layerObj::getResult, or NULL if index is outside the range of results.

scalebarObj

A scalebarObj is associated with mapObj.

```
+-----+ 0..1      1 +-----+
| Scalebar | <----- | Map |
+-----+          +-----+
```

and also with labelObj

```
+-----+ 1          1 +-----+
| Scalebar | -----> | Label |
+-----+          +-----+
```

scalebarObj Attributes

backgroundcolor [colorObj] Scalebar background color.

color [colorObj] Scalebar foreground color.

imagecolor [colorObj] Background color of scalebar.

height [int] Pixels.

intervals [int] Number of intervals.

label [labelObj] Scalebar label.

outlinecolor [colorObj] Foreground outline color.

position [int] MS_UL, MS_UC, MS_UR, MS_LL, MS_LC, or MS_LR.

postlabelcache [int] MS_TRUE or MS_FALSE.

status [int] MS_ON, MS_OFF, or MS_EMBED.

style [int] 0 or 1.

units [int] See MS_UNITS in mapserver.h.

width [int] Pixels.

scalebarObj Methods

None

shapefileObj

shapefileObj Attributes

bounds [rectObj] Extent of shapes

numshapes [int] Number of shapes

type [int] See mapshape.h for values of type.

shapefileObj Methods

new shapefileObj(string filename [, int type=-1]) [shapefileObj] Create a new instance. Omit the *type* argument or use a value of -1 to open an existing shapefile.

add(shapeObj shape) [int] Add shape to the shapefile. Returns MS_SUCCESS or MS_FAILURE.

get(int i, shapeObj shape) [int] Get the shapefile feature from index *i* and store it in *shape*. Returns MS_SUCCESS or MS_FAILURE.

getShape(int i) [shapeObj] Returns the shapefile feature at index *i*. More efficient than *get*.

TODO

shapeObj

Each feature of a layer's data is a shapeObj. Each part of the shape is a closed lineObj.

```

+-----+ 1      1..* +-----+
| Shape | -----> | Line |
+-----+          +-----+

```

shapeObj Attributes

bounds [rectObj] Bounding box of shape.

classindex [int] The class index for features of a classified layer.

index [int] Feature index within the layer.

numlines [int immutable] Number of parts.

numvalues [int immutable] Number of shape attributes.

text [string] Shape annotation.

tileindex [int] Index of tiled file for tileindexed layers.

type [int] MS_SHAPE_POINT, MS_SHAPE_LINE, MS_SHAPE_POLYGON, or MS_SHAPE_NULL.

shapeObj Methods

new shapeObj(int type) [shapeObj] Return a new shapeObj of the specified *type*. See the type attribute above. No attribute values created by default. *initValues* should be explicitly called to create the required number of values.

add(lineObj line) [int] Add *line* (i.e. a part) to the shape. Returns MS_SUCCESS or MS_FAILURE.

boundary() [shapeObj] Returns the boundary of the existing shape. Requires GEOS support. Returns NULL/undef on failure.

buffer(int distance) [shapeObj] Returns a new buffered shapeObj based on the supplied distance (given in the coordinates of the existing shapeObj). Requires GEOS support. Returns NULL/undef on failure.

contains(pointObj point) [int] Returns MS_TRUE if the point is inside the shape, MS_FALSE otherwise.

contains(shapeObj shape2) [int] Returns MS_TRUE if shape2 is entirely within the shape. Returns -1 on error and MS_FALSE otherwise. Requires GEOS support.

convexHull() [shapeObj] Returns the convex hull of the existing shape. Requires GEOS support. Returns NULL/undef on failure.

copy(shapeObj shape_copy) [int] Copy the shape to *shape_copy*. Returns MS_SUCCESS or MS_FAILURE.

clone() [shapeObj] Return an independent copy of the shape.

crosses(shapeObj shape2) [int] Returns MS_TRUE if shape2 crosses the shape. Returns -1 on error and MS_FALSE otherwise. Requires GEOS support.

difference(shapeObj shape) [shapeObj] Returns the computed difference of the supplied and existing shape. Requires GEOS support. Returns NULL/undef on failure.

disjoint(shapeObj shape2) [int] Returns MS_TRUE if shape2 and the shape are disjoint. Returns -1 on error and MS_FALSE otherwise. Requires GEOS support.

distanceToPoint(pointObj point) [float] Return distance to *point*.

distanceToShape(shapeObj shape) [float] Return the minimum distance to *shape*.

draw(mapObj map, layerObj layer, imageObj img) [int] Draws the individual shape using layer. Returns MS_SUCCESS or MS_FAILURE.

equals(shapeObj shape2) [int] Returns MS_TRUE if the shape and shape2 are equal (geometry only). Returns -1 on error and MS_FALSE otherwise. Requires GEOS support.

fromWKT(char *wkt) [shapeObj] Returns a new shapeObj based on a well-known text representation of a geometry. Requires GEOS support. Returns NULL/undef on failure.

get(int index) [lineObj] Returns a reference to part at *index*. Reference is valid only during the life of the shapeObj.

getArea() [double] Returns the area of the shape (if applicable). Requires GEOS support.

getCentroid() [pointObj] Returns the centroid for the existing shape. Requires GEOS support. Returns NULL/undef on failure.

getLength() [double] Returns the length (or perimeter) of a shape. Requires GEOS support.

getValue(int i) [string] Return the shape attribute at index *i*.

initValues(int numvalues) [void] Allocates memory for the requested number of values.

intersects(shapeObj shape) [int] Returns MS_TRUE if the two shapes intersect, MS_FALSE otherwise. Note, does not require GEOS support but will use GEOS functions if available.

intersection(shapeObj shape) [shapeObj] Returns the computed intersection of the supplied and existing shape. Requires GEOS support. Returns NULL/undef on failure.

overlaps(shapeObj shape2) [int] Returns MS_TRUE if shape2 overlaps shape. Returns -1 on error and MS_FALSE otherwise. Requires GEOS support.

project(projectionObj proj_in, projectionObj proj_out) [int] Reproject shape from *proj_in* to *proj_out*. Transformation is done in place. Returns MS_SUCCESS or MS_FAILURE.

setBounds [void] Must be called to calculate new bounding box after new parts have been added.

TODO: should return int and set msSetError.

setValue(int i, string value) [int] Set the shape value at index *i* to *value*.

symDifference(shapeObj shape) [shapeObj] Returns the computed symmetric difference of the supplied and existing shape. Requires GEOS support. Returns NULL/undef on failure.

touches(shapeObj shape2) [int] Returns MS_TRUE if the shape and shape2 touch. Returns -1 on error and MS_FALSE otherwise. Requires GEOS support.

toWKT() [string] Returns the well-known text representation of a shapeObj. Requires GEOS support. Returns NULL/undef on failure.

Union(shapeObj shape) [shapeObj] Returns the union of the existing and supplied shape. Shapes must be of the same type. Requires GEOS support. Returns NULL/undef on failure.

within(shapeObj shape2) [int] Returns MS_TRUE if the shape is entirely within shape2. Returns -1 on error and MS_FALSE otherwise. Requires GEOS support.

styleObj

An instance of styleObj is associated with one instance of classObj.

```
+-----+ 0..*      1 +-----+
| Style | <----- | Class |
+-----+          +-----+
```

An instance of styleObj can exist outside of a classObj container and be explicitly inserted into the classObj for use in mapping.

```
new_style = new styleObj()
the_class.insertStyle(new_style)
```

It is important to understand that insertStyle inserts a **copy** of the styleObj instance, not a reference to the instance itself.

The older use case

```
new_style = new styleObj(the_class)
```

remains supported. These will be the only ways to access the styles of a class. Programmers should no longer directly access the styles attribute.

styleObj Attributes

angle [double] Angle, given in degrees, to draw the line work. Default is 0. For symbols of Type HATCH, this is the angle of the hatched lines.

angleitem [string] Attribute/field that stores the angle to be used in rendering. Angle is given in degrees with 0 meaning no rotation.

antialias [int] MS_TRUE or MS_FALSE. Should TrueType fonts and Cartoline symbols be antialiased.

backgroundcolor [colorObj] Background pen color.

color [colorObj] Foreground or fill pen color.

mincolor [colorObj] Attribute for Color Range Mapping (*MS RFC 6: Color Range Mapping of Continuous Feature Values*). mincolor, minvalue, maxcolor, maxvalue define the range for mapping a continuous feature value to a continuous range of colors when rendering the feature on the map.

minsize [int] Minimum pen or symbol width for scaling styles.

minvalue [double] Attribute for Color Range Mapping (*MS RFC 6: Color Range Mapping of Continuous Feature Values*). mincolor, minvalue, maxcolor, maxvalue define the range for mapping a continuous feature value to a continuous range of colors when rendering the feature on the map.

minwidth [int] Minimum width of the symbol.

maxcolor [colorObj] Attribute for Color Range Mapping (*MS RFC 6: Color Range Mapping of Continuous Feature Values*). mincolor, minvalue, maxcolor, maxvalue define the range for mapping a continuous feature value to a continuous range of colors when rendering the feature on the map.

maxsize [int] Maximum pen or symbol width for scaling.

maxvalue [double] Attribute for Color Range Mapping (*MS RFC 6: Color Range Mapping of Continuous Feature Values*). mincolor, minvalue, maxcolor, maxvalue define the range for mapping a continuous feature value to a continuous range of colors when rendering the feature on the map.

maxwidth [int] Maximum width of the symbol.

offsetx [int] Draw with pen or symbol offset from map data.

offsety [int] Draw with pen or symbol offset from map data.

outlinecolor [colorObj] Outline pen color.

rangeitem [string] Attribute/field that stores the values for the Color Range Mapping (*MS RFC 6: Color Range Mapping of Continuous Feature Values*).

size [int] Pixel width of the style's pen or symbol.

sizeitem [string] Attribute/field that stores the size to be used in rendering. Value is given in pixels.

symbol [int] The index within the map symbolset of the style's symbol.

symbolname [string immutable] Name of the style's symbol.

width [int] Width refers to the thickness of line work drawn, in pixels. Default is 1. For symbols of Type HATCH, the with is how thick the hatched lines are.

styleObj Methods

new styleObj([classObj parent_class]) [styleObj] Returns new default style Obj instance. The *parent_class* is optional.

clone [styleObj] Returns an independent copy of the style with no parent class.

setSymbolByName(mapObj map, string symbolname) [int] Setting the symbol of the styleObj given the reference of the map object and the symbol name.

symbolObj

A symbolObj is associated with one symbolSetObj.

```
+-----+ 0..*      1 +-----+
| Symbol | <----- | SymbolSet |
+-----+          +-----+
```


A styleObj will often refer to a symbolObj by name or index, but this is not really an object association, is it?

symbolObj Attributes

antialias [int] MS_TRUE or MS_FALSE.

character [string] For TrueType symbols.

filled [int] MS_TRUE or MS_FALSE.

font [string] For TrueType symbols.

gap [int] **TODO** what is this?

imagepath [string] Path to pixmap file.

linecap [int] **TODO** unsure about the cartoline attributes.

linejoin [int] **TODO**

linejoinmaxsize [float] **TODO**

name [string] Symbol name

numpoints [int immutable] Number of points of a vector symbol.

position [int] **TODO** ?

sizeX [float] **TODO** what is this?

sizeY [float] **TODO** what is this?

stylelength [int] Number of intervals

transparent [int] **TODO** what is this?

transparentcolor [int] **TODO** is this a derelict attribute?

type [int] MS_SYMBOL_SIMPLE, MS_SYMBOL_VECTOR, MS_SYMBOL_ELLIPSE, MS_SYMBOL_PIXMAP, MS_SYMBOL_TRUETYPE, or MS_SYMBOL_CARTOLINE.

symbolObj Methods

new symbolObj(string symbolname [, string imagefile]) [symbolObj] Create new default symbol named *name*. If *imagefile* is specified, then the symbol will be of type MS_SYMBOL_PIXMAP.

getImage() [imageObj] Returns a pixmap symbol's imagery as an imageObj.

getPoints() [lineObj] Returns the symbol points as a lineObj.

setImage(imageObj image) [int] Set a pixmap symbol's imagery from *image*.

setPoints(lineObj line) [int] Sets the symbol points from the points of *line*. Returns the updated number of points.

setStyle(int index, int value) [int] Set the style at *index* to *value*. Returns MS_SUCCESS or MS_FAILURE.

symbolSetObj

A symbolSetObj is an attribute of a mapObj and is associated with instances of symbolObj.

```
+-----+ 1      0..* +-----+
| SymbolSet | -----> | Symbol |
+-----+          +-----+
```

symbolSetObj Attributes

filename [string] Symbolset filename

numsymbols [int immutable] Number of symbols in the set.

symbolSetObj Methods

new symbolSetObj([string symbolfile]) [symbolSetObj] Create new instance. If *symbolfile* is specified, symbols will be loaded from the file.

appendSymbol(symbolObj symbol) [int] Add a copy of *symbol* to the symbolset and return its index.

getSymbol(int index) [symbolObj] Returns a reference to the symbol at *index*.

getSymbolByName(string name) [symbolObj] Returns a reference to the symbol named *name*.

index(string name) [int] Return the index of the symbol named *name* or -1 in the case that no such symbol is found.

removeSymbol(int index) [symbolObj] Remove the symbol at *index* and return a copy of the symbol.

save(string filename) [int] Save symbol set to a file. Returns MS_SUCCESS or MS_FAILURE.

webObj

Has no other existence than as an attribute of a mapObj. Serves as a container for various run-time web application definitions like temporary file paths, template paths, etc.

webObj Attributes

empty [string] **TODO**

error [string] **TODO**

extent [rectObj] Clipping extent.

footer [string] Path to footer document.

header [string] Path to header document.

imagepath [string] Filesystem path to temporary image location.

imageurl [string] URL to temporary image location.

log [string] **TODO**

map [mapObj immutable] Reference to parent mapObj.

maxscale [float] Maximum map scale.

maxtemplate [string] **TODO**

metadata [hashTableObj immutable] metadata hash table.

minscale [float] Minimum map scale.

mintemplate [string] **TODO**

queryformat [string] **TODO**

template [string] Path to template document.

webObj Methods

None.

6.3 PHP MapScript

Author Daniel Morissette

Contact dmorissette at mapgears.com

Author Yewondwossen Assefa

Contact yassefa at dmsolutions.ca

Revision \$Revision: 8497 \$

Date \$Date: 2009-02-04 08:59:31 -0800 (Wed, 04 Feb 2009) \$

6.3.1 Introduction

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2008/07/16

Table of Contents

- Introduction
 - Abstract
 - Introduction
 - How to Get More Information on PHP/MapScript

Abstract

This document describes all of the classes, properties and methods associated with the PHP/MapScript module, and is the online version of the PHP/MapScript README file from the MapServer source code.

Introduction

PHP MapScript was originally developed for PHP-3.0.14 but after MapServer 3.5 support for PHP3 has been dropped and as of the last update of this document, PHP 4.1.2 or more recent was required.

The module has been tested and used on Linux, Solaris, *BSD, and WinNT.

This module is constantly under development.

How to Get More Information on PHP/MapScript

- The main resource for help is the [PHP/MapScript page](#) on MapTools.org.
- The [MapServer Wiki](#) might have more information on this module
- For installation questions regarding the PHP/MapScript module, see *PHP MapScript Installation*.
- Also, see the *MapScript*, *PHP MapScript* and the *Mapfile*
- Refer to the [main PHP site](#) for their official documentation

6.3.2 By Example

Author Vinko Vrsalovic

Contact el at vinko.cl

Revision \$Revision: 8365 \$

Date \$Date: 2008-12-31 07:49:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2005/12/12

Contents

- By Example
 - Introduction
 - MapScript overview
 - Our first application
 - Conclusions

Introduction

The purpose of this document is to be a step by step explanation of the *PHP MapScript* with practical examples for each of them. It is assumed a basic knowledge of *MAP* and MapServer, and familiarity with the [PHP \(scripting\)](#) and [HTML \(markup\)](#) languages . This document was originally created for MapServer v4.0, but the examples still apply to more recent versions.

Let's Begin...

Hello, kind reader. I am Tut, thank you for downloading me. I am sorry, but I am just a technical manual so I cannot answer any questions. The maintainer, a handsome, very nice and lazy guy according to what I saw from the other side of the screen, maybe will be able to answer your question(s). I am currently here to tell you about MapScript in its PHP incarnation. At my current age, I will be more useful to beginners than advanced users, even though I hope that some day I will be sufficiently old to be useful to advanced MapScript programmers.

Let's hope I live long enough... sigh.

But enough with my personal problems, let myself begin. My duty is to familiarize you with MapScript, and in particular with PHP MapScript. When I end, you are expected to understand what MapScript is, and to be able to write applications to display and navigate that is, zooming and panning over shapefiles via a web browser.

What follows are the questions you must answer affirmatively before accompanying me through the rest of this journey (I apologize for my maintainer's lack of literary taste).

Do you have running somewhere...

- a web server capable of running PHP as a CGI (Apache will do)?
- the PHP language configured as a CGI, version 4.1.2 or higher? I recommend 4.3 onwards.
- PHP MapScript, version 4.0 or later? *PHP MapScript Installation*

Can you...

- code PHP or are willing to [learn how to](#)?
- write and understand [HTML](#) documents? (Note that Javascript is a plus)
- tell somebody what on earth is a [shapefile](#) [or a [PostGIS](#) table]?

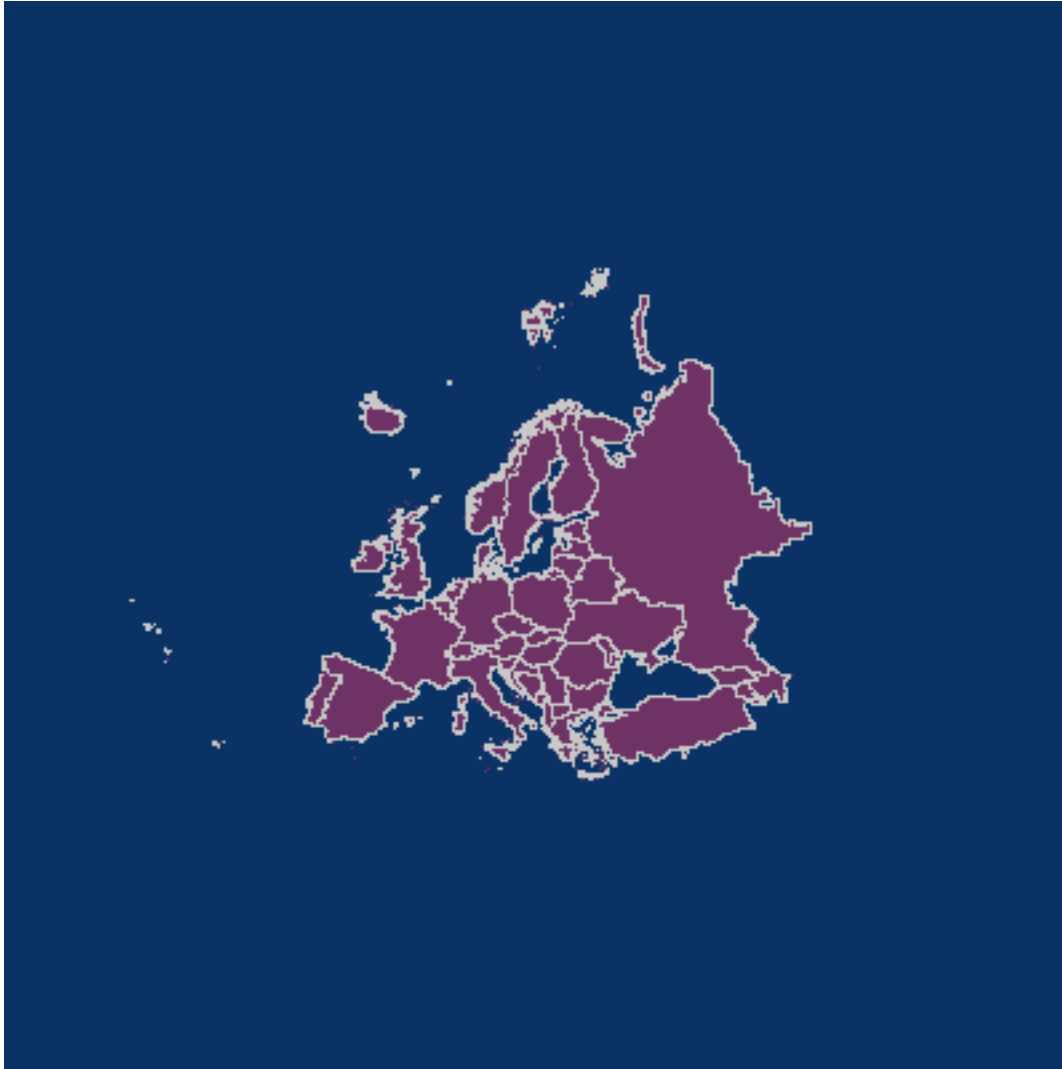
Outline of this Document

- A general overview of MapScript, in a language independent way
- A trivial example
- A simple example
- Conclusion

You can also go to each part directly through my table of contents located at the top, if you wish to skip some sections.

MapScript overview

Ok, now I'm at last arriving at a point I will enjoy. This overview intends to clear some common misconceptions beginners encounter when first facing MapScript and to give a general overview about MapScript's internals. For now, just look at the following diagram (I apologize again for the maintainer's lack of graphic design taste).



It all starts as everything on the Web. A browser requests a certain URL through HTTP. The request arrives at the web server, which, in turn, delivers a file or executes a program and then delivers its output back to the browser. Yes, I know you knew that, but I have been told to be as complete as possible, and I will try to.

In MapScript's case, the server executes a certain script, which contains standard language functionality, that is, the same functionality you would have in that language without MapScript, plus access to almost all of the MapServer C API, the level of completeness of MapServer API support varies a bit with the language you choose, but I think it is my duty to tell you almost every available flavor of MapScript is usable. This API, exposed now in your scripting language through the MapScript module, allows you to do many GIS-like operations on spatial data, including read-write access to shapefiles, reprojection of data, and many others. For more information on the API, click over the link above. For other flavors, you can check their own documentation, you will see there is not much difference.

The CGI version of MapServer is not required to run MapScript applications, just as you don't need a particular MapScript module to run the CGI. The CGI version has many features out-of-the-box, MapScript is just an API, so with MapScript you must start from scratch or with some of the examples available. Think of the CGI as of a MapScript application written directly in C, with direct access to the MapServer C API. Sometimes the out-of-the-box functionality has some limits which can be surpassed by MapScript, but not embedded within the CGI. In other words, the CGI is not scriptable, but you can program all the CGI and more with MapScript. This may seem a strange thing to clarify, but is a common misconception, just check the [list archives](#) if you are not inclined to believe me.

As with MapServer itself, MapScript can be configured using only map files, but, unlike the CGI, also includes the

possibility of dynamically create maps or modify existing ones and to (and here is the key to the flexibility that MapScript has) mix this information with other sources of non GIS data, such as user input, non spatial and spatial databases, text files, etc. and that you can use every single module your language provides. The power of this approach is tremendous, and the most restrictive limit is your imagination. As always, flexibility comes with a price, performance. It's generally slower to use a scripting language instead of C, but nowadays this shouldn't be a big worry. And you can still program directly in C (there are not much documents about how to do it, though you might want to check the [mapserver-dev list](#)) if you would like to.

The input and output formats MapScript can handle are exactly the same as the ones configured when you build MapServer/MapScript. But one of the most important things to remember is that, basically, you feed geographic data and relevant user input (for instance clicks over the map image) to MapScript and as a result get one or more file(s), typically standard image files such as a PNG or JPEG. So you can apply anything you've seen in any server side scripted web application, DHTML, Java applets, CSS, HTML templates, sessions, you name it.

Our first application

In this first example, I will tell you how to display a shapefile on a web page using a map file.

The Map File

Here's the map file:

```
NAME "Europe in purple"
SIZE 400 400
STATUS ON
SYMBOLSET "/var/www/html/maps/symbols/symbols.sym"
EXTENT -5696501 1923039 5696501 11022882
UNITS METERS
SHAPEPATH "/var/www/html/maps/data"

WEB
    IMAGEPATH "/var/www/html/maps/tmp/"
    IMAGEURL "/tmp/"
END

LAYER
    NAME "Europe"
    TYPE POLYGON
    STATUS ON
    DATA "europe"
    CLASS
        STYLE
            COLOR 110 50 100
            OUTLINECOLOR 200 200 200
            SYMBOL 0
        END
    END
END

END
```

Here I have shown a map with a single layer, where the europe.shp, europe.shx and europe.dbf files must be located in the subdirectory called data. The symbols are located in the symbols subdirectory. All this locations are relative from the place the map file is, but better safe than sorry, I guess. The web section is used to define where will the images be saved and in what URL will they be available.

Displaying the map with MapScript

To display a map the following MapScript objects and methods will be used:

- MapObj object
- imageObj object

MapObj methods:

- The constructor method: `MapObj ms_newMapObj(string map_file_name[,string new_map_path])`
- The draw method: `imageObj draw()`

imageObj methods:

- The `saveWebImage` method: `string saveWebImage()`

The code looks like this:

```
1 <?php
2 dl('php_mapscript.so');
3 $map_path="/var/www/html/ms/map_files/";
4 $map = ms_newMapObj($map_path."europe.map");
5 $image=$map->draw();
6 $image_url=$image->saveWebImage();
7 ?>
8 <HTML>
9 <HEAD>
10 <TITLE>Example 1: Displaying a map</TITLE>
11 </HEAD>
12 <BODY>
13 <IMG SRC=<?php echo $image_url; ?> >
14 </BODY>
15 </HTML>
```

The code I will present through the rest of this document will follow the following rule:

- Every non empty line is numbered

This code will render an image corresponding to the shapefile europe and display it on a HTML page.

Code Explanation

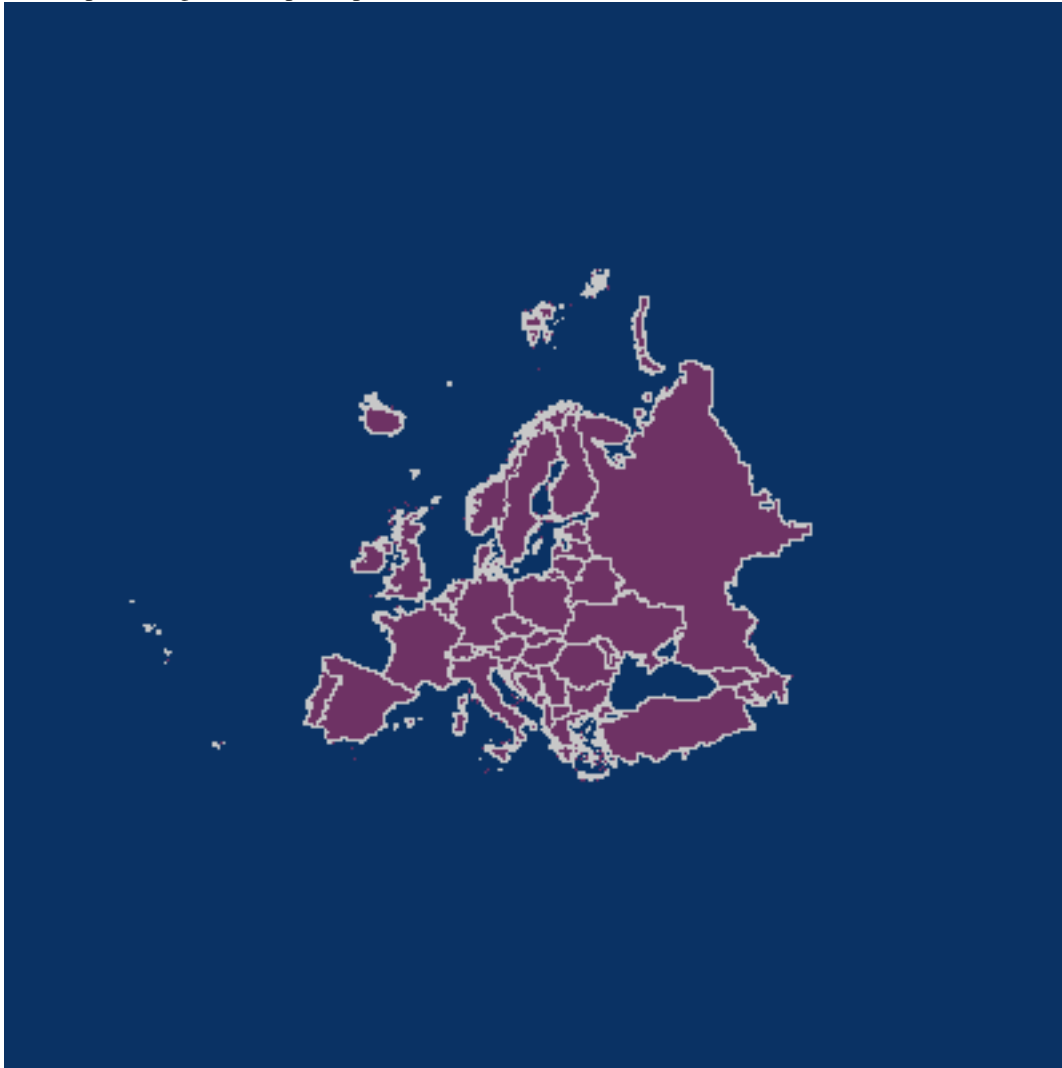
- In line 2 it is loaded the MapScript extension (you may not need it if your `php.ini` file is configured to automatically load it).
- Line 3 declares a variable that holds the absolute path for the mapfile.
- Line 4 creates an instance of the MapObj object using the constructor. As you can see, the constructor receives the location of the map file as its only required parameter, and the map file received the `europe.map` name.
- Afterwards the draw method of the map object is called to render the image defined by the map file (line 5). The result (an imageObj) is saved in the `$image` variable.

- Line 6 calls the `saveWebImage` method to generate the image file, it returns a string which represents the URL as defined in the mapfile (in this case, `/tmp/filename.png`).
- The rest of the lines are pure HTML, except line 13, that defines the source URL of the image will be the value stored in `$image_url`.

You should test the application on your system, to check that it really works and to solve the problems that may arise on your particular configuration before moving on to the more complex examples.

Output

The output (using the europe shapefile) should look like this:



Zooming and Panning

Now I will tell you how to add zoom and pan capabilities to the code.

Here goes the list of new methods and objects called.

New Objects:

- pointObj
- rectObj

New Methods and Members called:

- The zoompoint method of the map object: void zoompoint(int nZoomFactor, pointObj oPixelPos, int nImageWidth, int nImageHeight, rectObj oGeorefExt).
- The setextent method of the map object: \$map->setextent(double minx, double miny, double maxx, double maxy);.
- The extent, width and height members of the map object.
- The constructors of RectObj and PointObj: \$point = ms_newPointObj(); \$rect = ms_newRectObj();
- The setXY method of the point object: \$point->setXY(double x_coord, double y_coord);
- The setextent method of the rectangle object: \$rect->setextent(double minx, double miny, double maxx, double maxy);

The .map file remains the same as the one presented in the previous example.

PHP/MapScript Code

Here I present the new code.

```
1 <?php
2 dl('php_mapscript.so');
3 // Default values and configuration
4 $val_zsize=3;
5 $check_pan="CHECKED";
6 $map_path="/var/www/html/ms/map_files/";
7 $map_file="europe.map";
8 $map = ms_newMapObj($map_path.$map_file);
9 if ( isset($_POST["mapa_x"]) && isset($_POST["mapa_y"])
10     && !isset($_POST["full"]) ) {
11     $extent_to_set = explode(" ", $_POST["extent"]);
12     $map->setextent($extent_to_set[0], $extent_to_set[1],
13                   $extent_to_set[2], $extent_to_set[3]);
14     $my_point = ms_newpointObj();
15     $my_point->setXY($_POST["mapa_x"], $_POST["mapa_y"]);
16     $my_extent = ms_newrectObj();
17     $my_extent->setextent($extent_to_set[0], $extent_to_set[1],
18                           $extent_to_set[2], $extent_to_set[3]);
19     $zoom_factor = $_POST["zoom"]*$_POST["zsize"];
20     if ($zoom_factor == 0) {
21         $zoom_factor = 1;
```

```

22         $check_pan = "CHECKED";
23         $check_zout = "";
24         $check_zin = "";
25     } else if ($zoom_factor < 0) {
26         $check_pan = "";
27         $check_zout = "CHECKED";
28         $check_zin = "";
29     } else {
30         $check_pan = "";
31         $check_zout = "";
32         $check_zin = "CHECKED";
33     }

34     $val_zsize = abs($zoom_factor);

35     $map->zoompoint($zoom_factor,$my_point,$map->width,$map->height,
36                 $my_extent);

37 }

38 $image=$map->draw();
39 $image_url=$image->saveWebImage();

40 $extent_to_html = $map->extent->minx." ".$map->extent->miny." "
41                 .$map->extent->maxx." ".$map->extent->maxy;

42 ?>
43 <HTML>
44 <HEAD>
45 <TITLE>Map 2</TITLE>
46 </HEAD>
47 <BODY>
48 <CENTER>
49 <FORM METHOD=POST ACTION=<?php echo $HTTP_SERVER_VARS['PHP_SELF']?>>
50 <TABLE>
51 <TR>
52     <TD>
53         <INPUT TYPE=IMAGE NAME="mapa" SRC="<?php echo $image_url?>">
54     </TD>
55 </TR>
56 <TR>
57     <TD>
58         Pan
59     </TD>
60     <TD>
61         <INPUT TYPE=RADIO NAME="zoom" VALUE=0 <?php echo $check_pan?>>
62     </TD>
63 </TR>
64 <TR>
65     <TD>
66         Zoom In
67     </TD>
68     <TD>
69         <INPUT TYPE=RADIO NAME="zoom" VALUE=1 <?php echo $check_zin?>>
70     </TD>
71 </TR>
72 <TR>

```

```
73     <TD>
74         Zoom Out
75     </TD>
76     <TD>
77         <INPUT TYPE=RADIO NAME="zoom" VALUE=-1 <?php echo $check_zout?>>
78     </TD>
79 </TR>
80 <TR>
81     <TD>
82         Zoom Size
83     </TD>
84     <TD>
85         <INPUT TYPE=TEXT NAME="zsize" VALUE="<?php echo $val_zsize?>"
86         SIZE=2>
87     </TD>
88 </TR>
89 <TR>
90     <TD>
91         Full Extent
92     </TD>
93     <TD>
94         <INPUT TYPE=SUBMIT NAME="full" VALUE="Go"
95         SIZE=2>
96     </TD>
97 </TABLE>
98 <INPUT TYPE=HIDDEN NAME="extent" VALUE="<?php echo $extent_to_html?>">
99 </FORM>
100 </CENTER>
101 </BODY>
102 </HTML>
```

This code will zoom out, zoom in, pan, and restore to full extent the image displayed in the previous example.

It looks much more complicated than it really is, much of the lines are the HTML code, and much of the remaining PHP code is just to deal with the forms and such.

You should try it and look at how it works first. Try it in your own server by copying and pasting the code.

Now it's time for you to play with it a little and look at the source in your browser to check how it changes.

Done?, now let's start the explanation with the HTML part.

Code Explanation - HTML

Line 49 declares a form, and line 53 declares the image generated by MapScript to be part of that form, so when you click on it, the X and Y coordinates of the click (in pixels) will be sent along with the other data for the PHP code to process.

If you are familiar with HTML and PHP, the rest of the HTML code should be straightforward for you to understand with the exception of line 98, that will be explained in due time.

Code Explanation - PHP

Now look at the PHP code, it's almost the same code used in example 1, with the addition of lines 9 to 37. What do these lines do?

Line 9 checks the relevant variables from the form have been setted. 'mapa_x' and 'mapa_y' represent the X and Y coordinates of the click over the image, and 'full' represents the click on the 'Full Extent' button.

The first time the page is displayed the code between the if statement doesn't get executed, but the rest of the code does. Lines 40 and 41 set the '\$extent_to_html' variable with the values of the extent defined in the map file separated by spaces; that value will be put in the HTML variable 'extent' in line 98.

Now look at line 11 and 12. We are inside the if statement, that means the form has been submitted at least once. We grab the extent stored in the previous execution (the 'extent' HTML variable) of the code and set the extent of the map to be that last extent. This allows to zoom or pan with respect of the previous extent, not the extent that is set in the map file.

From that last paragraph you can deduce that all the default values are set in the map file, and anything that you change through MapScript and would like to remain in your code, must be stored somehow. In this case it is done through hidden variables in a form. For more advanced applications you could use session variables or a database.

Now you should be able to see why the 'Full Extent' button works. If you check line 10, it says that if you haven't pressed the button, skip the code in the if statement, so the extent is reset to the value that the map file has. You should also see that it isn't necessarily a full extent (in case the extent in the map file is not full extent).

Lines 14 and 15 declare a new point object and initialize it with the values the user clicked on. You should not forget that those values are in pixels, not in georeferenced coordinates.

Lines 16 through 18 create a new rectangle object and set it with the extent of the previous image, just like it is done on line 12. In fact this would work too: `$my_extent = $map->extent;`

To do all the zooming and panning, the zoompoint function is called on line 35, but first the arguments it receives must be prepared. You can determine the point the user clicked on, and the extent of the image (`$my_point` and `$my_extent`, respectively), but now you have to determine the zoom factor. That's what lines 19 to 33 do. If you wondered why the values of the radio buttons were 0, -1, and 1 for pan, zoom in and zoom out, now you will know the reason.

A zoom factor of 1 tells zoompoint that the operation is pan, a negative value indicates zoom out and a positive value indicates zoom in. So, by means of multiplying the value received for the radio buttons (HTML variable 'zoom') by the size of the zoom the user entered the zoom factor is calculated. If that value is 0, that means the user selected the pan operation, so '\$zoom_factor' is set to 1, otherwise the result of the multiplication is the zoom factor zoompoint needs to receive. The other lines are to preserve the button the user clicked on the next time. Line 34 tries to preserve the value of the zoom size the user entered (It doesn't do that all the time, when and why that line fails? That's for you to find out).

And finally, line 34 calls the zoompoint method with the zoom factor obtained, the point built from the pixel coordinates (I insist on that issue because zoompoint is almost the only method that receives the coordinates in pixels, for the other methods you must convert pixels to georeferenced coordinates on your own), the height and width of the image, and the extent.

After calling zoompoint, the extent of the image is changed accordingly to the operation performed (or, better put, the zoom factor). So then the image is drawn and the current extent saved (after the zooming) for use in the next iteration.

Conclusions

Well, it's time for me to go recharge my batteries. So I will use this last energy to share some final words. The examples I have managed to present here are very basic but you should now be able to devise ways to improve them and suit things to your needs. Keep in mind that you can preprocess, store, read, write data from any source you can usually read through PHP, plus all the sources MapServer can handle for GIS data. You can even process some GIS data with PHP only if the need would arise (SQL sources are a good example of this). You can also do hybrid approaches where some script prepares data which is then shown through the CGI interface to MapServer, or create data on the fly based on input from a GPS, etc, etc. The possibilities are just too many to enumerate completely. As I already said your imagination is the limit. The next version of this document will include examples that include more

than one layer, with different datasources (not just shapefiles) and creation of dynamic layers and classes. If you have a better idea or would like to see some other thing here first, please drop a note to my maintainer.

In the meantime, if you need bigger examples you can refer to the [GMap demo](#) (you can download the source [here](#) or as an [MS4W packaged application](#)), or the [MapTools site](#) (MapLab, Chameleon). Goodbye, and thanks for reading this far.

Contents

- PHP MapScript
 - Very important notes
 - Constants
 - Functions
 - MapObj Class
 - LayerObj Class
 - ClassObj Class
 - ImageObj Class
 - LabelObj Class
 - webObj Class
 - referenceMapObj Class
 - ColorObj Class
 - PointObj Class
 - LineObj Class
 - ShapeObj Class
 - RectObj Class
 - ShapefileObj Class
 - ResultCacheMemberObj Class
 - ProjectionObj Class
 - ScalebarObj Class
 - LegendObj Class
 - QuerymapObj Class
 - StyleObj Class
 - OutputformatObj Class
 - GridObj Class
 - ErrorObj Class
 - LabelcacheObj Class
 - SymbolObj Class
 - OwsrequestObj Class
 - hashTableObj Class

This is a PHP module to make MapServer's MapScript functionalities available in a PHP Dynamically Loadable Library.

PHP MapScript was originally developed for PHP-3.0.14 but after MapServer 3.5 support for PHP3 has been dropped and as of the last update of this document, PHP 4.1.2 or more recent was required.

The module has been tested and used on Linux, Solaris, *BSD, and WinNT.

There are documentation and examples for PHP MapScript available via the MapServer *documentation*

See also the MapServer Wiki for links to more information on this module:
<http://trac.osgeo.org/mapserver/wiki/PHPMapScript>

6.3.3 Very important notes

- Constant names and class member variable names are case-sensitive in PHP.
- Several MapScript functions (all those that access files in the back end such as `ms_newMapObj()`, `drawMap()`, etc) will affect the value of the current working directory (CWD) in the PHP environment. This will be fixed eventually but in the meantime you should be careful about these side-effects.

6.3.4 Constants

The following MapServer constants are available:

Boolean values MS_TRUE, MS_FALSE, MS_ON, MS_OFF, MS_YES, MS_NO

Map units MS_INCHES, MS_FEET, MS_MILES, MS_METERS, MS_KILOMETERS, MS_DD, MS_PIXELS

Layer types MS_LAYER_POINT, MS_LAYER_LINE, MS_LAYER_POLYGON, MS_LAYER_RASTER, MS_LAYER_ANNOTATION, MS_LAYER_QUERY, MS_LAYER_CIRCLE, MS_LAYER_TILEINDEX, MS_LAYER_CHART

Layer/Legend/Scalebar/Class Status MS_ON, MS_OFF, MS_DEFAULT, MS_EMBED, MS_DELETE

Layer alpha transparency allows alpha transparent pixmaps to be used with RGB map images MS_GD_ALPHA

Font types MS_TRUETYPE, MS_BITMAP

Label positions MS_UL, MS_LR, MS_UR, MS_LL, MS_CR, MS_CL, MS_UC, MS_LC, MS_CC, MS_AUTO, MS_XY, MS_FOLLOW

Bitmap font styles MS_TINY, MS_SMALL, MS_MEDIUM, MS_LARGE, MS_GIANT

Shape types MS_SHAPE_POINT, MS_SHAPE_LINE, MS_SHAPE_POLYGON, MS_SHAPE_NULL

Shapefile types MS_SHP_POINT, MS_SHP_ARC, MS_SHP_POLYGON, MS_SHP_MULTIPPOINT

Query/join types MS_SINGLE, MS_MULTIPLE

Querymap styles MS_NORMAL, MS_HILITE, MS_SELECTED

Connection Types MS_INLINE, MS_SHAPEFILE, MS_TILED_SHAPEFILE, MS_SDE, MS_OGR, MS_TILED_OGR, MS_POSTGIS, MS_WMS, MS_ORACLESPIATIAL, MS_WFS, MS_GRATICULE, MS_MYGIS, MS_RASTER, MS_PLUGIN

Error codes MS_NOERR, MS_IOERR, MS_MEMERR, MS_TYPEERR, MS_SYMMERR, MS_REGEXERR, MS_TTFERR, MS_DBFERR, MS_GDERR, MS_IDENTERR, MS_EOFERR, MS_PROJERR, MS_MISCERR, MS_CGIERR, MS_WEBERR, MS_IMGERR, MS_HASHERR, MS_JOINERR, MS_NOTFOUND, MS_SHPERR, MS_PARSEERR, MS_SDEERR, MS_OGRERR, MS_QUERYERR, MS_WMSERR, MS_WMSCONNERR, MS_ORACLESPIATIALERR, MS_WFSERR, MS_WFSCONNERR, MS_MAPCONTEXTERR, MS_HTTPERR, MS_WCSERR

Symbol types MS_SYMBOL_SIMPLE, MS_SYMBOL_VECTOR, MS_SYMBOL_ELLIPSE, MS_SYMBOL_PIXMAP, MS_SYMBOL_TRUETYPE, MS_SYMBOL_CARTOLINE

Image Mode types (*outputFormatObj*) MS_IMAGEMODE_PC256, MS_IMAGEMODE_RGB, MS_IMAGEMODE_RGBA, MS_IMAGEMODE_INT16, MS_IMAGEMODE_FLOAT32, MS_IMAGEMODE_BYTE, MS_IMAGEMODE_NULL

Style/Attribute binding MS_STYLE_BINDING_SIZE, MS_STYLE_BINDING_ANGLE, MS_STYLE_BINDING_COLOR, MS_STYLE_BINDING_OUTLINECOLOR, MS_STYLE_BINDING_SYMBOL

Label/Attribute binding MS_LABEL_BINDING_SIZE, MS_LABEL_BINDING_ANGLE,
 MS_LABEL_BINDING_COLOR, MS_LABEL_BINDING_OUTLINECOLOR,
 MS_LABEL_BINDING_FONT, MS_LABEL_BINDING_PRIORITY

Alignment MS_ALIGN_LEFT, MS_ALIGN_CENTER, MS_ALIGN_RIGHT

OwsRequest MS_GET_REQUEST, MS_POST_REQUEST

6.3.5 Functions

string ms_GetVersion() Returns the MapServer version and options in a string. This string can be parsed to find out which modules were compiled in, etc.

int ms_GetVersionInt() Returns the MapServer version number (x.y.z) as an integer ($x*10000 + y*100 + z$). (New in v5.0) e.g. V5.4.3 would return 50403.

array ms_TokenizeMap(string map_file_name) Prepares a mapfile through the MapServer parser and return an array with one item for each token from the mapfile. Strings, logical expressions, regex expressions and comments are returned as individual tokens.

void ms_ioinstallstdouttobuffer() Installs a mapserver IO handler directing future stdout output to a memory buffer.

void ms_ioinstallstdinfrombuffer() Installs a mapserver IO handler directing future stdin reading (ie. post request capture) to come from a buffer.

void ms_iogetstdoutbufferstring() Fetch the current stdout buffer contents as a string. This method does not clear the buffer.

int ms_iogetStdoutBufferBytes() Writes the current buffer to stdout. The PHP header() function should be used to set the documents's content-type prior to calling the function. Returns the number of bytes written if output is sent to stdout. See *MapScript Wrappers for WxS Services* for more info.

void ms_ioresethandlers() Resets the default stdin and stdout handlers in place of "buffer" based handlers.

void ms_iostripstdoutbuffercontenttype() Strip the Content-type header off the stdout buffer if it has one, and if a content type is found it is return. Otherwise return false.

6.3.6 MapObj Class

Constructor

mapObj ms_newMapObj(string map_file_name [, string new_map_path]) Returns a new object to deal with a MapServer map file.

mapObj ms_newMapObjFromString(string map_file_string [, string new_map_path]) Construct a new mapObj from a mapfile string. Returns a new object to deal with a MapServer map file.

Note: By default, the SYMBOLSET, FONTSET, and other paths in the mapfile are relative to the mapfile location. If new_map_path is provided then this directory will be used as the base path for all the relative paths inside the mapfile.

Members

Type	Name
Continued on next page	

Table 6.1 – continued from previous page

int	numlayers (read-only)
string	name
int	status
int	debug
int	width (see setSize())
int	height (see setSize())
int	maxsize
outputformatObj	outputformat
double	resolution (pixels per inch, defaults to 72)
rectObj	extent;
double	cellsize
int	units (map units type)
double	scaledenom (read-only, set by drawMap())
double	scale (Deprecated in v5.0. Use scaledenom instead)
string	shapepath
int	keysizeX
int	keysizeY
int	keyspacingX
int	keyspacingY
webObj	web
referenceMapObj	reference
colorObj	imagecolor
scalebarObj	scalebar
legendObj	legend
string	symbolsetfilename (read-only, set by setSymbolSet())
string	fontsetfilename (read-only, set by setFontSet())
labelcacheObj	labelcache (no members. Used only to be able to free the the label cache (ex : map->labelcache->free()))
projectionObj	projection
int	transparent (deprecated, use outputFormatObj)
int	interlace (deprecated, use outputFormatObj)
int	imagetype (deprecated, use outputFormatObj)
int	imagequality (deprecated, use outputFormatObj)

Methods

mapObj clone() Returns a handle to a new mapObj which is a clone of the current mapObj. All parameters in the current mapObj are copied to the new mapObj. Returns NULL (0) on error.

int set(string property_name, new_value) Set map object property to new value. Returns -1 on error.

int getsymbolbyname(string symbol_name) Returns the symbol index using the name.

symbol getsymbolobjectbyid(int symbolid) Returns the symbol object using a symbol id. Refer to the symbol object reference section for more details.

void preparequery() Calculate the scale of the map and set map->scaledenom.

imageObj prepareImage() Return handle on blank image object.

imageObj draw() Render map and return handle on image object.

imageObj drawQuery() Render a query map and return handle on image object.

imageObj drawLegend() Render legend and return handle on image object.

imageObj drawReferenceMap() Render reference map and return handle on image object.

imageObj drawScaleBar() Render scale bar and return handle on image object.

int embedlegend(imageObj image) embeds a legend. Actually the legend is just added to the label cache so you must invoke drawLabelCache() to actually do the rendering (unless postlabelcache is set in which case it is drawn right away).

int embedScalebar(imageObj image) embeds a scalebar. Actually the scalebar is just added to the label cache so you must invoke drawLabelCache() to actually do the rendering (unless postlabelcache is set in which case it is drawn right away).

int drawLabelCache(imageObj image) Renders the labels for a map. Returns -1 on error.

layerObj getLayer(int index) Returns a layerObj from the map given an index value (0=first layer)

layerObj getLayerByName(string layer_name) Returns a layerObj from the map given a layer name. Returns FALSE if layer doesn't exist.

colorObj getcolorbyindex(int iCloIndex) Returns a colorObj corresponding to the color index in the palette

void setExtent(double minx, double miny, double maxx, double maxy) Set the map extents using the georef extents passed in argument.

int setRotation(double rotation_angle) Set map rotation angle. The map view rectangle (specified in EXTENTS) will be rotated by the indicated angle in the counter- clockwise direction. Note that this implies the rendered map will be rotated by the angle in the clockwise direction. Returns MS_SUCCESS or MS_FAILURE.

int setSize(int width, int height) Set the map width and height. This method updates the internal geotransform and other data structures required for map rotation so it should be used instead of setting the width and height members directly. Returns MS_SUCCESS or MS_FAILURE.

void zoompoint(int nZoomFactor, pointObj oPixelPos, int nImageWidth, int nImageHeight, rectObj oGeorefExt) Zoom to a given XY postion.

Parameters are :

- Zoom factor : positive values do zoom in, negative values zoom out. Factor of 1 will recenter.

- Pixel position (pointObj) : x, y coordinates of the click, with (0,0) at the top-left
- Width : width in pixel of the current image.
- Height : Height in pixel of the current image.
- Georef extent (rectObj) : current georef extents.
- MaxGeoref extent (rectObj) : (optional) maximum georef extents. If provided then it will be impossible to zoom/pan outside of those extents.

void zoomrectangle(rectObj oPixelExt, int nImageWidth, int nImageHeight, rectObj oGeorefExt) Set the map extents to a given extents.

Parameters are :

- oPixelExt (rect object) : Pixel Extents, with (0,0) at the top-left The rectangle contains the coordinates of the LL and UR coordinates in pixel. (the maxy in the rect object should be < miny value)

```
----- UR (values in the rect object : maxx, maxy)
|         |
|         |
|         |
-----
LL (values in the rectobject minx, miny)
```

- Width : width in pixel of the current image.
- Height : Height in pixel of the current image.

- Georef extent (rectObj) : current georef extents.

void zoomscale(double nScaleDenom, pointObj oPixelPos, int nImageWidth, int nImageHeight, rectObj oGeorefExt [, rectObj oMaxGeorefExt]) Zoom in or out to a given XY position so that the map is displayed at specified scale.

Parameters are :

- ScaleDenom : Scale denominator of the scale at which the map should be displayed.

- Pixel position (pointObj) : x, y coordinates of the click, with (0,0) at the top-left
- Width : width in pixel of the current image.
- Height : Height in pixel of the current image.
- Georef extent (rectObj) : current georef extents.
- MaxGeoref extent (rectObj) : (optional) maximum georef extents. If provided then it will be impossible to zoom/pan outside of those extents.

int queryByPoint(pointObj point, int mode, double buffer) Query all selected layers in map at point location specified in georeferenced map coordinates (i.e. not pixels). The query is performed on all the shapes that are part of a *CLASS* that contains a *Templating* value or that match any class in a layer that contains a *LAYER TEMPLATE* value. Mode is MS_SINGLE or MS_MULTIPLE depending on number of results you want. Passing buffer -1 defaults to tolerances set in the map file (in pixels) but you can use a constant buffer (specified in ground units) instead. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int queryByRect(rectObj rect) Query all selected layers in map using a rectangle specified in georeferenced map coordinates (i.e. not pixels). The query is performed on all the shapes that are part of a *CLASS* that contains a *Templating* value or that match any class in a layer that contains a *LAYER TEMPLATE* value. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int queryByShape(shapeObj shape) Query all selected layers in map based on a single shape, the shape has to be a polygon at this point. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int queryByFeatures(int slayer) Perform a query based on a previous set of results from a layer. At present the results MUST be based on a polygon layer. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int queryByIndex(layerindex, tileindex, shapeindex, addtoquery) Add a specific shape on a given layer to the query result. If addtoquery (which is a non mandatory argument) is set to MS_TRUE, the shape will be added to the existing query list. Default behavior is to free the existing query list and add only the new shape.

int savequery(filename) Save the current query in a file. Returns MS_SUCCESS or MS_FAILURE. Can be used with loadquery

int loadquery(filename) Loads a query from a file. Returns MS_SUCCESS or MS_FAILURE. To be used with savequery.

void freequery(layerindex) Frees the query result on a specified layer. If the layerindex is -1, all queries on layers will be freed.

int save(string filename) Save current map object state to a file. Returns -1 on error. Use absolute path. If a relative path is used, then it will be relative to the mapfile location.

string getProjection() Returns a string representation of the projection. If no projection is set, MS_FALSE is returned.

int setProjection(string proj_params, boolean bSetUnitsAndExtents) Set map projection and coordinate system. Parameters are given as a single string of comma-delimited PROJ.4 parameters. The argument : bSetUnitsAndExtents is used to automatically update the map units and extents based on the new projection. Possible values are MS_TRUE and MS_FALSE. By default it is set at MS_FALSE

int setWKTProjection(string proj_params, boolean bSetUnitsAndExtents) Same as setProjection(), but takes an OGC WKT projection definition string as input.

Note: setWKTProjection requires GDAL support

int getMetaData(string name) Fetch metadata entry by name (stored in the *WEB* object in the map file). Returns "" if no entry matches the name.

Note: getMetaData's query is case sensitive.

int setMetaData(string name, string value) Set a metadata entry for the map (stored in the WEB object in the map file). Returns MS_SUCCESS/MS_FAILURE.

int removeMetaData(string name) Remove a metadata entry for the map (stored in the WEB object in the map file). Returns MS_SUCCESS/MS_FAILURE.

array getLayersIndexByGroup(string groupname) Return an array containing all the layer's indexes given a group name.

array getAllGroupNames() Return an array containing all the group names used in the layers.

array getAllLayerNames() Return an array containing all the layer names.

boolean moveLayerUp(int layerindex) Move layer up in the hierarchy of drawing.

boolean moveLayerDown(int layerindex) Move layer down in the hierarchy of drawing.

array getlayersdrawingorder() Return an array containing layer's index in the order which they are drawn.

boolean setlayersdrawingorder(array layerindex) Set the layer's order array. The argument passed must be a valid array with all the layer's index. Return TRUE on success or else FALSE.

string processtemplate(array params, boolean generateimages) Process the template file specified in the web object and return the result in a buffer. The processing consists of opening the template file and replace all the tags found in it. Only tags that have an equivalent element in the map object are replaced (ex [scaledenom]). The are two exceptions to the previous statement :

- [img], [scalebar], [ref], [legend] would be replaced with the appropriate url if the parameter generateimages is set to MS_TRUE. (Note : the images corresponding to the different objects are generated if the object is set to MS_ON in the map file)
- the user can use the params parameter to specify tags and their values. For example if the user have a specific tag call [my_tag] and would like it to be replaced by "value_of_my_tag" he would do

```
$tmparray["my_tag"] = "value_of_my_tag";  
$map->processtemplate($tmparray, MS_FALSE);
```

string processquerytemplate(array params, boolean generateimages) Process query template files and return the result in a buffer. Second argument generateimages is not mandatory. If not given it will be set to TRUE.

See Also:

processtemplate

string processlegendtemplate(array params) Process legend template files and return the result in a buffer.

See Also:

processtemplate

int setSymbolSet(string fileName) Load and set a symbol file dynamically.

int getNumSymbols() Return the number of symbols in map.

int setFontSet(string fileName) Load and set a new *FONTSET*.

int selectOutputFormat(string type) Selects the output format to be used in the map. Returns MS_SUCCESS/MS_FAILURE.

Note: the type used should correspond to one of the output formats declared in the map file. The type argument passed is compared with the mimetype parameter in the output format structure and then to the name parameter in the structure.

int saveMapContext(string filename) Available only if WMS support is enabled. Save current map object state in *WMS Map Context* format. Only WMS layers are saved in the WMS Map Context XML file. Returns MS_SUCCESS/MS_FAILURE.

int loadMapContext(string filename [, boolean unique_layer_name]) Available only if WMS support is enabled. Load a *WMS Map Context* XML file into the current mapObj. If the map already contains some layers then the layers defined in the WMS Map context document are added to the current map. The 2nd argument unique_layer_name is optional and if set to MS_TRUE layers created will have a unique name (unique prefix added to the name). If set to MS_FALSE the layer name will be the the same name as in the context. The default value is MS_FALSE. Returns MS_SUCCESS/MS_FAILURE.

int applySLD(string sldxml) Apply the *SLD* document to the map file. The matching between the sld document and the map file will be done using the layer's name. See *SLD HowTo* for more information on the SLD support.

int applySLDURL(string sldurl) Apply the SLD document pointed by the URL to the map file. The matching between the sld document and the map file will be done using the layer's name. See *SLD HowTo* for more information on the SLD support.

string generateSLD() Returns an SLD XML string based on all the classes found in all the layers.

string getconfiguration(string key) Returns the config value associated with the key. Returns an empty sting if key not found.

int setconfiguration(string key, string value) Sets a config parameter using the key and the value passed

int applyconfigoptions() Applies the config options set in the map file. For example setting the PROJ_LIB using the setconfiguration only modifies the value in the map object. applyconfigoptions will actually change the PROJ_LIB value that will be used when dealing with projection.

int loadowsparameters(owsrequest request, string version) Load OWS request parameters (BBOX, LAYERS, &c.) into map. Returns MS_SUCCESS or MS_FAILURE. 2nd argument version is not mandatory. If not given, the version will be set to 1.1.1

int owdispatch(owsrequest request) Processes and executes the passed OpenGIS Web Services request on the map. Returns MS_DONE (2) if there is no valid OWS request in the req object, MS_SUCCESS (0) if an OWS request was successfully processed and MS_FAILURE (1) if an OWS request was not successfully processed. OWS requests include *WMS*, *WFS*, *WCS* and *SOS* requests supported by MapServer. Results of a dispatched request are written to stdout and can be captured using the msIO services (ie. ms_ioinstallstdouttobuffer() and ms_iogetstdoutbufferstring())

int insertLayer(layerObj layer [, int nIndex=-1]) Insert a copy of *layer* into the Map at index *nIndex*. The default value of *nIndex* is -1, which means the last possible index. Returns the index of the new Layer, or -1 in the case of a failure.

layerObj removeLayer(int nIndex) Remove a layer from the mapObj. The argument is the index of the layer to be removed. Returns the removed layerObj on success, else null.

6.3.7 LayerObj Class

Constructor

Layer Objects can be returned by the MapObj class, or can be created using:

```
layerObj ms_newLayerObj(MapObj map [, layerObj layer])
```

A second optional argument can be given to ms_newLayerObj() to create the new layer as a copy of an existing layer. If a layer is given as argument then all members of a this layer will be copied in the new layer created.

Members

Type	Name
int	numclasses (read-only)
int	index (read-only)
int	status (MS_ON, MS_OFF, MS_DEFAULT or MS_DELETE)
int	debug
string	classitem
string	classgroup
string	name
string	group
string	data
int	type
int	dump
double	tolerance
int	toleranceunits
int	sizeunits
double	symbolscaledenom
double	minscaldenom
double	maxscaldenom
double	labelminscaldenom
double	labelmaxscaldenom
double	symbolscale (Deprecated in v5.0, use symbolscaledenom instead)
double	minscale (Deprecated in v5.0, use minscaldenom instead)
double	maxscale (Deprecated in v5.0, use maxscaldenom instead)
double	labelminscale (Deprecated in v5.0, use labelminscaldenom instead)
double	labelmaxscale (Deprecated in v5.0, use labelmaxscaldenom instead)
int	maxfeatures
colorObj	offsite
int	annotate
int	transform
int	labelcache
int	postlabelcache
string	labelitem
string	labelsizeitem
string	labelangleitem
string	tileitem
string	tileindex
string	header
string	footer
string	connection

Continued on next page

Table 6.2 – continued from previous page

int	connectiontype (read-only, use setConnectionType() to set it)
string	filteritem
string	template
int	opacity
int	transparency (Deprecated in v5.0. Use opacity instead.)
string	styleitem
gridObj	grid //only available on a layer defined as grid (MS_GRATICULE)
int	num_processing
string	requires
string	labelrequires
hashTableObj	metadata
projectionObj	projection

Methods

int updateFromString(string snippet) Update a layer from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

```
$oLayer->updateFromString('LAYER NAME land_fn2 END'); /*modify the name */
$oLayer->updateFromString('LAYER CLASS STYLE COLOR 255 255 0 END END END'); /*add a new class*/
```

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

int draw(imageObj image) Draw a single layer, add labels to cache if required. Returns -1 on error.

int drawQuery(imageObj image) Draw query map for a single layer.

classObj getClass(int classIndex) Returns a classObj from the layer given an index value (0=first class)

int queryByPoint(pointObj point, int mode, double buffer) Query layer at point location specified in georeferenced map coordinates (i.e. not pixels). The query is performed on all the shapes that are part of a CLASS that contains a TEMPLATE value or that match any class in a layer that contains a LAYER TEMPLATE value. Mode is MS_SINGLE or MS_MULTIPLE depending on number of results you want. Passing buffer -1 defaults to tolerances set in the map file (in pixels) but you can use a constant buffer (specified in ground units) instead. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int queryByRect(rectObj rect) Query layer using a rectangle specified in georeferenced map coordinates (i.e. not pixels). The query is performed on all the shapes that are part of a CLASS that contains a TEMPLATE value or that match any class in a layer that contains a LAYER TEMPLATE value. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int queryByShape(shapeObj shape) Query layer based on a single shape, the shape has to be a polygon at this point. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int queryByFeatures(int slayer) Perform a query set based on a previous set of results from another layer. At present the results MUST be based on a polygon layer. Returns MS_SUCCESS if shapes were found or MS_FAILURE if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int queryByAttributes(string qitem, string qstring, int mode) Query layer for shapes that intersect current map extents. qitem is the item (attribute) on which the query is performed, and qstring is the expression to match. The

query is performed on all the shapes that are part of a *CLASS* that contains a *TEMPLATE* value or that match any class in a layer that contains a *LAYER TEMPLATE* value. Note that the layer's *FILTER/FILTERITEM* are ignored by this function. Mode is *MS_SINGLE* or *MS_MULTIPLE* depending on number of results you want. Returns *MS_SUCCESS* if shapes were found or *MS_FAILURE* if nothing was found or if some other error happened (note that the error message in case nothing was found can be avoided in PHP using the '@' control operator).

int setFilter(string expression) Set layer filter *expression*.

string getFilterString() Returns the *expression* for this layer.

string getFilter() Deprecated in v5.0, use `getFilterString()` instead.

string getProjection() Returns a string representation of the *projection*. If no projection is set, *MS_FALSE* is returned.

int setProjection(string proj_params) Set layer *projection* and coordinate system. Parameters are given as a single string of comma-delimited PROJ.4 parameters.

int setWKTProjection(string proj_params) Same as `setProjection()`, but takes an OGC WKT projection definition string as input.

Note: `setWKTProjection` requires GDAL support

int getNumResults() Returns the number of results from this layer in the last query.

resultCacheMemberObj getResult(int index) Returns a `resultCacheMemberObj` by index from a layer object with index in the range 0 to `numresults-1`. Returns a valid object or `FALSE(0)` if index is invalid.

int open() Open the layer for use with `getShape()`. Returns *MS_SUCCESS/MS_FAILURE*.

int whichshapes(rectobj) Performs a spatial, and optionally an attribute based feature search. The function basically prepares things so that candidate features can be accessed by query or drawing functions (eg using `nextshape` function). Returns *MS_SUCCESS* or *MS_FAILURE*.

shapeobj nextShape() Called after `msWhichShapes` has been called to actually retrieve shapes within a given area returns a shape object or *MS_FALSE*

```
$map = ms_newmapobj("d:/msapps/gmap-ms40/htdocs/gmap75.map");
$layer = $map->getLayerByName('road');
$status = $layer->open();
$status = $layer->whichShapes($map->extent);
while ($shape = $layer->nextShape())
{
    echo $shape->index . "<br>\n";
}
$layer->close();
```

void close() Close layer previously opened with `open()`.

shapeObj getFeature(int shapeindex [, int tileindex = -1]) Retrieve `shapeObj` from a layer by index. `Tileindex` is optional and is used only for tiled shapefiles (you get it from the `resultCacheMemberObj` returned by `getResult()` for instance). Simply omit or pass `tileindex = -1` for other data sources.

shapeObj getShape(int tileindex, int shapeindex) Deprecated in v5.0, use `getFeature()` instead (note that the order of the arguments is reversed since `tileindex` is optional in `getFeature()`)

rectObj getExtent() Returns the layer's data extents. If the layer's *EXTENT* member is set then this value is used, otherwise this call opens/closes the layer to read the extents. This is quick on shapefiles, but can be an expensive operation on some file formats or data sources. This function is safe to use on both opened or closed layers: it is not necessary to call `open()/close()` before/after calling it.

int addFeature(shapeObj shape) Add a new feature in a layer. Returns -1 on error.

int getMetaData(string name) Fetch layer metadata entry by name. Returns "" if no entry matches the name. Note that the search is case sensitive.

Note: getMetaData's query is case sensitive.

int setMetaData(string name, string value) Set a metadata entry for the layer. Returns MS_SUCCESS/MS_FAILURE.

int removeMetaData(string name) Remove a metadata entry for the layer. Returns MS_SUCCESS/MS_FAILURE.

string getWMSFeatureInfoURL(int clickX, int clickY, int featureCount, string infoFormat) Return a WMS GetFeatureInfo URL (works only for WMS layers) clickX, clickY is the location of to query in pixel coordinates with (0,0) at the top left of the image. featureCount is the number of results to return. infoFormat is the format the format in which the result should be requested. Depends on remote server's capabilities. MapServer WMS servers support only "MIME" (and should support "GML.1" soon). Returns "" and outputs a warning if layer is not a WMS layer or if it is not queriable.

aString getItems() return a list of items. Must call open function first.

boolean setProcessing(string) Add the string to the processing string list for the layer. The layer->num_processing is incremented by 1.

```
$oLayer->setprocessing("SCALE_1=AUTO");
$oLayer->setprocessing("SCALE_2=AUTO");
```

aString getProcessing() Returns an array containing the processing strings

boolean clearProcessing() Clears all the processing strings

string executeWFSGetfeature() Executes a GetFeature request on a WFS layer and returns the name of the temporary GML file created. Returns an empty string on error.

int applySLD(string sldxml, string namedlayer) Apply the *SLD* document to the layer object. The matching between the sld document and the layer will be done using the layer's name. If a namedlayer argument is passed (argument is optional), the NamedLayer in the sld that matches it will be used to style the layer. See *SLD HowTo* for more information on the SLD support.

int applySLDURL(string sldurl, string namedlayer) Apply the *SLD* document pointed by the URL to the layer object. The matching between the sld document and the layer will be done using the layer's name. If a namedlayer argument is passed (argument is optional), the NamedLayer in the sld that matches it will be used to style the layer. See *SLD HowTo* for more information on the SLD support.

string generateSLD() Returns an SLD XML string based on all the classes found in the layers.

int moveclassup(int index) The class specified by the class index will be moved up into the array of layers. Returns MS_SUCCESS or MS_FAILURE. ex layer->moveclassup(1) will have the effect of moving class 1 up to position 0, and the class at position 0 will be moved to position 1.

int moveclassdown(int index) The class specified by the class index will be moved down into the array of layers. Returns MS_SUCCESS or MS_FAILURE. ex layer->moveclassdown(0) will have the effect of moving class 0 up to position 1, and the class at position 1 will be moved to position 0.

classObj removeClass(int index) Removes the class indicated and returns a copy, or NULL in the case of a failure. Note that subsequent classes will be renumbered by this operation. The numclasses field contains the number of classes available.

boolean isVisible() Returns MS_TRUE/MS_FALSE depending on whether the layer is currently visible in the map (i.e. turned on, in scale, etc.).

int setConenctionType(int connectiontype [,string plugin_library]) Changes the connectiontype of the layer and recreates the vtable according to the new connection type. This method should be used instead of setting the connectiontype parameter directly. In case when the layer.connectiontype = MS_PLUGIN the plugin_library

parameter should also be specified so as to select the library to load by mapserver. For the other connection types this parameter is not used.

6.3.8 ClassObj Class

Constructor

Class Objects can be returned by the LayerObj class, or can be created using:

```
classObj ms_newClassObj(layerObj layer [, classObj class])
```

The second argument class is optional. If given, the new class created will be a copy of this class.

Members

Type	Name
string	name
string	title
int	type
int	status (MS_ON, MS_OFF or MS_DELETE)
double	minscaledenom
double	maxscaledenom
double	minscale (Deprecated in v5.0, use minscaledenom instead)
double	maxscale (Deprecated in v5.0, use maxscaledenom instead)
string	template
labelObj	label
int	numstyles
string	keyimage
string	group
hashTableObj	metadata

Methods

int updateFromString(string snippet) Update a class from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

```
$oClass->updateFromString('CLASS STYLE COLOR 255 0 255 END END'); /*set the color */
```

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

int setExpression(string expression) Set the *expression* string for the class object.

string getExpressionString() Returns the *expression* string for the class object.

string getExpression() Deprecated in v5.0. Use getExpressionString() instead.

int setText(string text) Set the text string for the class object.

string getTextString() Returns the text string for the class object.

int drawLegendIcon(int width, int height, imageObj im, int dstX, int dstY) Draw the legend icon on im object at dstX, dstY. Returns MS_SUCCESS/MS_FAILURE.

imageObj createLegendIcon(int width, int height) Draw the legend icon and return a new imageObj.

styleObj getStyle(int index) Return the style object using an index. `index >= 0 && index < class->numstyles`.

classObj clone() Returns a cloned copy of the class.

int movestyleup(int index) The style specified by the style index will be moved up into the array of classes. Returns `MS_SUCCESS` or `MS_FAILURE`. ex `class->movestyleup(1)` will have the effect of moving style 1 up to position 0, and the style at position 0 will be moved to position 1.

int movestyledown(int index) The style specified by the style index will be moved down into the array of classes. Returns `MS_SUCCESS` or `MS_FAILURE`. ex `class->movestyledown(0)` will have the effect of moving style 0 up to position 1, and the style at position 1 will be moved to position 0.

int deletestyle(int index) Delete the style specified by the style index. If there are any style that follow the deleted style, their index will decrease by 1.

Note: if you are using the `numstyles` parameter while using the `deletestyle` function on the class object you need to refetch a new class object.

Example :

```
//class has 2 styles
$class = $oLayer->getclass(0);
$class->deletestyle(1);
echo $class->numstyles; : will echo 2

$class = $oLayer->getclass(0);
echo $class->numstyles; : will echo 1
```

int getMetaData(string name) Fetch class metadata entry by name. Returns "" if no entry matches the name. Note that the search is case sensitive.

Note: `getMetaData`'s query is case sensitive.

int setMetaData(string name, string value) Set a metadata entry for the class. Returns `MS_SUCCESS/MS_FAILURE`.

int removeMetaData(string name) Remove a metadata entry for the class. Returns `MS_SUCCESS/MS_FAILURE`.

6.3.9 ImageObj Class

Constructor

Instances of `ImageObj` are always created by the map class methods.

Members

Type	Name
int	width (read-only)
int	height (read-only)
string	imagepath
string	imageurl

Methods

void free() Destroys resources used by an image object.

int saveImage(string filename, MapObj oMap) Writes image object to specified filename. Passing an empty filename sends output to stdout. In this case, the PHP header() function should be used to set the document's content-type prior to calling saveImage(). The output format is the one that is currently selected in the map file. The second argument oMap is not mandatory. It is useful when saving to formats like GTIFF that needs georeference informations contained in the map file. The function returns -1 on error. On success, it returns either 0 if writing to an external file, or the number of bytes written if output is sent to stdout.

string saveWebImage() Writes image to temp directory. Returns image URL. The output format is the one that is currently selected in the map file.

void pasteImage(imageObj srcImg, int transparentColorHex [[, int dstX, int dstY], int angle] Copy srcImg on top of the current imageObj. transparentColorHex is the color (in 0xrrggbb format) from srcImg that should be considered transparent (i.e. those pixels won't be copied). Pass -1 if you don't want any transparent color. If optional dstx,dsty are provided then it defines the position where the image should be copied (dstx,dsty = top-left corner position). The optional angle is a value between 0 and 360 degrees to rotate the source image counterclockwise. Note that if an angle is specified (even if its value is zero) then the dstx and dsty coordinates specify the CENTER of the destination area. Note: this function works only with 8 bits GD images (PNG or GIF).

6.3.10 LabelObj Class

Constructor

LabelObj are always embedded inside other classes.

Members

Type	Name
string	font
int	type
colorObj	color
colorObj	outlinecolor
int	outlinewidth
colorObj	shadowcolor
int	shadowsizeX
int	shadowsizeY
colorObj	backgroundcolor
colorObj	backgroundshadowcolor
int	backgroundshadowsizeX
int	backgroundshadowsizeY
int	size
int	minsize
int	maxsize
int	position
int	offsetX
int	offsetY
double	angle
int	autoangle
int	buffer
int	antialias
int	wrap
int	minfeaturesize

Continued on next page

Table 6.3 – continued from previous page

int	autominfeaturesize
int	mindistance
int	partials
int	force
string	encoding
int	align
int	maxlength
int	minlength
int	priority

Methods

int updateFromString(string snippet) Update a label from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

int setBinding(const labelbinding, string value) Set the attribute binding for a specified label property. Returns true on success.

Example:

```
$oLabel->setbinding(MS_LABEL_BINDING_COLOR, "FIELD_NAME_COLOR");
```

This would bind the color parameter with the data (ie will extract the value of the color from the field called "FIELD_NAME_COLOR")

string getBinding(const labelbinding) Get the attribute binding for a specified label property. Returns null if there is no binding for this property.

Example:

```
$oLabel->setbinding(MS_LABEL_BINDING_COLOR, "FIELD_NAME_COLOR");
echo $oLabel->getbinding(MS_LABEL_BINDING_COLOR); // FIELD_NAME_COLOR
```

int removeBinding(const labelbinding) Remove the attribute binding for a specified style property. Returns true on success.

Example:

```
$oStyle->removebinding(MS_LABEL_BINDING_COLOR);
```

6.3.11 webObj Class

Constructor

Instances of webObj are always are always embedded inside the mapObj.

Members

Type	Name
string	log
string	imagepath
string	template
string	imageurl
string	header
string	footer
string	empty (read-only)
string	error (read-only)
string	mintemplate
string	maxtemplate
double	minscaledenom
double	maxscaledenom
double	minscale (Deprecated in v5.0, use minscaledenom instead)
double	maxscale (Deprecated in v5.0, use maxscaledenom instead)
rectObj	extent (read-only)
string	queryformat
string	legendformat
string	browseformat
hashTableObj	metadata

Methods

int updateFromString(string snippet) Update a web object from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

6.3.12 referenceMapObj Class

Constructor

Instances of referenceMapObj are always embedded inside the mapObj.

Members

Type	Name
string	image
int	width
int	height
int	status
rectObj	extent
ColorObj	color
ColorObj	outlinecolor

Methods

int updateFromString(string snippet) Update a referenceMap object from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

6.3.13 ColorObj Class

Constructor

Instances of ColorObj are always embedded inside other classes.

Members

Type	Name
int	red
int	green
int	blue

Methods

void setRGB(int red, int green, int blue) Set red, green, blue values.

6.3.14 PointObj Class

Constructor

PointObj ms_newPointObj()

Members

Type	Name
double	x
double	y
double	z (used for 3d shape files. set to 0 for other types)
double	m (used only for measured shape files. set to 0 for other types.)

Methods

int setXY(double x, double y [, double m]) Set X,Y coordinate values. Returns 0 on success, -1 on error.

Note: the 3rd parameter m is used for measured shape files only. It is not mandatory.

int setXYZ(double x, double y , double z, [, double m]) Set X,Y,Z coordinate values. Returns 0 on success, -1 on error.

Note: the 4th parameter m is used for measured shape files only. It is not mandatory.

int draw(mapObj map, layerObj layer, imageObj img, int class_index, string text) Draws the individual point using layer. The class_index is used to classify the point based on the classes defined for the layer. The text string is used to annotate the point. Returns MS_SUCCESS/MS_FAILURE.

double distanceToPoint(pointObj poPoint) Calculates distance between two points.

double distanceToLine(pointObj p1, pointObj p2) Calculates distance between a point and a line defined by the two points passed in argument.

double distanceToShape(shapeObj shape) Calculates the minimum distance between a point and a shape.

int project(projectionObj in, projectionObj out) Project the point from “in” projection (1st argument) to “out” projection (2nd argument). Returns MS_SUCCESS/MS_FAILURE.

void free() Releases all resources used by the object.

6.3.15 LineObj Class

Constructor

```
LineObj ms_newLineObj()
```

Members

Type	Name
int	numpoints (read-only)

Methods

int add(pointObj point) Add a point to the end of line.

int addXY(double x, double y [, double m]) Add a point to the end of line. It is not mandatory.

Note: the 3rd parameter m is used for measured shape files only. It is not mandatory.

int addXYZ(double x, double y, double z [, double m]) Add a point to the end of line.

Note: the 4th parameter m is used for measured shape files only. It is not mandatory.

PointObj point(int i) Returns a reference to point number i. Reference is valid only during the life of the lineObj that contains the point.

int project(projectionObj in, projectionObj out) Project the line from “in” projection (1st argument) to “out” projection (2nd argument). Returns MS_SUCCESS/MS_FAILURE.

void free() Destroys resources used by a line object.

6.3.16 ShapeObj Class

Constructor

```
ShapeObj ms_newShapeObj(int type)
```

‘type’ is one of MS_SHAPE_POINT, MS_SHAPE_LINE, MS_SHAPE_POLYGON or MS_SHAPE_NULL

```
ShapeObj ms_shapeObjFromWkt(string wkt)
```

Creates new shape object from WKT string.

Members

Type	Name
string	text
int	classindex
int	type (read-only)
int	numlines (read-only)
int	index
int	tileindex (read-only)
rectObj	bounds (read-only)
int	numvalues (read-only)
array	values (read-only)

The values array is an associative array with the attribute values for this shape. It is set only on shapes obtained from `layer->getShape()`. The key to the values in the array is the attribute name, e.g.

```
$population = $shape->values["Population"];
```

Methods

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

int add(lineObj line) Add a line (i.e. a part) to the shape.

LineObj line(int i) Returns a reference to line number i. Reference is valid only during the life of the shapeObj that contains the point.

int draw(mapObj map, layerObj layer, imageObj img) Draws the individual shape using layer. Returns MS_SUCCESS/MS_FAILURE.

boolean contains(pointObj point) Returns MS_TRUE if the point is inside the shape, MS_FALSE otherwise.

boolean intersects(shapeObj shape) Returns MS_TRUE if the two shapes intersect, MS_FALSE otherwise.

int project(projectionObj in, projectionObj out) Project the shape from “in” projection (1st argument) to “out” projection (2nd argument). Returns MS_SUCCESS/MS_FAILURE.

pointObj getpointusingmeasure(double m) Apply only on Measured shape files. Given a measure m, return the corresponding XY location on the shapeobject.

pointObj getmeasureusingpoint(pointObject point) Apply only on Measured shape files. Given an XY Location, find the nearest point on the shape object. Return a point object of this point with the m value set.

void free() Destroys resources used by a shape object.

string getvalue(layerObj layer, string filename) Returns the value for a given field name.

shapeobj buffer(width) Returns a new buffered shapeObj based on the supplied distance (given in the coordinates of the existing shapeObj). Only available if php/mapsript is built with GEOS library.

shapeobj convexhull() Returns a shape object representing the convex hull of shape. Only available if php/mapsript is built with GEOS library.

shapeobj boundary() Returns the boundary of the shape. Only available if php/mapsript is built with GEOS library.

int containsShape(shapeobj shape2) Returns true if shape2 passed as argument is entirely within the shape. Else return false. Only available if php/mapsript is built with GEOS library.

shapeobj union(shapeobj shape) Returns a shape object representing the union of the shape object with the one passed as parameter. Only available if php/mapsript is built with GEOS library

shapeobj union_geos(shapeobj shape) [(Deprecated in v5.2. Use union instead)] Returns a shape object representing the union of the shape object with the one passed as parameter. Only available if php/mapsript is built with GEOS library.

shapeobj intersection(shapeobj shape) Returns a shape object representing the intersection of the shape object with the one passed as parameter. Only available if php/mapsript is built with GEOS library.

shapeobj difference(shapeobj shape) Returns a shape object representing the difference of the shape object with the one passed as parameter. Only available if php/mapsript is built with GEOS library.

shapeobj symdifference(shapeobj shape) Returns the computed symmetric difference of the supplied and existing shape. Only available if php/mapsript is built with GEOS library.

int overlaps(shapeobj shape) Returns true if the shape passed as argument overlaps the shape. Else returns false. Only available if php/mapsript is built with GEOS library.

int within(shapeobj shape2) Returns true if the shape is entirely within the shape2 passed as argument. Else returns false. Only available if php/mapsript is built with GEOS library.

int crosses(shapeobj shape) Returns true if the shape passed as argument crosses the shape. Else return false. Only available if php/mapsript is built with GEOS library.

int touches(shapeobj shape) Returns true if the shape passed as argument touches the shape. Else return false. Only available if php/mapsript is built with GEOS library.

int equals(shapeobj shape) Returns true if the shape passed as argument is equal to the shape (geometry only). Else return false. Only available if php/mapsript is built with GEOS library.

int disjoint(shapeobj shape) Returns true if the shape passed as argument is disjoint to the shape. Else return false. Only available if php/mapsript is built with GEOS library.

pointObj getCentroid() Returns a point object representing the centroid of the shape. Only available if php/mapsript is built with GEOS library.

double getArea() Returns the area of the shape (if applicable). Only available if php/mapsript is built with GEOS library.

double getLength() Returns the length (or perimeter) of the shape. Only available if php/mapsript is built with GEOS library.

pointObj getLabelPoint() Returns a point object with coordinates suitable for labelling the shape.

string toWkt() Returns WKT representation of the shape's geometry.

int setBounds() Updates the bounds property of the shape. Must be called to calculate new bounding box after new parts have been added. Returns true if successful, else return false.

6.3.17 RectObj Class

Constructor

RectObj are sometimes embedded inside other objects. New ones can also be created with:

```
RectObj ms_newRectObj()
```

Note: the members (minx, miny,maxx,maxy) are initialized to -1;

Members:

Type	Name
double	minx
double	miny
double	maxx
double	maxy

Methods

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

void setextent(double minx, double miny, double maxx, double maxy) Set the rectangle extents.

int draw(mapObj map, layerObj layer, imageObj img, int class_index, string text) Draws the individual rectangle using layer. The class_index is used to classify the rectangle based on the classes defined for the layer. The text string is used to annotate the rectangle. Returns MS_SUCCESS/MS_FAILURE.

double fit(int width, int height) Adjust extents of the rectangle to fit the width/height specified.

int project(projectionObj in, projectionObj out) Project the rectangle from “in” projection (1st argument) to “out” projection (2nd argument). Returns MS_SUCCESS/MS_FAILURE.

void free() Destroys resources used by a rect object.

6.3.18 ShapefileObj Class**Constructor**

```
shapefileObj ms_newShapefileObj(string filename, int type)
```

Opens a shapefile and returns a new object to deal with it. Filename should be passed with no extension. To create a new file (or overwrite an existing one), type should be one of MS_SHP_POINT, MS_SHP_ARC, MS_SHP_POLYGON or MS_SHP_MULTIPPOINT. Pass type as -1 to open an existing file for read-only access, and type=-2 to open an existing file for update (append).

Members

Type	Name
int	numshapes (read-only)
int	type (read-only)
string	source (read-only)
rectObj	bounds (read-only)

Methods

shapeObj getShape(int i) Retrieve shape by index.

shapeObj getPoint(int i) Retrieve point by index.

shapeObj getTransformed(mapObj map, int i) Retrieve shape by index.

rectObj getExtent(int i) Retrieve a shape’s bounding box by index.

int addShape(shapeObj shape) Appends a shape to an open shapefile.

int addPoint(pointObj point) Appends a point to an open shapefile.

void free() Closes a shape file (and commits all changes in write mode) and releases all resources used by the object.

6.3.19 ResultCacheMemberObj Class

Constructor

Instances of ResultCacheMemberObj are always obtained through layerObj's getResult() method.

Members

Type	Name
int	shapeindex (read-only)
int	tileindex (read-only)
int	classindex (read-only)

6.3.20 ProjectionObj Class

Constructor

```
ProjectionObjObj ms_newProjectionObj(string projectionstring)
```

Creates a projection object based on the projection string passed as argument.

```
$projInObj = ms_newprojectionobj("proj=latlong")
```

will create a geographic projection class.

The following example will convert a lat/long point to an LCC projection:

```
$projInObj = ms_newprojectionobj("proj=latlong");
$projOutObj = ms_newprojectionobj("proj=lcc,ellps=GRS80,lat_0=49, ".
                                "lon_0=-95,lat_1=49,lat_2=77");

$poPoint = ms_newpointobj();
$poPoint->setXY(-92.0, 62.0);
$poPoint->project($projInObj, $projOutObj);
```

Methods

int getUnits() Returns the units of a projection object. Returns -1 on error.

6.3.21 ScalebarObj Class

Constructor

Instances of scalebarObj are always are always embedded inside the mapObj.

Members

Type	Name
int	height
int	width
int	style
int	intervals
colorObj	color
colorObj	backgroundcolor
colorObj	outlinecolor
int	units
int	status //MS_ON, MS_OFF, MS_EMBED
int	position //for embeded scalebars, MS_UL, MS_UC, ...
int	transparent
int	interlace
int	postlabelcache
labelObj	label
colorObj	imagecolor
int	align

Methods

int updateFromString(string snippet) Update a scalebar from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

int setimagecolor(int red, int green, int blue) Sets the imagecolor property (background) of the object. Returns false on error.

6.3.22 LegendObj Class

Constructor

Instances of legendObj are always are always embedded inside the mapObj.

Members

Type	Name
int	height
int	width
int	keysizeX
int	keysizeY
int	keyspacingX
int	keyspacingY
colorObj	outlinecolor //Color of outline of box, -1 for no outline
int	status //MS_ON, MS_OFF, MS_EMBED
int	position //for embeded legends, MS_UL, MS_UC, ...
int	transparent
int	interlace
int	postlabelcache //MS_TRUE, MS_FALSE
labelObj	label
colorObj	imagecolor
string	template

Methods

int updateFromString(string snippet) Update a legend from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

6.3.23 QuerymapObj Class

Constructor

Instances of querymapObj are always are always embedded inside the mapObj.

Members

Type	Name
int	width
int	height
int	style (MS_NORMAL, MS_HILITE, MS_SELECTED)
colorObj	color

Methods

int updateFromString(string snippet) Update a queryMap object from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

6.3.24 StyleObj Class

Constructor

Instances of styleObj are always embedded inside the classObj.

```
styleObj ms_newStyleObj(classObj class [, styleObj style])
```

The second argument 'style' is optional. If given, the new style created will be a copy of the style passed as argument.

Members

Type	Name
int	symbol
string	symbolname
double	size
double	minsize
double	maxsize
int	offsetx
int	offsety
int	antialias
colorObj	color
colorObj	backgroundcolor
colorObj	outlinecolor
double	width
double	minwidth
double	maxwidth
double	angle
string	angleitem
string	sizeitem
double	minvalue
double	maxvalue
string	rangeitem
int	opacity (this parameter is only supported for the AGG driver)

Methods

int updateFromString(string snippet) Update a style from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

int set(string property_name, new_value) Set object property to a new value. Returns -1 on error.

styleObj clone() Returns a cloned copy of the style.

int setBinding(const stylebinding, string value) Set the attribute binding for a specified style property. Returns true on success. Added in MapServer 5.0.

```
$oStyle->setbinding(MS_STYLE_BINDING_COLOR, "FIELD_NAME_COLOR");
```

This would bind the color parameter with the data (ie will extract the value of the color from the field called "FIELD_NAME_COLOR")

string getBinding(const stylebinding) Get the attribute binding for a specified style property. Returns null if there is no binding for this property.

```
$oStyle->setbinding(MS_STYLE_BINDING_COLOR, "FIELD_NAME_COLOR");  
echo $oStyle->getbinding(MS_STYLE_BINDING_COLOR); // FIELD_NAME_COLOR
```

int removeBinding(const stylebinding) Remove the attribute binding for a specified style property. Returns true on success. Added in MapServer 5.0.

```
$oStyle->removebinding(MS_STYLE_BINDING_COLOR);
```

6.3.25 OutputformatObj Class

Constructor

Instance of outputformatObj is always embedded inside the mapObj. It uses a read only.

No constructor available (coming soon, see bug 979)

Members

Type	Name
string	name
string	mimetype
string	driver
string	extension
int	renderer
int	imagemode // MS_IMAGEMODE_* value.
int	transparent

Methods

int set(string property_name, new_value) Set object property to a new value.

void setOption(string property_name, string new_value) Add or Modify the format option list. return true on success.

```
$oMap->outputformat->setOption("OUTPUT_TYPE", "RASTER");
```

string getOption(string property_name) Returns the associated value for the format option property passed as argument. Returns an empty string if property not found.

void setFormatOption(string property_name, string new_value) Deprecated. See setOption().

string getFormatOption(string property_name) Deprecated. See getOption().

6.3.26 GridObj Class

Constructor

The grid is always embedded inside a layer object defined as a grid (layer->connectiontype = MS_GRATICULE) (for more docs : <http://trac.osgeo.org/mapserver/wiki/MapServerGrid>)

A layer can become a grid layer by adding a grid object to it using : `ms_newGridObj(layerObj layer)`


```

$oLayer = ms_newlayerobj($oMap);
$oLayer->set("name", "GRID");
ms_newgridobj($oLayer);
$oLayer->grid->set("labelformat", "DDMMSS");

```

Members

Type	Name
double	minsubdivide
double	maxsubdivide
double	minarcs
double	maxacrs
double	mininterval
double	maxinterval
string	labelformat

Methods

int set(string property_name, new_value) Set object property to a new value.

6.3.27 ErrorObj Class

Instances of errorObj are created internally by MapServer as errors happen. Errors are managed as a chained list with the first item being the most recent error. The head of the list can be fetched using `ms_GetErrorObj()`, and the list can be cleared using `ms_ResetErrorList()`

Functions

errorObj ms_GetErrorObj() Returns a reference to the head of the list of errorObj.

void ms_ResetErrorList() Clear the current error list. Note that clearing the list invalidates any errorObj handles obtained via the `$error->next()` method.

Members

Type	Name
int	code //See error code constants above
string	routine
string	message

Method

errorObj next() Returns the next errorObj in the list, or NULL if we reached the end of the list.

Example: This example draws a map and reports all errors generated during the `draw()` call, errors can potentially come from multiple layers.

```
ms_ResetErrorList();
$img = $map->draw();
$error = ms_GetErrorObj();
while($error && $error->code != MS_NOERR)
{
    printf("Error in %s: %s<br>\n", $error->routine, $error->message);
    $error = $error->next();
}
```

6.3.28 LabelcacheObj Class

Accessible only through the map object (`map->labelcache`). This object is only used to give the possibility to free the label cache (`map->labelcache->free()`)

Method

boolean free() Free the label cache. Returns true on success or false if an error occurs. Ex : (`map->labelcache->free()`);

6.3.29 SymbolObj Class

Constructor

```
symbolid = ms_newSymbolObj(mapObj map, string symbolname);
```

Creates a new symbol with default values in the symbolist. Returns the Id of the new symbol. If a symbol with the same name exists, It's id will be returned.

To get a symbol object, you need to use a method on the map object:

```
$oSymbol = $map->getSymbolObjectById($nId);
```

Members

Type	Name
string	name
type	name //Please refer to symbol type constants
int	inmapfile If set to TRUE, the symbol will be saved inside the mapfile.
double	size _x
double	size _y
int	numpoints (Read-Only)
int	filled
int	patternlength (Read-Only)
int	stylelength (Deprecated in v5.0, use patternlength instead.) (Read-Only)
string	imagepath (Read-Only))
int	transparent
int	transparentcolor
string	character
int	antialias
string	font
int	gap
int	position

Methods

int set(string property_name, new_value) Set object property to a new value.

int setpoints(array double) Set the points of the symbol. Note that the values passed if an array containing the x and y values of the points. Example array[0] = 1 : x value of the first point array[1] = 0 : y values of the first point array[2] = 1 : x value of the 2nd point

int setpattern(array int) Set the pattern of the symbol (used for dash patterns)

array getpointsarray() Returns an array containing the points of the symbol. Refer to setpoints to see how the array should be interpreted.

array getpatternarray() Returns an array containing the pattern.

int setimagepath(char filename) Loads a pixmap symbol specified by the filename. The file should be of either Gif or Png format.

int setstyle(array int) Deprecated in v5.0, will be removed in a future release. Use setpattern() instead.

array getstylearray() Deprecated in v5.0, will be removed in a future release. Use getpatternarray() instead.

Example of usage:

1. create a symbol to be used as a dash line

```
$nId = ms_newsymbolobj($gpoMap, "mydash");
$oSymbol = $gpoMap->getsymbolobjectbyid($nId);
$oSymbol->set("filled", MS_TRUE);
$oSymbol->set("sizex", 1);
$oSymbol->set("sizey", 1);
$oSymbol->set("inmapfile", MS_TRUE);

$aPoints[0] = 1;
$aPoints[1] = 1;
$oSymbol->setpoints($aPoints);
```

```
$aPattern[0] = 10;
$aPattern[1] = 5;
$aPattern[2] = 5;
$aPattern[3] = 10;
$oSymbol->setpattern($aPattern);

$style->set("symbolname", "mydash");
```

1. Create a TrueType symbol

```
$nId = ms_newSymbolObj($gpoMap, "ttfSymbol");
$oSymbol = $gpoMap->getSymbolObjectById($nId);
$oSymbol->set("type", MS_SYMBOL_TRUETYPE);
$oSymbol->set("filled", true);
$oSymbol->set("character", "&#68;");
$oSymbol->set("font", "ttfFontName");
```

6.3.30 OwsrequestObj Class

Constructor

```
request = ms_newOwsrequestObj();
```

Create a new ows request object.

Members

Type	Name
int	numparams (read-only)

```
int type (read-only): MS_GET_REQUEST or MS_POST_REQUEST =====
=====
```

Methods

int loadparams() Initializes the OWSRequest object from the cgi environment variables REQUEST_METHOD, QUERY_STRING and HTTP_COOKIE. Returns the number of name/value pairs collected.

int setparameter(string name, string value) Set a request parameter. For example :

```
$request->setparameter('REQUEST', 'GetMap');
```

string getname(int index) Return the name of the parameter at *index* in the request's array of parameter names.

string getvalue(int index) Return the value of the parameter at *index* in the request's array of parameter values.

string getvaluebyname(string name) Return the value associated with the parameter *name*.

6.3.31 hashTableObj Class

Constructor

Instance of hashTableObj is always embedded inside the LayerObj, WebObj and ClassObj. It is uses a read only.

```
$hashTable = $oLayer->metadata;
$key = null;
while ($key = $hashTable->nextkey($key))
    echo "Key: ".$key." value: ".$hashTable->get($key)."<br/>";
```

Methods

string get(string key) Fetch class metadata entry by name. Returns "" if no entry matches the name. Note that the search is case sensitive.

int set(string key, string value) Set a metadata entry in the hashTable. Returns MS_SUCCESS/MS_FAILURE.

int remove(string key) Remove a metadata entry in the hashTable. Returns MS_SUCCESS/MS_FAILURE.

void clear() Clear all items in the hashTable (To NULL).

string nextkey(string previousKey) Return the next key or first key if previousKey = NULL. Return NULL if no item is in the hashTable or end of hashTable is reached

6.4 Python MapScript Appendix

Author Sean Gillies

Revision \$Revision: 8278 \$

Date \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Contents

- Python MapScript Appendix
 - Introduction
 - Classes
 - Exception Handling

6.4.1 Introduction

The Python MapScript module contains some class extension methods that have not yet been implemented for other languages.

6.4.2 Classes

References to sections below will be added here as the documentation grows.

imageObj

The Python Imaging Library, <http://www.pythonware.com/products/pil/>, is an indispensable tool for image manipulation. The extensions to imageObj are all geared towards better integration of PIL in MapScript applications.

imageObj Methods

imageObj(PyObject arg1, PyObject arg2 [, PyObject arg3]) [imageObj] Create a new instance which is either empty or read from a Python file-like object that refers to a GD format image.

The constructor has 2 different modes. In the blank image mode, arg1 and arg2 should be the desired width and height in pixels, and the optional arg3 should be either an instance of outputFormatObj or a GD driver name as a shortcut to a format. In the image file mode, arg1 should be a filename or a Python file or file-like object. If the file-like object does not have a “seek” attribute (such as a urllib resource handle), then a GD driver name *must* be provided as arg2.

Here’s an example of creating a 320 pixel wide by 240 pixel high JPEG using the constructor’s blank image mode:

```
image = mapscript.imageObj(320, 240, 'GD/JPEG')
```

In image file mode, interesting values of *arg1* to try are instances of *StringIO*:

```
s = StringIO()
pil_image.save(s)      # Save an image manipulated with PIL
ms_image = imageObj(s)
```

Or the file-like object returned from *urlopen*

```
url = urllib.urlopen('http://mapserver.gis.umn.edu/bugs/ant.jpg')
ms_image = imageObj(url, 'GD/JPEG')
```

write([PyObject file]) [void] Write image data to a Python file-like object. Default is stdout.

pointObj

pointObj Methods

__str__() [string] Return a string formatted like

```
{ 'x': %f , 'y': %f }
```

with the coordinate values substituted appropriately. Usage example:

```
>>> p = mapscript.pointObj(1, 1)
>>> str(p)
{ 'x': 1.000000 , 'y': 1.000000 }
```

Note that the return value can be conveniently eval’d into a Python dictionary:

```
>>> p_dict = eval(str(p))
>>> p_dict['x']
1.000000
```

rectObj

rectObj Methods

__contains__(pointObj point) [boolean] Returns True if *point* is inside the rectangle, otherwise returns False.

```

>>> r = mapscript.rectObj(0, 0, 1, 1)
>>> p = mapscript.pointObj(2, 0)      # outside
>>> p in r
False
>>> p not in r
True

```

__str__() [string] Return a string formatted like

```
{ 'minx': %f , 'miny': %f , 'maxx': %f , 'maxy': %f }
```

with the bounding values substituted appropriately. Usage example:

```

>>> r = mapscript.rectObj(0, 0, 1, 1)
>>> str(r)
{ 'minx': 0.000000 , 'miny': 0.000000 , 'maxx': 1.000000 , 'maxy': 1.000000 }

```

Note that the return value can be conveniently eval'd into a Python dictionary:

```

>>> r_dict = eval(str(r))
>>> r_dict['minx']
0.000000

```

6.4.3 Exception Handling

The Python MapScript module maps a few MapServer errors into Python exceptions. Attempting to load a non-existent mapfile raises an 'IOError', for example

```

>>> import mapscript
>>> mapfile = '/no/such/file.map'
>>> m = mapscript.mapObj(mapfile)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/usr/lib/python2.3/site-packages/mapscript.py", line 799, in __init__
    newobj = _mapscript.new_mapObj(*args)
IOError: msLoadMap(): Unable to access file. (/no/such/file.map)
>>>

```

The message of the error is written by 'msSetError' and so is the same message that CGI mapserv users see in error logs.

6.5 Python MapScript Image Generation

Author Sean Gillies

Revision \$Revision: 8295 \$

Date \$Date: 2008-12-26 21:08:04 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2008/07/15

Table of Contents

- Python MapScript Image Generation
 - Introduction
 - Imagery Overview
 - The imageObj Class
 - Image Output
 - Images and Symbols

6.5.1 Introduction

The MapScript HOWTO docs are intended to complement the API reference with examples of usage for specific subjects. All examples in this document refer to the mapfile and testing layers distributed with MapServer 4.2+ and found under `mapserver/tests`.

Pseudocode

All examples will use a pseudocode that is consistent with the language independent API reference. Each line is a statement. For object attributes and methods we use the dot, `.`, operator. Creation and deletion of objects will be indicated by `'new'` and `'del'` keywords. Other than that, the pseudocode looks a lot like Python.

6.5.2 Imagery Overview

The most common use of MapServer and MapScript is to create map imagery using the built-in GD format drivers: GD/GIF, GD/PNG, GD/PNG24, and GD/JPEG. This imagery might be saved to a file on disk or be streamed directly to another device.

6.5.3 The imageObj Class

Imagery is represented in MapScript by the `imageObj` class. Please see the API Reference (`MapScript.txt`) for class attribute and method details.

Creating imageObj from a mapObj

The `mapObj` class has two methods that return instances of `imageObj`: `'draw'`, and `'prepareImage'`. The first returns a full-fledged map image just as one would obtain from the `mapserv` CGI program

```
test_map = MapScript.mapObj('tests/test.map')
map_image = test_map.draw()
```

A properly sized and formatted blank image, without any layers, symbols, or labels, will be generated by `'prepareImage'`

```
blank_image = test_map.prepareImage()
```


Creating a new imageObj

The imageObj class constructor creates new instances without need of a map

```
format = MapScript.outputFormatObj('GD/JPEG')
image = MapScript.imageObj(300, 200, format)    # 300 wide, 200 high JPEG
```

and can even initialize from a file on disk

```
# First three args are overridden by attributes of the disk image file
disk_image = MapScript.imageObj(-1, -1, NULL, 'tests/test.png')
```

6.5.4 Image Output

Creating files on disk

Imagery is saved to disk by using the ‘save’ method. By accessing the ‘extension’ attribute of an image’s format, the proper file extension can be used without making any assumptions

```
filename = 'test.' + map_image.format.extension
map_image.save(filename)
```

If the image is using a GDAL/GTiff-based format, a GeoTIFF file can be created on disk by adding a mapObj as a second optional argument to ‘save’

```
map_image.save(filename, test_map)
```

Direct Output

An image can be dumped to an open filehandle using the ‘write’ method. By default, the filehandle is ‘stdout’

```
# Send an image to a web browser
print "Content-type: " + map_image.format.mimetype + "\n\n"
map_image.write()
```

This method is not fully functional for all SWIG MapScript languages. See the API Reference (MapScript.txt) for details. The ‘write’ method is new in 4.4.

6.5.5 Images and Symbols

The symbolObj::getImage() method will return an instance of imageObj for pixmap symbols

```
symbol = test_map.symbolset.getSymbolByName('home-png')
image = symbol.getImage()
```

There is a symmetric ‘setImage’ method which loads imagery into a symbol, allowing pixmap symbols to be created dynamically

```
new_symbol = MapScript.symbolObj('from_image')
new_symbol.type = MapScript.MS_SYMBOL_PIXMAP
new_symbol.setImage(image)
index = test_map.symbolset.appendSymbol(new_symbol)
```

The get/setImage methods are new in MapServer 4.4.

6.6 Mapfile Manipulation

Author Sean Gillies

Revision \$Revision: 8365 \$

Date \$Date: 2008-12-31 07:49:02 -0800 (Wed, 31 Dec 2008) \$

Contents

- Mapfile Manipulation
 - Introduction
 - Mapfile Overview
 - The mapObj Class
 - Children of mapObj
 - Metadata

6.6.1 Introduction

The MapScript HowTo docs are intended to complement the API reference with examples of usage for specific subjects. All examples in this document refer to the mapfile and testing layers distributed with MapServer 4.2+ and found under `mapserver/tests`.

Pseudocode

All examples will use a pseudocode that is consistent with the language independent API reference. Each line is a statement. For object attributes and methods we use the dot, '.', operator. Creation and deletion of objects will be indicated by 'new' and 'del' keywords. Other than that, the pseudocode looks a lot like Python.

6.6.2 Mapfile Overview

By “Mapfile” here, I mean all the elements that can occur in (nearly) arbitrary numbers within a MapScript mapObj: Layers, Classes, and Styles. MapServer 4.4 has greatly improved capability to manipulate these objects.

6.6.3 The mapObj Class

An instance of mapObj is a parent for zero to many layerObj children.

New instances

The mapfile path argument to the `mapscript.mapObj` constructor is now optional

```
empty_map = new mapscript.mapObj
```

generates a default `mapObj` with no layers. A `mapObj` is initialized from a mapfile on disk in the usual manner:

```
test_map = new mapscript.mapObj('tests/test.map')
```

Cloning

An independent copy, less result and label caches, of a `mapObj` can be produced by the new `mapObj.clone()` method:

```
clone_map = test_map.clone()
```

Note: the Java MapScript module implements a “cloneMap” method to avoid conflict with the clone method of Java’s Object class.

Saving

A `mapObj` can be saved to disk using the save method:

```
clone_map.save('clone.map')
```

Frankly, the `msSaveMap()` function which is the foundation for `mapObj::save` is incomplete. Your mileage may vary.

6.6.4 Children of mapObj

There is a common parent/child object API for Layers, Classes, and Styles in MapServer 4.4.

Referencing a Child

References to Layer, Class, and Style children are obtained by “getChild”-like methods of their parent:

```
layer_i   = test_map.getLayer(i)
class_ij  = layer_i.getClass(j)
style_ijk = class_ij.getStyle(k)
```

These references are for convenience only. MapScript doesn’t have any reference counting, and you are certain to run into trouble if you try to use these references after the parent `mapObj` has been deleted and freed from memory.

Cloning a Child

A completely independent Layer, Class, or Style can be created using the clone method of `layerObj`, `classObj`, and `styleObj`:

```
clone_layer = layer_i.clone()
```

This instance has no parent, and is self-owned.

New Children

Uninitialized instances of `layerObj`, `classObj`, or `styleObj` can be created with the new constructors:

```
new_layer = new mapscript.layerObj
new_class = new mapscript.classObj
new_style = new mapscript.styleObj
```

and are added to a parent object using “insertChild”-like methods of the parent which returns the index at which the child was inserted:

```
li = test_map.insertLayer(new_layer)
ci = test_map.getLayer(li).insertClass(new_class)
si = test_map.getLayer(li).getClass(ci).insertStyle(new_style)
```

The insert* methods create a completely new copy of the object and store it in the parent with all ownership taken on by the parent.

see the API reference for more details.

Backwards Compatibility

The old style child object constructors with the parent object as a single argument:

```
new_layer = new mapscript.layerObj(test_map)
new_class = new mapscript.classObj(new_layer)
new_style = new mapscript.styleObj(new_class)
```

remain in MapServer 4.4.

Removing Children

Child objects can be removed with “removeChild”-like methods of parents, which return independent copies of the removed object:

```
# following from the insertion example ...
# remove the inserted style, returns a copy of the original new_style
removed_style = test_map.getLayer(li).getClass(ci).removeStyle(si)
removed_class = test_map.getLayer(li).removeClass(ci)
removed_layer = test_map.removeLayer(li)
```

6.6.5 Metadata

Map, Layer, and Class metadata are the other arbitrarily numbered elements (well, up to the built-in limit of 41) of a mapfile.

New API

In MapServer 4.4, the metadata attributes of `mapObj.web`, `layerObj`, and `classObj` are instances of `hashTableObj`, a class which functions like a limited dictionary

```
layer.metadata.set('wms_name', 'foo')
name = layer.metadata.get('wms_name') # returns 'foo'
```

You can iterate over all keys in a hashTableObj like

```
key = NULL
while (1):
    key = layer.metadata.nextKey(key)
    if key == NULL:
        break
    value = layer.metadata.get(key)
    ...
```

See the API Reference (mapscript.txt) for more details.

Backwards Compatibility for Metadata

The old getMetaData and setMetaData methods of mapObj, layerObj, and classObj remain for use by older programs.

6.7 Querying

Author Sean Gillies

Revision \$Revision: 8278 \$

Date \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Contents

- Querying
 - Introduction
 - Querying Overview
 - Attribute Queries
 - Spatial Queries

6.7.1 Introduction

All examples in this document refer to the mapfile and testing layers distributed with MapServer 4.2+ and found under mapserver/tests.

Pseudocode

All examples will use a pseudocode that is consistent with the language independent API reference. Each line is a statement. For object attributes and methods we use the dot, '.', operator. Creation and deletion of objects will be indicated by 'new' and 'del' keywords. Other than that, the pseudocode looks a lot like Python.

6.7.2 Querying Overview

The Query Result Set

Map layers can be queried to select features using spatial query methods or the attribute query method. Ignoring for the moment whether we are executing a spatial or attribute query, results are obtained like so:

```
layer.query()                                # not an actual method!
results = layer.getResults()
```

In the case of a failed query or query with zero results, 'getResults' returns NULL.

Result Set Members

Individual members of the query results are obtained like:

```
...                                           # continued

if results:
    for i in range(results.numresults):      # iterate over results
        result = results.getResult(i)
```

This result object is a handle, of sorts, for a feature of the layer, having 'shapeindex' and 'tileindex' attributes that can be used as arguments to 'getFeature'.

Resulting Features

The previous example code can now be extended to the case of obtaining all queried features:

```
layer.query()
results = layer.getResults()
if results:
    # open layer in preparation of reading shapes
    layer.open()

    for i in range(results.numresults):
        result = results.getResult(i)

        layer.getFeature(result.shapeindex, result.tileindex)

        ... # do something with this feature

    # Close when done
    layer.close()
```

Backwards Compatibility

Scripts using the 4.2 API can continue to access query result members through layer methods:

```
for i in range(layer.getNumResults()):
    result = layer.getResult(0)
```

but should adopt the new API for use in new work.

6.7.3 Attribute Queries

By Attributes

queryByAttributes()

6.7.4 Spatial Queries

By Rectangle

queryByRect()

By Point

queryByRect()

By Shape

queryByShape()

By Selection

queryByFeatures()

6.8 MapScript Variables

Author Howard Butler

Contact hobu.inc at gmail.com

Revision \$Revision: 8278 \$

Date \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Contents

- MapScript Variables
 - Version
 - Logical Control - Boolean Values
 - Logical Control - Status Values
 - Map Units
 - Layer Types
 - Font Types
 - Label Positions
 - Label Size (Bitmap only)
 - Shape Types
 - Measured Shape Types
 - Shapefile Types
 - Query Types
 - File Types
 - Querymap Styles
 - Connection Types
 - DB Connection Types
 - Join Types
 - Line Join Types (for rendering)
 - Image Types
 - Image Modes
 - Symbol Types
 - Return Codes
 - Limiters
 - Error Return Codes

6.8.1 Version

Name	Type	Value
MS_VERSION	character	5.2

6.8.2 Logical Control - Boolean Values

Name	Type	Value
MS_TRUE	integer	1
MS_ON	integer	1
MS_YES	integer	1
MS_FALSE	integer	0
MS_OFF	integer	0
MS_NO	integer	0

6.8.3 Logical Control - Status Values

Name	Type	Value
MS_DEFAULT	integer	2
MS_EMBED	integer	3
MS_DELETE	integer	4

6.8.4 Map Units

Name	Type	Value
MS_DD	integer	
MS_FEET	integer	
MS_INCHES	integer	
MS_METERS	integer	
MS_MILES	integer	
MS_PIXELS	integer	

6.8.5 Layer Types

Name	Type	Value
MS_LAYER_POINT	integer	
MS_LAYER_LINE	integer	
MS_LAYER_POLYGON	integer	
MS_LAYER_RASTER	integer	
MS_LAYER_ANNOTATION	integer	
MS_LAYER_QUERY	integer	
MS_LAYER_CIRCLE	integer	
MS_LAYER_TILEINDEX	integer	

6.8.6 Font Types

Name	Type	Value
MS_TRUETYPE	integer	
MS_BITMAP	integer	

6.8.7 Label Positions

Name	Type	Value
MS_UL	integer	
MS_LL	integer	
MS_UR	integer	
MS_LR	integer	
MS_CL	integer	
MS_CR	integer	
MS_UC	integer	
MS_LC	integer	
MS_CC	integer	
MS_AUTO	integer	

6.8.8 Label Size (Bitmap only)

Name	Type	Value
MS_TINY	integer	
MS_SMALL	integer	
MS_MEDIUM	integer	
MS_LARGE	integer	
MS_GIANT	integer	

6.8.9 Shape Types

Name	Type	Value
MS_SHAPE_POINT	integer	
MS_SHAPE_LINE	integer	
MS_SHAPE_POLYGON	integer	
MS_SHAPE_NULL	integer	

6.8.10 Measured Shape Types

Name	Type	Value
MS_SHP_POINTM	integer	21
MS_SHP_ARCM	integer	23
MS_SHP_POLYGONM	integer	25
MS_SHP_MULTIPPOINTM	integer	28

6.8.11 Shapefile Types

Name	Type	Value
MS_SHAPEFILE_POINT	integer	1
MS_SHAPEFILE_ARC	integer	3
MS_SHAPEFILE_POLYGON	integer	5
MS_SHAPEFILE_MULTIPPOINT	integer	8

6.8.12 Query Types

Name	Type	Value
MS_SINGLE	integer	0
MS_MULTIPLE	integer	1

6.8.13 File Types

Name	Type	Value
MS_FILE_MAP	integer	
MS_FILE_SYMBOL	integer	

6.8.14 Querymap Styles

Name	Type	Value
MS_NORMAL	integer	
MS_HILITE	integer	
MS_SELECTED	integer	

6.8.15 Connection Types

Name	Type	Value
MS_INLINE	integer	
MS_SHAPEFILE	integer	
MS_TILED_SHAPEFILE	integer	
MS_SDE	integer	
MS_OGR	integer	
MS_POSTGIS	integer	
MS_WMS	integer	
MS_ORACLESPATIAL	integer	
MS_WFS	integer	
MS_GRATICULE	integer	
MS_MYGIS	integer	
MS_RASTER	integer	

6.8.16 DB Connection Types

Name	Type	Value
MS_DB_XBASE	integer	
MS_DB_CSV	integer	
MS_DB_MYSQL	integer	
MS_DB_ORACLE	integer	
MS_DB_POSTGRES	integer	

6.8.17 Join Types

Name	Type	Value
MS_JOIN_ONE_TO_ONE	integer	
MS_JOIN_ONE_TO_MANY	integer	

6.8.18 Line Join Types (for rendering)

Name	Type	Value
MS_CJC_NONE	integer	
MS_CJC_BEVEL	integer	
MS_CJC_BUTT	integer	
MS_CJC_MITER	integer	
MS_CJC_ROUND	integer	
MS_CJC_SQUARE	integer	
MS_CJC_TRIANGLE	integer	

6.8.19 Image Types

Name	Type	Value
GD/GIF	integer	
GD/PNG	integer	
GD/PNG24	integer	
GD/JPEG	integer	
GD/WBMP	integer	
swf	integer	
imagemap	integer	
pdf	integer	
GDAL/GTiff	integer	

6.8.20 Image Modes

Name	Type	Value
MS_IMAGEMODE_PC256	integer	
MS_IMAGEMODE_RGB	integer	
MS_IMAGEMODE_RGBA	integer	
MS_IMAGEMODE_INT16	integer	
MS_IMAGEMODE_FLOAT32	integer	
MS_IMAGEMODE_BYTE	integer	
MS_IMAGEMODE_NULL	integer	
MS_NOOVERRIDE	integer	
MS_GD_ALPHA	integer	1000

6.8.21 Symbol Types

Name	Type	Value
MS_SYMBOL_SIMPLE	integer	
MS_SYMBOL_VECTOR	integer	
MS_SYMBOL_ELLIPSE	integer	
MS_SYMBOL_PIXMAP	integer	
MS_SYMBOL_TRUETYPE	integer	
MS_SYMBOL_CARTOLINE	integer	

6.8.22 Return Codes

Name	Type	Value
MS_SUCCESS	integer	
MS_FAILURE	integer	
MS_DONE	integer	

6.8.23 Limiters

Name	Type	Value
MS_MAXSYMBOLS	long	
MS_MAXVECTORPOINTS	long	
MS_MAXSTYLELENGTH	long	
MS_IMAGECACHESIZE	long	

6.8.24 Error Return Codes

Name	Type	Value
MS_NOERR	long	0
MS_IOERR	long	1
MS_MEMERR	long	2
MS_TYPEERR	long	3
MS_SYMERR	long	4
MS_REGEXERR	long	5
MS_TTFERR	long	6
MS_DBFERR	long	7
MS_GDERR	long	8
MS_IDENTERR	long	9
MS_EOFERR	long	10
MS_PROJERR	long	11
MS_MISCERR	long	12
MS_CGIERR	long	13
MS_WEBERR	long	14
MS_IMGERR	long	15
MS_HASHERR	long	16
MS_JOINERR	long	17
MS_NOTFOUND	long	18
MS_SHPERR	long	19
MS_PARSEERR	long	20
MS_SDEERR	long	21
MS_OGRERR	long	22
MS_QUERYERR	long	23
MS_WMSERR	long	24
MS_WMCONNERR	long	25
MS_ORACLESPATIALERR	long	26
MS_WFSERR	long	27
MS_WFCONNERR	long	28
MS_MAPCONTEXTERR	long	29
MS_HTTPERR	long	30
MS_CHILDERR	long	31
MS_WCSERR	long	32
MS_NUMERRORCODES	long	33
MESSAGELENGTH	long	33
ROUTINELENGTH	long	33

DATA INPUT

7.1 Vector Data

Date 2008/09/09

Author Tyler Mitchell

Contact tmitchell at osgeo.org

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit: <http://creativecommons.org/licenses/by-sa/2.0/ca/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

What is vector data? This quote from is a good description of what vector data is:

Vector: “An abstraction of the real world where positional data is represented in the form of coordinates. In vector data, the basic units of spatial information are points, lines and polygons. Each of these units is composed simply as a series of one or more coordinate points. For example, a line is a collection of related points, and a polygon is a collection of related lines. Vector images are defined mathematically as a series of points joined by lines. Vector-based drawings are resolution independent. This means that they appear at the maximum resolution of the output device, such as a printer or monitor. Each object is self-contained, with properties such as color, shape, outline, size, and position on the screen.”

From: http://coris.noaa.gov/glossary/glossary_1_z.html#v

The rest of this document is the data format guide. This guide is structured to show the fundamentals of each MapServer supported data format. Each section discusses one format, ranging from one to several pages in length. The sections typically start with a summary of the most important information about the format, followed by examples of file listings, connection methods, ogrinfo usage and MapServer map file syntax examples.

Each section has been designed to stand alone, so you may notice that certain warnings and comments are repeated or redundant. This is intentional. Each format is presented in rough order of popular use, based on a survey of the MapServer community.

The following formats are included:

7.1.1 Data Format Types

Each type of data is made up of a data source and (one or more) layers. These two definitions apply to MapServer and OGR.

Data Source - a group of layers stored in a common repository. This may be a file that handles several layers within it, or a folder that has several files.

Layer - a sub-set of a data source often containing information in one type of vector format (point, line, polygon).

There are three types of data mapping and GIS data formats. Each type is handled differently. Below are the types and some example formats:

- File-based- Shapefiles, Microstation Design Files (DGN), GeoTIFF images
- Directory-based - ESRI ArcInfo Coverages, US Census TIGER
- Database connections - PostGIS, ESRI ArcSDE, MySQL

File-based Data

File-based data consists of one or more files stored in any arbitrary folder. In many cases a single file is used (e.g. DGN) but ESRI Shapefiles, for example, consist of at least 3 files each with a different filename extension: SHP, DBF, SHX. In this case all 3 files are required because they each perform a different task internally.

Filenames usually serve as the data source name and contain layers that may or may not be obvious from the filename. In Shapefiles, for example, there is one data source per shapefile and one layer which has the same name as that of the file.

Directory-based Data

Directory-based data consists of one or more files stored in a particular way within a parent folder. In some cases (e.g. Coverages) they may also require additional folders in other locations in the file tree in order to be accessed. The directory itself may be the data source. Different files within the directory often represent the layers of data available.

For example, ESRI ArcInfo Coverages consist of more than one file with an ADF file extension, within a folder. The PAL.ADF file represents the Polygon data. ARC.ADF holds the arc or line string data. The folder holds the data source and each ADF file is a layer.

Database Connections

Database Connections are very similar to file and directory-based structures in one respect: they provide geographic coordinate data for MapServer to interpret. That may be oversimplifying what is happening inside MapServer, but in essence all you need is access to the coordinates making up the vector datasets.

Database connections provide a stream of coordinate data that is temporarily stored (e.g. in memory) and read by MapServer to create the map. Other attribute or tabular data may also be required, but the focus of this guide is coordinate data.

One important distinction between databases must be made. The databases discuss here are spatial databases, those which can hold geographic data in its own data type. This is opposed to strictly tabular databases which cannot hold geographic coordinates in the same way. It is possible to store some very simple coordinate data in regular tables, but for anything but the most simple use a spatial database is required. There are spatial extensions to many databases (open source and commercial). One of the most robust is the PostGIS extension to the PostgreSQL database. This database not only allows the storage of geographic data, but also allows the manipulation of that data using SQL commands. The other open source database with spatial capabilities is MySQL.

Connections to databases usually consist of the following pieces of connection information:

Host - Directions to the server or computer hosting the database.

Database name - The name of the database you wish to access that is running on the host.

User name / passwords - Access privileges are usually restricted by user.

Note: Some databases (e.g. Oracle) use a name service identifier that includes both the host and database names.

Access to specific pieces of coordinate data usually require:

Table/View name - The name of the table or view holding the coordinate data.

Geographic column name - Where the geometry or coordinates are stored.

7.1.2 ESRI Shapefiles (SHP)

Also known as ESRI ArcView Shapefiles or ESRI Shapefiles. ESRI is the company that introduced this format. ArcView was the first product to use shapefiles.

File listing

Shapefiles are made up of a minimum of three similarly named files, with different suffixes:

```
Countries_area.dbf
Countries_area.shp
Countries_area.shx
```

Data Access / Connection Method

Shapefile access is built directly into MapServer. It is also available through OGR, but direct access without OGR is recommended and discussed here. The path to the shapefile is required. No file extension should be specified. Shapefiles only hold one layer of data, therefore no distinction needs to be made.

OGRINFO Examples

- The directory can serve as a data source.
- Each shapefile in a directory serves as a layer.
- A shapefile can also be a data source. In this case the layer has the same prefix as the shapefile.

Using ogrinfo on a directory with multiple shapefiles:

```
> ogrinfo /data/shapefiles/
INFO: Open of `'/data/shapefiles/'
using driver 'ESRI Shapefile' successful.
1: wpg_h2o (Line String)
2: wpg_roads (Line String)
3: wpg_roads_dis (Line String)
4: wpgrestaurants (Point)
```

Using ogrinfo on a single shapefile:

```
> ogrinfo /data/shapefiles/Countries_area.shp
Had to open data source read-only.
INFO: Open of `Countries_area.shp'
using driver 'ESRI Shapefile' successful.
1: Countries_area (Polygon)
```

Using ogrinfo to examine the structure of the file/layer:

```
> ogrinfo -summary /data/shapefiles/Countries_area.shp Countries_area
Had to open data source read-only.
INFO: Open of `Countries_area.shp`
using driver `ESRI Shapefile` successful.
```

```
Layer name: Countries_area
Geometry: Polygon
Feature Count: 27458
Extent: (-180.000000, -90.000000) - (180.000000, 83.627419)
Layer SRS WKT:
(unknown)
FAC_ID: Integer (5.0)
TILE: Integer (3.0)
ARCLIST: String (254.0)
NAM: String (77.0)
PERIMETER: Real (22.17)
POLYGONCOU: Integer (6.0)
NA2DESC: String (45.0)
```

Map File Example:

```
LAYER
    NAME my_shapefile
    TYPE POLYGON
    DATA countries_area
    STATUS OFF
    CLASS
        NAME "Countries"
        OUTLINECOLOR 0 0 0
    END
END
```

7.1.3 PostGIS/PostgreSQL

PostGIS/PostgreSQL

PostGIS spatially enables the Open Source PostgreSQL database.

The [PostGIS wiki page](#) may include additional information.

Data Access /Connection Method

PostGIS is supported directly by MapServer and must be compiled into MapServer to work.

The PostgreSQL client libraries (libpq.so or libpq.dll) must be present in the system's path environment for functionality to be present.

The CONNECTIONTYPE parameter must be set to POSTGIS.

The CONNECTION parameter is used to specify the parameters to connect to the database. CONNECTION parameters can be in any order. Most are optional. dbname is required. user is required. host defaults to localhost, port defaults to 5432 (the standard port for PostgreSQL).

The DATA parameter is used to specify the data used to draw the map. The form of DATA is “[geometry_column] from [table_name|sql_subquery] using unique [unique_key] using srid=[spatial_reference_id]”. The “using unique” and “using srid=” clauses are optional, but using them improves performance.

Here is a simple generic example:

```
CONNECTIONTYPE POSTGIS
CONNECTION "host=yourhostname dbname=yourdatabasename user=yourdbusername password=yourdbpassword port=yourdbport"
DATA "geometrycolumn from yourtablename"
```

This example shows specifying the unique key and srid in the DATA line:

```
CONNECTIONTYPE POSTGIS
CONNECTION "dbname=yourdatabasename user=yourdbusername"
DATA "the_geom from the_database using unique gid using srid=4326"
```

This example shows using a SQL subquery to perform a join inside the database and map the result in MapServer. Note the “as subquery” string in the statement – everything between “from” and “using” is sent to the database for evaluation:

```
CONNECTIONTYPE POSTGIS
CONNECTION "dbname=yourdatabasename user=yourdbusername"
DATA "the_geom from (select g.gid, g.the_geom, a.attr1, a.attr2 from geotable g join attrtable a on g.gid=a.gid) as subquery"
```

This example shows using a geometry function and database sort to limit the number of features and vertices returned to MapServer:

```
CONNECTIONTYPE POSTGIS
CONNECTION "dbname=yourdatabasename user=yourdbusername"
DATA "the_geom from (select g.gid, ST_Simplify(g.the_geom, 10.0) as the_geom from geotable g order by g.gid)"
```

This example shows the use of the !BOX! substitution string to over-ride the default inclusion of the map bounding box in the SQL. By default the spatial box clause is appended to the SQL in the DATA clause, but you can use !BOX! to insert it anywhere you like in the statement. In general, you won’t need to use !BOX!, because the PostgreSQL planner will generate the optimal plan from the generated SQL, but in some cases (complex sub-queries) a better plan can be generated by placing the !BOX! closer to the middle of the query:

```
CONNECTIONTYPE POSTGIS
CONNECTION "dbname=yourdatabasename user=yourdbusername"
DATA "the_geom from (select g.gid, ST_Union(g.the_geom, 10.0) as the_geom from geotable g where ST_Intersection(g.the_geom, !BOX!) is not null)"
```

OGRINFO Examples

OGRINFO can be used to read out metadata about PostGIS tables directly from the database.

First you should make sure that your GDAL/OGR build contains the PostgreSQL driver, by using the ‘-formats’ command:

```
>ogrinfo --formats
Loaded OGR Format Drivers:
...
-> "PGeo" (readonly)
-> "PostgreSQL" (read/write)
-> "MySQL" (read/write)
...
```

If you don't have the driver, you might want to try the [FWTools](#) or [MS4W](#) packages, which include the driver.

Once you have the driver you are ready to try an ogrinfo command on your database to get a list of spatial tables:

```
>ogrinfo PG:"host=127.0.0.1 user=postgres password=postgres dbname=canada port=5432"
      using driver 'PostgreSQL' successful.
      1: province (Multi Polygon)
```

Now use ogrinfo to get information on the structure of the spatial table:

```
>ogrinfo PG:"host=127.0.0.1 user=postgres password=postgres dbname=canada port=5432" province -summary
INFO: Open of 'PG:host=127.0.0.1 user=postgres password=postgres dbname=canada'
      using driver 'PostgreSQL' successful.

Layer name: province
Geometry: Multi Polygon
Feature Count: 1068
Extent: (-2340603.750000, -719746.062500) - (3009430.500000, 3836605.250000)
Layer SRS WKT:
(unknown)
FID Column = gid
Geometry Column = the_geom
area: Real (0.0)
island: String (30.0)
island_e: String (30.0)
island_f: String (30.0)
name: String (30.0)
...
```

Mapfile Example

```
LAYER
  NAME "province"
  STATUS ON
  TYPE POLYGON
  CONNECTIONTYPE POSTGIS
  CONNECTION "host=127.0.0.1 port=5432 dbname=canada user=postgres password=postgres"
  DATA "the_geom from province"
  CLASS
    ...
  END
END
```

For more info about PostGIS and MapServer see the PostGIS docs: <http://postgis.refrations.net/docs/>

7.1.4 OGR

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8597 \$

Date \$Date: 2009-02-19 07:10:12 -0800 (Thu, 19 Feb 2009) \$

Table of Contents

- OGR
 - Introduction
 - What is OGR?
 - Obtaining and Compiling MapServer with OGR Support
 - Integrating OGR Support with MapServer Applications
 - STYLEITEM “AUTO” - Rendering Layers Using Style Information from the OGR File
 - Sample Sites Using OGR/MapServer
 - FAQ / Common Problems

Introduction

Starting with version 3.5, MapServer included the ability to access vector data sets in formats other than Shapefile in their native format using the OGR library. The following document describes the process for implementing OGR support within MapServer applications.

Note: Experimental OGR support was included in MapServer version 3.4 but this initial implementation had some limitations and is not covered in this document.

This document assumes that you are already familiar with certain aspects of MapServer:

- MapServer application development and especially setting up *.map files*.
- Some compilation skills if you don't have ready access to a pre-compiled installation and need to compile your own copy of MapServer with OGR support.
- access to OGR utilities, such as *ogrinfo*, which are available in the [FWTools](#) and [MS4W](#) packages.

Readers should also check out the [Vector Data Access Guide](#), which has lots of examples of how to access specific vector formats.

What is OGR?

The OGR Simple Features Library is a C++ open source library (and command-line tools) providing read (and sometimes write) access to a variety of vector file formats including ESRI Shapefiles, and MapInfo mid/mif and TAB formats.

OGR is actually part of the GDAL library, so you will notice that some references point to GDAL (such as the mailing list).

What Does OGR Add to MapServer?

The OGR Simple Features Library allows MapServer users to display several types of vector data files in their native formats. For example, MapInfo Mid/Mif and TAB data do not need to be converted to ESRI shapefiles when using OGR support with MapServer.

What Data Formats are Supported?

See http://www.gdal.org/ogr/ogr_formats.html for the latest list of supported formats. At the date this document was written, the following formats were supported:

- ArcInfo Binary Coverages

- ArcInfo E00 Coverages
- Atlas BNA
- Comma Separated Value (.csv)
- DODS/OPeNDAP
- ESRI ArcSDE
- ESRI Personal GeoDatabase
- ESRI Shapefiles
- FMEObjects Gateway
- Géoconcept Export
- GeoJSON
- GeoRSS
- GML
- GMT
- GRASS
- GPX
- Informix DataBlade
- INGRES
- INTERLIS
- KML
- MapInfo files
- Memory
- Microstation DGN files
- MySQL
- ODBC
- OGDI Vectors
- Oracle Spatial
- PostgreSQL
- SDTS
- SQLite
- UK.NTF (National Transfer Format)
- US Census TIGER/Line
- VRT - Virtual Datasource
- X-Plane/Flighgear aeronautical data

Note: Some of the above formats (e.g. OGDI) have external dependencies and are not always included in the pre-compiled binary distributions of MapServer with OGR support.*

Note: Some of the above formats are not well suited for random access by nature, that's the case of MapInfo MIF/MID files which is a TEXT format and will give very poor performance for a web application. On the other hand, some

binary formats such as MapInfo TAB are better suited for random access and will give performance comparable to native shapefile access in MapServer.*

How to Get More Information on the OGR Project

- More information on the OGR Simple Features Project can be found at <http://www.gdal.org/ogr/>.
- The [GDAL mailing list](#) can be used for OGR related questions. Always search the list archives before sending new questions.
- The [GDAL Wiki](#) has lots of good information for users and developers.
- The [#gdal IRC channel](#) on irc.freenode.net might also be of help. For info on IRC see the [MapServer IRC page](#).

The main developer of the OGR library is Frank Warmerdam and the integration of OGR within MapServer was done by Daniel Morissette.

Obtaining and Compiling MapServer with OGR Support

- Follow the instructions on the [OGR page](#) to compile/install OGR/GDAL.
- Obtain the MapServer [source](#).

For UNIX users, see the README.CONFIGURE file in the MapServer source, or see the [UNIX Compilation and Installation](#). If GDAL/OGR is normally installed it should be sufficient to add `-with-ogr` to the configure line before (re)building MapServer. Linux users might want to try [FGS](#), a Linux installer for MapServer.

For Windows users, it is recommended to look for a pre-compiled binary on the MapServer site ([MS4W](#) is recommended). If you want to compile your own then see the README.WIN32 file in the MapServer source.

Integrating OGR Support with MapServer Applications

The only change that is needed to integrate OGR support with a MapServer application is with the `.map` file. The LAYER's DATA parameter is expanded to three parameters (CONNECTIONTYPE OGR, CONNECTION and DATA).

The syntax for this differs depending on the type of data being used (the [Vector Data Access Guide](#) is an excellent resource for this). In OGR, a data source can be either a set of files that share a common basename (e.g. `.shp/.shx/.dbf` for ArcView Shapefiles, or `.tab/.map/.dat/.ind/.id` for MapInfo TAB files) or a whole directory of files (e.g. TIGER).

Let's call the former "File-based data sources" and the later "Directory-based data sources". When accessing a **file-based data source** you specify the filename of one of the files in the set (e.g. `roads.shp` or `roads.tab`) and when accessing a **directory-based data source** you specify the directory name and OGR reads all the files in the directory as a single data source with potentially several layers (e.g. TIGER files).

Some OGR drivers (e.g. SHP, TAB) can have dual behaviors, that is if they're pointed to a single file then they behave as a file-based data source and if they're pointed to a directory then they will behave as a directory-based data source and then every file in the directory becomes a new layer in the data source.

See the [OGR formats page](#) for more info on the specific file format you're using. (Click on the format name for more specific driver info on that format)

Using OGR Data Sources in the Map File

The `.map` file LAYER definition for file-based sources is as follows:

```
LAYER
...
CONNECTIONTYPE OGR
CONNECTION "<datasource_name>"
DATA "<layer_definition>"
...
END
```

<datasource_name> is the name of the datasource to read from and is prefixed by the CONNECTION keyword. The exact organization depends on the format driver in use. The format driver to use is automatically selected by OGR based on the nature of the string passed as the datasource, and/or the format of the file referenced by it.

- For file based datasources this is the name of the file, including the extension, using an absolute path, or a relative path. Relative paths are interpreted relative to the SHAPEPATH first, if not found then we try again relative to the .map file location.

Note: Before version 4.1 the SHAPEPATH was ignored for OGR datasources.

- For directory based datasources, such as TIGER/Line, or Arc/Info Binary Coverages this is the name of the directory containing the files. If the path is relative it is interpreted relative to the .map file.
- For virtual datasources such as database systems, and OGDII this is the service connection string and is generally not related to the filesystem. For instance, for Oracle Spatial this might be "OCI:warmerda/Password@gdal800.velocet.ca".

<layer_definition> is the name, number or SQL definition of the layer to use from the datasource. It is indicated via the DATA keyword in the map file.

- Layer Name: The (case insensitive) layer name may be used to select a layer.
- Layer Number: The layer number (starting from 0 for the first layer) may be used to select a layer. Generally the layer name is preferred to this since it is more self describing.
- Omitted: If no DATA keyword is provided, this is equivalent to selecting layer 0.
- SQL SELECT: If an SQL SELECT statement is used, it is interpreted in a driver specific manner to try and generate a temporary pseudo-layer. For some formats this a restricted subset of SQL is interpreted within OGR. For RDBMS based drivers (such as PostGIS and Oracle) this is passed through to the underlying database.

The OGRINFO utility can be used to find out the list of layers and their names in a data source.

Examples of Layer Definitions Using OGR

Please see the *Vector Data Access Guide* for details and examples of each data format supported.

Example 1. MapInfo TAB file

```
LAYER
  NAME "Builtup_Areas_tab"
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "data/tab/092b06_builtup_a.tab"
  STATUS ON
  CLASS
  ...
END
...
END
```


Example 2. Microstation DGN file using <layer_index>

The entire DGN file is represented in OGR as one layer (see the [DGN driver page](#) for more details):

```
LAYER
  NAME "dgn"
  TYPE LINE
  CONNECTIONTYPE OGR
  CONNECTION "dgn/santabarbara02.dgn"
  DATA "0"
  STATUS ON
  STYLEITEM "AUTO"
  CLASS
  END
END # Layer
```

Example 3. TIGER/Line file using <layer_name>

```
LAYER
  NAME "Roads_tig"
  TYPE line
  CONNECTIONTYPE OGR
  CONNECTION "full/path/to/tiger/TGR25001"
  DATA "CompleteChain"
  STATUS ON
  CLASS
  ...
  END
END
```

Example 4. Directory of Shapefiles using SQL JOIN

```
LAYER
  NAME "Parks_cov"
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "data/shppoly"
  DATA "SELECT eas_id, idlink.Name FROM poly LEFT JOIN idlink ON poly.eas_id = idlink.eas_id"
  STATUS ON
  CLASSITEM "idlink.Name"
  CLASS
  ...
  END
END
```

How to Use “OGRINFO”

OGRINFO is part of the GDAL/OGR distribution (it is also included in [FWTools](#) and [MS4W](#)). It is an executable that can be used to obtain layer information about OGR supported files. The parameters are:

```
ogrinfo [-ro] [-q] datasource_name [layer [layer...]]
```

- -ro opens the file as read only (optional)
- -q executes in quiet mode, only the layer index line will be returned (optional)
- *datasource_name* is the filename including extension (eg. roads.tab); for TIGER/Line files, *datasource_name* is the directory containing the TIGER files (eg. ogrinfo TGR25001)

Example 5. To get the list of layers in a file:

```
$ ogrinfo popplace.tab

Had to open data source read-only.
INFO: Open of `popplace.tab'
using driver `MapInfo File' successful.
1: popplace (Point)
```

which shows that there is one point layer in the popplace.tab file.

Example 6. To get a dump of a specific layer, including field names, projection, etc:

```
$ ogrinfo popplace.tab popplace

Had to open data source read-only.
INFO: Open of `popplace.tab'
using driver `MapInfo File' successful.

Layer name: popplace
Geometry: Point
Feature Count: 497
Layer SRS WKT: PROJCS["unnamed",GEOGCS["unnamed",DATUM["North ...snipped...
AREA: Real (15.3)
PERIMETER: Real (15.3)
POPPLACE_: Real (11.0)
POPPLACE_I: Real (15.0)
NAME: String (50.0)
OGRFeature(popplace):1
  AREA (Real) =          0.000
  PERIMETER (Real) =          0.000
  POPPLACE_ (Real) =          1
  POPPLACE_I (Real) =          1
  NAME (String) = Port Hope Simpson
  POINT (2437287.249 1153656.751)

OGRFeature(popplace):2
  AREA (Real) =          0.000
  PERIMETER (Real) =          0.000
  POPPLACE_ (Real) =          2
  POPPLACE_I (Real) =          1
  NAME (String) = Hopedale

...
...
```

Example 7. To get a list of layers in a TIGER/Line Directory:

```
$ ogrinfo TGR25001

Had to open data source read-only.
INFO: Open of `TGR25001'
using driver `TIGER' successful.
1: CompleteChain (Line String)
2: AltName (None)
3: FeatureIds (None)
4: ZipCodes (None)
5: Landmarks (Point)
```

```

6: AreaLandmarks (None)
7: KeyFeatures (None)
8: Polygon (None)
9: EntityNames (Point)
10: IDHistory (None)
11: PolyChainLink (None)
12: PIP (Point)
13: TLIDRange (None)
14: ZipPlus4 (None)

```

The above example shows that there are 14 layers in the TGR25001 directory.

Example 8. To get a summary of a specific TIGER layer, including only field names, projection, and extent

```
$ ogrinfo TGR25001 Landmarks -summary
```

```

Had to open data source read-only.
INFO: Open of 'TGR25001'
using driver 'TIGER' successful.

```

```

Layer name: Landmarks
Geometry: Point
Feature Count: 777
Extent: (-70.674324, 41.519817) - (-69.969211, 42.046868)
Layer SRS WKT: GEOGCS["NAD83",DATUM["North_American_Datum_1983",
SPHEROID["GRS 1980",6378137,298.257222101]],PRIMEM["Greenwich",0],
UNIT["degree",0.0174532925199433]]
MODULE: String (8.0)
FILE: String (5.0)
STATE: Integer (2.0)
COUNTY: Integer (3.0)
LAND: Integer (10.0)
SOURCE: String (1.0)
CFCC: String (3.0)
LANAME: String (30.0)

```

Queries Through OGR Format

OGR layers can be queried the same way as regular shapefiles in MapServer.

TILEINDEX with OGR

OGR layers can utilize tile indexes in a similar fashion to Shapefile based layers. The TILEINDEX keyword should contain the connection string for the tile index file. The tile index file may be any supported OGR format, including shapefiles.

The TILEITEM keyword in the LAYER definition indicates what attribute from the tile index file should be used as the datasource location. If omitted, the default TILEITEM value is "location". The value in the location field should be a connection string the same as would have been used in the CONNECTION field for OGR layers. The CONNECTION keyword is not needed (and will be ignored) for layers using the OGR connection type and having the TILEINDEX keyword.

Tileindex files can be prepared in an external GIS, or using the OGR utility ogrtindex. Details can be found on the [OGR Utilities Page](#).

The following is a simple example of a point layer using a tile index.

```
LAYER
  NAME "ogr_points"
  TYPE POINT
  CONNECTIONTYPE OGR
  TILEINDEX "PIP_ogr_tiles.shp,0"
  STATUS ON
  CLASS
    NAME "points"
    STYLE
      SYMBOL "default-circle"
      COLOR 255 0 0
      SIZE 6
    END
  END
END
```

OGR tileindex layers should support all normal query and attribute fetching mechanisms, including from MapScript; however, this has not been heavily tested as of April/2002. Please report problems via the MapServer Trac. If auto projection support is used for tileindexed OGR layers, the tileindex is read for the projection (not the component tiles). Problems may (or may not) be encountered if the component tiles have differing schemas (different sets of attributes).

Connection Pooling

For some OGR supported formats, connecting to the dataset is quite expensive in terms of CPU use and amount of disk IO. For instance, establishing access to an S-57 dataset results in a complete read into memory of the data files. Connection pooling control aims at reducing this overhead in situations where the same file is used for several different map layers.

To ensure that an OGR supported dataset is only opened once per map render (instead of separately for each map LAYER referencing the dataset, use the CLOSE_CONNECTION PROCESSING option. The default value is for CLOSE_CONNECTION is NORMAL, but if set to DEFER the dataset will be kept open till the map render is complete. It will be reused by any other layers with using the same datasource.

Example 9. Preserve S-57 connection for two layers

In this example, we are using the same dataset (NO410810.000) for two layers. To avoid re-reading the dataset, we mark the first layer to defer closing the connection till layer. In the second (or last) layer we request NORMAL connection handling (though this could have been left out as normal handling is the default).

```
LAYER
  NAME "AdminAreas"
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "NO410810.000"
  DATA "ADMARE"
  PROCESSING "CLOSE_CONNECTION=DEFER"
  STATUS ON
  ...
END
LAYER
  NAME "Land Areas"
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "NO410810.000"
  DATA "LNDARE"
```

```

PROCESSING "CLOSE_CONNECTION=NORMAL"
STATUS ON
...
END

```

1. The text of the CONNECTION keyword must match exactly between layers for the connection to be reused.
2. Some dataset connections are quite memory expensive, and keeping them open may result in increased memory use.
3. If all layers rendered for a particular connection defer closing the connection, it will remain open till MapServer terminates. For normal cgi or MapScript use this is likely OK.
4. This use of CLOSE_CONNECTION handling is unique to OGR layers, and may be changed at some point in the future as part of a broader implementation of connection pooling in MapServer.

STYLEITEM "AUTO" - Rendering Layers Using Style Information from the OGR File

Note: This feature is only supported with MapInfo TAB and Microstation DGN files at the moment, but eventually other formats that carry colors and styles at the shape-level may also be supported through OGR.*

In MapServer, ArcView, and other shapefile-based applications, colors and styles are usually defined at the layer level. This means that all the shapes in a given layer are usually rendered using the same color and styles.

On the other hand, some formats supported by OGR such as MapInfo TAB do have color and style information attached to each shape. OGR adds support for the 'STYLEITEM "AUTO"' layer parameter which allows you to request that the shapes in a layer be rendered using colors and styles coming from the data source instead of being driven by CLASSes as was traditionally done with MapServer.

How to Implement

In order to have a layer rendered using colours and styles coming from the OGR data source, you must do the following:

- Your layer definition must contain the STYLEITEM "AUTO" parameter.
- Your layer definition needs to contain at least one CLASS (which may be empty) and optionally a CLASSITEM to match the expressions if your CLASS contains an expression. The empty CLASS in the layer will be updated dynamically at runtime to contain colours and styles coming from the data source for each shape.

Examples

Example 10. Layer Definition Using STYLEITEM "AUTO" without a CLASSITEM

```

LAYER
  NAME "test_dgn"
  STATUS ON
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "../data/dgn/test.dgn"

  # This enables use of colors and styles from the source file.
  STYLEITEM "AUTO"

  # Define an empty class that will be filled at runtime from the color and

```

```
    # styles read on each shape in the source file.
    CLASS
  END
END # layer
```

Example 11. Layer Definition Using STYLEITEM “AUTO” with a CLASSITEM

```
LAYER
  NAME "Builtup_Areas_tab"
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "data/tab/092b06_builtup_a.tab"
  STATUS ON

  # This enables use of colors and styles from the source file.
  STYLEITEM "AUTO"

  # Define an empty class that will be filled at runtime from the color and
  # styles read on each shape in the source file.
  CLASSITEM "CATEGORY"
  CLASS
    EXPRESSION "1"
  END
END
```

Please Note:

CLASS EXPRESSIONs are still working, so it is still possible to query and classify layers that are using STYLEITEM “AUTO”. The only difference is that instead of using static class definitions, the colors and style will be read from the data file.

Important Notes

NOTE 1 Even though MapInfo and other OGR data sources may support layers with mixed geometry types (e.g. points, lines and polygons in the same file) this is not yet supported in MapServer. So you still have to define a layer ‘TYPE’ and make sure that all the shapes in the OGR data source are compatible with that layer type, otherwise MapServer may produce an error about incompatible geometry types at runtime.

NOTE 2 Due to the dynamic nature of this feature, it is not compatible with the labelcache, so the labelcache is automatically disabled for layers that make use of ‘STYLEITEM “AUTO”’.

NOTE 3 When you use STYLEITEM AUTO, MapServer tries to match symbol names returned by OGR to names in your symbol file. For a quick solution, try using the following symbol file:

http://www2.dmsolutions.ca/msapps/yk_demo/etc/symbols_mapinfo.sym

The name of the symbols returned by OGR to MapServer depends on the file format. In the case of MapInfo files, it will be:

- For “old-style” symbols (default MapInfo 3.0 symbols numbered 32 to 67) the symbol name will be ‘mapinfo-sym-##’ where ‘##’ is the symbol number, e.g. ‘mapinfo-sym-32’.
- For “Font Symbols”, the symbol name is also ‘mapinfo-sym-##’ where ‘##’ is the symbol number in the font. In this case, the name of the font itself is ignored by MapServer.
- MapInfo also supports “custom symbols” (bitmap symbols)... I’m not sure what you would get from OGR for this, but I’m pretty sure that MapServer doesn’t do anything useful with them.

The OGRINFO utility can be used to find out exactly which symbol names OGR will return to MapServer. Look at the “Style” string in the ogrinfo output for each shape that is read.

Mapping of OGR Style Info to the MapServer CLASS Members

Here is the list of style parameters that are currently supported from OGR data sources and how they’re mapped in MapServer:

Line color The line colour is mapped to CLASS.COLOR

Line thickness The default will be 1 pixel line (as it always is with MapServer). In MapServer, in order to get lines thicker than 1 pixel, one has to define a circle symbol that will be used to render the line (by applying a size to the circle symbol).

So if your data source contains lines thicker than 1 pixel then you need to provide a circle symbol in your symbolset and you must name it “default-circle”. If this symbol is present then the lines will be drawn using their real thickness, otherwise all lines will be 1 pixel wide.

Polygon fill color Polygon fill color is mapped directly to CLASS.COLOR

Note that at this time, transparent polygons are not supported (they’re always opaque).

Polygon outline If a polygon has an outline color and thickness defined in the data source then the same rule as for line color and thickness above will apply, except that the outline color is mapped to CLASS.OUTLINECOLOR

Point symbols Point symbol color is directly mapped to CLASS.COLOR Point symbol size is directly mapped to CLASS.SIZE

If your symbolset contains a symbol called “default-marker” then this symbol will be used, otherwise the default will be CLASS.SYMBOL=0 (i.e. a 1 pixel dot)

It is also possible (with a bit of work) to control which symbol gets used in rendering point symbols. OGR provides MapServer with symbol names, and if the symbol name returned by OGR to MapServer matches the name of one of the symbols in your symbolset then this symbol will be used.

For MapInfo point symbols (numbered 32 to 67 in the MapInfo MIF spec), the name returned by OGR is “mapinfo-sym-X” where X should be replaced with the MapInfo symbol number (e.g. “mapinfo-sym-35” is the star symbol).

Text labels The text string is mapped to CLASS.TEXT

Text color is mapped to CLASS.LABEL.COLOR

Text background color is mapped to CLASS.LABEL.BACKGROUNDCOLOR

Text height is mapped to CLASS.LABEL.SIZE

Text angle is mapped to CLASS.LABEL.ANGLE

Text font mapping follows the following rules:

1. If TTF fonts are supported:
 - (a) If the native font name (e.g. “Arial”) is found in your fontset then this font will be used.
 - (b) If 1a. failed and a font called “default” is present in your fontset then this “default” font will be used.
2. If TTF fonts are not supported or if all above cases failed, then BITMAP MEDIUM font will be used.

Accessing OGR STYLEITEMAUTO Label Styles Through MapScript

OGR STYLEITEMAUTO label styles can be accessed through MapScript, such as PHP/MapScript's `getshape()` or `getvalue()` methods, by setting the LAYER's PROCESSING parameter to "GETSHAPE_STYLE_ITEMS=all". Therefore, the LAYER may contain:

```
LAYER
...
PROCESSING "GETSHAPE_STYLE_ITEMS=all"
...
END
```

The following label styles are supported:

Label Style	Description	MapServer Version Implemented
OGR:LabelFont	Comma-delimited list of fonts names	5.4
OGR:LabelSize	Numeric value with units	5.2.0
OGR:LabelText	Label text string	5.2.0
OGR:LabelAngle	Rotation angle (in degrees)	5.2.0
OGR:LabelFColor	Foreground color	5.4
OGR:LabelBColor	Background color	5.4
OGR:LabelPlacement	How is the text drawn relative to the feature's geometry	5.4
OGR:LabelAnchor	A value from 1 to 12 defining the label's position relative to the point to which it is attached.	5.4
OGR:LabelDx	X offset	5.4
OGR:LabelDy	Y offset	5.4
OGR:LabelPerp	Perpendicular offset	5.4
OGR:LabelBold	Bold text	5.4
OGR:LabelItalic	Italic text	5.4
OGR:LabelUnderline	Underlined text	5.4
OGR:LabelPriority	Numeric value defining the order in which style parts should be drawn.	5.4
OGR:LabelStrikeout	Strike out text (gdal >= 1.4.0)	5.4
OGR:LabelStretch	Stretch factor changes the width of all characters in the font by factor percent. (gdal >= 1.4.0)	5.4
OGR:LabelAdjHor	Horizontally adjacent text (gdal >= 1.4.0)	5.4
OGR:LabelAdjVer	Vertically adjacent text (gdal >= 1.4.0)	5.4
OGR:LabelHColor	Shadow color (gdal >= 1.4.0)	5.4
OGR:LabelOColor	Outline color (gdal > 1.6.0)	5.4

Please see the [OGR Feature Style Specification](#) document for more details on those specific styles.

Sample Sites Using OGR/MapServer

The following sites use OGR's STYLEITEM "AUTO" feature:

- http://www2.dmsolutions.ca/msapps/yk_demo/demo_init.html
- http://www2.dmsolutions.ca/msapps/nfld_demo/demo_init.html

The following site uses OGR, as well as MapInfo's 'Seamless Map Layers' feature:

- http://www2.dmsolutions.ca/msapps/ro_demo/demo_init.html

The following site uses OGR to display TIGER 2000 files:

- http://www2.dmsolutions.on.ca/msapps/tig_demo/demo_init.html

FAQ / Common Problems

Q What Does “OGR” Stand For?

A Basically, OGR does not stand for anything. For a detailed explanation of how OGR was named, see GDAL’s FAQ at <http://trac.osgeo.org/gdal/wiki/FAQ>.

Q When using STYLEITEM AUTO, what should I have in my .sym symbols file?

A When you use STYLEITEM AUTO, MapServer tries to match symbol names returned by OGR to names in your symbol file. For a quick solution, try using the following symbol file:

http://www2.dmsolutions.ca/msapps/yk_demo/etc/symbols_mapinfo.sym

The name of the symbols returned by OGR to MapServer depends on the file format. In the case of MapInfo files, it will be:

- For “old-style” symbols (default MapInfo 3.0 symbols numbered 32 to 67) the symbol name will be ‘mapinfo-sym-##’ where ‘##’ is the symbol number, e.g. ‘mapinfo-sym-32’.
- For “Font Symbols”, the symbol name is also ‘mapinfo-sym-##’ where ‘##’ is the symbol number in the font. In this case, the name of the font itself is ignored by MapServer.
- MapInfo also supports “custom symbols” (bitmap symbols)... I’m not sure what you would get from OGR for this, but I’m pretty sure that MapServer doesn’t do anything useful with them.

The OGRINFO utility can be used to find out exactly which symbol names OGR will return to MapServer. Look at the “Style” string in the ogrinfo output for each shape that is read.

7.1.5 MapInfo

File listing

The following files are also associated with .TAB files: .DAT, .ID, .MAP. An example is:

```
border.DAT
border.ID
border.MAP
border.TAB
```

The term MID/MIF refers to files with .MID and .MIF extension.

Data Access / Connection Method

TAB and MID/MIF access is available in MapServer through OGR.

- The CONNECTIONTYPE OGR parameter must be used.
- The path to the (*.tab or *.mif) file is required, and the file extension is needed.
- The path may be relative to the SHAPEPATH
- MapInfo files already contain styling information. This styling information can be used optionally by specifying the STYLEITEM “AUTO” parameter in the LAYER object of the map file.

Note: If you use STYLEITEM “AUTO” you must have an empty class in the layer.

OGRINFO Examples

Using ogrinfo on a single TAB file

```
> ogrinfo elev5_poly.TAB
Had to open data source read-only.
INFO: Open of `elev5_poly.TAB'
using driver `MapInfo File' successful.
1: elev5_poly (Polygon)
```

Using ogrinfo to examine the structure of the file/layer

```
> ogrinfo elev5_poly.TAB elev5_poly
Had to open data source read-only.
INFO: Open of `elev5_poly.TAB'
using driver `MapInfo File' successful.

Layer name: elev5_poly
Geometry: Polygon
Feature Count: 2236
Extent: (-141.000000, 60.000000) - (-124.403310, 69.300251)
Layer SRS WKT:
GEOGCS["unnamed",
DATUM["MIF 0",
    SPHEROID["WGS 84 (MAPINFO Datum 0)",6378137.01,298.257223563],
    TOWGS84[0,0,0,0,0,0,0]],
PRIMEM["Greenwich",0],
UNIT["degree",0.0174532925199433]]
AREA: Real (0.0)
PERIMETER: Real (0.0)
ELEV5_: Integer (0.0)
ELEV5_ID: Integer (0.0)
TYPE: Real (4.0)
ELEV5: Real (4.0)
...
```

Map File Example

```
LAYER
NAME Elevation_Poly_5
TYPE POLYGON
STATUS DEFAULT
CONNECTIONTYPE OGR
CONNECTION "../hypso/elev5_poly.TAB"
STYLEITEM "AUTO"
CLASS
    NAME "Elevation Poly 5"
END
END # Layer
```

7.1.6 WFS

WFS is an Open Geospatial Consortium (OGC) specification. For more information about the format itself, see: <http://www.opengeospatial.org/standards/wfs>

WFS allows a client to retrieve geospatial data encoded in Geography Markup Language (GML) from multiple Web Feature Services. GML is built on the standard web language XML.

WFS differs from the popular Web Map Service (WMS) specification in that WFS returns a subset of the data in valid GML format, not just a graphic image of data.

Capabilities

Requesting the capabilities using the GetCapabilities request to a WFS server returns an XML document showing what layers and projections are available, etc. Example of a WFS GetCapabilities URL:

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?VERSION=1.0.0&SERVICE=wfs&REQUEST=GetCapabilities
```

Example of the Resulting XML from GetCapabilities:

```
...
<FeatureTypeList>
<Operations>
  <Query/>
</Operations>
<FeatureType>
  <Name>park</Name>
  <Title>Parks</Title>
  <SRS>EPSG:42304</SRS>
  <LatLongBoundingBox minx="-173.433" miny="41.4271" maxx="-13.0481" maxy="83.7466" />
</FeatureType>
<FeatureType>
  <Name>road</Name>
  <Title>Roads</Title>
  <SRS>EPSG:42304</SRS>
  <LatLongBoundingBox minx="-148.059" miny="35.882" maxx="-33.7745" maxy="72.5503" />
</FeatureType>
<FeatureType>
  <Name>popplace</Name>
  <Title>Cities</Title>
  <SRS>EPSG:42304</SRS>
  <LatLongBoundingBox minx="-172.301" miny="36.3541" maxx="-12.9698" maxy="83.4832" />
</FeatureType>
</FeatureTypeList>
...
```

Data Access / Connection Method

- WFS access is a core MapServer feature. MapServer currently supports WFS version 1.0.0
- The CONNECTIONTYPE WFS parameter must be used.
- WFS layers can be requested through a layer in a map file, or you can request the GML directly through the browser with a GetFeature request. You can specify a specific layer with the TypeName request. In a map file the name/value pairs should be put into a METADATA object.
- You can limit the number of features returned in the GML by using the MaxFeatures option (e.g. &MAXFEATURES=100).

Example of a WFS Request Directly Through the Browser:

The following URL requests the GML for the layer road. (see the GetCapabilities above for the possible layers available on this test server) . The URL is all one line, broken up here for readability.

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap
?VERSION=1.0.0
&SERVICE=wfs
&REQUEST=getfeature&TYPENAME=road
```

Map File Example:

```
LAYER
NAME "wfs_gmap_roads"
STATUS DEFAULT
TYPE LINE
CONNECTIONTYPE WFS
CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?"
METADATA
    "wfs_version" "1.0.0"
    "wfs_srs" "EPSG:42304"
    "wfs_typename" "road"
    "wfs_request_method" "GET"
    "wfs_service" "WFS"
END
CLASS
    NAME "roads"
    STYLE
        COLOR 0 0 0
    END
END
END # layer
```

7.1.7 GML

Also known as Geographic Markup Language and GML/XML. GML is a text-based, XML format that can represent vector and attribute data. This is an Open Geospatial Consortium specification for data interchange. More information is available at <http://www.opengeospatial.org/standards/gml>

File listing

GML files are usually a single text file with a GML filename extension. Some may use XML as the filename extension:

```
coal_dep.gml
```

XML schema documents often accompany GML files that have been translated from some other format (e.g. using the ogr2ogr utility).

GML uses sets of nested tags to define attributes and geometry coordinates. Example of text in a GML file:

```
<gml:featureMember>
<Coal_Deposits fid="1">
<UNKNOWN>0.000</UNKNOWN>
<NA>0.000</NA>
<ID>2</ID>
<ID2>2</ID2>
<MARK>7</MARK>
<COALKEY>110</COALKEY>
<COALKEY2>110</COALKEY2>
```

```

<ogr:geometryProperty>
<gml:Point>
<gml:coordinates>78.531,50.694</gml:coordinates>
</gml:Point>
</ogr:geometryProperty>
</Coal_Deposits>
</gml:featureMember>

```

Data Access / Connection Method

- GML access is available in MapServer through OGR. More information on OGR GML support is available at http://www.gdal.org/ogr/drv_gml.html
- The CONNECTIONTYPE OGR parameter must be used.
- The path to the GML file is required, including file extension. There can be multiple layers in a GML file, including multiple feature types.

OGRINFO Examples

Using ogrinfo on a single GML file:

```

> ogrinfo /data/gml/coal_dep.gml
Had to open data source read-only.
INFO: Open of 'coal_dep.gml'
using driver 'GML' successful.
1: Coal_Deposits

```

Using ogrinfo to examine the structure of one layer:

```

> ogrinfo -summary /data/gml/coal_dep.gml Coal_Deposits
Had to open data source read-only.
INFO: Open of 'coal_dep.gml'
using driver 'GML' successful.

```

```

Layer name: Coal_Deposits
Geometry: Unknown (any)
Feature Count: 266
Extent: (23.293650, 37.986340) - (179.272550, 80.969670)
Layer SRS WKT:
(unknown)
UNKNOWN: Real (0.0)
NA: Real (0.0)
ID: Integer (0.0)
ID2: Integer (0.0)
MARK: Integer (0.0)
COALKEY: Integer (0.0)
COALKEY2: Integer (0.0)
LONG: Real (0.0)
LAT: Real (0.0)

```

Map File Example:

```
LAYER
NAME coal_deposits
TYPE POINT
STATUS DEFAULT
CONNECTIONTYPE OGR
CONNECTION "gml/coal_dep.gml"
CLASS
    STYLE
        COLOR 0 0 0
        SYMBOL 'circle'
        SIZE 6
    END
END
END
```

7.1.8 Virtual Spatial Data

Table of Contents

- Virtual Spatial Data
 - Types of Databases
 - Types of Flat Files
 - Steps for Display

This is an OGR extension to MapServer. It allows you to connect to databases that do not explicitly hold spatial data, as well as flat text files. Your data must have an X and a Y column, and the data may be accessed through an ODBC connection or a direct pointer to a text file.

The original [VirtualSpatialData](#) wiki page may contain additional information.

Types of Databases

The VirtualSpatialData OGR extension has been tested with the following databases and should, in theory, support all ODBC data sources.

- Oracle
- MySQL
- SQL Server
- Access
- PostgreSQL

Types of Flat Files

Comma, tab or custom delimited text/flat files work with VirtualSpatialData.

Steps for Display

1. Create the Datasource Name (DSN)

- Specific notes about creating a DSN on Windows and Linux can be found by searching the MapServer reference documents site
- On some Windows systems you must create a SYSTEM DSN.

2. Test your Connection

Test your connection with ogrinfo. The syntax for this command is:

```
> ogrinfo ODBC:user/pass@DSN table
```

Windows users may not be required to specify a user/password, so the syntax would be:

```
> ogrinfo ODBC:@DSN table
```

Example: Accessing a comma separated text file through ODBC using ogrinfo

The following is a snippet of the flat text file coal_dep.txt containing lat/long points:

```
unknown,na,id,id2,mark,coalkey,coalkey2,long,lat
0.000,0.000,1,1,7,87,87,76.90238,51.07161
0.000,0.000,2,2,7,110,110,78.53851,50.69403
0.000,0.000,3,3,3,112,112,83.22586,71.24420
0.000,0.000,4,4,6,114,114,80.79896,73.41175
```

If the DSN name is Data_txt, the ogrinfo command to see a list of applicable files in the directory is:

```
> ogrinfo ODBC:jeff/test@Data_txt
INFO: Open of 'ODBC:jeff/test@Data_txt'
using driver 'ODBC' successful.
1: coal_dep.csv
2: coal_dep.txt
3: coal_dep_nf.txt
4: coal_dep_trim.txt
5: Copy of coal_dep.txt
6: deposit.csv
7: maruia.asc
8: oahuGISbathy.csv
9: oahuGISbathy.txt
10: on_pts.txt
11: on_pts_utm.txt
12: test.txt
13: utm_test.txt
```

Username and password may be optional, so the following may also be valid:

```
> ogrinfo ODBC:@Data_txt
```

Therefore, the command to see more information about one of the specific layers is:

```
> ogrinfo ODBC:@Data_txt coal_dep.txt
INFO: Open of 'ODBC:@Data_txt'
using driver 'ODBC' successful.
```

```
Layer name: coal_dep.txt
Geometry: Unknown (any)
Feature Count: 266
Layer SRS WKT:
(unknown)
UNKNOWN: String (255.0)
NA: String (255.0)
ID: String (255.0)
ID2: String (255.0)
MARK: String (255.0)
COALKEY: String (255.0)
COALKEY2: String (255.0)
LONG: String (255.0)
LAT: String (255.0)
OGRFeature(coal_dep.txt):0
UNKNOWN (String) = 0.000
....
```

3. Create a Virtual Data File

This is a file with an ovf extension and looks like the following:

```
<OGRVRTDataSource>
  <OGRVRTLayer name="mylayer">
    <SrcDataSource>ODBC:user/pass@DSN</SrcDataSource>
    <SrcLayer>tablename</SrcLayer>
    <GeometryType>wkbPoint</GeometryType>
    <LayerSRS>WGS84</LayerSRS>
    <GeometryField encoding="PointFromColumns" x="x" y="y"/>
  </OGRVRTLayer>
</OGRVRTDataSource>
```

More information on ovf files can be found at: http://www.gdal.org/ogr/drv_vrt.html

Example ovf file for coal_dep.txt:

```
<OGRVRTDataSource>
  <OGRVRTLayer name="coal-test">
    <SrcDataSource>ODBC:Data_txt</SrcDataSource>
    <SrcLayer>coal_dep.txt</SrcLayer>
    <GeometryField encoding="PointFromColumns" x="Long" y="Lat"/>
    <GeometryType>wkbPoint</GeometryType>
  </OGRVRTLayer>
</OGRVRTDataSource>
```

4. Test Virtual Data File with ogrinfo

Use ogrinfo to test your new ovf file, such as:


```
> ogrinfo coal.ovf coal-test
ERROR 4: Update access not supported for VRT datasources.
Had to open data source read-only.
INFO: Open of 'myfile.ovf'
using driver 'VRT' successful.
```

```
Layer name: coal_dep.txt
Geometry: Unknown (any)
Feature Count: 266
Layer SRS WKT:
(unknown)
UNKNOWN: String (255.0)
NA: String (255.0)
ID: String (255.0)
ID2: String (255.0)
MARK: String (255.0)
...
```

5. Mapfile Layer

Using an ovf file your layer may look like:

```
LAYER
  CONNECTION "coal.ovf"
  CONNECTIONTYPE OGR
  DATA "coal-test"
  METADATA
    "wms_srs" "4326"
    "wms_title" "coal-test"
  END
  NAME "coal-test"
  SIZEUNITS PIXELS
  STATUS ON
  TOLERANCE 0
  TOLERANCEUNITS PIXELS
  TYPE POINT
  UNITS METERS
  CLASS
    STYLE
      COLOR 255 0 0
      MAXSIZE 100
      MINSIZE 1
      SIZE 6
      SYMBOL "star"
    END
  END
END
```

Or you may specify the ovf contents inline such as:

```
LAYER
  CONNECTION "<OGRVRTDataSource>"
  <OGRVRTLayer name=' coal-test'>
  <SrcDataSource>ODBC:@Data_txt</SrcDataSource>
  <SrcLayer>coal_dep.txt</SrcLayer>
```

```
<GeometryField encoding='PointFromColumns' x='Long' y='Lat' />
<GeometryType>wkbPoint</GeometryType>
</OGRVRTLayer>
</OGRVRTDataSource>"
CONNECTIONTYPE OGR
DATA "coal-test"
METADATA
    "wms_srs"    "4326"
    "wms_title"  "coal-test"
END
NAME "coal-test"
SIZEUNITS PIXELS
STATUS ON
TOLERANCE 0
TOLERANCEUNITS PIXELS
TYPE POINT
UNITS METERS
CLASS
    STYLE
        COLOR 255 0 0
        MAXSIZE 100
        MINSIZE 1
        SIZE 6
        SYMBOL "star"
    END
END
END
```

6. Test your Mapfile

The first thing you should try is to use the *shp2img* utility:

```
shp2img -m mymapfile.map -o test.png
```

Once you successfully created a map image, then try your application. Note Windows users may come across a problem where *shp2img* works but their application throws an error similar to this:

```
Warning: [MapServer Error]: msOGRFileOpen(): Open failed for OGR connection 'coal.ovf'.
Unable to initialize ODBC connection to DSN for jeff/test@Data_txt,
[Microsoft][ODBC Driver Manager] Data source name not found
and no default driver specified in D:\ms4w\Apache\htdocs\quickmap.php on line 40
```

If that happens you should make sure you have created a System DSN.

7.1.9 ArcInfo

ESRI ArcInfo Coverage Files are also known as simply as Coverages and less commonly as ADF files.

File listing

Coverages are made up of a set of files within a folder. The folder itself is the coverage name. The files roughly represent different layers, usually representing different types of topology or feature types.

```
> ls /data/coverage/brazil
aat.adf  arc.adf  arx.adf  bnd.adf  lab.adf  prj.adf  tic.adf  tol.adf
```

A folder with the name INFO is also part of the coverage. It sits at the same hierarchical level as the coverage folder itself. Therefore, to copy a coverage (using regular file system tools) the coverage folder and the INFO folder must both be copied. The INFO folder holds some catalogue information about the coverage.

```
> ls /data/coverage/info
arc0000.dat  arc0001.dat  arc0002.dat  arc.dir
arc0000.nit  arc0001.nit  arc0002.nit
```

Data Access / Connection Method

- CONNECTIONTYPE OGR must be used. The ability to use coverages is not built into MapServer.
- The path to the coverage folder name is required.
- The layer name (feature type) is specified in the DATA parameter

OGRINFO Examples

The directory is the data source. Layers are found within the directory. Using ogrinfo on a coverage directory:

```
> ogrinfo /data/coverage/brazil -summary
INFO: Open of 'brazil'
using driver 'AVCBin' successful.
1: ARC (Line String)
2: CNT (Point)
3: LAB (Point)
4: PAL (Polygon)
```

Using ogrinfo to examine the structure of a layer:

```
> ogrinfo /data/coverage/brazil PAL -summary
Had to open data source read-only.
INFO: Open of 'brazil'
using driver 'AVCBin' successful.

Layer name: PAL
Geometry: Polygon
Feature Count: 1
Extent: (1272793.274958, 795381.617050) - (1287078.382785, 807302.747284)
Layer SRS WKT:
(unknown)
ArcIds: IntegerList (0.0)
AREA: Real (18.5)
PERIMETER: Real (18.5)
F_OPER#: Integer (5.0)
F_OPER-ID: Integer (5.0)
OPER: String (2.0)
FCODE: String (10.0)
```

Map File Example:

```
LAYER
  NAME Brazil_bounds
  TYPE POLYGON
  STATUS DEFAULT
  CONNECTIONTYPE OGR
  CONNECTION "/data/coverage/brazil"
  DATA "PAL"
  CLASS
    NAME "Brazil Admin Areas"
    STYLE
      OUTLINECOLOR 153 102 0
      SIZE 2
    END
  END
END
```

7.1.10 ArcSDE

Spatial Database Engine (SDE) is one of ESRI's products which enables spatial data to be stored, managed, and quickly retrieved from leading commercial database management systems like Oracle, Microsoft SQL Server, Sybase, IBM DB2, and Informix.

Supported ArcSDE Operations

- Versioned queries (query geometry and attributes from a specified version)
- queryByAttributes (select geometry and attributes based on the values of an attribute)
- Limited join support for within-database tables
- queryByRect (select geometry based on an extent)
- Projection on the fly
- SDE for Coverages (a read-only type of SDE for coverage, shapefile, and ArcStorm/ArcLibrarian repositories)
- SDE 8.1, 8.2, 8.3, 9.0, 9.1, and 9.2
- Linux, Windows, and Solaris (platforms that have SDE C API support)

Unsupported ArcSDE Operations

- queryByShape (pass in a shape with MapScript and use it for queries)
- Direct Connect (bypass SDE to connect directly to the database with the SDE C API)

How to make a connection to SDE:

- Install the SDE C API client libraries for your platform (preferably matched to the server version you are using, ie 8.2 client -> 8.2 server, 8.3 client -> 8.3 server)
- Compile MapServer with SDE support *MapServer Unix Compilation Howto* for specific details)
- Define a LAYER block in a MapFile that uses SDE as the CONNECTIONTYPE

```

LAYER
  NAME          states
  TYPE          POLYGON
  CONNECTION    "sdemachine.iastate.edu,port:5151,sde,username,password"
  CONNECTIONTYPE SDE
  DATA        "HOBU.STATES_LAYER,SHAPE,SDE.DEFAULT"
  FILTER       "where MYCOLUMN is not NULL"
  PROCESSING   "QUERYORDER=ATTRIBUTE" # <-- MapServer 4.10 and above

  # Within database one-to-one join support

  # MapServer 5.0 and above
  PROCESSING   "JOINTABLE=SDE_MASTER.GEOSERVWRITE.JOINTABLE"

  # MapServer 5.0 and above
  CLASSITEM   "SDE_MASTER.GEOSERVWRITE.JOINTABLE.VAL"

  # MapServer 5.0 and above
  FILTER      "SDE_MASTER.GEOSERVWRITE.JOINTABLE.AQ_TAG=SDE_MASTER.GEOSERVWRITE.JOINTESTLAYER.AQ_TAG"

  # ObjectID column manipulation
  # MapServer 5.0 and above
  PROCESSING   "OBJECTID=OBJECTID"

  TEMPLATE    '/where/the/template/file/is/located'
  CLASS
    STYLE
      SYMBOL 'circle'
      SIZE 3
      COLOR -1 -1 -1
      OUTLINECOLOR 0 0 0
    END
  END
END

```

CONNECTION - Order is important!

- **sdemachine.iastate.edu** - The name of the machine you are connecting to. In some instances, this may need to be the IP address of the machine rather than the name if the server running MapServer is not configured to cascade DNS lookups
- **port:5151** - The port number of SDE. The *port:* is important as SDE expects you to define the **service** in this slot, and it can be other names like **sde:oracle** (for direct connect) or **esri_sde** (for systems with port 5151 defined as *esri_sde* in */etc/services*)
- **sde** - The database username that the SDE server is using to connect to your database. It is often only important for SDE setups that are connecting to Oracle (and even then, not so important). Just leave it as **sde** if you don't know what it should be.
- **username** - The username that will be connecting to SDE. This user must have been granted rights to select the layer that you will be specifying in the *DATA* directive. You can use ArcCatalog or the SDE command-line utilities to grant the appropriate rights to layers.
- **password** - Password of the user connecting to SDE. **Case Sensitive.**

DATA - Order is important!

- **HOBU.STATES_LAYER** - The layer name you are querying. This the *full* name of the table in which the layer resides. If you are using Oracle or Microsoft SQL Server as the DB for SDE, the schema name must also be supplied.
- **SHAPE** - The column that contains the geometry. SDE technically allows for storage of multiple geometry types in the same layer, but in practice this isn't desirable. Also, expect to have problems if there are invalid or null geometries in the layer (or versions of the layer).
- **SDE.DEFAULT** - As of MapServer 4.2, you can query against a specific version of the layer. SDE supports multi-user editing with versions. If a layer has been Registered with the GeoDatabase and Registered as Versioned (ArcGIS terms), MapServer can query against specified versions of those edits. If not specified, *SDE.DEFAULT* will be used for all queries. **Case Sensitive**.

Note: The version parameter is located in a different spot than MapServer 4.2, which had it on the CONNECTION string.

TEMPLATE

- **/where/the/template/file/is/located** - A template directive must be specified (can point to a dummy file) in order for MapServer to be able to query attributes from SDE. If you are only going to be drawing layers, this directive is unnecessary and will slow down the query operations of SDE (especially for layers with lots of attribute columns).

PROCESSING

- **PROCESSING "QUERYORDER=ATTRIBUTE"** - Allows you to force SDE to use the WHERE clause that was defined in your FILTER statement first, without attempting to hit the spatial index. Only in very special cases will you want to do this.
- **PROCESSING "OBJECTID=OBJECTID"** - If you are having trouble with the SDE driver detecting your unique ID column, you can override it with this processing parameter. Doing so will also have a slight performance benefit because it will save a couple of extra queries to the database.
- **PROCESSING "ATTRIBUTE_QUALIFIED=TRUE"** - User can set this option to always use fully qualified attribute names.

Within-database Join Support

MapServer's SDE driver, as of MapServer 5.0, allows you to join a single attribute table that has no geometries to the layer that you are rendering. This feature allows you to use the data in the joined table much as you would in a composite query that was made with something like PostGIS or Oracle Spatial. That is, the columns in the right table of the join are available for CLASSITEM, LABELITEM and so on. The biggest constraint, however, is that **fully qualified** names must be used or it most likely will not work. The join support is activated through PROCESSING options.

- **PROCESSING "JOINTABLE=SDE_MASTER.GEOSERVWRITE.JOINTABLE"** - The JOINTABLE processing option tells the driver which table you are joining the current layer to.
- **CLASSITEM "SDE_MASTER.GEOSERVWRITE.JOINTABLE.VAL"** - A CLASSITEM or LABELITEM for a joined table using this mechanism must be fully qualified.

- **FILTER** “SDE_MASTER.GEOSERVWRITE.JOINTABLE.AQ_TAG=SDE_MASTER.GEOSERVWRITE.JOINTESTL/”
- An important part of the join is defining how the join is to be made. Use a FILTER to do so.

7.1.11 DGN

File listing

Data are encapsulated in a single file, usually with the suffix .dgn.

```
0824t.dgn
```

Data Access / Connection Method

- Access is available in MapServer through OGR.
- The CONNECTIONTYPE OGR parameter must be used.
- The path to the dgn file is required, file extension is needed.
- All types of features in a DGN file are held in one “layer” of data. The layer is called elements and is the first and only layer.
- The type of feature to be read from the DGN depends on the TYPE parameter in the map file.
- DGN files typically contain POINT, LINE, POLYGON and ANNOTATION feature types.
- DGN files contain “styling” information - how to color and present the data. This is used, optionally, by specifying the STYLEITEM “AUTO” parameter.

Note: DGN files typically use white as a color for their features and therefore are not visible on maps with white backgrounds.

OGRINFO Examples

Using ogrinfo on a single DGN file:

```
> ogrinfo /data/dgn/0824t.dgn
Had to open data source read-only.
INFO: Open of `0842t.dgn'
using driver `DGN' successful.
1: elements
```

Note: No geometry/feature type for the layer is identified because it can be multiple types.

DGN files are not really GIS data files. They evolved from drafting formats used by computer aided drafting/design (CADD) programs.

They carry a few key attributes which are usually consistent across all DGN files. Most of the attributes relate to graphical styling of features for map presentation, such as ColorIndex, Style, etc.

Spatial reference system information is not always encoded into DGN files. This can be a major problem when trying to adequately reference the DGN data in another mapping program.

Measurement units can be a problem. In some cases the features could be located in kilometres or feet even though it is not obvious from the output of ogrinfo. Sometimes the only way to identify or correct a problem with units is to open the file in Microstation software.

Using ogrinfo to examine the structure of the file/layer:

```
> ogrinfo -summary /data/dgn/0824t.dgn elements
INFO: Open of '0824t.dgn'
using driver 'DGN' successful.

Layer name: elements
Geometry: Unknown (any)
Feature Count: 22685
Extent: (-513183.050000, 150292.930000) - (-224583.220000, 407463.360000)
Layer SRS WKT:
(unknown)
Type: Integer (2.0)
Level: Integer (2.0)
GraphicGroup: Integer (4.0)
ColorIndex: Integer (3.0)
Weight: Integer (2.0)
Style: Integer (1.0)
EntityNum: Integer (8.0)
MSLink: Integer (10.0)
Text: String (0.0)
```

Map File Example:

```
LAYER
  NAME dgn
  TYPE LINE
  STATUS DEFAULT
  CONNECTIONTYPE OGR
  CONNECTION "dgn/0824t.dgn"
  STYLEITEM "AUTO"
  CLASS
  END
END # Layer
```

7.1.12 S57

Also known as S57. The IHO S-57 format is a vector interchange format used for maritime charts. It was developed by the International Hydrographic Organisation (IHO). For more information about the IHO see: <http://www.iho.shom.fr/>

File listing

Individual S57 data files have an extension of *.000. For example:

```
US1BS02M.000
```

Data Access / Connection Method

- S57 access in MapServer occurs through OGR, CONNECTIONTYPE OGR must be used.
- Specify a full path or a relative path from the SHAPEPATH to the .000 file for the CONNECTION
- Use the DATA parameter to specify the s57 layer name

Special Notes

The underlying OGR code requires two files from your GDAL/OGR installation when reading S57 data in MapServer: `s57objectclasses.csv` and `s57attributes.csv`. These files can be found in the `/GDAL/data/` folder (unix: `/usr/local/share/gdal` windows: `/ms4w/gdaldata`). If you receive an error in MapServer such as:

```
msDrawMap(): Image handling error. Failed to draw layer named 's57'.
msOGRFileOpen(): OGR error. xxx failed for OGR connection
```

you may have to point MapServer to these files using the `CONFIG` parameter in the main section of your map file:

```
CONFIG GDAL_DATA "C:\ms4w\gdaldata"
```

OGRINFO Examples

Using `ogrinfo` on an S57 file to get the layer name:

```
> ogrinfo uslbs02m.000
ERROR 4: S57 Driver doesn't support update.
Had to open data source read-only.
INFO: Open of 'uslbs02m.000'
using driver 'IHO S-57 (ENC)' successful.
1: ADMARE (Polygon)
2: CBLSUB (Line String)
3: CTNARE
4: COALNE (Line String)
5: DEPARE
6: DEPCNT (Line String)
7: LNDARE
8: LNDELV
9: LNDRGN
10: LNDMRK
11: LIGHTS (Point)
12: OBSTRN
13: RDOSTA (Point)
14: SEAARE
15: SBDARE
16: SILCONS
17: SOUNDG (Multi Point)
18: UWTROC (Point)
19: WATTUR
20: WRECKS
21: M_COVR (Polygon)
22: M_NPUB (Polygon)
23: M_NSYS (Polygon)
24: M_QUAL (Polygon)
25: C_ASSO (None)
```

Using `ogrinfo` to examine the structure of an S57 layer:

```
> ogrinfo uslbs02m.000 DEPARE -summary
ERROR 4: S57 Driver doesn't support update.
Had to open data source read-only.
INFO: Open of 'uslbs02m.000'
using driver 'IHO S-57 (ENC)' successful.
```

```
Layer name: DEPARE
Geometry: Unknown (any)
Feature Count: 297
Extent: (165.666667, 48.500000) - (180.000000, 60.750000)
Layer SRS WKT:
GEOGCS["WGS 84",
DATUM["WGS_1984",
    SPHEROID["WGS 84",6378137,298.257223563]],
PRIMEM["Greenwich",0],
UNIT["degree",0.0174532925199433]]
GRUP: Integer (3.0)
OBJL: Integer (5.0)
RVER: Integer (3.0)
AGEN: Integer (2.0)
FIDN: Integer (10.0)
FIDS: Integer (5.0)
LNAM: String (16.0)
LNAM_REFS: StringList (16.0)
DRVAL1: Real (0.0)
DRVAL2: Real (0.0)
QUASOU: String (0.0)
SOUACC: Real (0.0)
VERDAT: Integer (0.0)
INFORM: String (0.0)
NINFOM: String (0.0)
NXTDS: String (0.0)
SCAMAX: Integer (0.0)
SCAMIN: Integer (0.0)
TXTDSC: String (0.0)
RECDAT: String (0.0)
RECIND: String (0.0)
...
```

Map File Example:

```
LAYER
NAME s57
TYPE POLYGON
STATUS DEFAULT
CONNECTIONTYPE OGR
CONNECTION "./s57/us1bs02m.000"
DATA "DEPARE"
CLASS
STYLE
COLOR 247 237 219
OUTLINECOLOR 120 120 120
END
END
END # Layer
```

7.1.13 ESRI Personal Geodatabase (MDB)

ESRI Personal Geodatabases are basically Microsoft Access files that contain spatial information. For more information see the [ESRI description page](#).

File listing

Similar to other database formats, the mdb file consists of several tables. The geometry is held in a BLOB table column.

Data Access / Connection Method

Personal geodatabase access is available through OGR. See the [OGR driver page](#) for specific driver information. The driver is standard in any win32 build of GDAL/OGR version 1.3.2 or later. For Linux/Unix, [MDBTools](#) ODBC drivers can be used for this (with some difficulty).

OGR uses the names of spatial tables within the personal geodatabase (tables with a Shape column) as layers.

The CONNECTION parameter must include the mdb extension, and the DATA parameter should be the name of the spatial table (or OGR layer).

```
CONNECTIONTYPE ogr
CONNECTION "pgeodatabase.mdb"
DATA "layername"
```

OGRINFO Examples

First you should make sure that your GDAL/OGR build contains the personal geodatabase “PGeo” driver, by using the ‘-formats’ command:

```
>ogrinfo --formats
  Loaded OGR Format Drivers:
  ...
  -> "ODBC" (read/write)
  -> "PGeo" (readonly)
  -> "PostgreSQL" (read/write)
  ...
```

If you don’t have the driver, you might want to try the [FWTools](#) or [MS4W](#) packages, which include the driver.

Once you have the PGeo driver you are ready to try an ogrinfo command on your database to get a list of spatial tables:

```
>ogrinfo test.mdb
  INFO: Open of 'test.mdb'
  using driver 'PGeo' successful.
  1: counties
```

Now use ogrinfo to get information on the structure of the spatial table:

```
>ogrinfo test.mdb counties -summary
  INFO: Open of 'test.mdb'
  using driver 'PGeo' successful.

  Layer name: counties
  Geometry: Unknown (any)
  Feature Count: 67
  Extent: (-87.634943, 24.543945) - (-80.031369, 31.000975)
  Layer SRS WKT:
  GEOGCS["GCS_WGS_1984",
    DATUM["WGS_1984",
```

```
    SPHEROID["WGS_1984",6378137.0,298.257223563]],
    PRIMEM["Greenwich",0.0],
    UNIT["Degree",0.0174532925199433]]
OBJECTID_1: Integer (10.0)
OBJECTID: Integer (10.0)
NAME: String (32.0)
STATE_NAME: String (25.0)
STATE_FIPS: String (2.0)
CNTY_FIPS: String (3.0)
FIPS: String (5.0)
...
```

Note that you can also use an ODBC connection to access all of the tables in your geodatabase:

```
>ogrinfo ODBC:jeff/pass@testDSN counties -summary
INFO: Open of 'ODBC:jeff/pass@testDSN'
using driver 'ODBC' successful.

1: counties
2: counties_Shape_Index
3: GDB_AnnoSymbols
4: GDB_AttrRules
5: GDB_CodedDomains
6: GDB_DefaultValues
7: GDB_Domains
8: GDB_EdgeConnRules
9: GDB_Extensions
10: GDB_FeatureClasses
11: GDB_FeatureDataset
12: GDB_FieldInfo
13: GDB_GeomColumns
...
```

Mapfile Example

```
LAYER
  NAME my_geodatabase
  TYPE POLYGON
  CONNECTIONTYPE ogr
  CONNECTION "test.mdb"
  DATA "counties"
  STATUS ON
  CLASS
    NAME "counties"
    STYLE
      COLOR 255 255 120
    END
  END
END
```

7.1.14 Inline

Inline features refer to coordinates entered directly into the map file. They are not a file or database format and do not require any DATA or CONNECTION parameters. Instead they use a FEATURE section to define the coordinates.

Inline features can be used to define points, lines and polygons as if taken from an external file. This requires direct entry of coordinate pairs in the map file using a particular syntax.

Data Access / Connection Method

This is a native MapServer option that doesn't use any external libraries to support it.

Map File Example

Points

- Each FEATURE..END section defines a feature.
- Multiple points can be defined in a FEATURE section. If multiple points are defined in the same layer, they will have the same CLASS settings, e.g. for colours and styles.
- Coordinates are entered in the units set in the layer's projection. In this case it is assuming the map file projection is using decimal degrees.

```
LAYER
  NAME inline_stops
  TYPE POINT
  STATUS DEFAULT
  FEATURE
    POINTS
      72.36 33.82
    END
    TEXT "My House"
  END
  FEATURE
    POINTS
      69.43 35.15
      71.21 37.95
      72.02 38.60
    END
    TEXT "My Stores"
  END
  CLASS
    STYLE
      COLOR 0 0 250
      SYMBOL 'circle'
      SIZE 6
    END
  END
END
```

Lines

Lines are simply a list of points strung together, but the layer must be TYPE LINE instead of TYPE POINT.

```
LAYER
  NAME inline_track
  TYPE LINE
  STATUS DEFAULT
```

```
MAXSCALE 10000000
FEATURE
  POINTS
    72.36 33.82
    70.85 34.32
    69.43 35.15
    70.82 36.08
    70.90 37.05
    71.21 37.95
  END
END
CLASS
  STYLE
    COLOR 255 10 0
    SYMBOL 'circle'
    SIZE 2
  END
END
END
```

Polygons

Polygons are the same as the line example, just a list of points. They require the TYPE POLYGON parameter. Polygons also require the final coordinate pair to be the same as the first, making it a closed polygon.

7.1.15 KML - Keyhole Markup Language

Keyhole Markup Language (KML) is an XML-based language for managing the display of 3D geospatial data.

Data Access / Connection Method

KML access is available through OGR. See the [OGR driver page](#) for specific driver information. Read support was initially added to GDAL/OGR version 1.5.0.

The CONNECTION parameter must include the kml extension, and the DATA parameter should be the name of the layer.

```
CONNECTIONTYPE ogr
CONNECTION "myplaces.kml"
DATA "layername"
```

OGRINFO Examples

First you should make sure that your GDAL/OGR build contains the “KML” driver, by using the ‘–formats’ command:

```
>ogrinfo --formats
Loaded OGR Format Drivers:
...
-> "GML" (read/write)
-> "GPX" (read/write)
-> "KML" (read/write)
...
```

If you don't have the driver, you might want to try the [FWTools](#) or [MS4W](#) packages, which include the driver.

Once you have the KML driver you are ready to try an ogrinfo command on your file to get a list of available layers:

```
>ogrinfo myplaces.kml
INFO: Open of 'myplaces.kml'
using driver 'KML' successful.
1: Layer #0 (Point)
```

Now use ogrinfo to get information on the structure of the layer:

```
>ogrinfo fountains-hotel.kml "Layer #0" -summary
Had to open data source read-only.
INFO: Open of 'fountains-hotel.kml'
using driver 'KML' successful.

Layer name: Layer #0
Geometry: Point
Feature Count: 1
Extent: (18.424930, -33.919627) - (18.424930, -33.919627)
Layer SRS WKT:
GEOGCS["WGS 84",
  DATUM["WGS_1984",
    SPHEROID["WGS 84",6378137,298.257223563,
      AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0,
    AUTHORITY["EPSG","8901"]],
  UNIT["degree",0.01745329251994328,
    AUTHORITY["EPSG","9122"]],
  AUTHORITY["EPSG","4326"]]
Name: String (0.0)
Description: String (0.0)
```

Mapfile Example

```
LAYER
  NAME kml_places
  TYPE POINT
  STATUS DEFAULT
  CONNECTIONTYPE OGR
  CONNECTION "kml/fountains-hotel.kml"
  DATA "Layer #0"
  LABELITEM "NAME"
  CLASS
    NAME "My Places"
    STYLE
      COLOR 250 0 0
      OUTLINECOLOR 255 255 255
      SYMBOL 'circle'
      SIZE 6
    END
  LABEL
    SIZE TINY
    COLOR 0 0 0
    OUTLINECOLOR 255 255 255
```

```
POSITION AUTO
END
END
END
```

7.1.16 Oracle Spatial

Author Bart van den Eijnden

Last Updated 2005/12/12

Table of Contents

- Oracle Spatial
 - What MapServer 5.2 with Oracle Spatial
 - Binaries
 - Installation
 - Two options for using Oracle Spatial with MapServer
 - Mapfile syntax for native Oracle Spatial support
 - Using subselects in the DATA statement
 - Additional keywords - [FUNCTION]
 - Additional keywords - [VERSION]
 - More information
 - Example of a *LAYER*
 - *Mapfile* syntax for OGR Oracle Spatial support

Oracle Spatial is a spatial cartridge for the Oracle database. Remember that all Oracle databases come with Locator, which has less features than Oracle Spatial. The differences between Locator and Spatial can be found in the [Oracle Spatial FAQ](#).

You can also see the original [OracleSpatial wiki page](#) that this document was based on.

What MapServer 5.2 with Oracle Spatial

- mode=map
- query modes: query, nquery, itemnquery
- *MapScript* query functions such as querybyattributes
- *OGC:WMS*: GetCapabilities, GetMap, GetFeatureInfo, DescribeLayer
- *OGC:WFS*, GetCapabilities, DescribeFeatureType, GetFeature

Binaries

MapServer Windows plugins with Oracle spatial support can be downloaded from [MS4W](#). But you need Oracle client software in the server on which you are running MapServer. Oracle client software can be obtained for development purposes from the Oracle website, but you need to register, which by the way is free. The most recent version is Oracle Database 10g Release 1 Client. The ORACLE TECHNOLOGY NETWORK DEVELOPMENT LICENSE AGREEMENT applies to this software. The instant client will be satisfactory, and you can download the [instant client](#). Make sure though your MapServer is compiled against the same version as your Oracle client, for compiling you need a full client install, not just the instant client.

Installation

See *Oracle Installation* for more configuration and installation information for MapServer's native Oracle support

Note: If you receive error messages like "Error: .". It's likely related to MapServer being unable access or locate the ORACLE_HOME.

Two options for using Oracle Spatial with MapServer

Oracle Spatial layers in MapServer can be used through 2 interfaces:

- The native built-in support through `maporaclespatial.c`
- OGR, but watch out: OGR is not compiled with Oracle Spatial support so it won't work without compiling in OCI (Oracle client) yourself. This requires both recompiling GDAL/OGR as well as recompiling MapServer itself against the new GDAL/OGR !!!!

Mapfile syntax for native Oracle Spatial support

The DATA statement for a LAYER of CONNECTIONTYPE `oraclespatial` can now have 4 options. This change is backwards compatible, i.e. the old ways of specifying DATA still work. The new options are an extension to the old DATA statements, as they needed to include identification of the primary key to be used for the query modes (UNIQUE).

The following options are valid DATA statements:

```
"[geom_column]
FROM
[table] | ((
    SELECT [...]
    FROM [table] | [Spatial Operator]
    [WHERE condition] ) )
[USING [UNIQUE column]
| [SRID #srid]
| [FUNCTION]
| [VERSION #version]
]"
```

Example 1

The most simple DATA statement, in this case you only need to define one geometry column and one table. This option assumes you do not have an SRID defined.

```
LAYER
...
CONNECTIONTYPE oraclespatial
DATA "MYGEOMETRY FROM MYTABLE"
...
END
```

Example 2

It's composed of the first option plus the USING UNIQUE parameter. These new features are necessary when you want to use any query function. When it is used you must pass a numeric column type. This option assumes you do not have an SRID defined.

```
LAYER
...
  CONNECTIONTYPE oraclespatial
  DATA "MYGEOMETRY FROM MYTABLE USING UNIQUE MYTABLE_ID"
...
END
```

Example 3

This option is an extension to the first option. In this mode you must define the USING SRID parameter when the SRID value in your data is different from NULL.

```
LAYER
...
  CONNECTIONTYPE oraclespatial
  DATA "MYGEOMETRY FROM MYTABLE USING SRID 90112"
...
END
```

Example 4

This option is a combination of examples 2 and 3.

```
LAYER
...
  CONNECTIONTYPE oraclespatial
  DATA "MYGEOMETRY FROM MYTABLE USING UNIQUE MYTABLE_ID SRID 90112"
...
END
```

Using subselects in the DATA statement

It is possible to define the source of the data as a subselect and not only as a table. As source of data, used in FROM token, you can define any SQL, table, function, or operator that returns a SDO_GEOMETRY. For example:

```
DATA "[geom_column] FROM (SELECT [columns] FROM [table]|[Spatial function])"
```

If the LAYER definition contains a CLASSITEM, LABELITEM or FILTER, it is necessary that the fields used are returned by the query. When you define CLASSITEM you can use an expression without any problems.

Additional keywords - [FUNCTION]

You can add three keywords to the DATA statement for [FUNCTION] option that influence the query which will be executed in Oracle:

USING FILTER

```
"[geom_column] FROM [table]|([Subselect]) USING FILTER"
```

Using this keyword triggers MapServer to use the Oracle Spatial SDO_FILTER operator. This operator executes only the Oracle Spatial primary filter over your query data. In the Oracle User guide they explain: The primary filter compares geometric approximations, it returns a superset of exact result. The primary filter therefore should be as efficient (that is, selective yet fast) as possible. This operator uses the spatial index, so you need to define your spatial index correctly to retrieve an exact result. If the result of the query is not exact you can try the next option.

USING RELATE

```
"[geom_column] FROM [table]|([Subselect]) USING RELATE"
```

Using this keyword triggers MapServer to use the Oracle Spatial SDO_RELATE operator. This operator applies the primary and secondary Oracle Spatial filters. It's performance can be slightly slow but the result is extremely correct. You can use this mode when you want a perfect result or when you can't readjust the spatial index.

USING GEOMRELATE

```
"[geom_column] FROM [table]|([Subselect]) USING GEOMRELATE"
```

Using this keyword triggers MapServer to use the geometry function SDO_GEOM.RELATE, a function that searches the relations between geometries. SDO_GEOM.RELATE does not use the spatial index and your performance is more slow than operators but it's very accurate. You can use this mode when you can't use the spatial index or when it doesn't exist.

USING NONE

```
"[geom_column] FROM [table]|([Subselect]) USING NONE"
```

Using this keyword triggers MapServer to don't use any geometry function or spatial operator. So, the internal SQL don't restrict, bases in the extent, the data from source. All the data from source will be returned for MapServer. The NONE token is very useful when the source of the data don't contains any spatial index. It's usually occur when the source is a function like SDO_BUFFER, SDO_XOR, SDO_INTERSECTION..... So this mode is recommended when you can't use the spatial index or when it doesn't exist.

Additional keywords - [VERSION]

You can define what version of database you are using to improve the internal sql. This is very useful when using geodetic SRIDs and MapServer functions that retrieve the extent from data.

USING VERSION 8i

```
"[geom_column] FROM [table]|([Subselect]) USING VERSION 8i"
```

This indicates MapServer to use a internal SQL that it's compatible with Oracle 8i version.

USING VERSION 9i

```
"[geom_column] FROM [table]|([Subselect]) USING VERSION 9i"
```

The second indicates MapServer to use 9i version, is recommended to use this parameter if you are using 9i version because the internal SQL will use specific spatial functions that is need to retrieve data correctly from 9i Oracle Spatial versions.

USING VERSION 10g

```
"[geom_column] FROM [table]|([Subselect]) USING VERSION 10g"
```

This indicates MapServer to use a internal SQL that it's compatible with Oracle 10g version.

More information

- You can define any *PROJECTION* to your *LAYER* without problem, can be used for data with or without an SRID in Oracle.
- The native support for Oracle Spatial supports the defaults definition for SDO_GEOMETRY in database, the Oracle Spatial SDO package.
- Information about the primary and secondary Oracle Spatial filters can be found in the Oracle Spatial User Guide (the "Query Model" section). Information about the SDO_FILTER and SDO_RELATE operators can be found in the "Spatial Operators" section, and information about the SDO_GEOM.RELATE function can be found in the "Geometry Function" section of the Oracle Spatial User Guide.

Example of a *LAYER*

```
LAYER
  NAME kwadranten
  TYPE POLYGON
  CONNECTIONTYPE oraclespatial
  CONNECTION "user/pwd"
  DATA "GEOMETRIE FROM KWADRANTEN USING SRID 90112"
  DUMP TRUE
  CLASS
    STYLE
      OUTLINECOLOR 0 0 0
      COLOR 0 128 128
    END
  END
END
```

You can specify the SID for your database, the SID alias needs to be supplied in the tnsnames.ora file of the Oracle client, e.g.

Example for tnsnames.ora:

```

MYDB =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP) (HOST = server_ip) (PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = DB1)
    )
  )

```

So after this you can define you layer connection as:

```
CONNECTION "user/pwd@MYDB"
```

Mapfile syntax for OGR Oracle Spatial support

Syntax for your MAP file:

```

CONNECTION "OCI:user/pwd@service"
CONNECTIONTYPE OGR
DATA "Tablename"

```

Note: Make sure you set the wms_extent METADATA for the LAYER, as otherwise the “Getcapabilities” request takes a lot of time.

7.1.17 MySQL

Revision \$Revision: 8472 \$

Date \$Date: 2009-01-28 13:08:43 -0800 (Wed, 28 Jan 2009) \$

Author David Fawcett

Contact david.fawcett at moea.state.mn.us

Contents

- MySQL
 - Introduction
 - Create .ovf file
 - Test Connection with ogrinfo
 - Create MapServer Layer
 - More Information

Introduction

This method takes advantage of OGR’s MySQL driver and avoids the need to set up an ODBC connection.

This is the primitive MySQL support for point data, not the recently added [MySQL spatial support](#) found in [GDAL/OGR 1.3.2](#) .

Support for this functionality is found in GDAL/OGR 1.2.6 and older on Windows and GDAL/OGR 1.3.2 on Linux.

The [MySQL wiki page](#) might contain additional information.

Needed

- MySQL database containing a table with fields containing x and y coordinates
- .ovf file, a small xml file you will create
- MapServer compiled with OGR version supporting this functionality

Create .ovf file

Here is the .ovf file named aqidata.ovf

```
<OGRVRTDataSource>
  <OGRVRTLayer name="aqidata">
    <SrcDataSource>MYSQL:aqiTest,user=uuuuu,password=ppppp,host=192.170.1.100,port=3306,tables=t
    <SrcSQL>SELECT areaID, x, y, sampleValue FROM testdata</SrcSQL>
    <GeometryType>wkbPoint</GeometryType>
    <GeometryField encoding="PointFromColumns" x="x" y="y"/>
  </OGRVRTLayer>
</OGRVRTDataSource>
```

If you look at the connection string in <SrcDataSource>

- The MySQL database name is 'aqiTest'
- 'testdata' is the table containing the coordinate data
- host and port are for MySQL server

Use the GeometryField element to tell OGR which fields store the x and y coordinate data. Mine are simply named x and y.

Test Connection with ogrinfo

```
# usr/local/bin/ogrinfo /maps/aqidata.ovf
```

ogrinfo returns

```
ERROR 4: Update access not supported for VRT datasources.
Had to open data source read-only.
INFO: Open of '/maps/aqidata.ovf'
using driver 'VRT' successful.
1: aqidata (Point)
```

Don't worry about the error, this is just telling you that it is a read-only driver. If it really bugs you, call ogrinfo with the -ro (read only) flag.

To see the actual data

```
# usr/local/bin/ogrinfo /maps/aqidata.ovf -al
```

Create MapServer Layer

```

LAYER
  NAME "MyAqi"
  STATUS DEFAULT
  TYPE POINT
  CONNECTIONTYPE OGR
  CONNECTION "aqidata.ovf"
  DATA "aqidata"
  CLASS
    NAME "MyClass"
    STYLE
      SYMBOL 'circle'
      SIZE 15
      COLOR 0 255 0
    END
  END
END

```

DATA in the LAYER definition should be the same as the name attribute of the OGRVRTLayer element in the ovf file.

For this to draw, you need to have a SYMBOLSET defined in your mapfile and have a symbol called 'circle' in your symbols.sym file.

More Information

- *OGR*
- *Vector Data*

7.1.18 NTF

NTF files are mostly used by the United Kingdom Ordnance Survey (OS). For more on the Ordnance Survey, see their website at: <http://www.ordnancesurvey.co.uk/oswebsite/>

File listing

NTF files have an NTF extension.

Data Access / Connection Method

- NTF access requires OGR.
- The path to the NTF file is required in the CONNECTION string. It may be relative to the SHAPEPATH setting in the map file or the full path.
- The DATA parameter is used to specify the layer to use

OGRINFO Examples

Using ogrinfo on an NTF file to retrieve layer names:

```
> ogrinfo llcontours.ntf
ERROR 4: NTF Driver doesn't support update.
Had to open data source read-only.
INFO: Open of `llcontours.ntf'
using driver `UK .NTF' successful.
1: LANDLINE_POINT (Point)
2: LANDLINE_LINE (Line String)
3: LANDLINE_NAME (Point)
4: FEATURE_CLASSES (None)
```

Using ogrinfo to examine the structure of an NTF layer:

```
> ogrinfo llcontours.ntf LANDLINE_LINE -summary
ERROR 4: NTF Driver doesn't support update.
Had to open data source read-only.
INFO: Open of `llcontours.ntf'
using driver `UK .NTF' successful.

Layer name: LANDLINE_LINE
Geometry: Line String
Feature Count: 491
Extent: (279000.000000, 187000.000000) - (280000.000000, 188000.000000)
Layer SRS WKT:
PROJCS["OSGB 1936 / British National Grid",
    GEOGCS["OSGB 1936",
        DATUM["OSGB_1936",
            SPHEROID["Airy 1830",6377563.396,299.3249646,
                AUTHORITY["EPSG","7001"]],
            AUTHORITY["EPSG","6277"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433],
        AUTHORITY["EPSG","4277"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",49],
    PARAMETER["central_meridian",-2],
    PARAMETER["scale_factor",0.999601272],
    PARAMETER["false_easting",400000],
    PARAMETER["false_northing",-100000],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
        AUTHORITY["EPSG","27700"]]
LINE_ID: Integer (6.0)
FEAT_CODE: String (4.0)
...
```

Map File Example:

```
LAYER
  NAME ntf_uk
  TYPE LINE
  CONNECTIONTYPE OGR
  CONNECTION "./ntf/llcontours.ntf"
DATA "LANDLINE_LINE"
  STATUS DEFAULT
  CLASS
    NAME "Contours"
  STYLE
```



```

        COLOR 0 150 200
END
    END
END

```

7.1.19 SDTS

This is a United States Geological Survey (USGS) format. SDTS has a raster and a vector format. The raster format is not supported in MapServer. Only the vector formats are supported, including VTP and DLG files.

File listing

- SDTS files are often organized into state-sized pieces. For example, all of the state of Maryland (MD), U.S.A.
- Files are also available for multiple types of features including hydrography, transportation and administrative boundaries.

This example uses transportation data, which consists of 35 separate files, each with the suffix DDF:

```

MDTRAHDR.DDF  MDTRARRF.DDF  MDTRCATS.DDF
MDTRDQCG.DDF  MDTRFF01.DDF  MDTRLE02.DDF
MDTRNA03.DDF  MDTRNO03.DDF  MDTRSPDM.DDF
MDTRAMTF.DDF  MDTRBFPS.DDF  MDTRCATX.DDF
MDTRDQHL.DDF  MDTRIDEN.DDF  MDTRLE03.DDF
MDTRNE03.DDF  MDTRPC01.DDF  MDTRSTAT.DDF
MDTRARDF.DDF  MDTRBMTA.DDF  MDTRDDSH.DDF
MDTRDQLC.DDF  MDTRIREF.DDF  MDTRNA01.DDF
MDTRNO01.DDF  MDTRPC02.DDF  MDTRXREF.DDF
MDTRARDM.DDF  MDTRCATD.DDF  MDTRDQAA.DDF
MDTRDQPA.DDF  MDTRLE01.DDF  MDTRNA02.DDF
MDTRNO02.DDF  MDTRPC03.DDF

```

Data Access / Connection Method

- SDTS access is available in MapServer through OGR.
- The CONNECTIONTYPE OGR parameter must be used.
- The path (which can be relative) to the catalog file (????CATD.DDF) is required, including file extension.
- There are multiple layers in the SDTS catalog, some of which are only attributes and have no geometries.
- The layer name is specified with the DATA parameter

OGRINFO Examples

Using ogrinfo on a catalog file (note that the first 7 layers do not have geometries):

```

> ogrinfo /data/sdts/MD/MDTRCATD.DDF
Had to open data source read-only.
INFO: Open of 'MDTRCATD.DDF'
using driver 'SDTS' successful.
1: ARDF (None)
2: ARRF (None)

```

```
3: AMTF (None)
4: ARDM (None)
5: BFPS (None)
6: BMTA (None)
7: AHDR (None)
8: NE03 (Point)
9: NA01 (Point)
10: NA02 (Point)
11: NA03 (Point)
12: NO01 (Point)
13: NO02 (Point)
14: NO03 (Point)
15: LE01 (Line String)
16: LE02 (Line String)
17: LE03 (Line String)
18: PC01 (Polygon)
19: PC02 (Polygon)
20: PC03 (Polygon)
```

Using ogrinfo to examine the structure of the file/layer:

```
> ogrinfo /data/sdts/MD/MDTRCATD.DDF LE01 -summary
Had to open data source read-only.
INFO: Open of 'MDTRCATD.DDF'
using driver 'SDTS' successful.

Layer name: LE01
Geometry: Line String
Feature Count: 780
Extent: (-80.000289, 36.999774) - (-74.999711, 40.000225)
Layer SRS WKT:
GEOGCS["NAD27",
DATUM["North_American_Datum_1927",
    SPHEROID["Clarke 1866",6378206.4,294.978698213901]],
PRIMEM["Greenwich",0],
UNIT["degree",0.0174532925199433]]
  RCID: Integer (0.0)
  SNID: Integer (0.0)
  ENID: Integer (0.0)
  ENTITY_LABEL: String (7.0)
  ARBITRARY_EXT: String (1.0)
  RELATION_TO_GROUND: String (1.0)
  VERTICAL_RELATION: String (1.0)
  OPERATIONAL_STATUS: String (1.0)
  ACCESS_RESTRICTION: String (1.0)
  OLD_RAILROAD_GRADE: String (1.0)
  WITH_RAILROAD: String (1.0)
  COVERED: String (1.0)
  HISTORICAL: String (1.0)
  LIMITED_ACCESS: String (1.0)
  PHOTOREVISED: String (1.0)
  LANES: Integer (2.0)
  ROAD_WIDTH: Integer (3.0)
  BEST_ESTIMATE: String (1.0)
  ROUTE_NUMBER: String (7.0)
  ROUTE_TYPE: String (9.0)
```

Map File Example:

```
LAYER
  NAME sdt_s_maryland
  TYPE LINE
  CONNECTIONTYPE OGR
  CONNECTION "data/sdt_s/MD/MDTRCATD.DDF"
  DATA "LE01"
  STATUS DEFAULT
  CLASS
    STYLE
      COLOR 0 0 0
    END
  END
END
```

7.1.20 USGS TIGER

TIGER/Line files are created by the US Census Bureau and cover the entire US. They are often referred simply as TIGER files. For more information see: <http://www.census.gov/geo/www/tiger/>.

File listing

TIGER/Line files are text files and directory-based data sources. For example, one county folder TGR06059 contains several associated files:

```
TGR06059.RT1 TGR06059.RT2 TGR06059.RT4 TGR06059.RT5
TGR06059.RT6 TGR06059.RT7 TGR06059.RT8 TGR06059.RTA
TGR06059.RTC TGR06059.RTH TGR06059.RTI TGR06059.RTP
TGR06059.RTR TGR06059.RTS TGR06059.RTT TGR06059.RTZ
```

Data Access / Connection Method

- TIGER/Line access occurs through an OGR CONNECTION
- The full path to the directory containing the associated files is required in the CONNECTION string.
- The feature type is specified in the DATA parameter

OGRINFO Examples

Using ogrinfo on a TIGER directory to retrieve feature types:

```
> ogrinfo TGR06059 (NOTE that this is a directory)
ERROR 4: Tiger Driver doesn't support update.
Had to open data source read-only.
INFO: Open of 'TGR06059'
using driver 'TIGER' successful.
1: CompleteChain (Line String)
2: AltName (None)
3: FeatureIds (None)
4: ZipCodes (None)
5: Landmarks (Point)
6: AreaLandmarks (None)
```

```
7: Polygon (None)
8: PolygonCorrections (None)
9: EntityNames (Point)
10: PolygonEconomic (None)
11: IDHistory (None)
12: PolyChainLink (None)
13: PIP (Point)
14: TLIDRange (None)
15: ZeroCellID (None)
16: OverUnder (None)
17: ZipPlus4 (None)
```

Using ogrinfo to examine the structure of the TIGER feature type CompleteChain:

```
> ogrinfo TGR06059 CompleteChain -summary
ERROR 4: Tiger Driver doesn't support update.
Had to open data source read-only.
INFO: Open of `TGR06059'
using driver `TIGER' successful.

Layer name: CompleteChain
Geometry: Line String
Feature Count: 123700
Extent: (-118.125898, 33.333992) - (-117.412987, 33.947512)
Layer SRS WKT:
GEOGCS["NAD83",
DATUM["North_American_Datum_1983",
    SPHEROID["GRS 1980",6378137,298.257222101]],
PRIMEM["Greenwich",0],
UNIT["degree",0.0174532925199433]]
MODULE: String (8.0)
TLID: Integer (10.0)
SIDE1: Integer (1.0)
SOURCE: String (1.0)
FDIRP: String (2.0)
FENAME: String (30.0)
FETYPE: String (4.0)
FEDIRS: String (2.0)
CFCC: String (3.0)
FRADDL: String (11.0)
TOADDL: String (11.0)
FRADDR: String (11.0)
TOADDR: String (11.0)
FRIADDL: String (1.0)
TOIADDL: String (1.0)
FRIADDR: String (1.0)
TOIADDR: String (1.0)
ZIPL: Integer (5.0)
```

Map File Example:

```
LAYER
  NAME Complete_Chain
  TYPE LINE
  STATUS DEFAULT
  CONNECTIONTYPE OGR
  CONNECTION "/path/to/data/tiger/TGR06059"
```

```

DATA "CompleteChain"
CLASS
  STYLE
    COLOR 153 102 0
  END
END
END # Layer

```

7.1.21 GPS Exchange Format (GPX)

GPX (the GPS Exchange Format) is a light-weight XML data format containing GPS data (waypoints, routes, and tracks). For more information see the official [GPX site](#).

File listing

All waypoints, routes, and tracks are stored in a single .gpx file.

Data Access / Connection Method

- GPX access is available through OGR. See the [OGR driver page](#) for specific driver information.
- A relative path to the .gpx file can be used in the mapfile LAYER's CONNECTION string.
- **The feature type is specified in the DATA parameter** – the “tracks” feature type will usually be the track line
 - the “track_points” feature type will usually be the points that make up the track line

OGRINFO Examples

First you should make sure that your GDAL/OGR build contains the “GPX” driver, by using the ‘–formats’ command:

```

>ogrinfo --formats
Loaded OGR Format Drivers:
...
-> "CSV" (read/write)
-> "GML" (read/write)
-> "GPX" (read/write)
-> "KML" (read/write)
...

```

If you don't have the driver, you might want to try the [FWTools](#) or [MS4W](#) packages, which include the driver.

Once you have the GPX driver you are ready to try an ogrinfo command on your file to get a list of feature types:

```

>ogrinfo test.gpx
INFO: Open of 'test.gpx'
      using driver 'GPX' successful.
1: waypoints (Point)
2: routes (Line String)
3: tracks (Multi Line String)
4: route_points (Point)
5: track_points (Point)

```

Now use ogrinfo to get information on one of the feature types:

```
>ogrinfo test.gpx track_points -summary
INFO: Open of 'test.gpx'
      using driver 'GPX' successful.

Layer name: track_points
Geometry: Point
Feature Count: 661
Extent: (-66.694270, 47.985570) - (-66.675222, 47.990791)
Layer SRS WKT:
GEOGCS["WGS 84",
  DATUM["WGS_1984",
    SPHEROID["WGS 84",6378137,298.257223563,
      AUTHORITY["EPSG","7030"]],
    AUTHORITY["EPSG","6326"]],
  PRIMEM["Greenwich",0,
    AUTHORITY["EPSG","8901"]],
  UNIT["degree",0.01745329251994328,
    AUTHORITY["EPSG","9122"]],
  AUTHORITY["EPSG","4326"]]
track_fid: Integer (0.0)
track_seg_id: Integer (0.0)
track_seg_point_id: Integer (0.0)
ele: Real (0.0)
time: DateTime (0.0)
magvar: Real (0.0)
geoidheight: Real (0.0)
name: String (0.0)
cmt: String (0.0)
desc: String (0.0)
src: String (0.0)
...
```

Mapfile Example

Since you have confirmed that OGR can read your GPX file, now you can create a MapServer layer:

```
LAYER
  NAME gpx
  TYPE POINT
  STATUS ON
  CONNECTIONTYPE OGR
  CONNECTION test.gpx
  DATA "track_points"
  CLASS
    NAME "gpx"
    STYLE
      SYMBOL 'circle'
      COLOR 0 119 255
      SIZE 2
    END
  END
END # layer
```

7.2 Raster Data

Author Frank Warmerdam

Contact warmerdam at pobox.com

Revision \$Revision: 8480 \$

Date \$Date: 2009-01-29 12:04:25 -0800 (Thu, 29 Jan 2009) \$

Last Updated 2007/12/09

Table of Contents

- Raster Data
 - Introduction
 - How are rasters added to a Map file?
 - Supported Formats
 - Rasters and Tile Indexing
 - Raster Warping
 - 24bit RGB Rendering
 - Special Processing Directives
 - Raster Query
 - Raster Display Performance Tips
 - Preprocessing Rasters
 - Georeference with World Files

7.2.1 Introduction

MapServer supports rendering a variety of raster file formats in maps. The following describes some of the supported formats, and what capabilities are supported with what formats.

This document assumes that you are already familiar with setting up MapServer *Mapfile*, but does explain the raster specific aspects of mapfiles.

7.2.2 How are rasters added to a Map file?

A simple raster layer declaration looks like this. The DATA file is interpreted relative to the SHAPEPATH, much like shapefiles.

```
LAYER
  NAME "JacksonvilleNC_CIB"
  DATA "Jacksonville.tif"
  TYPE RASTER
  STATUS ON
END
```

Though not shown rasters can have PROJECTION, METADATA, PROCESSING, MINSCALE, and MAXSCALE information. It cannot have labels, CONNECTION, CONNECTIONTYPE, or FEATURE information.

Classifying Rasters

Rasters can be classified in a manner similar to vectors, with a few exceptions.

There is no need to specify a CLASSITEM. The raw pixel value itself (“[pixel]”) and, for paletted images, the red, green and blue color associated with that pixel value (“[red]”, “[green]” and “[blue]”) are available for use in classifications. When used in an evaluated expression the pixel, red, green and blue keywords must be in lower case.

```
LAYER
  NAME "JacksonvilleNC_CIB"
  DATA "Jacksonville.tif"
  TYPE RASTER
  STATUS ON
  CLASSITEM "[pixel]"
  # class using simple string comparison, equivalent to ([pixel] = 0)
  CLASS
    EXPRESSION "0"
    STYLE
      COLOR 0 0 0
    END
  END
  # class using an EXPRESSION using only [pixel].
  CLASS
    EXPRESSION ([pixel] >= 64 AND [pixel] < 128)
    STYLE
      COLOR 255 0 0
    END
  END
  # class using the red/green/blue values from the palette
  CLASS
    NAME "near white"
    EXPRESSION ([red] > 200 AND [green] > 200 AND [blue] > 200)
    STYLE
      COLOR 0 255 0
    END
  END
  # Class using a regular expression to capture only pixel values ending in 1
  CLASS
    EXPRESSION /*1/
    STYLE
      COLOR 0 0 255
    END
  END
END
```

As usual, CLASS definitions are evaluated in order from first to last, and the first to match is used. If a CLASS has a NAME attribute it may appear in a LEGEND. Only the COLOR, EXPRESSION and NAME parameters within a CLASS definition are utilized for raster classifications. The other styling or control information is ignored.

Raster classifications always take place on only one raster band. It defaults to the first band in the referenced file, but this can be altered with the BANDS PROCESSING directive. In particular this means that including even a single CLASS declaration in a raster layer will result in the raster layer being rendered using the one band classification rules instead of other rules that might have applied (such as 3 band RGB rendering).

Classifying Non-8bit Rasters

As of MapServer 4.4 support has been added for classifying non-8bit raster inputs. That is input rasters with values outside the range 0-255. Mostly this works transparently but there are a few caveats and options to provide explicit control.

Classifying raster data in MapServer is accomplished by pre-classifying all expected input values and using that table

of classification results to lookup each pixel as it is rendered. This is done because evaluating a pixel value against a series of CLASS definitions is relatively expensive to do for the hundreds of thousands of pixels in a typical rendered image.

For simple 8bit inputs, only 256 input values need to be pre-classified. But for non-8bit inputs more values need to be classified. For 16bit integer inputs all 65536 possible input values are pre-classified. For floating point and other input data types, up to 65536 values are pre-classified based on the maximum expected range of input values.

The PROCESSING directive can be used to override the range of values to be pre-classified, or the number of values (aka Buckets) in that range to classify. The SCALE=min,max PROCESSING directive controls the range. The SCALE_BUCKETS PROCESSING directive controls the number of buckets. In some cases rendering can be accelerated considerable by selecting a restricted range of input values and a reduced number of scaling values (buckets).

The following example classifies a floating raster, but only 4 values over the range -10 to 10 are classified. In particular, the values classified would be -7.5, -2.5, 2.5, and 7.5 (the middle of each “quarter” of the range). So those four value are classified, and one of the classification results is selected based on which value is closest to the pixel value being classified.

```
LAYER
  NAME grid1
  TYPE raster
  STATUS default
  DATA data/float.tif
  PROCESSING "SCALE=-10,10"
  PROCESSING "SCALE_BUCKETS=4"
  CLASS
    NAME "red"
    EXPRESSION ([pixel] < -3)
    STYLE
      COLOR 255 0 0
    END
  END
  CLASS
    NAME "green"
    EXPRESSION ([pixel] >= -3 and [pixel] < 3)
    STYLE
      COLOR 0 255 0
    END
  END
  CLASS
    NAME "blue"
    EXPRESSION ([pixel] >= 3)
    STYLE
      COLOR 0 0 255
    END
  END
END
```

7.2.3 Supported Formats

What raster formats are supported by MapServer is largely controlled by configuration time options. Some formats are considered to be built-in while the remainder are handled by the optional GDAL raster library.

More information on GDAL can be found at <http://www.gdal.org>, including the [supported formats list](#). Some of the advanced MapServer raster features, such as resampling, RGB color cube generation and automatic projection capture only work with raster formats used through GDAL. GDAL is normally built and installed separately from MapServer, and then enabled during the build of MapServer using the `-with-gdal` configuration switch.

To find out what is built into a particular MapServer executable, use the `-v` flags to discover what build options are enabled. To find out what GDAL formats are available, the “`gdalinfo -formats`” command may be used. For example:

```
warmerda@gdal2200[124]% mapserv -v
MapServer version 4.4.0-beta2 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER SUPPORTS=FASTCGI
INPUT=EPPL7 INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE DEBUG=MSDEBUG
warmerda@gdal2200[18]% gdalinfo --formats
Supported Formats:
  GRASS (ro): GRASS Database Rasters (5.7+)
  GTiff (rw+): GeoTIFF
  NITF (rw+): National Imagery Transmission Format
  HFA (rw+): Erdas Imagine Images (.img)
  SAR_CEOS (ro): CEOS SAR Image
  ...
```

The following formats are potential builtins:

TIFF/GeoTIFF If built with `INPUT=TIFF` MapServer will have builtin support for reading TIFF or GeoTIFF files. The builtin TIFF support has some limitations with regard to the organization of files that can be read (no tiled, 16bit, RGB, or odd color models). This driver supports world files, or simple builtin GeoTIFF coordinates for georeferencing.

Full featured TIFF/GeoTIFF support is available through GDAL. Note that only GDAL supports tiled TIFF files and TIFF files with overviews. Tiled TIFF files with overviews pre-built are one of the highest performance ways of serving large raster images.

GIF If GD is configured with `GIF (OUTPUT=GIF)` support, then MapServer will also be able to read GIF files for raster layers. The only way to georeference GIF files is with a world file.

If GD is not configured with GIF support, it may still be available in GDAL.

PNG If GD is configured with `PNG (OUTPUT=PNG)` support, then MapServer will also be able to read PNG files for raster layers. The only way to georeference PNG files is with a world file.

If GD is not configured with PNG support, it may still be available in GDAL.

JPEG If MapServer is built with `JPEG (INPUT=JPEG)` support then greyscale JPEG files may be rendered in raster layers. RGB files (the more common kind) will not be able to be displayed. Georeferencing is via world files.

If MapServer is not built with native JPEG support, GDAL may still support the format. In this case RGB files are also supported (via the RGB color cube mechanism). Georeferencing is still via world file.

Erdas .LAN/GIS If configured with `INPUT=EPPL7` (the default) MapServer will support one band eight bit Erdas LAN/GIS files. The `.trl` file is read for a colormap, and if not found the layer is treated as greyscale. Georeferencing is read from the file.

If MapServer is built with GDAL it is generally better to access all possible formats through GDAL rather than via the built-in drivers. The built-in drivers are less featureful, and not as well maintained.

7.2.4 Rasters and Tile Indexing

When handling very large raster layers it is often convenient, and higher performance to split the raster image into a number of smaller images. Each file is a tile of the larger raster mosaic available for display. The list of files forming a layer can be stored in a shapefile with polygons representing the footprint of each file, and the name of the files. This is called a `TILEINDEX` and works similarly to the same feature in vector layers. The result can be represented in the `MAP` file as one layer, but MapServer will first scan the tile index, and ensure that only raster files overlapping the current display request will be opened.

The following example shows a simple example. No DATA statement is required because MapServer will fetch the filename of the raster files from the Location attribute column in the hp2.dbf file for records associated with polygons in hp2.shp that intersect the current display region. The polygons in hp2.shp should be rectangles representing the footprint of the corresponding file. Note that the files do not have to be all the same size, the formats can vary and they can even overlap (later files will be drawn over earlier ones); however, they must all be in the same coordinate system (projection) as the layer.

```
LAYER
  NAME "hpool"
  STATUS ON
  TILEINDEX "hp2.shp"
  TILEITEM "Location"
  TYPE RASTER
END
```

The filenames in the tileindex are searched for relative to the SHAPEPATH or map file, not relative to the tileindex. Great care should be taken when establishing the paths put into the tileindex to ensure they will evaluate properly in use. Often it is easiest to place the tileindex in the SHAPEPATH directory, and to create the tileindex with a path relative to the SHAPEPATH directory. When all else fails, absolute paths can be used in tileindex, but then they cannot be so easily moved from system to system.

While there are many ways to produce TILEINDEX shapefiles for use with this command, one option is the `gdaltindex` program, part of the GDAL utility suite. The `gdaltindex` program will automatically generate a tile index shapefile from a list of GDAL supported raster files passed on the command line.

```
Usage: gdaltindex [-tileindex field_name] index_file [gdal_file]*
```

```
% gdaltindex doq_index.shp doq/*.tif
```

Tile Index Notes

- The shapefile (index_file) will be created if it doesn't already exist.
- The default tile index field is 'location'.
- Simple rectangular polygons are generated in the same coordinate system as the rasters.
- Raster filenames will be put in the file exactly as they are specified on the commandline.
- Many problems with tile indexes relate to how relative paths in the tile index are evaluated. They should be evaluated relative to the SHAPEPATH if one is set, otherwise relative to the tileindex file. When in doubt absolute paths may avoid path construction problems.

The `gdaltindex` program is built as part of GDAL. Prebuilt binaries for GDAL including the `gdaltindex` program can be downloaded as part of the [OSGeo4W](#), [FWTools](#) and [MS4W](#) distributions.

See Also:

Tile Indexes

7.2.5 Raster Warping

MapServer is able to resample GDAL rasters on the fly into new projections. Non-GDAL rasters may only be up or down sampled without any rotation or warping.

Raster warping kicks in if the projection appears to be different for a raster layer than for the map being generated. Warped raster layers are significantly more expensive to render than normal raster layers with rendering time being

perhaps 2-4 times long than a normal layer. The projection and datum shifting transformation is computed only at selected points, and generally linearly interpolated along the scanlines (as long as the error appears to be less than 0.333 pixels).

In addition to reprojecting rasters, the raster warping ability can also apply rotation to GDAL rasters with rotational coefficients in their georeferencing information. Currently rotational coefficients won't trigger raster warping unless the map and layer have valid (though matching is fine) projection definitions.

7.2.6 24bit RGB Rendering

Traditionally MapServer has been used to produce 8 bit pseudo-colored map displays generated from 8bit greyscale or pseudocolored raster data. However, if the raster file to be rendered is actually 24bit (a red, green and blue band) then additional considerations come into play. Currently rendering of 24bit imagery is only supported via the GDAL renderer. The built-in PNG, JPEG and other drivers do not support 24bit input images.

If the output is still 8bit pseudo-colored (the `IMAGEMODE` is `PC256` in the associated `OUTPUT` format declaration) then the full 24bit RGB colors for input pixels will be converted to a color in the colormap of the output image. By default a color cube is used. That is a fixed set of 175 colors providing 5 levels of red, 7 levels of green and 5 levels of blue is used, plus an additional 32 greyscale color entries. Colors in the input raster are mapped to the closest color in this color cube on the fly. This substantially degrades color quality, especially for smoothly changing images. It also fills up the colors table, limited to 256 colors, quite quickly.

A variation on this approach is to dither the image during rendering. Dithering selects a color for a pixel in a manner that "diffuses error" over pixels. In an area all one color in the source image, a variety of output pixel colors would be selected such that the average of the pixels would more closely approximate the desired color. Dithering also takes advantage of all currently allocated colors, not just those in the color cube. Dithering requires GDAL 1.1.9 or later, and is enabled by providing the `PROCESSING "DITHER=YES"` option in the mapfile. Dithering is more CPU intensive than using a simple color cube, and should be avoided if possible in performance sensitive situations.

The other new possibility for handling 24bit input imagery in MapServer 4.0 or later, is to produce 24bit output images. The default `"IMAGETYPE png24"` or `"IMAGETYPE jpeg"` declaration may be used to produce a 24bit PNG output file, instead of the more common 8bit pseudo-colored PNG file. The `OUTPUTFORMAT` declaration provides for detailed control of the output format. The `IMAGEMODE RGB` and `IMAGEMODE RGBA` options produce 24bit and 32bit (24bit plus 8bit alpha/transparency) for supported formats.

7.2.7 Special Processing Directives

As of MapServer 4.0, the `PROCESSING` parameter was added to the `LAYER` of the `Mapfile`. It is primarily used to pass specialized raster processing options to the GDAL based raster renderer. The following processing options are supported in MapServer 4.0 and newer.

DITHER=YES This turns on error diffusion mode, used to convert 24bit images to 8bit with error diffusion to get better color results.

Example:

```
PROCESSING "DITHER=YES"
```

BANDS=red_or_grey[,green,blue[,alpha]] This directive allows a specific band or bands to be selected from a raster file. If one band is selected, it is treated as greyscale. If 3 are selected, they are treated as red, green and blue. If 4 are selected they are treated as red, green, blue and alpha (opacity).

Example:

```
PROCESSING "BANDS=4, 2, 1"
```

SCALE[_n]=AUTO or min,max This directive instructs the GDAL reader to pre-scale the incoming raster data. It is primarily used to scale 16bit or floating point data to the range 0-255, but can also be used to contrast stretch 8bit data. If an explicit min/max are provided then the input data is stretch (or squished) such that the minimum value maps to zero, and the maximum to 255. If AUTO is used instead, a min/max is automatically computed. To control the scaling of individual input bands, use the SCALE_1, SCALE_2 and SCALE_3 keywords (for red, green and blue) instead of SCALE which applies to all bands.

Example:

```
PROCESSING "SCALE=AUTO"
  or
PROCESSING "SCALE_1=409,1203"
PROCESSING "SCALE_2=203,296"
PROCESSING "SCALE_3=339,1004"
```

LUT[_n]=<lut_spec> This directive (MapServer 4.9+) instructs the GDAL reader to apply a custom LUT (lookup table) to one or all color bands as a form of on the fly color correction. If LUT is used, the LUT is applied to all color bands. If LUT_n is used it is applied to one color band (n is 1 for red, 2 for green, 3 for blue, 4 for alpha).

The LUT can be specified inline in the form:

```
<lut_spec> = <in_value>:<out_value>[,<in_value>:<out_value>]*
```

This essentially establish particular input values which are mapped to particular output values. The list implicitly begins with 0:0, and 255:255. An actual 256 entry lookup table is created from this specification, linearly interpolating between the values. The in values must be in increasing order. The LUT specification may also be in a text file with the <lut_spec> being the filename. The file contents should be in the same syntax, and the file is searched relative to the mapfile.

Example:

```
PROCESSING "LUT_1=red.lut"
PROCESSING "LUT_2=green.lut"
PROCESSING "LUT_3=blue.lut"
  or
PROCESSING "LUT=100:30,160:128,210:200"
```

As a special case there is also support for GIMP format curve files. That is the text files written out by the Tools->Color->Curves tool. If this is specified as the filename then it will be internally converted into linear segments based on the curve control points. Note that this will not produce exactly the same results as the GIMP because linear interpolation is used between control points instead of curves as used in the GIMP. For a reasonable number of control points the results should be similar. Also note that GIMP color curve files include an overall "value" curve, and curves for red, green, blue and alpha. The value curve and the appropriate color curve will be composed internally to produce the final LUT.

Example:

```
PROCESSING "LUT=munich.crv"
```

COLOR_MATCH_THRESHOLD=n Alter the precision with which colors need to match an entry in the color table to use it when producing 8bit colormapped output (IMAGEMODE PC256). Normally colors from a raster colormap (or greyscale values) need to match exactly. This relaxes the requirement to being within the specified color distance. So a COLOR_MATCH_THRESHOLD of 3 would mean that an existing color entry within 3 (sum of difference in red, green and blue) would be used instead of creating a new colormap entry. Especially with greyscale raster layers, which would normally use all 256 color entries if available, this can be a good way to avoid "stealing" your whole colormap for a raster layer. Normally values in the range 2-6 will give good results.

Example:

```
PROCESSING "COLOR_MATCH_THRESHOLD=3"
```

RESAMPLE=NEAREST/AVERAGE/BILINEAR This option can be used to control the resampling kernel used sampling raster images. The default (and fastest) is NEAREST. AVERAGE will perform compute the average pixel value of all pixels in the region of the disk file being mapped to the output pixel (or possibly just a sampling of them). BILINEAR will compute a linear interpolation of the four pixels around the target location. This topic is discussed in more detail in *MS RFC 4: MapServer Raster Resampling*.

Resampling options other than NEAREST result in use of the generalized warper and can dramatically slow down raster processing. Generally AVERAGE can be desirable for reducing noise in dramatically downsampled data, and can give something approximating antialiasing for black and white linework. BILINEAR can be helpful when oversampling data to give a smooth appearance.

Example (chose one):

```
PROCESSING "RESAMPLE=NEAREST"  
PROCESSING "RESAMPLE=AVERAGE"  
PROCESSING "RESAMPLE=BILINEAR"
```

7.2.8 Raster Query

A new feature added in MapServer 4.4 is the ability to perform queries on rasters in a manner similar to queries against vector layers. Raster queries on raster layers return one point feature for each pixel matching the query. The point features will have attributes indicating the value of different bands at that pixel, the final rendering color and the class name. The resulting feature can be directly access in MapScript, or processed through templates much like normal vector query results. Only raster layers with a query TEMPLATE associated can be queried, even for the query methods that don't actually use the query template (much like vector data).

Raster query supports QueryByPoint, QueryByRect, and QueryByShape. QueryByPoint supports single and multiple result queries. Other query operations such as QueryByIndex, QueryByIndexAdd, QueryByAttributes and QueryByFeature are not supported for raster layers.

Raster layers do **not** support saving queries to disk, nor **query maps**.

Raster queries return point features with some or all of the following attributes:

- x** georeferenced X location of pixel.
- y** georeferenced Y location of pixel.
- value_list** a comma separated list of the values of all selected bands at the target pixel.
- value_n** the value for the n'th band in the selected list at this pixel (zero based). There is one value_n entry for each selected band.
- class** Name of the class this pixel is a member of (classified layers only).
- red** red component of the display color for this pixel.
- green** green component of the display color for this pixel.
- blue** blue component of the display color for this pixel.

The red, green and blue attribute are intended to be the final color the pixel would be rendered with, but in some subtle cases it can be wrong (ie. classified floating point results). The selected bands are normally the band that would be used to render the layer. For a pure query-only layer BANDS PROCESSING directive can be used to select more bands than could normally be used in a render operation. For instance for a 7 band landsat scene a PROCESSING "BANDS=1,2,3,4,5,6,7" directive could be used to get query results for all seven bands in results to a query operation.

Care should be taken to avoid providing a large query area (selecting a lot of pixels) as each selected pixel requires over 100 bytes of memory for temporary caching. The `RASTER_QUERY_MAX_RESULT_PROCESSING` item can be used to restrict the maximum number of query results that will be returned. The default is one million which would take on the order of 100MB of RAM.

Query results can be returned as HTML via the normal substitution into query template HTML. Query results are also accessible via WMS GetFeatureInfo calls, and from MapScript. The following example shows executing a feature query from Python MapScript and fetching back the results:

```
map = mapscript.Map('rquery.map')
layer = map.getLayer(0)

pnt = mapscript.Point()
pnt.x = 440780
pnt.y = 3751260

layer.queryByPoint( map, pnt, mapscript.MS_MULTIPLE, 180.0 )

layer.open()
for i in range(1000):
    result = layer.getResult( i )
    if result is None:
        break

    print '(%d,%d)' % (result.shapeindex, result.tileindex)

    s = layer.getShape( result.shapeindex, result.tileindex )
    for i in range(layer.numitems):
        print '%s: %s' % (layer.getItem(i), s.getValue(i))

layer.close()
```

The following is a simple example query *TEMPLATE* file. The raster pixel attributes will be substituted in before the query result is returned to the user as HTML.

```
Pixel:<br>
  values=[value_list]<br>

  value_0=[value_0]<br>
  value_1=[value_1]<br>
  value_2=[value_2]<br>
  RGB = [red],[green],[blue]<p>
  Class = [class]<br>
```

Internally raster query results are essentially treated as a set of temporary features cached in RAM. Issuing a new query operation clears the existing query cache on the layer. The transitory in-memory representation of raster query results is also responsible for the inability to save raster query results since saved query results normally only contain the feature ids, not the entire features. Some additional information is available in the [RasterQuery Wiki](#) topic.

7.2.9 Raster Display Performance Tips

- Build overview levels for large rasters to ensure only a reasonable amount of data needs to be touched to display an overview of a large layer. Overviews can be implemented as a group of raster layers at different resolutions, using `MINSCALE`, and `MAXSCALE` to control which layers are displayed at different resolutions. Another, perhaps easier way, is to build overviews for GDAL supported formats using the `gdaladdo` utility.

- When using tileindexes to manage many raster files as a single file, it is especially important to have an overview layer that kicks in at high scales to avoid having to open a large number of raster files to fulfill the map request.
- Preprocess RGB images to eightbit with a colormap 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. This can be accomplished using the TILEINDEX mechanism of the mapfile, or by creating a tiled format file (ie. TIFF with GDAL).
- Ensure that the image is kept on disk in the most commonly requested projection to avoid on-the-fly image warping which is fairly expensive.
- If you are getting debug output from MapServer in your web server log file, check to see if the message `msRe-sampleGDALToMap` in effect appears. If so, the raster layer is being resampled. If you don't think it should be resampled carefully review your map file to ensure that the layer projection exactly matches the map projection or that the layer has no projection definition.

7.2.10 Preprocessing Rasters

The following operations use GDAL commandline utilities, some of which are python scripts. They are generally available on any GDAL installation with python support.

Producing Tiled Datasets

The TIFF and Erdas Imagine formats support internal tiling within files, and will generally give better display speed for local map requests from large images. To produce a GeoTIFF file in internally tiled format using the `TILED=YES` creation option with the `gdal_translate` utility:

```
gdal_translate -co TILED=YES original.tif tiled.tif
```

Erdas Imagine (HFA) files are always tiled, and can be larger than 4GB (the GeoTIFF limit). Use a command like the following to translate a raster to Imagine format:

```
gdal_translate -of HFA original.tif tiled.img
```

Reducing RGB to 8bit

Rendering and returning 24bit images (especially as PNG) can be quite expensive in render/compress time and bandwidth. Pre-reducing raster data to 8bit can save disk space, processing time, and bandwidth. However, such a color reduction also implicitly reduces the quality of the resulting map. The color reduction can be done on the fly by MapServer but this requires even more processing. A faster approach is to pre-reduce the colors of 24bit imagery to 8bit. This can be accomplished with the GDAL `rgb2pct.py` script like this:

```
rgb2pct.py original.tif 8bit.tif
```

By default images will be reduced to 256 colors but this can mean there are not enough colors to render other colors in the map. So it may be desired to reduce to even less colors:

```
rgb2pct.py -n 200 original.tif 8bit.tif
```

Downsampling to 8bit should be done before internal tiling and overview building. The `rgb2pct.py` script tries to compute an optimal color table for a given image, and then uses error diffusion during the 24bit to 8bit reduction. Other packages (such as ImageMagick or Photoshop) may have alternative color reduction algorithms that are more appropriate for some uses.

Building Internal Overviews

Most GDAL supported raster formats can have overviews pre-built using the `gdaladdo` utility. However, a few formats, such as JPEG2000, MrSID, and ECW already contain implicit overviews in the format themselves and will not generally benefit from external overviews. For other formats (such as GeoTIFF, and Erdas Imagine format) use a command like the following to build overviews:

```
gdaladdo tile.tif 2 4 8 16 32 64 128
```

The above would build overviews at x2 through x128 decimation levels. By default it uses “nearest neighbour” downsampling. That is one of the pixels in the input downsampled area is selected for each output pixel. For some kinds of data averaging can give much smoother overview results, as might be generated with this command:

```
gdaladdo -r average tiled.tif 2 4 8 16 32 64 128
```

Note that overview building should be done after translating to a final format. Overviews are lost in format conversions using `gdal_translate`. Also, nothing special needs to be done to make MapServer use GDAL generated overviews. They are automatically picked up by GDAL when mapserver requests a reduced resolution map.

Building External Overviews

When working with large collections of raster files using a MapServer tileindex, it is desirable to build reduced resolution overview layers that kick in at high scales (using `MINSCALE/MAXSCALE` to control which layer activates). Preparing the overviews can be a somewhat complex process. One approach is to use the `gdal_merge.py` script to downsample and mosaic all the images. For instance if we want to produce an overview of many 1meter ortho photos with 250 meter pixels we might do something like:

```
gdal_merge.py -o overview.tif -ps 250 250 ortho_*.tif
```

The `gdal_merge.py` utility suffers from a variety of issues, including no support for different resampling kernels. With GDAL 1.3.2 or later it should be able to accomplish something similar with the more flexible `gdalwarp` utility:

```
gdalwarp -rc -tr 250 250 ortho_*.tif overview.tif
```

In some cases the easiest way of generating an overview is to let MapServer do it using the `shp2img` utility. For instance if the tileindex layer is called “orthos” we could do something like:

```
shp2img -m ortho.map -l orthos -o overview.png
```

Note that the overview will be generated with the extents and size in the `.map` file, so it may be necessary to temporarily adjust the map extents and size values to match the raster extents and the desired output size. Also, if using this method, don’t leave large files in PNG (or GIF or JPEG) format as they are slow formats to extract subareas from.

7.2.11 Georeference with World Files

World files are a simple mechanism for associating georeferencing (world coordinates) information with raster files. ESRI was the first company to propagate the use of world files, and they often used with TIFF instead of embedding georeferencing information in the file itself.

The world file contents look like the following. The first coefficient is the X pixel size. The second and third are rotational/shear coefficients (and should normally be 0.0). The fourth is the Y pixel size, normally negative indicating that Y decreases as you move down from the top left origin. The final two values are the X and Y location of the

center of the top left pixel. This example is for an image with a 2m x 2m pixel size, and a top left origin at (356800E, 5767999N):

```
2  
0.0000000000  
0.0000000000  
-2  
356800.00  
5767999.00
```

The name of the world file is based on the file it relates to. For instance, the world file for aerial.tif might be aerial.tfw. Conventions vary for appropriate endings, but with MapServer the extension .wld is always OK for world files.

OUTPUT GENERATION

8.1 AGG Rendering Specifics

Author Thomas Bonfort

Contact thomas.bonfort at gmail

Revision \$Revision: 8295 \$

Date \$Date: 2008-12-26 21:08:04 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2008/11/24

Table of Contents

- AGG Rendering Specifics
 - Introduction
 - Setting the OutputFormat
 - New Features
 - Modified Behavior

8.1.1 Introduction

MapServer 5.0 released with a new rendering backend. This howto details the changes and new functionality that this adds to map creation. This howto assumes you already know the basics of mapfile syntax. If not, you should probably be reading *the mapfile syntax*.

8.1.2 Setting the OutputFormat

24 bit png (high quality, large file size):

```
OUTPUTFORMAT
  NAME 'AGG'
  DRIVER AGG/PNG
  IMAGEMODE RGB
END
```

24 bit png, transparent background:

```
OUTPUTFORMAT
  NAME 'AGGA'
  DRIVER AGG/PNG
  IMAGEMODE RGBA
END
```

24 bit jpeg (jpeg compression artifacts may appear, but smaller file size):

```
OUTPUTFORMAT
  NAME 'AGG_JPEG'
  DRIVER AGG/JPEG
  IMAGEMODE RGB
END
```

png output, with number of colors reduced with quantization.

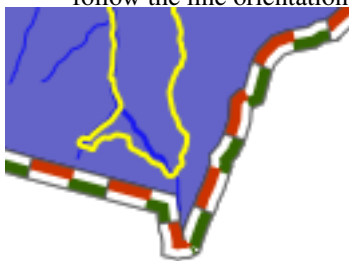
```
OUTPUTFORMAT
  NAME 'AGG_Q'
  DRIVER AGG/PNG
  IMAGEMODE RGB
  FORMATOPTION "QUANTIZE_FORCE=ON"
  FORMATOPTION "QUANTIZE_DITHER=OFF"
  FORMATOPTION "QUANTIZE_COLORS=256"
END
```

8.1.3 New Features

- All rendering is now done antialiased by default. All ANTIALIAS keywords are now ignored, as well as TRANSPARENCY ALPHA. Pixmaps and fonts are now all drawn respecting the image's internal alpha channel (unless a backgroundcolor is specified).
- As with GD in ver. 4.10, using a SYMBOL of type ELLIPSE to draw thick lines isn't mandatory anymore. To draw a thick line just use:

```
STYLE
  WIDTH 5
  COLOR 0 0 255
END
```

- A line symbolizer has been added, that works with vector or pixmap symbols, to draw textured lines. This happens by default if a line's style is given a symbol of type vector or pixmap. To enable "shield" symbolization, i.e. a marker placed only on some points of the line, you must add a GAP parameter to your symbol definition. This GAP value is scaled w.r.t the style's SIZE parameter. Specify a positive gap value for symbols always facing north (optionally rotated by the ANGLE of the current style), or a negative value for symbols that should follow the line orientation



- This happens by default if a line's style is given a symbol of type vector or pixmap. To enable "shield" symbolization, i.e. a marker placed only on some points of the line, you must add a GAP parameter to your symbol definition. This GAP value is scaled w.r.t the style's SIZE parameter - specify a positive gap value for symbols always facing north (optionally rotated by the ANGLE of the current style), or a negative value for symbols that should follow the line orientation
- Pixmap and font symbols can now be rotated without losing their transparency
- For POLYGON layers with no specific SYMBOL, the WIDTH keyword specifies the width of the outline, if an OUTLINECOLOR was specified. This is a shorthand that avoids having to create multiple styles for basic rendering, and will provide a marginal performance gain. Note that in this case, the width of the outline is /not/ scale dependent.

8.1.4 Modified Behavior

- When specifying a SYMBOL for a polygon shape, the GAP parameter of the symbol is used as a separation between each rendered symbol. This works for symbols of type vector, pixmap and ellipse. For example a symbol defined by

```
SYMBOL
NAME 'triangle'
TYPE VECTOR
FILLED TRUE
POINTS
0 1
.5 0
1 1
0 1
END
GAP 1
END
```

that is rendered in a class where SIZE is 15 will be rendered like



- layers of type CIRCLE support hatch type symbol filling
- the ENCODING keyword for labels is now enforced. If unset, MapServer will treat your label text byte-by-byte (resulting in corrupt special characters).

8.2 AntiAliasing with MapServer

Author Pericles Nacionales

Contact naci0002 at umn.edu

Revision \$Revision: 8440 \$

Date \$Date: 2009-01-17 07:39:26 -0800 (Sat, 17 Jan 2009) \$

Last Updated 2009/01/17

Note: For quality antialiased output from mapserver, it is **highly** recommended to use the *AGG* rendering. This document applies only if you wish to stick to the GD rendering, or if you are using a version predating the 5.0 release of mapserver.

Table of Contents

- AntiAliasing with MapServer
 - What needs to be done

8.2.1 What needs to be done

1. Change (or add) `IMAGETYPE` keyword in `MAP` object to `PNG24` (24-bit PNG output) or `JPEG`

MAP

```
...
IMAGETYPE PNG24
...
```

END

1. Add `TRANSPARENCY` to the `LAYER` object and set value to `ALPHA`

MAP

```
...
IMAGETYPE PNG24
...
```

LAYER

```
...
TRANSPARENCY ALPHA
...
```

END

END

1. Add `ANTIALIAS` keyword to the `STYLE` object within the `CLASS` object within the `LAYER` and set value to `TRUE`

MAP

```
...
IMAGETYPE PNG24
...
```

LAYER

```
...
TRANSPARENCY ALPHA
...
```

CLASS

```
...
STYLE
...
ANTIALIAS TRUE
...
```

END

```
...
END # end class
```

```

END # end layer
END # end map

```

Note: Don't use the SYMBOL or the SIZE keywords within the CLASS object, instead use WIDTH to specify width of line or polygon outline. Don't use WIDTH unless you have to. If you must define a SYMBOL, use symbol of type CARTOLINE or ELLIPSE—it supports antialiasing.

Here's an example of a real-world mapfile:

```

1  MAP
2  NAME 'ms101'
3  EXTENT -2198022.00 -2444920.25 2707932.00 1234545.25 # CONUS LAEA (US)
4  SIZE 640 480
5  SHAPEPATH 'data'
6  SYMBOLSET 'symbols/symbols.txt'
7
8  IMAGETYPE PNG24
9
10 PROJECTION
11     "init=epsg:2163"
12 END
13
14 # The layer below will be rendered as 1-pixel wide, antialiased line
15 # If you'd like to change the line thickness add the WIDTH keyword
16 # in the STYLE object with a value of 3 or greater.
17 LAYER # begin antialiased country boundary (line) layer
18     NAME 'country_line'
19     DATA 'shapefile/WorldCountryBorders'
20     TYPE LINE
21     STATUS ON
22     TRANSPARENCY ALPHA
23
24     PROJECTION
25         "init=epsg:4326"
26     END
27
28     CLASS
29         NAME 'Country Boundary'
30         STYLE
31             COLOR 96 96 96
32             ANTIALIAS TRUE
33         END
34     END
35 END # end country boundary layer
36
37 # The layer below shows one way to draw a polygon with antialiased outline
38 LAYER # begin antialiased country boundary (polygon) layer
39     NAME 'country_line'
40     DATA 'shapefile/Countries_area'
41     TYPE POLYGON
42     STATUS ON
43     TRANSPARENCY ALPHA
44
45     PROJECTION
46         "init=epsg:4326"
47     END
48
49     CLASS

```

```
50     NAME 'Country Boundary'
51     STYLE
52         COLOR 212 212 212
53         OUTLINECOLOR 96 96 96
54         WIDTH 3
55         ANTIALIAS TRUE
56     END
57 END
58 END # end country boundary polygon layer
59
60 # The layer below shows one way to draw a polygon with antialiased outline
61 LAYER # begin antialiased state boundary (line) layer
62     NAME 'state_line'
63     DATA 'shapefile/us_states'
64     TYPE LINE
65     STATUS ON
66     TRANSPARENCY ALPHA
67
68     PROJECTION
69         "init=epsg:4326"
70     END
71
72     CLASS
73         NAME 'State Boundary'
74         STYLE
75             COLOR 144 144 144
76             SYMBOL 'cartoline'
77             ANTIALIAS TRUE
78         END
79     END
80 END # end state line layer
81 END # end of map file
```

Here's how the 'cartoline' symbol is defined:

```
SYMBOL
    NAME 'cartoline'
    TYPE CARTOLINE
    LINECAP "round"
    LINEJOIN "round"
    LINEJOINMAXSIZE 3
END
```

Note: The examples provided here are for illustrative purposes only—keep your map file definitions simple. Antialiasing adds computing overhead on the server and could slow/degrade its performance. Don't use it unless you must and certainly don't use symbols with it unless you really have to.

8.3 Dynamic Charting

Author Thomas Bonfort

Contact thomas.bonfort at gmail.com

Revision \$Revision: 8439 \$

Date \$Date: 2009-01-17 07:27:10 -0800 (Sat, 17 Jan 2009) \$

Last Updated 2009/01/17

Table of Contents

- Dynamic Charting
 - Setup
 - Adding a Chart Layer to a Mapfile
 - Pie Charts
 - Bar Graphs

Starting with version 5.0, MapServer included the ability to automatically draw pie or bar graphs whose values are taken and adjusted from attributes of a datasource.

This document assumes that you are already familiar with MapServer application development and especially setting up *Mapfiles*. You can also check out the *Vector Data Access Guide*, which has lots of examples of how to access specific data sources.

8.3.1 Setup

Supported Renderers

Dynamic charts are supported solely with the GD and *AGG* renderers.

Attempting to add a chart layer with any other renderer (e.g. PDF or SWF) will result in an error. Rendering quality with the GD renderer is less than optimal, especially with small graphs, due to the lack of subpixel rendering functions.

Output from AGG and GD Renderers

MapServer AGG Rendering



MapServer GD Rendering



8.3.2 Adding a Chart Layer to a Mapfile

Layer Type

A new type of layer has been added to the mapfile syntax. To specify a chart layer, use

```
LAYER
...
    TYPE CHART
    ...
END
```

No other specific keywords have been added in order to keep the number of different keywords to a minimum in the mapfile syntax, therefore all the chart specific configuration is determined by PROCESSING directives.

Specifying the Size of each Chart

The size of each chart is specified by the CHART_SIZE directive. If two values are given for this parameter, this will specify the width and height of each chart (this only applies for bar graphs). By default, the charts are 20x20 pixels.

```
LAYER
    TYPE CHART
    PROCESSING "CHART_SIZE=21" # specify size of the chart for pie or bar graphs
    #PROCESSING "CHART_SIZE=20 10" # specify width and height for bar graphs
    ...
END
```

From version 5.2 and onwards, the diameter of a pie chart can be bound to an attribute, using the CHART_SIZE_RANGE PROCESSING attribute:

```
PROCESSING "CHART_SIZE_RANGE = itemname minsize maxsize minval maxval"
```

where:

- itemname is the name of the attribute that drives the chart size (e.g. total_sales)
- minsize and maxsize are the minimum and maximum chart size values in pixels (e.g. "10 100")
- minval and maxval are the minimum values of the attribute that correspond to chart sizes of minsize and maxsize (e.g. 10000 1000000).

If the attribute value is smaller than 'minval' then the chart size will be minsize pixels, and if the attribute value is larger than maxval, the chart size will be maxsize pixels.

Specifying the Values to be Plotted

Each value to be plotted (i.e. a slice in a pie chart, or a bar in a bar graph) is specified in a CLASS of the chart layer. The value to be plotted is taken from the SIZE keyword from the first STYLE block of the class. This is semantically a bit awkward, but keeps the number of different keywords to a minimum in the mapfile syntax. The value given to the SIZE keyword could of course be given a static value, but dynamic charting really only makes sense with attribute binding.

```
LAYER
...
    CLASS
        # include a NAME keyword if you want this class to be included
        # in the legend
        NAME "value 1"
        STYLE
            # specify which value from the data source will be used as the
            # value for the graph
```

```

        SIZE [a t t r i b u t e]
        ...
    END
END
CLASS
    ...
END
...
END

```

At least 2 CLASS blocks must be specified before charting can occur (but you already knew this if you want your charts to convey at least *some* information ;)).

Specifying Style

The styling of each value in the charts is specified by the usual MapServer syntax. Only one style per class is supported, any other STYLE block will be silently ignored. Only a subset of the styling keywords are supported:

```

STYLE
    SIZE [a t t r i b u t e]
    # specify the fill color
    COLOR r g b

    # if present will draw an outline around the corresponding bar or slice
    OUTLINECOLOR r g b

    #specify the width of the outline if OUTLINECOLOR is present (defaults to 1)
    WIDTH w

    # only for pie charts. 'a' is the number of pixels the corresponding
    # slice will be offset relative to the center of the pie. This is useful
    # for emphasizing a specific value in each chart. 'b' is required by the
    # mapfile parser but is ignored.
    OFFSET a b
END

```

8.3.3 Pie Charts

This is the default type of chart that is rendered. This can also be specifically set with a PROCESSING keyword in the layer attributes:

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

For each shape in the layer's datasource, the STYLE SIZE is used to set the relative size (value) of each pie slice, with the angles of the slices that are automatically computed so as to form a full pie. For example:

```

1 LAYER
2   NAME "Ages"
3   TYPE CHART
4   CONNECTIONTYPE postgis
5   CONNECTION "blabla"
6   DATA "the_geom from demo"
7   PROCESSING "CHART_TYPE=pie"
8   PROCESSING "CHART_SIZE=30"

```

```

9     STATUS ON
10    CLASS
11      NAME "Population Age 0-19"
12      STYLE
13        SIZE [v]1006]
14        COLOR 255 244 237
15      END
16    END
17    CLASS
18      NAME "Population Age 20-39"
19      STYLE
20        SIZE [v]1007]
21        COLOR 255 217 191
22      END
23    END
24    CLASS
25      NAME "Population Age 40-59"
26      STYLE
27        SIZE [v]1008]
28        COLOR 255 186 140
29      END
30    END
31  END

```

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

8.3.4 Bar Graphs

Bar graph drawing is set with a `PROCESSING` keyword in the layer attributes:

```
PROCESSING "CHART_TYPE=BAR"
```

For each shape in the layer's datasource, the `STYLE SIZE` is used to set the relative size (value) of each bar in the graph. By default, the vertical axis of each bar graph is scaled for the values of the corresponding shape, and will always include the origin (=0). For example

- a shape whose `STYLE SIZES` contains values $\{5,8,10,3\}$ will be plotted on a graph whose vertical axis spans 0 to 10.
- a shape whose `STYLE SIZES` contains values $\{-5,-8,-10,-3\}$ will be plotted on a graph whose vertical axis spans -10 to 0.
- a shape whose `STYLE SIZES` contains values $\{-5,-8,10,3\}$ will be plotted on a graph whose vertical axis spans -8 to 10.

Additional `PROCESSING` directives are used to optionally specify the bounds of vertical axes so that the graphs for all the shapes can be plotted with the same scale:

```
PROCESSING "CHART_BAR_MINVAL=val"
PROCESSING "CHART_BAR_MAXVAL=val"
```

Values in the datasource that are above `CHART_BAR_MAXVAL` or below `CHART_BAR_MINVAL` will be clipped respectively to these values. If only one of these directives is included, the other will be automatically adjusted for each shape to include at least the origin, i.e. the graphs for all the shapes will be in the same scale *only if* all the values are of the same sign (positive or negative).

8.4 Flash Output

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Yewondwossen Assefa

Contact assefa at dmsolutions.ca

Revision \$Revision: 8370 \$

Date \$Date: 2008-12-31 12:32:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2008/07/15

Table of Contents

- Flash Output
 - Introduction
 - Installing MapServer with Flash Support
 - How to Output SWF Files from MapServer
 - What is Currently Supported and Not Supported

8.4.1 Introduction

Since MapServer 4.0, MapServer can output Flash files, in SWF format (or “Shockwave Flash Format”). The following document outlines how to enable Flash output in MapServer.

Links to Flash-Related Information

- [Open Source Flash Viewer](#)
- [Flash maps demo](#)

8.4.2 Installing MapServer with Flash Support

To check that your mapserv executable includes Flash support, use the “-v” command-line switch and look for “OUTPUT=SWF”.

```
$ ./mapserv -v
MapServer version 5.2.0-rc1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
OUTPUT=PDF OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=AGG
SUPPORTS=FREETYPE SUPPORTS=ICONV SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=FASTCGI SUPPORTS=THREADS
SUPPORTS=GEOS SUPPORTS=RGBA_PNG INPUT=JPEG INPUT=POSTGIS
INPUT=ORACLESPATIAL INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

Using Pre-compiled Binaries

Windows users can use [MS4W](#), which supports SWF output.

Compiling MapServer with Flash Support

The library chosen to output SWF files is the [Ming library](#). Ming is a C library for generating SWF (“Flash”) format movies, and it contains a set of wrappers for using the library from C++ and popular scripting languages like PHP, Python, and Ruby.

Building on Windows

- download the [Ming library](#) (the version currently supported is 0.2a)
- as of Ming 0.3 there was no makefile for Windows available in the distribution yet, but you can download a MS VC++ makefile (makefile.vc) from [here](#) (contains makefile and also libming.lib)
- copy makefile.vc under the src directory (ming-0.2/src)
- execute:

```
nmake /f makefile.vc
```

- at this point you should have a libming.lib that will be linked with MapServer
- edit the nmake.opt in your MapServer directory and uncomment the MING=-DUSE_MING_FLASH flag, and point MING_DIR to your Ming directory.
- build MapServer as usual

Building on Unix

Use the “-with-ming” configure flag to enable MING support on Unix. “-with-ming=dir” will try to find the include files and library in the indicated directory.

Note: compiling MapServer 4.4.2 with flash support (mingbeta version 0.3) requires the `-DMING_VERSION_03` option otherwise the make fails. This option should be included in the `configure.in` after `-DUSE_MING_FLASH` as below:

```
MING_ENABLED= "-DUSE_MING_FLASH -DMING_VERSION_03"
```

8.4.3 How to Output SWF Files from MapServer

SWF output is specified by using the *OUTPUTFORMAT* object. There are 2 possible output types:

1. A single movie containing the raster output for all the layers. To enable this, declare the following in the map file:

```
OUTPUTFORMAT
  NAME swf
  MIMETYPE "application/x-shockwave-flash"
  DRIVER swf
  IMAGEMODE PC256
  FORMATOPTION "OUTPUT_MOVIE=SINGLE"
END
```

2. A movie for every layer (vector movies for vector layers and raster movies for raster layers). To enable this, declare the following in the map file:

```

OUTPUTFORMAT
  NAME swf
  MIMETYPE "application/x-shockwave-flash"
  DRIVER swf
  IMAGEMODE PC256
  FORMATOPTION "OUTPUT_MOVIE=MULTIPLE"
END

```

Other OutputFormat Options

- **FORMATOPTION “FULL_RESOLUTION=FALSE”**

The FULL_RESOLUTION applies only for vector layers. If set to FALSE, filtering will be applied to the vector elements. It results in a smaller SWF file. The default value is TRUE.

- **FORMATOPTION “LOAD_AUTOMATICALLY=OFF”**

Setting this option to OFF will not load the SWF files for each layer. The default value is ON.

Composition of the Resulting SWF Files

Several SWF Files will be produced from a single map file: there will be one SWF file for each layer defined in the map file and one ‘main’ SWF file containing critical information on the map file and the layers produced.

- The ‘main’ SWF File will contain Action Script (AS) code that gives critical information on the map file and the SWF layers produced. Basically there will be an object called mapObj containing the height, width, extent, scale, number of layers, etc. Here is an example (in AS) of the contents of this main movie:

```

mapObj = new Object ();
mapObj.name = "DEMO_SWF";
mapObj.width = 400;
mapObj.height = 300;
mapObj.extent = "-2594561.353333,3467361.353333,3467361.353333,3840000.000000"; ;
mapObj.numlayers = 4;
mapObj.layers = new Array ();
function LayerObj (name, type, fullname, relativename) {
  this.name = name;
  this.type = type;
  this.fullname = fullname;
  this.relativename = relativename;
}
mapObj.layers[0] = new LayerObj ("park", "2", "c:/tmp/ms_tmp/102389536132841_layer_0.swf", "1023
mapObj.layers[1] = new LayerObj ("popplace", "4", "c:/tmp/ms_tmp/102389536132841_layer_1.swf", "
mapObj.layers[2] = new LayerObj ("rail", "1", "c:/tmp/ms_tmp/102389536132841_layer_2.swf", "1023
mapObj.layers[3] = new LayerObj ("road", "1", "c:/tmp/ms_tmp/102389536132841_layer_3.swf", "1023

```

This example is produced based on a mapfile with two layers defined in it. We create a layer class object containing useful information on a layer. The parameters are:

- Name : the name found in the map file
- Type : the type of layer (0 = Point Layer; 1=Line; 2=Polygon; 3=Raster; 4=Annotation; 6=Circle)
- Fullname : Full name of the file with path included
- Relative name : Relative Name

For example you can use `mapObj.layers[0].name` to extract the name of the first layer.

Note: All map parameters from MapServer are not exported at this time. We should come up with a list of information of what we want to output. Note that this information can be used in a Flash application to load the SWF file, to build a legend, to build a scale bar, etc.

- SWF Files for each layer

Each layer defined in the mapfile will have an associated SWF file created. The names of these SWF files are based on the name of the main file with an addition of 'layer_X' at the end of the name (where X is the layer index).

These SWF files will contain vector and raster data as well as some Action Script depending on the layer and some configurations in the map file. We will see these configurations in detail in the following section.

Exporting Attributes

Exporting attributes works on a layer basis (it is only available for Vector Layers). To be able to export attributes to the SWF files, you have to define a metadata item called SWFDUMPATTRIBUTES in the layer section of the mapfile. Here is an example :

```
...
LAYER
NAME park
METADATA
  "DESCRIPTION" "Parks"
  "RESULT_FIELDS" "NAME_E YEAR_EST AREA_KMSQ"
  "SWFDUMPATTRIBUTES" "NAME_E, AREA_KMSQ "
END
TYPE POLYGON
STATUS ON
DATA park
...
```

In the above example, the values for the attributes `NAME_E` and `AREA_KMSQ` will be exported for each element in the layer.

The resulting SWF File will have the values of these attributes (written in Action Script). Here is an example related to the above layer:

```
nAttributes= 2;
Attributes = new Array();
Attributes[0] = "NAME_E";
Attributes[1] = "AREA_KMSQ";
Element = new Array ();
Element[0] = new Array();
Element[0][0] = "Ellesmere Island National Park Reserve";
Element[0][1] = "1500";
Element[1][0] = " Aulavik National park";
Element[1][1] = "1500";
```

Events and Highlights

Here is what is currently implemented concerning events (events here refer to mouse events happening on an element. The available events are `MOUSEUP`, `MOUSEDOWN`, `MOUSEOVER`, `MOUSEOUT`):

- Events are only available for layers that have defined attributes exported (using SWFDUMPATTRIBUTES). This is like defining that a certain layer is queryable.
- When a mouse event happens on one of the elements, there is an Action Script call that is made: `_root.ElementSelected(LayerId, ShapeId, Event)`. The Flash application who wants to receive these events should define the function `ElementSelected` and use the information received to do actions like retrieving the attribute values from the specific SWF for the specified shape and display it.

In order to have highlighting, it has to be defined when the SWF is produced (basically highlighting means that the shape is redrawn using a different color).

As of MapServer 5.0, highlighting is available on queryable layers by using the `QueryMap` object in the map file to extract the color and do a highlight when on `MOUSEOVER`. The current implementation will highlight all objects that are in a layer that uses `SWFDUMPATTRIBUTES`, using the `COLOR` set in the `QueryMap` object in the mapfile.

Before MapServer 5.0, all objects that are in a layer that uses `SWFDUMPATTRIBUTES` are highlighted using a red color.

Fonts

Ming uses a special type of font called FDB files. It does not yet support Truetype fonts. Please refer to ming documentation on how to [produce FDB files](#).

Outputting Raster SWF for Vector Layers

One mechanism would be to use the metadata for layer objects to define a raster output for vector layers. We could use something like “SWFOUTPUT” “RASTER”. If this sounds desirable, please file an enhancement [ticket](#) with this request, specifying the “Output-SWF” component.

8.4.4 What is Currently Supported and Not Supported

1. Vector layers

- Layer Point (case `MS_LAYER_POINT`) : *done*
 - `msDrawMarkerSymbol`
 - `msDrawLabel`
- Layer line (case `MS_LAYER_LINE`) : *done*
 - `msDrawLineSymbol`
 - `msDrawLabel`
- Layer circle (case `MS_LAYER_CIRCLE`) : *not done* (should be done easily but missing data for testing)
 - `omsCircleDrawLineSymbol`
 - `omsCircleDrawShadeSymbol`
- Layer annotation (case `MS_LAYER_ANNOTATION`): *done*
 - `omsDrawMarkerSymbol`
 - `omsDrawLabel`
- Layer Polygon (`MS_SHAPE_POLYGON`): *done*
 - `omsDrawShadeSymbol`

- omsDrawLineSymbol
 - omsDrawLabel
 - Vector Low Level functions
 - omsDrawMarkerSymbol
 - * case(MS_SYMBOL_TRUETYPE) : *done*
 - * case(MS_SYMBOL_PIXMAP) : *done*
 - * case(MS_SYMBOL_ELLIPSE) : *done*
 - * case(MS_SYMBOL_VECTOR) : *done*
 - omsDrawLineSymbol
 - * case : simple line : *done*
 - drawing with the symbols : *not done*
 - omsDrawShadeSymbol
 - * case : solid fill polygon : *done*
 - * case : filled with symbols : cannot be implemented for now (tried to create a GD image to fill the shape but files created were huge)
 - omsCircleDrawLineSymbol : *not done*
 - omsCircleDrawShadeSymbol : *not done*
 - omsDrawLabel : *done*
 - omsDrawLabelCache : *done*
 - obillboard (shadow for texts) : *not done*
2. Raster Layer
 - msDrawRasterLayer: *done*
 3. WMS Layer
 - msDrawWMSLayer: *done*
 4. Surround components (Legend, scalebar) : *not supported*

8.5 HTML Legends with MapServer

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8278 \$

Date \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Last Updated 2006/01/09

Contents

- HTML Legends with MapServer
 - Introduction
 - Sample Site Using the HTML Legend

8.5.1 Introduction

The HTML legend is an alternative to the traditional GIF legend in MapServer. The following document describes the process for implementing an HTML legend in MapServer CGI applications (NOTE: MapServer version > 3.5 is required).

This document assumes that you are already familiar with certain aspects of MapServer:

- Setting up MapServer mapfiles and templates.

Implementation

Key components for generating HTML legends are 1) a template parameter in the legend object, 2) a CGI [legend] tag in the HTML file, and 3) an HTML legend template file. So that means that if the HTML page has the CGI [legend] parameter set, and the mapfile has a LEGEND object with its TEMPLATE set to a valid HTML legend file then an HTML legend will be returned. The following sections discuss these components.

Legend Object of Mapfile

The HTML legend is enabled by a new TEMPLATE parameter in the Legend Object of the mapfile. If TEMPLATE is set in the Legend Object, then the HTML legend template file is read and used to generate an HTML legend which will be inserted at the location of the [legend] tag in the main HTML template. Similar to other MapServer templates, the HTML legend template filename MUST end with an “.html” extension.

Example 1. Sample Legend Object with the new TEMPLATE parameter

```
...
# LEGEND object
LEGEND
  STATUS ON
  KEYSIZE 18 12
  # LABEL object
  LABEL
    TYPE BITMAP
    SIZE MEDIUM
    COLOR 0 0 89
  END
  TEMPLATE "legend.html"  ### HTML template file
END
...
```

If TEMPLATE is not set, then the [legend] tag produces a regular image in a GIF/PNG image (the traditional behaviour).

CGI [legend] tag

The traditional CGI [legend] tag returns the URL of an image, so it is usually used inside an `` tag in the HTML file. The new HTML [legend] tag returns a block of HTML, so when converting an existing application template from using a traditional image legend to the new HTML legend, you have to remove the IMG tag in the main application template. Also note that if legend mode is specified in the URL, then MapServer will return a gif containing the whole legend if no template is specified.

See the *CGI Reference doc* for more information on CGI parameters.

Example 2. [legend] tag in the main HTML template (with TEMPLATE set)

```
...
<FONT SIZE=+1><B>Legend</B></FONT><BR><HR> [legend] <HR>
...
```

Example 3. [legend] tag in the main HTML template (with TEMPLATE not set)

```
...
<FONT SIZE=+1><B>Legend</B></FONT><BR><HR><IMG SRC="[legend]"><HR>
...
```

HTML Legend Template File

The HTML legend template file is a separate file that contains 0 or 1 of each of the following tags that define blocks of HTML to use in building the legend:

```
[leg_group_html] ... [/leg_group_html]
[leg_layer_html <OPTIONAL PARAMS>] ... [/leg_layer_html]
[leg_class_html <OPTIONAL PARAMS>] ... [/leg_class_html]
```

Note

Any text or HTML tags outside the [leg_*_html] tag pairs in the legend template file are ignored by the template parser.

The following example shows what an HTML legend TEMPLATE file could look like:

Example 4. An HTML legend TEMPLATE file

```
[leg_group_html]
<tr>
  <td colspan=3 bgcolor=#cccccc><b>[leg_group_name]</b></td>
</tr>
[/leg_group_html]

[leg_layer_html order_metadata=legend_order opt_flag=5]
<tr>
  <td>
    <input type=checkbox name="map_[leg_layer_name]_status"
      value=1 [if name=layer_status oper=eq value=2]CHECKED[/if]>
  </td>
  <td colspan=2>
    <a href="[metadata name=href]">[metadata name=layer_title]</a>
  </td>
</tr >
[/leg_layer_html]
```

```
[leg_class_html]
<tr>
  <td width=15> </td>
  <td>
    
  </td>
  <td>
    [leg_class_name]
  </td>
</tr>
[/leg_class_html]
```

Supported Tags for the TEMPLATE file:

HEADER block

Tag [leg_header_html]...[/leg_header_html]

Description HTML block to use as the header of the legend.

FOOTER block

Tag [leg_footer_html]...[/leg_footer_html]

Description HTML block to use as the footer of the legend.

Example 5. HTML Legend File Using Header/Footer Blocks

```
[leg_header_html]
  <p><b>my header</b></p>
[/leg_header_html]

[leg_layer_html]
  ...
[/leg_layer_html]

[leg_footer_html]
  <p><b>my footer</b></p>
[/leg_footer_html]
```

GROUP block

Tag [leg_group_html <OPTIONAL PARAMS>]...[/leg_group_html]

Description HTML block to use for layer group headers if layers should be grouped in the legend. If not set then layers are not grouped in the legend.

When the [leg_group_html] tag is used, then layers that don't belong to any group (i.e. LAYER GROUP not set in the mapfile) and their classes will not show up at all in the legend. The group list is decided by the order_metadata parameter, which is explained later.

SUPPORTED PARAMETERS:

Parameter opt_flag=<bit_mask>

Description Control the group's display, by adding the following values (default is 15). The `opt_flag` is applied on all layers in the group. If at least one layer matches the flag, the group will show up in the legend.

- 1** If set, show group even if all layers in group are out of scale (default: hide groups out of scale).
- 2** If set, show group even if all layers in group have status OFF (default: hide groups with STATUS OFF).
- 4** If set, show group even if all layers in group are of type QUERY (default: hide group of TYPE QUERY)
- 8** If set, show group even if all layers in group are of type ANNOTATION (default: hide groups of TYPE ANNOTATION)

e.g. `opt_flag=12` (shown below) means show all layer types, including QUERY and ANNOTATION layers (4 + 8)

```
[leg_group_html opt_flag=12]
...
[/leg_group_html]
```

SUPPORTED TAGS:

Tag `[leg_group_name]`

Description Returns the group's name.

Tag `[layer_status]`

Description Returns the status of the first layer in the group.

Tag `[leg_icon width=<optional_width> height=<optional_height>]`

Description In the group context, the `[leg_icon]` tag returns the URL of a legend icon for the first class in the first layer that's part of this group.

Tag `[metadata name=<metadata_field_to_display>]`

Description Returns specified metadata value from web's metadata.

e.g. the group block below simply displays the name of the group in the legend:

```
[leg_group_html]
  <tr><td colspan=2><b>[leg_group_name]</b></td></tr>
[/leg_group_html]
```

LAYER block

Tag `[leg_layer_html <OPTIONAL PARAMS>] ... [/leg_layer_html]`

Description HTML block to use for layer header. If not set then no layer headers are displayed (could allow a legend with only classes in it).

SUPPORTED PARAMETERS:

Parameter `order_metadata=<field_to_order_by>`

Description Specifies that the value of the layer metadata `<field_to_order_by>` controls the order and visibility of the layers in the legend.

- Layers with `<field_to_order_by> >= 0` are sorted in order of this value, with multiple layers with same value being accepted, in which case the map layer order applies between those layers.
- Layers with `<field_to_order_by> < 0` are always hidden in the legend.

Parameter `opt_flag=<bit_mask>`

Description Control the layer display process. Add the values below to acquire the desired options (default is 15):

- 1** If set, show layer even if out of scale (default: hide layers out of scale).
- 2** If set, show layer even if status is OFF (default: hide layers with STATUS OFF).
- 4** If set, show layer even if type is QUERY (default: hide layers of TYPE QUERY)
- 8** If set, show layer even if type is ANNOTATION (default: hide layers of TYPE ANNOTATION)

e.g. `opt_flag=14` (shown below) means do not show layers in the legend that are out of scale.

```
[leg_layer_html opt_flag=14]
...
[/leg_layer_html]
```

SUPPORTED TAGS:

Tag `[leg_layer_group]`

Description Returns the group name of the layer. This was added to MapServer v4.8.

Tag `[leg_layer_index]`

Description Returns the mapfile index value of the layer, which is useful for ordering. This was added to MapServer v4.8.

Tag `[leg_layer_maxscale]`

Description Returns the maximum scale set for the layer. This was added to MapServer v4.8.

Tag `[leg_layer_minscale]`

Description Returns the minimum scale set for the layer. This was added to MapServer v4.8.

Tag `[leg_layer_name]`

Description Returns the current LAYER NAME value.

Tag `[leg_icon width=<optional_width> height=<optional_height>]`

Description In the layer context, the `[leg_icon]` tag returns the URL of a legend icon for the first class in this layer.

Tag `[metadata name=<metadata_field_to_display>]`

Description Returns specified metadata value from this layer's metadata and web's metadata.

e.g. the layer block below simply displays an icon of the layer's class and the layer name:

```
[leg_layer_html]
  <tr><td><img src=[leg_icon width=15 height=15]><b>[leg_layer_name]</b></td></tr>
[/leg_layer_html]
```

CLASS block

Tag [leg_class_html <OPTIONAL PARAMS>] ... [/leg_class_html]

Description HTML block to use for classes. If not set then no classes are displayed (could allow a legend with only layer headers in it). Note that classes with NULL (i.e. empty) NAMES are not displayed.

SUPPORTED PARAMETERS:

Parameter opt_flag=<bit_mask>

Description Control the layer (i.e. class) display process. Add the values below to acquire the desired options (default is 15). Note that using this parameter for the CLASS block has the same effect as using the opt_flag parameter in the LAYER block.

- 1** If set, show layer even if out of scale (default: hide layers out of scale).
- 2** If set, show layer even if status is OFF (default: hide layers with STATUS OFF).
- 4** If set, show layer even if type is QUERY (default: hide layers of TYPE QUERY)
- 8** If set, show layer even if type is ANNOTATION (default: hide layers of TYPE ANNOTATION)

e.g. opt_flag=14 (shown below) means do not show classes in the legend that are out of scale.

```
[leg_class_html opt_flag=14]
...
[/leg_class_html]
```

SUPPORTED TAGS:

Tag [leg_class_index]

Description Returns the mapfile index value of the class, which is useful for ordering and legend icon creation. This was added to MapServer v4.8.

Tag [leg_class_maxscale]

Description Returns the maximum scale set for the class. This was added to MapServer v4.8.

Tag [leg_class_minscale]

Description Returns the minimum scale set for the class. This was added to MapServer v4.8.

Tag [leg_class_name]

Description Returns the CLASS NAME value.

Tag [leg_class_title]

Description Returns the CLASS TITLE value.

Tag [leg_layer_name]

Description Returns the parent layer name. This was added to MapServer v4.8.

Tag [leg_icon width=<optional_width> height=<optional_height>]

Description In the layer context, the [leg_icon] tag returns the URL of a legend icon for the first class in this layer.

Tag [metadata name=<metadata_field_to_display>]

Description Returns specified metadata value from the metadata of the layer to which this class belongs and web's metadata.

e.g. the class block below simply displays an icon of the layer's class and the class name:

```
[leg_class_html]
  <tr><td><img src=[leg_icon width=15 height=15]><b>[leg_class_name]</b></td></tr>
[/leg_class_html]
```

CONDITIONAL text

[if] tags can be used in any of the [leg_*_html] tags above to place conditional text. The syntax is:

```
[if name=<field_to_check> oper=<eq|neq|isset|isnull> value=<to_compare_with_field>] ... [/if]
```

Note:

Nested IF's are supported. Parameter "oper" can be "eq" for equal, "neq" for not equal, "isset" (self-explanatory), or "isnull" (self-explanatory). The default value is equal.

Example 6. [if] tag can be used to maintain the state of a layer checkbox

```
[leg_layer_html order_metadata=legend_order opt_flag=5]
<tr>
  <td>
    <input type=checkbox name="map_[leg_layer_name]_status"
      value=1 [if name=layer_status oper=eq value=2]CHECKED[/if] >
  </td>
  <td colspan=2>
    <a href="[metadata name=href]">[metadata name=layer_title]</a>
  </td>
</tr >
[/leg_layer_html]
```

The possible values that can be tested in an [if] tag depend on the context in which the [if] tag is used. At the moment, the number of values that can be tested is limited, but new values may be added as needed.

Note that the order of the items in the following [if] contexts are listed by their order of precedence. The rule is always that special keywords have top priority (e.g. layer_status, etc.), followed by layer-level metadata, and ending with map-level metadata. The possible values that can be tested are as follows:

In a [leg_group_html] context:

- [if name=layer_status value=...] ... [/if]
 - value is the layer status of the first layer that belongs to the group in integer format: 0=OFF, 1=ON, 2=DEFAULT
- [if name=layer_visible value=...] ... [/if]
 - value is the visibility of the first layer in the group: 0=NOT VISIBLE, 1=VISIBLE
- [if name=group_name value=...] ... [/if]
- [if name=any_layer_metadata value=...] ... [/if]
 - Uses metadata value from the first layer in the mapfile that belongs to that group
- [if name=any_web_metadata value=...] ... [/if]

In a `[leg_layer_html]` context:

- `[if name=layer_status value=...] ... [/if]`
value is the layer's status in integer format: 0=OFF, 1=ON, 2=DEFAULT
- `[if name=layer_type value=...] ... [/if]`
value is the layer's type in integer format: 0=POINT, 1=LINE, 2=POLYGON, 3=RASTER, 4=ANNOTATION, 5=QUERY, 6=CIRCLE
- `[if name=layer_name value=...] ... [/if]`
value is the layer's name in string format
- `[if name=layer_group value=...] ... [/if]`
value is the layer's group name in string format
- `[if name=layer_visible value=...] ... [/if]`
value is the visibility of the layer: 0=NOT VISIBLE, 1=VISIBLE
- `[if name=any_layer_metadata value=...] ... [/if]`
- `[if name=any_web_metadata value=...] ... [/if]`

In a `[leg_class_html]` context:

- `[if name=layer_status value=...] ... [/if]`
value is the status of the layer in which the class is located
- `[if name=layer_type value=...] ... [/if]`
value is the layer's type in integer format: 0=POINT, 1=LINE, 2=POLYGON, 3=RASTER, 4=ANNOTATION, 5=QUERY, 6=CIRCLE
- `[if name=layer_name value=...] ... [/if]`
value is the layer's name in string format
- `[if name=layer_group value=...] ... [/if]`
value is the layer's group name in string format
- `[if name=layer_visible value=...] ... [/if]`
value is the visibility of the layer: 0=NOT VISIBLE, 1=VISIBLE
- `[if name=class_name value=...] ... [/if]`
- `[if name=any_layer_metadata value=...] ... [/if]`
- `[if name=any_web_metadata value=...] ... [/if]`

8.5.2 Sample Site Using the HTML Legend

http://www2.dmsolutions.ca/msapps/itasca_legend/demo_init.html

This demo is based on the MapServer Itasca demo and contains several variations of HTML Legends, some of which are listed below:

- “HTML Legend 1” - displays classes only, similar to the traditional legends:

```
[leg_class_html opt_flag=15]
  <img src=[leg_icon]> [leg_class_name]<br>
[/leg_class_html]
```

- “HTML Legend 2” - displays layer titles with HREF links and classes:

```
[leg_layer_html order_metadata=WMS_ORDER visibility_flag=15]
  <a href="[leg_layer_name]">[metadata name=WMS_TITLE]</a><BR>
[/leg_layer_html]

[leg_class_html visibility_flag=15]
  <img src=[leg_icon]> [leg_class_name]<br>
[/leg_class_html]
```

- “HTML Legend 3” - displays layers by group, with checkboxes to turn layers on/off:

```
[leg_group_html]
  <tr><td colspan=2><b>[leg_group_name]</b></td></tr>
[/leg_group_html]

[leg_layer_html order_metadata=WMS_ORDER opt_flag=15]
  <tr>
    <td><input type=checkbox name=layer value=[leg_layer_name]
      [if name=layer_status value=1]CHECKED[/if]>
      [if name=layer_type value=4]
        <img src=[leg_icon width=22 height=18]>
      [/if]
      [if name=layer_type oper=neq value=4]<img src=[leg_icon]>[/if]
    </td>
    <td>
      <a href="[leg_layer_name]">[metadata name=WMS_TITLE]</a>
    </td>
  </tr>
[/leg_layer_html]
```

8.6 HTML Imagemaps

Author David Fawcett

Contact david.fawcett at gmail.com

Revision \$Revision: 8370 \$

Date \$Date: 2008-12-31 12:32:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2008/10/08

Contents

- HTML Imagemaps
 - Introduction
 - Mapfile Layer Definition
 - Templates
 - Request URL
 - Additional Notes
 - More Information

8.6.1 Introduction

The shpxy method of creating imagemaps uses MapServer query functionality to build a html imagemap. Just like a regular MapServer query, you send a query request and MapServer uses the templates to build a block of html that it sends back to the browser. The first example shows you how to build an imagemap based on a point layer. An example template for a polygon layer is also included.

Components

- MapServer mapfile
- query template file
- query header template
- query footer template

8.6.2 Mapfile Layer Definition

Here is a simple mapfile for our example

```
MAP
NAME "myMapFile"
STATUS ON
SIZE 200 200
EXTENT 178784 4804000 772653 5483346

UNITS METERS
STATUS ON
SHAPEPATH "/web/maps/data"
IMAGECOLOR 255 255 255

WEB
  IMAGEPATH "/web/maps/tmp/"
  IMAGEURL "/maps/tmp/"
END

QUERYMAP
  STATUS ON
  STYLE NORMAL
END

LAYER
  NAME "sites"
  STATUS DEFAULT
  TYPE point
```

```

DATA 'aqiAreas'
  TEMPLATE "bodytemplate.html"
  HEADER "imapheader.html"
  FOOTER "imapfooter.html"
END
END

```

You can see that we have a mapfile with one point layer, and that it contains references to three query templates.

8.6.3 Templates

In MapServer, the query header and footers get processed only once. The main query template, 'bodytemplate.html' in this example, gets processed once for each record in the record set returned by the query.

Point Layers

Here is the query header, 'imapheader.html'. It creates the opening tag for your html imagemap.

```
<map id="mymap" name="mymap">
```

Here is the query template, 'bodytemplate.html'. It creates the body of the html imagemap.

```
<area shape="circle" coords="[shpxy precision=0 proj=image yf=",7" xf=",]" href="http://my.url/mypage"
```

This template is used to create circular imagemap elements for a point layer. NAME is a fieldname in the data source, the value for NAME for each individual record gets substituted as the template is processed. The href specifies the URL link if the element is clicked. Title and alt will display the value when an element is moused over.

The resulting html element looks like

```
<area shape="circle" coords="80,103,7" href="http://my.url/mypage.cfm?region=Northern" >
```

The key part here is

```
coords="[shpxy precision=0 proj=image xf=", " yf=",7"]"
```

This is where MapServer will substitute the image coordinates for that query record. With Precision=0, the coordinates will be integers.

You also see shpxy template formatting options 'xf' and 'yf'. 'xf=","' tells MapServer to place a comma after the x coordinate. 'yf=","7"' after the y coordinate. This is done to specify a radius of 7 pixels for the circle. More options can be found in the [Template Reference](#).

The query footer template simply adds the closing tag for the html imagemap

```
</map>
```

Polygon Layers

Here is a query template for a polygon layer

```
<area shape="poly" coords="[shpxy precision=0 proj=image]" href="http://my.url/mypage.cfm?ID=[SITE_ID]"
```

8.6.4 Request URL

To get the imagemap, you need to send a GET or POST request to MapServer with several URL variables defined. The below URL tells MapServer where the mapfile is located, what layer we are querying, and that we are using nquery mode to return multiple results.

```
http://myurl/cgi-bin/mapserv?map=/web/maps/demoimap.map&qlayer=sites&mode=nquery&searchmap=true
```

8.6.5 Additional Notes

If you use separate map files to generate your imagemap and your map image, make sure that the EXTENT and SIZE specified in both mapfiles are identical. If they are not, your features will not align properly.

8.6.6 More Information

[Steve Lime's SHPXY Example](#)

8.7 PDF Output

Author Yewondwossen Assefa

Contact yassefa at dmsolutions.ca

Revision \$Revision: 8278 \$

Date \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Last Updated 2006/01/12

Table of Contents

- PDF Output
 - Introduction
 - What is currently supported and not supported
 - Implementing PDF Output
 - PHP/MapScript and PDF Output

8.7.1 Introduction

PDF output support was added to MapServer 3.7. Previous versions of MapServer had support for pdf output using a utility program (shp2pdf) to output a pdf file given a MapServer mapfile.

The difference in this new version is that the output to PDF can now be directly specified in the mapfile using the *IMAGETYPE* or the *OUTPUTFORMAT* parameters in the mapfile. Additionally, raster layers are now supported for pdf output.

8.7.2 What is currently supported and not supported

1. Vector Layers

- Layer Point: supported
- Layer Line: supported
- Layer Polygon: supported
- Layer Circle : not supported
- Layer Annotation: supported

Note: Note: Dashed lines are supported with PDFlib version 6 or greater.

Note: Polygons filled with symbols are not supported.

1. Raster Layers

Raster layers are supported. Note that at this point all raster layers are transformed to jpeg format before being written to the PDF file.

2. WMS Layers

Not yet supported

3. Surround components

Legend, scalebar are not supported.

4. Fonts

Standard PostScript fonts are supported. For use of other fonts (such as truetype), see the pdflib documentation for use of UPR description files (some notes on it are [here](#)).

8.7.3 Implementing PDF Output

Note that the following instructions were developed for MapServer 3.7 and pdflib 4.0.3, but the general steps should be similar for recent versions of both.

Build the PDF Library

In order to have access to the PDF support in MapServer, you should download and build the PDF library from <http://www.pdfliib.com/products/pdfliib/>. Please follow the instructions on the PDFLib site to build on your specific platforms.

Here are some quick notes on how to build on windows:

- download and extract the source code from <http://www.pdfliib.com/products/pdfliib/>
- open the project PDFliib.dsw in MS Visual C++
- build the project pdfliib_dll
- after a successful build, you should have a pdfliib.lib and pdbliib.dll under the pdfliib directory
- copy the pdfliib.dll under your system directory (ex : c:/winnt/system32)
- the pdfliib.lib will be used while building mapserver with the PDF support

Build MapServer with PDF support

Windows platform

Edit the `makefile.vc` and uncomment the following lines (make sure that the paths are adapted to your installation):

```
PDF_LIB=../pdflib-4.0.3/pdflib/pdflib.lib
PDF_INC=-I../pdflib-4.0.3/pdflib
PDF=-DUSE_PDF
```

See the *Windows Compilation* document for general MapServer compile instructions.

Unix platforms

Add `with-pdf` to your configure command line before compiling.

See the *Unix Compilation* document for general MapServer compile instructions.

Mapfile definition

The `IMAGETYPE` parameter in the *Mapfile* should be set to `pdf` in order to output to PDF:

```
NAME pdf-test
STATUS ON
...
IMAGETYPE pdf
..

WEB
...
END

LAYER
...
END

END
```

You can also specify the output using the `OUTPUTFORMAT` tag (this tag was introduced in mapserver 3.7) :

```
OUTPUTFORMAT
  NAME pdf
  MIMETYPE "application/x-pdf"
  DRIVER pdf
  FORMATOPTION "OUTPUT_TYPE=RASTER" ##not mandatory
END
```

If the `OUTPUT_TYPE=RASTER` all the layers will be rendered as rasters. Note that when WMS layers are included in the mapfile, this option should be set since there is a problem with transparency and wms layers. See the `OUTPUTFORMAT` object in the *Mapfile* reference for parameter explanations.

Testing

The easiest way to test your pdf output mapfile is with the MapServer *shp2img utility*. Windows users can find this utility in *MS4W*, as well as *FWTools*.

You simply pass a mapfile to the executable and a name for the output pdf, and a pdf file is generated:

```
shp2img -m gmap_pdf.map -o test.pdf
```

Possible Errors

```
PDFlib I/O error: Resource configuration file 'pdflib.upr' not found
```

This is related to fonts. If you remove the LABEL object from your mapfile you will see this error go away. The pdf error is described [here](#). Basically, until this issue is 'fixed', if you want to use a font other than the included standard PostScript fonts in pdf output (such as truetype fonts), consult the PDFlib documentation.

8.7.4 PHP/MapScript and PDF Output

MapServer can render to PDF directly, another option is to render to a PNG and insert that into a PDF document. This is not the only way to create a PDF document of course. You will need to have support for [PDFLib](#) compiled into your PHP install.

This example shows the key parts of the process, you will need to furnish parts of the script yourself (depending on your app) and repeat the process for each map element that you want to include.

Refer to the *PHP/MapScript Reference* wherever necessary.

How does it work?

In brief, we will pass parameters required to render a map to a PHP script that will:

- create a PDF document
- render a PNG view at a suitably higher resolution
- insert the PNG
- buffer it and send it to the user

Create the PDF document

Here is an example similar to the one given on the [PHP website](#) to create a new PDF document:

```
$my_pdf = pdf_new();  
...
```

Get this stage and section 4.5 working before you try inserting MapServer elements.

Render PNG views at a suitable resolution

Work back from the assumption that you will need no more than 300 dpi on your page for your map to look presentable. For an A4 map, I am using 150 dpi for an 8' x 8' main map, which is 1200 x 1200 pixels.

```
$map->set(width,1200);  
$map->set(height,1200);
```

Of course, our map will not be very useful unless it is zoomed in to the extent our user requested, and the layers they selected are switched on. Maintain arrays in your application that record:

- The current extent (say `$ext[]`)
- Layer status (say `$layer[]`)

Open your map file and pass these back through to set the map file into the state the user is expecting, something like:

```
$map->setextent($ext[0], $ext[1], $ext[2], $ext[3]);

while($layer[]) {
    $layer=$map->getLayer($n);
    if($layer[$n]==1) {
        $layer->set(status,1);
    } else {
        $layer->set(status,0);
    }
}
```

Now you will need to save a rendered view to a PNG file.

```
$img = $map->draw();
$url = $img->saveWebImage(MS_PNG, 0, 0, 0);
```

Use the same method for all your map elements, such as `drawReferenceMap?()`, `drawScaleBar?()` and `drawLegend()`.

Insert the PNG elements into your PDF document

This is really easy, use the `pdf_open_image_file()` function to import the map elements into your PDF document:

```
$element = pdf_open_image_file($my_pdf, "png", "$webroot/$url");
pdf_place_image($my_pdf, $element, $xpos, $ypos);
pdf_close_image($my_pdf, $element);
```

Repeat as needed for any map elements you created.

Buffer the PDF and send it to the user

Assuming we have been creating the document `$my_pdf`, when we are done, we merely buffer it and send it to the user using `echo()`:

```
<?php

....
pdf_close($my_pdf);

$data = pdf_get_buffer($my_pdf);

header('Content-type: application/pdf');
header('Content-disposition: inline; filename=my_pdf.pdf');
header('Content-length: ' . strlen($data) );

echo $data;

?>
```

Gotcha: remember that you cannot send headers if you have at any stage outputted text to the browser.

Additional stuff to try

Rendering everything as PNG can look ugly, so I step through the key and extract labels so I can render them using PDF's text functions.

This can be done for other map element, such as map titles, layer descriptions, or anything else that can be read from the mapfile.

8.8 SVG

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8365 \$

Date \$Date: 2008-12-31 07:49:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2005/12/13

Table of Contents

- SVG
 - Introduction
 - Feature Types and SVG Support Status
 - Testing your SVG Output
 - goSVG

8.8.1 Introduction

SVG (or Scalable Vector Graphics) is a standardized XML language for describing 2D graphics via vector graphics, text and raster graphics. As of version 4.5, MapServer can output SVG v1.1 maps. The following documentation is based on the [World Wide Web Consortium's \(W3C\) Scalable Vector Graphics \(SVG\) 1.1 Specification](#).

This document assumes that you are already familiar with certain aspects of MapServer:

- [MapServer application development and setting up map files.](#)

Links to SVG-Related Information

- [SVG 1.1 specification](#)
- [SVG Discussion Paper](#)
- [G-XML Project Page](#)
- [SVG Tiny Profile](#)
- [MapFile Reference Doc](#)

8.8.2 Feature Types and SVG Support Status

Annotation Layers

Annotation layers are supported (see the *Text Features* section below for details).

Circle Layers

Circle layers are not yet supported.

Line Layers

The following items describe how line layers are handled by MapServer for SVG output:

- Lines are converted to SVG `polyline` elements.
- The STYLE object's WIDTH parameter is used for SYMBOL 0 for line thickness.
- The STYLE object's SIZE parameter is used for other symbols for line thickness.
- All lines are drawn without symbols - only line thickness changes.
- If a style uses a symbol and this symbol has a dashed style, it will be transformed into an SVG `stroke-dasharray` element.

Point Layers

The following items describe how point layers are handled by MapServer for SVG output:

- VECTOR, ELLIPSE, and TRUETYPE symbols are supported.
- PIXMAP symbols are not currently supported.
- Labels attached with the symbols are supported (see the *Text Features* section below for details).

Polygon Layers

The following items describe how polygon layers are handled by MapServer for SVG output:

- Polygons are converted to SVG `polygon` elements.
- The STYLE's COLOR is used for the fill.
- The STYLE's OUTLINECOLOR is used for the stroke.
- SVG `patterns` are not currently supported.

Raster Layers

The following items describe how raster layers are handled by MapServer for SVG output:

- Temporary image is created through the GD library, and GD functions are used to draw the layer.
- You must have at least PNG or JPEG support compiled in MapServer.
- You must have the WEB object's IMAGEPATH and IMAGEURL set properly in your mapfile.

Text Features

The following items describe how text features are handled by MapServer for SVG output:

- Text is converted to SVG `text` element.
- Only TRUETYPE fonts are supported.
- Supports labels with ENCODING (output as UTF-8 hexadecimal values).
- The FONT name used in MapServer is parsed to form the SVG `font-family`, `font-style`, and `font-weight`.

WMS Layers

WMS layers are not yet supported.

Setting up a Mapfile for SVG Output

- You must have valid IMAGEPATH and IMAGEURL parameters set in the WEB object of the mapfile.
- To be able to output a valid SVG file, the user needs to define an OUTPUTFORMAT object in the map file and set the IMAGETYPE parameter to `svg`. Here is an example:

```
MAP
  ...
  IMAGETYPE  svg
  ...
  OUTPUTFORMAT
    NAME  svg
    MIMETYPE  "image/svg+xml"
    DRIVER  svg
    FORMATOPTION  "COMPRESSED_OUTPUT=TRUE"
    FORMATOPTION  "FULL_RESOLUTION=TRUE"
  END
  ...
  WEB
    IMAGEPATH  "/tmp/ms_tmp/"
    IMAGEURL  "/ms_tmp/"
  END
  ...
  LAYER
    ...
  END
END
```

Note:

If FORMATOPTION “COMPRESSED_OUTPUT=TRUE” is set MapServer will produce a compressed SVG file (svgz). By default this option is FALSE. Note that to be able to create compressed output, MapServer must be built with the compile flag USE_ZLIB.

If FORMATOPTION “FULL_RESOLUTION=TRUE” is set MapServer will not eliminate duplicate points and collinear lines when outputting SVG. By default this option is set to FALSE.

8.8.3 Testing your SVG Output

- The easiest way to test your SVG mapfile is to use *MapServer CGI*. For example, you might enter the following URL in a browser:

```
http://127.0.0.1/cgi-bin/mapserv.exe?map=my/path/to/my-svg.map&mode=map&layers=layer1 layer2
```

- You can also use *PHP/MapScript* to test your SVG mapfile. Your php file might look like the following:

```
<?php
    dl("php_mapscript_45.dll");

    $oMap = ms_newmapObj("my/path/to/my-svg.map");

    $img = $oMap->draw();

    header("Content-type: image/svg+xml");

    $url = $img->saveImage("");

?>
```

An SVG file should be created in your IMAGEPATH directory. If you open the SVG file in a text editor you can see that it is an XML file. Below is a sample SVG file of a point layer with labels:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-flat.dtd" [
<svg version="1.1" width="400" height="300" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/2000/xlink">

<!-- START LAYER popplace -->
<ellipse cx="252" cy="130" rx="3" ry="3" fill="#000000" />
<ellipse cx="37" cy="227" rx="3" ry="3" fill="#000000" />
<ellipse cx="127" cy="239" rx="3" ry="3" fill="#000000" />
<ellipse cx="255" cy="282" rx="3" ry="3" fill="#000000" />
<polygon fill="#000000" stroke-width="1" points=" 267,263 270,263 271,260 272,263 275,263 273,265 />
<ellipse cx="288" cy="247" rx="3" ry="3" fill="#000000" />
<ellipse cx="313" cy="243" rx="3" ry="3" fill="#000000" />
<ellipse cx="328" cy="233" rx="3" ry="3" fill="#000000" />
<ellipse cx="331" cy="245" rx="3" ry="3" fill="#000000" />
<ellipse cx="366" cy="196" rx="3" ry="3" fill="#000000" />
<ellipse cx="161" cy="246" rx="3" ry="3" fill="#000000" />
<ellipse cx="92" cy="208" rx="3" ry="3" fill="#000000" />
<ellipse cx="40" cy="125" rx="3" ry="3" fill="#000000" />
<ellipse cx="108" cy="146" rx="3" ry="3" fill="#000000" />
<text x="40" y="143" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="43" y="121" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="34" y="205" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="164" y="258" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="316" y="190" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="334" y="258" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="249" y="230" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="241" y="242" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="223" y="260" font-family="fritqat-italic" font-size="8pt" fill="#ff0000" stroke="ffffff" stroke-width="1" />
<text x="210" y="279" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="82" y="234" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="40" y="223" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="214" y="125" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
</svg>
```

You can now view the SVG file in a supported browser (see the official [list of SVG implementations](#) for possible SVG viewers). The [Adobe Viewer plugin](#) is very popular.

8.8.4 goSVG

goSVG is now supported as a vector output format in MapServer 4.5 (and later).

Definition

This definition of goSVG was obtained from [here](#).

goSVG is short for “G-XML over SVG” and “g-contents over SVG”. This is a subset for mobiles specified within the [G-XML](#) (a Japanese Spatial Information Format which is an XML based protocol with the ability to describe, communicate and exchange Spatial Information and Electric Maps), and is a Spatial Information Exchanging format that determines the method to expand spatial information and connect to the backend system(G-XML standard mark format). goSVG is an expanded [SVG Tiny profile](#) (a Mobile profile of [SVG 1.1](#). suited for cellular phones) that adds functions that are useful for Spatial Information Services (SVG Map Service).

Support for Specific goSVG Elements

- Name space extension: supported
- Content Area Definition (bounding box): supported
- Geographic Coordinate System: supported
- Map Request Protocol: supported

Setting up a Mapfile for goSVG Output

Requirements

- A valid MapServer *Mapfile*.
- Valid IMAGEPATH and IMAGEURL parameters set in the WEB object of the mapfile.
- A PROJECTION object defined beneath the MAP object, using an EPSG code. For example:

```
MAP
  ...
  WEB
    IMAGEPATH "/tmp/ms_tmp/"
    IMAGEURL  "/ms_tmp/"
  END
  ...
  PROJECTION
    "init=epsg:42304"
  END
  ...
  LAYER
    ...
  END
END
```

Setting the OUTPUTFORMAT

To be able to output a valid goSVG file, you must define an *OUTPUTFORMAT* object in the mapfile and set the *IMAGETYPE* to *svg*. Here is an example:

```
MAP
...
IMAGETYPE svg
...
OUTPUTFORMAT
  NAME svg
  MIMETYPE "image/svg+xml"
  DRIVER svg
  FORMATOPTION "GOSVG=TRUE"
  FORMATOPTION "GOSVG_ZoomInTH=20"
  FORMATOPTION "GOSVG_ZoomOutTH=40"
  FORMATOPTION "GOSVG_ScrollTH=60"
END
...
WEB
  IMAGEPATH "/tmp/ms_tmp/"
  IMAGEURL  "/ms_tmp/"
END
...
PROJECTION
  "init=epsg:42304"
END
...
LAYER
...
END
END
```

Specific FORMATOPTIONS Related to goSVG

GOSVG should be set to **TRUE**. The default is **false**.

GOSVG_ZoomInTH controls the zoomin threshold when outputting the Map Request Protocol. If it is not defined the default value is set to 70.

GOSVG_ZoomOutTH controls the zoomout threshold when outputting the Map Request Protocol. If it is not defined the default value is set to 100.

GOSVG_ScrollTH controls the scrolling threshold when outputting the Map Request Protocol. If it is not defined the default value is set to 10.

Testing your goSVG Output

Refer to the section [Testing your SVG Output](#) to generate and test your goSVG output. goSVG can be read by regular SVG viewers (they will just ignore the goSVG headers).

Sample goSVG File Produced by MapServer

Below is a sample goSVG file of a point layer with labels:


```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11-flat.dtd" [
<svg version="1.1" width="400" height="300" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/2000/xlink" >
<title>DEMO</title>
<metadata>
<rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:crs = "http://www.opengis.org/2000/coordinateReferenceSystem" xmlns:svg="http://www.w3.org/2000/svg">
<rdf:Description>
<crs:CoordinateReferenceSystem svg:transform="matrix(0.000066,0.000000,0.000000,-0.000066,171.243002,0.000000)"
rdf:resource="http://www.opengis.net/gml/srs/epsg.xml#42304"/>
</rdf:Description>
</rdf:RDF>
<au:lbs protocol="maprequest">
<au:zoomin th="20" xlink:href="."/>
<au:zoomout th="40" xlink:href="."/>
<au:scroll th="60" xlink:href="."/>
</au:lbs>
</metadata>

<!-- START LAYER popplace -->
<ellipse cx="252" cy="130" rx="3" ry="3" fill="#000000" />
<ellipse cx="37" cy="227" rx="3" ry="3" fill="#000000" />
<ellipse cx="127" cy="239" rx="3" ry="3" fill="#000000" />
<ellipse cx="255" cy="282" rx="3" ry="3" fill="#000000" />
<polygon fill="#000000" stroke-width="1" points=" 267,263 270,263 271,260 272,263 275,263 273,265 />
<ellipse cx="288" cy="247" rx="3" ry="3" fill="#000000" />
<ellipse cx="313" cy="243" rx="3" ry="3" fill="#000000" />
<ellipse cx="328" cy="233" rx="3" ry="3" fill="#000000" />
<ellipse cx="331" cy="245" rx="3" ry="3" fill="#000000" />
<ellipse cx="366" cy="196" rx="3" ry="3" fill="#000000" />
<ellipse cx="161" cy="246" rx="3" ry="3" fill="#000000" />
<ellipse cx="92" cy="208" rx="3" ry="3" fill="#000000" />
<ellipse cx="40" cy="125" rx="3" ry="3" fill="#000000" />
<ellipse cx="108" cy="146" rx="3" ry="3" fill="#000000" />
<text x="40" y="143" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="43" y="121" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="34" y="205" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="164" y="258" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="316" y="190" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="334" y="258" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="249" y="230" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="241" y="242" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="223" y="260" font-family="fritqat-italic" font-size="8pt" fill="ff0000" stroke="ffffff" stroke-width="1" />
<text x="210" y="279" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="82" y="234" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="40" y="223" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="214" y="125" font-family="fritqat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
</svg>

```

8.9 Tile Mode

Author Paul Ramsey

Contact pramsey at cleverelephant.ca

Revision \$Revision: 8287 \$

Date \$Date: 2008-12-26 07:25:23 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2008/04/30

Table of Contents

- Tile Mode
 - Introduction
 - Configuration
 - Utilization

8.9.1 Introduction

MapServer can feed tile-based map clients directly using the CGI “tile mode”. Tile-based map clients work by dividing the map of the world up into a discrete number of zoom levels, each partitioned into a number of identically sized “tiles”. Instead of accessing a map by requesting a bounding box, a tile client builds a map by accessing individual tiles.

8.9.2 Configuration

Tile requests are handled by the ‘mapserv’ CGI program. In order to return tiles in the correct projection, MapServer must be built with the `-use-proj` option turned on. You can check if your version of ‘mapserv’ has projection support by running it with the `-v` option and looking for ‘SUPPORTS=PROJ’.

Example 1. On Unix:

```
$ ./mapserv -v
MapServer version 4.6.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP OUTPUT=PDF
OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER
SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
INPUT=JPEG INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE DEBUG=MSDEBUG
```

Example 2. On Windows:

```
C:\apache\cgi-bin> mapserv -v
MapServer version 4.6.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP OUTPUT=PDF
OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER
SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
INPUT=JPEG INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE DEBUG=MSDEBUG
```

MapServer requires that each `LAYER` in your map file have a valid `PROJECTION` block to support reprojection. Because the tile mode uses reprojection, you will have to ensure each `LAYER` has a valid `PROJECTION` block.

Configuration checklist:

- MapServer compiled with PROJ support
- Map file with a *PROJECTION* defined for every *LAYER*

8.9.3 Utilization

The MapServer tile support adds three new directives to the CGI interface:

- `mode=tile` tells the server to generate tiles based on the other tile mode parameters

- `tilemode=gmap` tells the server use the Google Maps tile scheme for the tiles
- `tile=x+y+z` tells the server what tile you want to retrieve, using the Google Maps tile addressing system
- `tilemode=ve` tells the server use the Virtual Earth tile naming scheme for the tiles
- `tile=10231` tells the server what tile you want to retrieve, using the Virtual Earth tile addressing system

About Spherical Mercator

Spherical Mercator (also called “web mercator” by some) is a world projection. All the major tile-based map interfaces (Google Maps, Microsoft Virtual Earth, Yahoo Maps, OpenLayers) use the spherical mercator system to address tiles.

A spherical mercator set of tiles has the following properties:

- The map has been reprojected to mercator using a spherical mercator algorithm
- There is one tile in the top zoom level, zoom level zero
- Each successive zoom level (z) has 2^z tiles along each axis
- Tiles are 256x256 in size

Google Maps and Virtual Earth both use spherical mercator as their underlying tile projection, but use different formats to address the individual tiles.

Google Maps uses an “x”, “y”, “zoom” format. The zoom indicates which level to pull tiles from, and the “x” and “y” indicate while tile in that zoom level to pull.

Virtual Earth uses a single string to address each tile. The top zoom level in Virtual Earth has four tiles (equivalent to Google’s zoom level 1). The top left tile in the Virtual Earth top zoom level is addressed as “0”, top right as “1”, bottom left as “2” and bottom right as “3”. Each tile the next level is addressed by first referencing the top level tile that contains it, then its address relative to that tile. So the top left tile in the second zoom level is “00” and the bottom right one is “33”. See the Virtual Earth site for more details: <http://msdn.microsoft.com/en-us/library/bb545006.aspx>

Using Google Maps

The Google Maps API includes support for using alternative tile sets as overlays, or as alternate base maps. Here is an example of an `GTileLayerOverlay`

```

1  <!DOCTYPE html
2  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml">
5  <head>
6  <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
7  <title>Google/MapServer Tile Example</title>
8  <script src="http://maps.google.com/maps?file=api&v=2&key=[YOUR KEY HERE]"
9  type="text/javascript"></script>
10 <script type="text/javascript">
11
12 function load() {
13     if (GBrowserIsCompatible()) {
14         var urlTemplate = 'http://localhost/cgi-bin/mapserv?';
15         urlTemplate += 'map=/var/map.map&';
16         urlTemplate += 'layers=layer1 layer2&';
17         urlTemplate += 'mode=tile&';
18         urlTemplate += 'tilemode=gmap&';
19         urlTemplate += 'tile={X}+{Y}+{Z}';

```

```

20     var myLayer = new GTileLayer(null,0,18,{
21         templateUrl:urlTemplate,
22         isPng:true,
23         opacity:1.0 });
24     var map = new GMap2(document.getElementById("map"));
25     map.addControl(new GLargeMapControl());
26     map.addControl(new GMapTypeControl());
27     map.setCenter(new GLatLng(35.35, -80.55), 15);
28     map.addOverlay(new GTileLayerOverlay(myLayer));
29 }
30 }
31
32 </script>
33 </head>
34 <body onload="load()" onunload="GUnload()" >
35     <div id="map" style="width: 500px; height: 500px"></div>
36 </body>
37 </html>

```

Note the format of the `tileUrlTemplate`: a valid URL, with {X}, {Y} and {Z} substitution tokens that Google Maps will replace with the tile coordinates and zoom level on the fly to retrieve tiles from your server.

You can also use a MapServer tile layer as an alternate base map:

```

1  <!DOCTYPE html
2     PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml">
5  <head>
6  <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
7  <title>Google/MapServer Tile Example</title>
8  <script src="http://maps.google.com/maps?file=api&v=2&key=[YOUR KEY HERE]"
9     type="text/javascript"></script>
10 <script type="text/javascript">
11
12 function load() {
13     if (GBrowserIsCompatible()) {
14         var urlTemplate = 'http://localhost/cgi-bin/mapserv?';
15         urlTemplate += 'map=/var/map.map&';
16         urlTemplate += 'layers=layer1 layer2&';
17         urlTemplate += 'mode=tile&';
18         urlTemplate += 'tilemode=gmap&';
19         urlTemplate += 'tile={X}+{Y}+{Z}';
20         var myLayer = new GTileLayer(null,0,18,{
21             templateUrl:urlTemplate,
22             isPng:true,
23             opacity:0.3 });
24         var map = new GMap2(document.getElementById("map"));
25         map.addControl(new GLargeMapControl());
26         map.addControl(new GMapTypeControl());
27         map.setCenter(new GLatLng(35.35, -80.55), 15);
28         var myMapType = new GMapType([myLayer], new GMercatorProjection(18), 'MapServer');
29         map.addMapType(myMapType);
30     }
31 }
32
33 </script>
34 </head>

```

```

35 <body onload="load()" onunload="GUnload()">
36   <div id="map" style="width: 500px; height: 500px"></div>
37 </body>
38 </html>

```

The only change from the previous example is that we don't create a `GTileLayerOverlay`, we create a `GMapType`, and use `addMapType()`, instead of `addOverlay()`.

Using Virtual Earth

The Virtual Earth API also includes support for using alternative tile sets as overlays, or as alternate base maps. Here is an example:

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4    <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
5    <title>Virtual Earth Example</title>
6    <script type="text/javascript" src="http://dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.1"></script>
7    <script type="text/javascript">
8
9      var map = null;
10
11     function OnLoadMap () {
12       map = new VEMap("myMap");
13       map.LoadMap();
14
15       var url = "http://localhost/cgi-bin/mapserv?";
16       url += "map=/var/map.map&";
17       url += "mode=tile&";
18       url += "layers=layer1 layer2&";
19       url += "tilemode=ve&";
20       url += "tile=%4";
21
22       var tileSourceSpec = new VETileSourceSpecification( "myLayer", url );
23       tileSourceSpec.Opacity = 0.3;
24       map.AddTileLayer(tileSourceSpec, true);
25     }
26
27   </script>
28 </head>
29 <body onload="OnLoadMap();" >
30   <div id="myMap" style="position:relative; width:500px; height:500px;"></div>
31 </body>
32 </html>

```


OGC SUPPORT AND CONFIGURATION

Interoperability is increasingly becoming a focus point for organizations that distribute and share data over the Internet. The [Open Geospatial Consortium \(OGC\)](#) focuses on the development of publicly available geospatial web standards. MapServer supports numerous OGC standards, allowing users to publish and consume data and services in an application neutral implementation manner.

9.1 WMS Server

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8907 \$

Date \$Date: 2009-04-14 06:38:45 -0700 (Tue, 14 Apr 2009) \$

Table of Contents

- WMS Server
 - Introduction
 - Setting Up a WMS Server Using MapServer
 - WMS 1.3.0 Support
 - Reference Section
 - FAQ / Common Problems

9.1.1 Introduction

A WMS (or Web Map Server) allows for use of data from several different servers, and enables for the creation of a network of Map Servers from which clients can build customized maps. The following documentation is based on the Open Geospatial Consortium's (OGC) [Web Map Server Interfaces Implementation Specification v1.1.1](#).

MapServer v3.5 or more recent is required to implement WMS features. At the time this document was written, MapServer supports the following WMS versions: 1.0.0, 1.0.7, 1.1.0 (a.k.a. 1.0.8), 1.1.1 and 1.3.0

This document assumes that you are already familiar with certain aspects of MapServer:

- MapServer application development and setting up .map files.
- Familiarity with the WMS spec would be an asset. A link to the WMS specification document is included in the "WMS-Related Information" section below.

Links to WMS-Related Information

- *MapServer WMS Client Howto*
- WMS 1.1.1 specification
- WMS 1.3.0 specification
- Open Geospatial Consortium (OGC) home page
- WMS-Dev mailing list and archive
- WMS Cookbook
- MapServer OGC Web Services Workshop package
- *MapServer Styled Layer Descriptor (SLD) Howto*
- *MapServer WMS Time Support Howto*

How does a WMS Work

WMS servers interact with their clients via the HTTP protocol. In most cases, a WMS server is a CGI program. This is also the case with MapServer.

The WMS specification defines a number of request types, and for each of them a set of query parameters and associated behaviors. A WMS-compliant server **MUST** be able to handle at least the following 2 types of WMS requests:

1. **GetCapabilities:** return an XML document with metadata of the Web Map Server's information
2. **GetMap:** return an image of a map according to the user's needs.

And support for the following types is optional:

1. **GetFeatureInfo:** return info about feature(s) at a query (mouse click) location. MapServer supports 3 types of responses to this request:
 - text/plain output with attribute info.
 - text/html output using MapServer query templates specified in the CLASS template parameter. The MIME type returned by the Class templates defaults to text/html and can be controlled using the metadata "wms_feature_info_mime_type".
 - application/vnd.ogc.gml, GML.1 or GML for GML features.
2. **DescribeLayer:** return an XML description of one or more map layers. To execute this:
 - for vector layers: to have a valid return the user needs to setup wfs_onlineresource (or ows_onlineresource) metadata either at the map level or at the layer level (the layer level metadata is the one which is used if both are defined)
 - for raster layers: the metadata is wcs_onlineresource with the same logic as above.
3. **GetLegendGraphic:** returns a legend image (icon) for the requested layer, with label(s). More information on this request can be found in the GetLegendGraphic section later in this doc.

With respect to MapServer specifically, it is the "mapserv" CGI program that knows how to handle WMS requests. So setting up a WMS server with MapServer involves installing the *mapserv* CGI program and a setting up a mapfile with appropriate metadata in it. This is covered in the rest of this document.

9.1.2 Setting Up a WMS Server Using MapServer

Install the Required Software

WMS requests are handled by the *mapserv CGI* program. Not all versions of the *mapserv* program do include WMS support (it is included by default when you compile together with the PROJ library), so the first step is to check that your *mapserv* executable includes WMS support. One way to verify this is to use the “-v” command-line switch and look for “SUPPORTS=WMS_SERVER”.

(Unix users should refer to the *Compiling on Unix* document for any compiling instructions, and Windows users might want to use *MS4W*, which comes ready with WMS/WFS support)

Example 1. On Unix:

```
$ ./mapserv -v
MapServer version 4.6.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP OUTPUT=PDF
OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER
SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
INPUT=JPEG INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE DEBUG=MSDEBUG
```

Example 2. On Windows:

```
C:\apache\cgi-bin> mapserv -v
MapServer version 4.6.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP OUTPUT=PDF
OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER
SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
INPUT=JPEG INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE DEBUG=MSDEBUG
```

Setup a Mapfile For Your WMS

Each instance of WMS server that you setup needs to have its own mapfile. It is just a regular MapServer mapfile in which some parameters and some metadata entries are mandatory. Most of the metadata is required in order to produce a valid GetCapabilities output.

Here is the list of parameters and metadata items that usually optional with MapServer, but are **required (or strongly recommended) for a WMS configuration**:

At the MAP level:

- Map NAME
- Map PROJECTION
- Map Metadata (in the WEB Object):
 - wms_title
 - wms_onlineresource
 - wms_srs (unless PROJECTION object is defined using “init=epsg:...”)

And for each LAYER:

- Layer NAME
- Layer PROJECTION
- Layer METADATA
 - wms_title
 - wms_srs (optional since the layers inherit the map’s SRS value)

- Layer STATUS
 - Layers set to STATUS DEFAULT will always be sent to the client.
 - Layers set to STATUS ON or STATUS OFF can be requested by the client.
- Layer TEMPLATE (required for GetFeatureInfo requests)
- Layer DUMP TRUE (only required for GetFeatureInfo GML requests)

Let's go through each of these parameters in more detail:

- **Map Name and wms_title:**

WMS Capabilities requires a Name and a Title tag for every layer. The Map's NAME and wms_title metadata will be used to set the root layer's name and title in the GetCapabilities XML output. The root layer in the WMS context corresponds to the whole mapfile.

- **Layer Name and wms_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).

- **Map PROJECTION and wms_srs metadata:**

WMS servers have to advertise the projection in which they are able to serve data using EPSG projection codes (see <http://www.epsg.org/> for more background on EPSG codes). Recent versions of the PROJ4 library come with a table of EPSG initialization codes and allow users to define a projection like this:

```
PROJECTION
    "init=epsg:4269"
END
```

(Note that "epsg" has to be in lowercase when used in the PROJ4 'init' directive.)

If the *MAP PROJECTION* block is provided in the format "init=epsg:xxxx" then MapServer will also use this information to generate a <BoundingBox> tag for the top-level layer in the WMS capabilities document. Note that the BoundingBox is an optional element of WMS capabilities, but it is good practice to allow MapServer to include it when possible.

The above is sufficient for MapServer to recognize the EPSG code and include it in SRS tags in the capabilities output (wms_srs metadata is not required in this case). However, it is often impossible to find an EPSG code to match the projection of your data. In those cases, the "wms_srs" metadata is used to list one or more EPSG codes that the data can be served in, and the PROJECTION object contains the real PROJ4 definition of the data's projection.

Here is an example of a server whose data is in an Lambert Conformal Conic projection (42304). It's capabilities output will advertise EPSG:4269 and EPSG:4326 projections (lat/lon), but the PROJECTION object is set to the real projection that the data is in:

```
NAME "DEMO"
...

WEB
...
METADATA
    "wms_title"          "WMS Demo Server"
    "wms_onlineresource" "http://my.host.com/cgi-bin/mapserv?map=wms.map&"
    "wms_srs"           "EPSG:4269 EPSG:4326"
END
```

```

END

PROJECTION
  "init=epsg:42304"
END
...
END

```

In addition to EPSG:xxxx projections, a WMS server can advertize projections in the AUTO:xxxx namespace. AUTO projections 42001 to 42005 are internally supported by MapServer. However, AUTO projections are useful only with smart WMS clients, since the client needs to define the projection parameters in the WMS requests to the server. For more information see Annex E of the [WMS 1.1.1 specification](#) and section 6.5.5.2 of the same document. See also the FAQ on AUTO projections at the end of this document.

- **Layer PROJECTION and wms_srs metadata:**

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 wms_srs for every layer. If a layer PROJECTION is not provided then the top-level map projection will be assumed.

Layer PROJECTION and wms_srs metadata are defined exactly the same way as the map's PROJECTION and wms_srs metadata.

For vector layers, if a PROJECTION block is provided in the format "init=epsg:xxxx" then MapServer will also use this information to generate a <BoundingBox> tag for this layer in the WMS capabilities document. Note that the BoundingBox is an optional element of WMS capabilities, but it is good practice to allow MapServer to include it when possible.

- **"wms_onlineresource" metadata:**

The wms_onlineresource metadata is set in the map's web object metadata and specifies the URL that should be used to access your server. This is required for the GetCapabilities output. If wms_onlineresource is not provided then MapServer will try to provide a default one using the script name and hostname, but you shouldn't count on that too much. It is strongly recommended that you provide the wms_onlineresource metadata.

See section 6.2.2 of the [WMS 1.1.1 specification](#) for the whole story about the online resource URL. Basically, what you need is a complete HTTP URL including the http:// prefix, hostname, script name, potentially a "map=" parameter, and and terminated by "?" or "&".

Here is a valid online resource URL:

```
http://my.host.com/cgi-bin/mapserv?map=mywms.map&
```

By creating a wrapper script on the server it is possible to hide the "map=" parameter from the URL and then your server's online resource URL could be something like:

```
http://my.host.com/cgi-bin/mywms?
```

This is covered in more detail in the section "More About the Online Resource URL" below.

- **Configuring for GetFeatureInfo Requests:**

You must set the layer TEMPLATE parameter for the layer to be queryable by GetFeatureInfo requests. For requests of type "text/html" you should also set the layer HEADER and FOOTER parameters.

For GetFeatureInfo requests of GML you must set the layer to DUMP TRUE in the mapfile. As of MapServer 4.6 you must also set the gml_* metadata for the layer attributes to be served (see the Layer Object metadata in the Reference Section later in this document).

Here are working examples of GetFeatureInfo requests: [text/plain](#) / [text/html](#) / [gml](#) (for gml, your browser might ask you to save the file, if so save it locally as a .gml file and view it in a text editor)

Test Your WMS Server

Validate the Capabilities Metadata

OK, now that we've got a mapfile, we have to check the XML capabilities returned by our server to make sure nothing is missing.

Using a web browser, access your server's online resource URL to which you add the parameters "SERVICE=WMS&VERSION=1.1.1&REQUEST=GetCapabilities" to the end, e.g.

```
http://my.host.com/cgi-bin/mapserv?map=mywms.map&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetCapabilities
```

Here is a working GetCapabilities request (note that the SERVICE parameter is required for all GetCapabilities requests):

```
http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetCapabilities
```

This should return a document of MIME type `application/vnd.ogc.wms_xml`, so your browser is likely going to prompt you to save the file. Save it and open it in a text editor (Emacs, Notepad, etc.), and you will see the returned XML from the WMS server.

If you get an error message in the XML output then take necessary actions. Common problems and solutions are listed in the FAQ at the end of this document.

If everything went well, you should have a complete XML capabilities document. Search it for the word "WARNING"... MapServer inserts XML comments starting with "`<!--WARNING:` " in the XML output if it detects missing mapfile parameters or metadata items. If you notice any warning in your XML output then you have to fix all of them before you can register your server with a WMS client, otherwise things are likely not going to work.

Note that when a request happens, it is passed through WMS, WFS, and WCS in MapServer (in that order) until one of the services respond to it.

Test With a GetMap Request

OK, now that we know that our server can produce a valid XML GetCapabilities response we should test the GetMap request. MapServer only checks for a few of the required GetMap parameters, so both of the minimum MapServer parameters and a valid GetMap request will be explained below.

The following is a list of the required GetMap parameters according to the WMS spec:

VERSION=version: Request version

REQUEST=GetMap: Request name

LAYERS=layer_list: Comma-separated list of one or more map layers. Optional if SLD parameter is present.

STYLES=style_list: Comma-separated list of one rendering style per requested layer. Optional if SLD parameter is present. NOTE that MapServer does not support named styles, so most times you would specify "STYLES=" with an empty value. MapServer does support STYLES when used with an SLD.

SRS=namespace:identifier: Spatial Reference System.

BBOX=minx,miny,maxx,maxy: Bounding box corners (lower left, upper right) in SRS units.

WIDTH=output_width: Width in pixels of map picture.

HEIGHT=output_height: Height in pixels of map picture.

FORMAT=output_format: Output format of map.

Note: WMS Servers only advertise supported formats that are part of the gd / gdal libraries.

A valid example would therefore be:

```
http://my.host.com/cgi-bin/mapserv?map=mywms.map&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&LAYERS=pro
```

Here is a working [valid request](#).

Test with a Real Client

If you have access to a WMS client, then register your new server's online resource with it and you should be off and running.

If you don't have your own WMS client installed already, here are a few pointers:

- MapServer itself can be used as a WMS client, see the *MapServer WMS Client Howto*.
- [Quantum GIS](#) is a full GIS package which includes WMS client support. (recommended)
- [OpenJUMP](#) is a desktop GIS package which includes WMS client support.
- [uDig](#) is a desktop package that allows users to add WMS layers.
- [Deegree](#) provides a WMS client.
- [owsview](#) Viewer Client Generator is an online application that allows users to add WMS layers.

This list is not exhaustive, there are several Open Source or proprietary packages that offer WMS support and could be used to interact with your new MapServer WMS server instance.

More About the Online Resource URL

As mentioned in the section "Setup a Mapfile / wms_onlineresource metadata" above, the following Online Resource URL is perfectly valid for a MapServer WMS according to section 6.2.2 or the WMS 1.1.1 specification:

```
http://my.host.com/cgi-bin/mapserv?map=mywms.map&
```

However, some people will argue that the above URL contains mandatory vendor-specific parameters and that this is illegal. First we would like to point that "map=..." is not considered a vendor-specific parameter in this case since it is part of the Online Resource URL which is defined as an opaque string terminated by "?" or "&" (See [WMS 1.1.1 section 6.2.2](#)).

But anyway, even if it's valid, the above URL is still ugly. And you might want to use a nicer URL for your WMS Online Resource URL. Here are some suggestions:

1. On Unix servers, you can setup a wrapper shell script that sets the MS_MAPFILE environment variable and then passes control to the mapserv executable... that results on a cleaner OnlineResource URL:

```
#!/bin/sh
MS_MAPFILE=/path/to/demo.map
export MS_MAPFILE
/path/to/mapserv
```

1. Another option is to use the “setenvif” feature of Apache: use symbolic links that all point to a same mapserv binary, and then for each symbolic link test the url, and set the MAP environment accordingly.

For Windows and Apache users the steps are as follows (this requires Apache 1.3 or newer):

- Copy mapserv.exe to a new name for your WMS, such as “mywms.exe”.
- In httpd.conf, add:

```
SetEnvIf Request_URI "/cgi-bin/mywms" MS_MAPFILE=/path/to/mymap.map
```

2. On IIS servers (Windows), you can use the following ASP script:

Note: The script below, while functional, is intended only as an example of using ASP to filter MapServer requests. Using ASP in a production WMS server will likely require additional ASP especially in the area of error handling and setting timeouts.*

```
<%
Server.ScriptTimeout = 360

Select Case Request.ServerVariables("REQUEST_METHOD")
    Case "GET" strRequest = Request.QueryString
    Case "POST" strRequest = Request.Form
End Select

strURL = "http://myserver/cgi-bin/mapserv.exe?map=C:\inetpub\wwwroot\workshop\itasca.map&" & s

Dim objHTTP
Set objHTTP = Server.CreateObject("MSXML2.ServerXMLHTTP")
objHTTP.open "GET", strURL, false
objHTTP.send ""

Response.ContentType = objHTTP.getResponseHeader("content-type")
Response.BinaryWrite objHTTP.responseBody

Set objHTTP = Nothing
%>
```

3. Some OGC services (WFS, SOS) support both GET and POST requests. Here, you can use a minimal MapScript WxS wrapper. Here’s a Python example:

```
#!/usr/bin/python

import mapscript

req = mapscript.OWSRequest()
req.loadParams()
map = mapscript.mapObj('/path/to/config.map')
map.OWSDispatch(req)
```

GetLegendGraphic Request

This request returns a legend image (icon) for the specified layer. The request will draw an icon and a label for all classes defined on the layer.

Requirements

The following are required in the WMS server mapfile to enable this request:

- a LEGEND object.
- a CLASS object for each layer.
- a NAME in the CLASS object.
- the STATUS of each LAYER must be set to ON.

Parameters

The following are valid parameters for this request:

- **LAYER** - (Required) Name of the WMS layer to return the legend image of. Note that this is the <Name> parameter of the Layer in the GetCapabilities.
- **FORMAT** - (Required) Format of the legend image (e.g. "image/png").
- **WIDTH** - (Optional) Width of the legend image. Note that the Width parameter is only used when the Rule parameter is also used in the request.
- **HEIGHT** - (Optional) Height of the legend image. Note that the Height parameter is only used when the Rule parameter is also used in the request.
- **SLD** - (Optional) The URL to the SLD. Applies the SLD on the layer and the legend is drawn after the SLD is applied (using the classes specified by the SLD). Note here that you need to put a <Name>class|</Name> inside the Rule element so that a class name is created from the SLD and therefore a correct legend image.
- **SLD_BODY** - (Optional) The body (code) of the SLD, instead of specifying a URL (as in the 'SLD' parameter).
- **SCALE** - (Optional) Specify a scale so that only layers that fall into that scale will have a legend.
- **RULE** - (Optional) Specify the name of the CLASS to generate the legend image for (as opposed to generating an icon and label for ALL classes for the layer).

Note: All rules that are used to draw the legend in normal CGI mode apply here. See the *CGI Reference doc* if necessary.

The *CLASS* object's KEYIMAGE parameter can also be used to specify a legend image for a CLASS. See the *MapFile Reference doc* if necessary. Example Request

An example request might look like:

```
http://127.0.0.1/cgi-bin/mapserv.exe?SERVICE=WMS&VERSION=1.1.1&layer=park&
REQUEST=getlegendgraphic&FORMAT=image/png
```

9.1.3 WMS 1.3.0 Support

MapServer 5.4 adds support for WMS 1.3.0. Although the general mechanism in Mapserver to support this new specification are the same, there are some notable upgrades.

Major features related to the WMS 1.3.0 support

- Support WMS 1.3.0 basic operations: GetCapabilities, GetMap and GetFeatureInfo.
- Implement the [Styled Layer Descriptor profile of the Web Map Service Implementation Specification](#). This specification extends the WMS 1.3.0 and allows to advertise styling capabilities (Styled Layer Descriptor (SLD) support). It also defines two addition operations GetLegendGraphic and DescribeLayer
- Implement the [Symbology Encoding Implementation Specification](#), which is the new version of the SLD. Read support was added for Point, Line, Polygon, Raster symbolizers
- Upgrade the generation of SLD to version 1.1.0 (SLD generated through through the GetStyles operation or through MapScript)

Coordinate Systems and Axis Orientation

The most notable changes introduced in WMS 1.3.0 are the:

- the axis changes
- the introduction of new coordinate reference systems
- the use of CRS parameter (instead of SRS)

The axis order in previous versions of the WMS specifications was to always use easting (x or lon) and northing (y or lat). WMS 1.3.0 specifies that, depending on the particular CRS, the x axis may or may not be oriented West-to-East, and the y axis may or may not be oriented South-to-North. The WMS portrayal operation shall account for axis order. This affects some of the EPSG codes that were commonly used such as EPSG:4326. The current implementation makes sure that coordinates passed to the server (as part of the GetMap BBOX parameter) as well as those advertised in the capabilities document reflect the inverse axe orders for EPSG codes between 4000 and 5000.

In addition, the WMS 1.3.0 defines a series of new coordinate system. These are the once that are currently supported in MapServer:

- CRS:84 (WGS 84 longitude-latitude)
- CRS:83 (NAD83 longitude-latitude)
- CRS:27 (NAD27 longitude-latitude)
- AUTO2:420001 (WGS 84 / Auto UTM)
- AUTO2:420002 (WGS 84 / Auto Tr. Mercator)
- AUTO2:420003 (WGS 84 / Auto Orthographic)
- AUTO2:420004 (WGS 84 / Auto Equirectangular)
- AUTO2:420005 (WGS 84 / Auto Mollweide)

Example of requests

Users can use the CRS:84 coordinate system and order the BBOX coordinates as long/lat: - ...&CRS=CRS:84&BBOX=-180.0,-90.0,180.0,90.0&...

Users can also use the EPSG:4326 coordinates and use the axis odering of lat/long: - ...&EPSG:4326&BBOX=-90.0,-180.0,90,180.0&...

Some Missing features

- WMS 1.3.0 Post request should be an XML document containing the different operations and parameters.
- SLD documents containing elements from the Feature Encoding 1.1 specification could potentially use ESPG projections with some filters. It is not yet clear nor implemented if the axis ordering should be taken into account in these specific cases.

OCG compliance tests

As of version 5.4, MapServer passes all the basic and query tests of the OGC CITE test suite for WMS 1.3.0.

9.1.4 Reference Section

The following metadata are available in the setup of the mapfile:

(Note that each of the metadata below can also be referred to as ‘ows_*

Web Object Metadata

ows_http_max_age

- *Description:* (Optional) an integer (in seconds) to specify how long a given map response should be considered new. Setting this directive allows for aware WMS clients to use this resulting HTTP header value as a means to optimize (and minimize) requests to a WMS Server. More info is available at http://www.mnot.net/cache_docs/#CACHE-CONTROL

ows_updatesequence

- *Description:* (Optional) The updateSequence parameter can be used for maintaining the consistency of a client cache of the contents of a service metadata document. The parameter value can be an integer, a timestamp in [ISO 8601:2000] format, or any other number or string.

ows_sld_enabled

- *Description:* (Optional) A value (true or false) which, when set to “false”, will ignore SLD and SLD_BODY parameters in order to disable remote styling of WMS layers. Also, SLD is not advertised in WMS Capabilities as a result

ows_schemas_location

- *Description:* (Optional) (Note the name ows_schemas_location and not wms_... this is because all OGC Web Services (OWS) use the same metadata) Root of the web tree where the family of OGC WMS XMLSchema files are located. This must be a valid URL where the actual .xsd files are located if you want your WMS output to validate in a validating XML parser. Default is <http://schemas.opengespatial.net>. See <http://ogc.dmsolutions.ca> for an example of a valid schema tree.

wms_abstract

- *WMS TAG Name:* Abstract (WMS1.1.1, sect. 7.1.4.2)
- *Description:* (Optional) A blurb of text providing more information about the WMS server.

wms_accessconstraints

- *WMS TAG Name:* AccessConstraints (WMS1.1.1, sect. 7.1.4.2)

- *Description:* (Optional) Access constraints information. Use the reserved word “none” if there are no access constraints.

wms_adresstype, wms_address, wms_city, wms_stateorprovince, wms_postcode, wms_country

- *WMS TAG Name:* ContactAddress and family (WMS1.1.1, sect. 7.1.4.2)
- *Description:* Optional contact address information. If provided then all six metadata items are required.

wms_attribution_logourl_format

- *Description:* (Optional) The MIME type of the logo image. (e.g. “image/png”). Note that the other wms_attribution_logourl_* metadata must also be specified.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_logourl_height

- *Description:* (Optional) Height of the logo image in pixels. Note that the other wms_attribution_logourl_* metadata must also be specified.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_logourl_href

- *Description:* (Optional) URL of the logo image. Note that the other wms_attribution_logourl_* metadata must also be specified.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_logourl_width

- *Description:* (Optional) Width of the logo image in pixels. Note that the other wms_attribution_logourl_* metadata must also be specified.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_onlineresource

- *Description:* (Optional) The data provider’s URL.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_title

- *Description:* (Optional) Human-readable string naming the data provider.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_contactelectronicmailaddress

- *WMS TAG Name:* ContactElectronicMailAddress (WMS1.1.1, sect. 7.1.4.2)
- *Description:* Optional contact Email address.

wms_contactfacsimiletelephone

- *WMS TAG Name:* ContactFacsimileTelephone (WMS1.1.1, sect. 7.1.4.2)
- *Description:* Optional contact facsimile telephone number.

wms_contactperson, wms_contactorganization, wms_contactposition

- *WMS TAG Name:* ContactInformation, ContactPerson, ContactOrganization, ContactPosition (WMS1.1.1, sect. 7.1.4.2)
- *Description:* Optional contact information. If provided then all three metadata items are required.

wms_contactvoicetelephone

- *WMS TAG Name:* ContactVoiceTelephone (WMS1.1.1, sect. 7.1.4.2)
- *Description:* Optional contact voice telephone number.

wms_encoding

- *WMS TAG Name:* Encoding
- *Description:* Optional XML capabilities encoding type. The default is ISO-8859-1.

wms_feature_info_mime_type

- *WMS TAG Name:* Feature_info_mime_type
- *Description:*
 - Used to specify an additional MIME type that can be used when responding to the GetFeature request.
For example if you want to use the layer's HTML template as a base for its response, you need to add "WMS_FEATURE_INFO_MIME_TYPE" "text/html". Setting this will have the effect of advertizing text/html as one of the MIME types supported for a GetFeature request. You also need to make sure that the layer points to a valid html template. The client can then call the server with INFO_FORMAT=text/html.
 - If not specified, MapServer by default has text/plain and GML implemented.
 - Note that for GML to be returned the layer in the wms-server mapfile must be set to DUMP TRUE.

wms_fees

- *WMS TAG Name:* Fees (WMS1.1.1, sect. 7.1.4.2)
- *Description:* (Optional) Fees information. Use the reserved word "none" if there are no fees.

wms_keywordlist

- *WMS TAG Name:* KeywordList (WMS1.1.1, sect. 7.1.4.2)
- *Description:* (Optional) A comma-separated list of keywords or keyword phrases to help catalog searching. As of WMS 1.1.0 no controlled vocabulary has been defined.

wms_onlineresource

- *WMS TAG Name:* OnlineResource (WMS1.1.1, sect. 6.2.2)
- *Description:* (Recommended) The URL that will be used to access this WMS server. This value is used in the GetCapabilities response.
- *See Also:* sections "Setup a Mapfile / wms_onlineresource metadata" and "More About the Online Resource URL" above.

wms_resx, wms_resy

- *WMS TAG Name:* BoundingBox (WMS1.1.1, sect. 6.5.6)
- *Description:* (Optional) Used in the BoundingBox tag to provide info about spatial resolution of the data, values are in map projection units.

wms_service_onlineresource

- *Description:* (Optional) Top-level onlineresource URL. MapServer uses the onlineresource metadata (if provided) in the following order:
 1. wms_service_onlineresource
 2. ows_service_onlineresource
 3. wms_onlineresource (or automatically generated URL, see the onlineresource section of this document)

wms_srs

- *WMS TAG Name:* SRS (WMS1.1.1, sect. 6.5.5)
- *Description:* (Recommended) Contains a list of EPSG projection codes that should be advertized as being available for all layers in this server. The value can contain one or more EPSG:<code> pairs separated by spaces (e.g. “EPSG:4269 EPSG:4326”) This value should be upper case (EPSG:42304.....not epsg:42304) to avoid problems with case sensitive platforms.
- See Also: section “Setup a Mapfile / Map PROJECTION and wms_srs metadata” above.

wms_timeformat

- *Description:* The time format to be used when a request is sent. (e.g. “wms_timeformat” “%Y-%m-%d %H,%Y-%m-%d %H:%M”). Please see the [WMS Time Support Howto](#) for more information.

wms_title

- *WMS TAG Name:* Title (WMS1.1.1, sect. 7.1.4.1)
- *Description:* (Required) A human-readable name for this Layer.

Layer Object Metadata

gml_exclude_items

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) A comma delimited list of items to exclude. As of MapServer 4.6, you can control how many attributes (fields) you expose for your data layer with metadata. The previous behaviour was simply to expose all attributes all of the time. The default is to expose no attributes at all. An example excluding a specific field would be:

```
"gml_include_items" "all"  
"gml_exclude_items" "Phonenumber"
```

gml_groups

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) A comma delimited list of group names for the layer.

gml_[group name]_group

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) A comma delimited list of attributes in the group. Here is an example:

```
"gml_include_items" "all"  
"gml_groups" "display"  
"gml_display_group" "Name_e,Name_f"
```

gml_include_items

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) A comma delimited list of items to include, or keyword “all”. As of MapServer 4.6, you can control how many attributes (fields) you expose for your data layer with this metadata. The previous behaviour was simply to expose all attributes all of the time. You can enable full exposure by using the keyword “all”, such as:

```
"gml_include_items" "all"
```

You can specify a list of attributes (fields) for partial exposure, such as:

```
"gml_include_items" "Name, ID"
```

The new default behaviour is to expose no attributes at all.

gml_[item name]_alias

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) An alias for an attribute's name. The served GML will refer to this attribute by the alias. Here is an example:

```
"gml_province_alias" "prov"
```

gml_[item name]_type

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) Specifies the type of the attribute. Valid values are Integer|Real|Character|Date|Boolean.

gml_xml_items

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) A comma delimited list of items that should not be XML-encoded.

wms_abstract

- Same as wms_abstract in the Web Object.

wms_attribution_logourl_format

- *Description:* (Optional) The MIME type of the logo image. (e.g. "image/png"). Note that the other wms_attribution_logourl_* metadata must also be specified.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_logourl_height

- *Description:* (Optional) Height of the logo image in pixels. Note that the other wms_attribution_logourl_* metadata must also be specified.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_logourl_href

- *Description:* (Optional) URL of the logo image. Note that the other wms_attribution_logourl_* metadata must also be specified.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_logourl_width

- *Description:* (Optional) Width of the logo image in pixels. Note that the other wms_attribution_logourl_* metadata must also be specified.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_onlineresource

- *Description:* (Optional) The data provider's URL.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_attribution_title

- *Description:* (Optional) Human-readable string naming the data provider.
- refer to section 7.1.4.5.11 of the WMS 1.1.1 spec.

wms_dataurl_format

- *Description:* (Optional) Non-standardized file format of the metadata. The layer metadata wms_dataurl_href must also be specified.
- refer to section 7.1.4.5.14 of the WMS 1.1.1 spec.

wms_dataurl_href

- *Description:* (Optional) The URL to the layer's metadata. The layer metadata wms_dataurl_format must also be specified.
- refer to section 7.1.4.5.14 of the WMS 1.1.1 spec.

wms_extent

- *WMS TAG Name:* BoundingBox (WMS1.1.1, sect. 6.5.6)
- *Description:* (Optional) Used for the layer's BoundingBox tag for cases where it is impossible (or very inefficient) for MapServer to probe the data source to figure its extents. The value for this metadata is "minx miny maxx maxy" separated by spaces, with the values in the layer's projection units. If wms_extent is provided then it has priority and MapServer will NOT try to read the source file's extents.

For Rasters served through WMS, MapServer can now use the wms_extent metadata parameter to register the image. If a .wld file cannot be found, MapServer will then look for the wms_extent metadata parameter and use the extents of the image and the size of the image for georegistration.

wms_group_abstract

- *Description:* (Optional) A blurb of text providing more information about the group. Only one layer for the group needs to contain wms_group_abstract, MapServer will find and use the value. The value found for the first layer in the group is used. So if multiple layers have wms_group_abstract set then only the first value is used.

wms_group_title

- *WMS TAG Name:* Group_title (WMS1.1.1, sect. 7.1.4.1)
- *Description:* (Optional) A human-readable name for the group that this layer belongs to. Only one layer for the group needs to contain wms_group_title, MapServer will find and use the value. The value found for the first layer in the group is used. So if multiple layers have wms_group_title set then only the first value is used.

wms_keywordlist

- Same as wms_keywordlist in the Web Object .

wms_layer_group

- *Description:* (Optional) Can be used to assign a layer to a number of hierarchically nested groups. This grouped hierarchy will be expressed in the capabilities.

WMS_LAYER_GROUP is different from the GROUP keyword in that it does not publish the name of the group in the capabilities, only the title. As a consequence the groups set with WMS_LAYER_GROUP can not be requested with a GetMap or GetFeatureInfo request (see section 7.1.4.5.2 of the WMS implementation specification version 1.1.1. (OGC 01-068r2)). Another difference is that GROUP does not support nested groups. The purpose of this metadata setting is to enable making a WMS client aware of layer grouping.

All group names should be preceded by a forward slash (/). It is not allowed to use both the WMS_LAYER_GROUP setting and the GROUP keyword for a single layer.

```
LAYER
  NAME "mylayer"
  DATA "mylayer"
  TYPE LINE
  CLASS
    STYLE
      COLOR 100 100 255
    END
  END
METADATA
```

```

    "WMS_LAYER_GROUP" "/rootgroup/subgroup"
  END
END

```

wms_metadatal_format

- *Description:* (Optional) The file format MIME type of the metadata record (e.g. “text/plain”). The layer metadata wms_metadatal_type and wms_metadatal_href must also be specified.
- refer to section 7.1.4.5.10 of the WMS 1.1.1 spec.

wms_metadatal_href

- *Description:* (Optional) The URL to the layer’s metadata. The layer metadata wms_metadatal_format and wms_metadatal_type must also be specified.
- refer to section 7.1.4.5.10 of the WMS 1.1.1 spec.

wms_metadatal_type

- *Description:* (Optional) The standard to which the metadata complies. Currently only two types are valid: “TC211” which refers to [ISO 19115], and “FGDC” which refers to [FGDC-STD-001-1988]. The layer metadata wms_metadatal_format and wms_metadatal_href must also be specified.
- refer to section 7.1.4.5.10 of the WMS 1.1.1 spec.

wms_opaque

- *WMS TAG Name:* Opaque (WMS1.1.1, sect. 7.1.4.6.3)
- *Description:* (Optional) Set this metadata to “1” to indicate that the layer represents an area-filling coverage of space (e.g. a bathymetry and elevation layer). This should be taken by the client as a hint that this layer should be placed at the bottom of the stack of layers.

wms_srs

- Same as wms_srs in the Web Object .

wms_style

- *Description:* (Optional) The LegendURL style name. Requires the following metadata: wms_style_<style’s_name>_width, wms_style_<style’s_name>_legendurl_height, wms_style_<style’s_name>_legendurl_format, wms_style_<style’s_name>_legendurl_href
- refer to section 7.1.4.5.4 of the WMS 1.1.1 spec.

wms_style_<style’s_name>_legendurl_height

- *Description:* (Optional) The height of the legend image in pixels. Requires the following metadata: wms_style_<style’s_name>_width, wms_style, wms_style_<style’s_name>_legendurl_format, wms_style_<style’s_name>_legendurl_href.
- refer to section 7.1.4.5.4 of the WMS 1.1.1 spec.

wms_style_<style’s_name>_legendurl_href

- *Description:* (Optional) The URL to the layer’s legend. Requires the following metadata: wms_style_<style’s_name>_width, wms_style_<style’s_name>_legendurl_height, wms_style_<style’s_name>_legendurl_format, wms_style.
- refer to section 7.1.4.5.4 of the WMS 1.1.1 spec.

wms_style_<style’s_name>_legendurl_format

- *Description:* (Optional) The file format MIME type of the legend image. Requires the following metadata: `wms_style_<style's_name>_width`, `wms_style_<style's_name>_legendurl_height`, `wms_style_<style's_name>_legendurl_href`.
- refer to section 7.1.4.5.4 of the WMS 1.1.1 spec.

wms_style_<style's_name>_legendurl_width

- *Description:* (Optional) The width of the legend image in pixels. Requires the following metadata: `wms_style_<style's_name>_format`, `wms_style_<style's_name>_legendurl_height`, `wms_style_<style's_name>_legendurl_href`.
- refer to section 7.1.4.5.4 of the WMS 1.1.1 spec.

wms_timedefault

- *Description:* (Optional for Time Support) This value is used if it is defined and the Time value is missing in the request. Please see the [WMS Time Support Howto](#) for more information.

wms_timeextent

- *Description:* (Mandatory for Time Support) This is used in the capabilities to return the valid time values for the layer. The value defined here should be a valid time range. Please see the [WMS Time Support Howto](#) for more information.

wms_timeitem

- *Description:* (Mandatory for Time Support) This is the name of the field in the DB that contains the time values. Please see the [WMS Time Support Howto](#) for more information.

wms_title

- Same as `wms_title` in the Web Object.

Sample WMS Server Mapfile

The following is a very basic WMS Server mapfile:

```
1  NAME "WMS-test"
2  STATUS ON
3  SIZE 400 300
4  SYMBOLSET ../etc/symbols.sym
5  EXTENT -2200000 -712631 3072800 3840000
6  UNITS METERS
7  SHAPEPATH "../data"
8  IMAGECOLOR 255 255 255
9  FONTSET ../etc/fonts.txt
10
11  WEB
12  IMAGEPATH "/ms4w/tmp/ms_tmp/"
13  IMAGEURL "/ms_tmp/"
14  METADATA
15  "wms_title" "WMS Demo Server" ##required
16  "wms_onlineresource" "http://yourpath/cgi-bin/mapserv.exe?" ##required
17  "wms_srs" "EPSG:42304 EPSG:42101 EPSG:4269 EPSG:4326" ##recommended
18  END
19  END
20
21  PROJECTION
22  "init=epsg:42304" ##required
23  END
```



```

24
25 #
26 # Start of layer definitions
27 #
28
29 LAYER
30   NAME "park"
31   METADATA
32     "wms_title"      "Parks"    ##required
33   END
34   TYPE POLYGON
35   STATUS OFF
36   DATA p a r k
37   PROJECTION
38     "init=epsg:42304"    ##recommended
39   END
40   CLASS
41     NAME "Parks"
42     STYLE
43       COLOR 200 255 0
44       OUTLINECOLOR 120 120 120
45     END
46   END
47 END # Layer
48
49 LAYER
50   NAME p o p p l a c e
51   METADATA
52     "wms_title"      "Cities"    ##required
53   END
54   TYPE POINT
55   STATUS ON
56   DATA p o p p l a c e
57   PROJECTION
58     "init=epsg:42304"    ##recommended
59   END
60   CLASS
61     NAME "Cities"
62     STYLE
63       SYMBOL 2
64       SIZE 8
65       COLOR 0 0 0
66     END
67   END
68 END # Layer
69
70 END # Map File

```

9.1.5 FAQ / Common Problems

Q How can I find the EPSG code for my data's projection?

A If you know the parameters of your data's projection, then you can browse the "epsg" file that comes with PROJ4 and look for a projection definition that matches your data's projection. It's a simple text file and the EPSG code is inside brackets (<...>) at the beginning of every line.

The "epsg" file is usually located in /usr/local/share/proj/ on Unix systems and in C:/PROJ/ or

C:/PROJ/NAD in Windows systems (depending on the installation). MS4W users will find the epsg file in /MS4W/proj/nad/.

Q My WMS server produces the error “msProcessProjection(): no system list, errno: ..”

A That’s likely PROJ4 complaining that it cannot find the “epsg” projection definition file. Make sure you have installed PROJ 4.4.3 or more recent and that the “epsg” file is installed at the right location. On Unix it should be under /usr/local/share/proj/, and on Windows PROJ looks for it under C:/PROJ/ or C:/PROJ/NAD (depending on the installation). You should also check the *error documentation* to see if your exact error is discussed.

If you don’t have the “epsg” file then you can get it as part of the PROJ4 distribution at <http://trac.osgeo.org/proj/> or you can download it at <http://www.maptools.org/dl/proj4-epsg.zip>.

Q How do AUTO projections work?

A When a WMS client calls a WMS server with an auto projection, it has to specify the SRS in the form: AUTO: proj_id,unit_id,lon0,lat0 where:

- proj_id is one of 42001, 42002, 42003, 42004, or 42005 (only five auto projections are currently defined).
- unit_id is always 9001 for meters. (It is uncertain whether anyone supports any other units.)
- lon0 and lat0 are the coordinates to use as the origin for the projection.

When using an AUTO projection in WMS GetCapabilities, you include only the “AUTO:42003” string in your wms_srs metadata, you do not include the projection parameters. Those are added by the application (client) at runtime depending on the map view. For example:

```
NAME "DEMO"
...

WEB
...

METADATA
  "wms_title"           "WMS Demo Server"
  "wms_onlineresource" "http://my.host.com/cgi-bin/mapserv?map=wms.map&"
  "wms_srs"            "AUTO:42001 AUTO:42002"

END
END
```

The above server advertises the first two auto projections.

9.2 WMS Client

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8371 \$

Date \$Date: 2008-12-31 12:44:04 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2008/07/15

Table of Contents

- WMS Client
 - Introduction
 - Compilation / Installation
 - MapFile Configuration
 - Limitations/TODO

9.2.1 Introduction

A WMS (or Web Map Server) allows for use of data from several different servers, and enables for the creation of a network of Map Servers from which clients can build customized maps. The following document contains information about using MapServer's WMS connection type to include layers from remote WMS servers in MapServer applications.

This document assumes that you are already familiar with certain aspects of MapServer:

- MapServer application development and setting up .map files.
- Familiarity with the WMS spec would be an asset. A link to the WMS specification document is included below.

WMS-Related Information

- [MapServer WMS Server HowTo](#)
- [WMS 1.1.1 specification](#)
- [MapServer OGC Web Services Workshop package](#)

9.2.2 Compilation / Installation

The WMS connection type is enabled by the `–with-wmsclient` configure switch. It requires PROJ4, libcurl version 7.10.1 or more recent, and optionally GDAL (see below). Windows users who do not want to compile MapServer should use [MS4W](#) (which comes ready for WMS/WFS client and server use), or check for the availability of other [Windows binaries](#) with WMS support.

- For PROJ4 and GDAL installation, see the MapServer Compilation HowTo ([Compiling on Unix / Compiling on Win32](#))
- For libcurl, make sure you have version 7.10.1 or more recent installed on your system. You can find out your libcurl version using `curl-config –version`. (if your system came with an older version of libcurl preinstalled then you MUST uninstall it prior to installing the new version)

You might want to also include GDAL support if you want your application to be able to reproject map slides received from remote servers. This is because raster resampling works only together with the GDAL library in MapServer. If GDAL is not included in your MapServer build then your application can only serve maps in the subset of the projections supported by all the remote servers (this should be sufficient for most applications). If you compile with GDAL then make sure that your GDAL includes GIF and/or PNG support, depending on which image format you request from remote servers.

Once the required libraries are installed, then configure MapServer using the `–with-wmsclient` switch (plus all the other switches you used to use) and recompile. This will give you a new set of executables (and possibly `php_mapscript.so` if you requested it). See the MapServer Compilation HowTo (links above) for installation details.

Check your MapServer executable

To check that your mapserv executable includes WMS support, use the “-v” command-line switch and look for “SUPPORTS=WMS_CLIENT”.

Example 1. Mapserv Version Info on Unix:

```
$ ./mapserv -v
MapServer version 5.2.0-rc1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
OUTPUT=PDF OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=AGG
SUPPORTS=FREETYPE SUPPORTS=ICONV SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=FASTCGI SUPPORTS=THREADS
SUPPORTS=GEOS SUPPORTS=RGBA_PNG INPUT=JPEG INPUT=POSTGIS
INPUT=ORACLESPATIAL INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

Example 2. Mapserv Version Info on Windows:

```
C:\ms4w\apache\cgi-bin> mapserv -v
MapServer version 5.2.0-rc1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
OUTPUT=PDF OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=AGG
SUPPORTS=FREETYPE SUPPORTS=ICONV SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=FASTCGI SUPPORTS=THREADS
SUPPORTS=GEOS SUPPORTS=RGBA_PNG INPUT=JPEG INPUT=POSTGIS
INPUT=ORACLESPATIAL INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

Install Optional PROJ4 EPSG Codes

(Note: installing these PROJ4 codes is optional, install only if you need them)

Some Canadian WMS servers will use some non-standard projection codes not included in the default distribution (e.g. EPSG:42304, etc.). If you are planning to use MapServer to connect to Canadian WMS servers then you might want to [download a custom Canadian epsg file](#) with those codes, and unzip it in the /usr/local/share/proj directory (or /ms4w/proj/nad/ for MS4W users).

Finally, ESRI WMS servers also come with their own series of non-standard codes. If you are planning to connect to ESRI WMS servers then you might want to get a custom epsg file that contains the canadian codes and the ESRI codes, allowing you to connect to any server. Download the [custom ESRI epsg file](#) and unzip it in /usr/local/share/proj (or /ms4w/proj/nad/ for MS4W users).

Q But why not always install and distribute the proj4-epsg-with-42xxx-and-esri.zip file then since it's more complete?

A You should install only the epsg projection codes that you need, the epsg file with all ESRI codes in it is 20% larger than the default one, so it comes with extra overhead that you may not need. Also note that when creating WMS servers, in order to be really interoperable, only EPSG codes that are part of the standard EPSG list should be used. i.e. it is a bad idea for interoperability to use the custom canadian codes or the custom ESRI codes and we do not want to promote their use too much.

9.2.3 MapFile Configuration

Note: A PROJECTION must be set in the mapfile for the MAP unless you are sure that all your WMS layers support only a single projection which is the same as the PROJECTION of the map. The MAP PROJECTION can be set using “init=epsg:xxxx” codes or using regular PROJ4 parameters. Failure to set a MAP PROJECTION may result in blank

maps coming from remote WMS servers (because of inconsistent BBOX+SRS combination being used in the WMS connection URL).

Storing Temporary Files

You have to set the IMAGEPATH value in the WEB object of your mapfile to point to a valid and writable directory. MapServer will use this directory to store temporary files downloaded from the remote servers. The temporary files are automatically deleted by MapServer so you won't notice them... but a valid IMAGEPATH is still required.

Example 3. Setting IMAGEPATH Parameter in Mapfile

```
MAP
...
WEB
  IMAGEPATH "/tmp/ms_tmp/"
  IMAGEURL ...
END
...
END
```

If you want to keep this temporary file for debugging purposes, you should add the following statement to the LAYER object of your mapfile:

```
LAYER
...
  DEBUG ON
...
END
```

Adding a WMS Layer

WMS layers are accessed via the WMS connection type in the *Mapfile*. Here is an example of a layer using this connection type:

```
LAYER
  NAME "prov_bound"
  TYPE RASTER
  STATUS ON
  CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?"
  CONNECTIONTYPE WMS
  METADATA
    "wms_srs"           "EPSG:42304"
    "wms_name"          "prov_bound"
    "wms_server_version" "1.1.1"
    "wms_format"        "image/gif"
  END
END
```

Required Layer Parameters and Metadata

- **CONNECTIONTYPE WMS**
- **CONNECTION** - this is the remote server's online resource URL, just the base URL without any of the WMS parameters. The server version, image format, layer name, etc. will be provided via metadata, see below.

Note: Note that if the CONNECTION parameter value is not set the the value of the “wms_onlineresource” metadata will be used. If both CONNECTION and “wms_onlineresource” are set then the “wms_onlineresource” metadata takes precedence.

- **“wms_format” metadata** - the image format to use in GetMap requests.

Note: If wms_formatlist is provided then wms_format is optional and MapServer will pick the first supported format in wms_formatlist for use in GetMap requests. If both wms_format and wms_formatlist are provided then wms_format takes precedence. Also note that WMS Servers only advertize supported formats that are part of the GD/GDAL libraries.

- **“wms_name” metadata** - comma-separated list of layers to be fetched from the remote WMS server. This value is used to set the LAYERS and QUERY_LAYERS WMS URL parameters.
- **“wms_server_version” metadata** - the version of the WMS protocol supported by the remote WMS server and that will be used for issuing GetMap requests.
- **“wms_srs” metadata** - space-delimited list of EPSG projection codes supported by the remote server. You normally get this from the server’s capabilities output. This value should be upper case (EPSG:4236.....not epsg:4236) to avoid problems with case sensitive platforms. The value is used to set the SRS WMS URL parameter.

Optional Layer Parameters and Metadata

- **MINSCALE, MAXSCALE** - if the remote server’s capabilities contains a ScaleHint value for this layer then you might want to set the MINSCALE and MAXSCALE in the LAYER object in the mapfile. This will allow MapServer to request the layer only at scales where it makes sense
- **PROJECTION object** - it is optional at this point. MapServer will create one internally if needed.
- **“wms_auth_username” metadata** - msEncrypt-style authorization string. Empty strings are also accepted.

METADATA

```
"wms_auth_username" "foo"  
"wms_auth_password" "{FF88CFDAAE1A5E33}"
```

END

- **“wms_auth_type” metadata** - Authorization type. Supported types include:
 - basic
 - digest
 - ntlm
 - any (the underlying http library picks the best among the options supported by the remote server)
 - anysafe (the underlying http library picks only safe methods among the options supported by the remote server)

METADATA

```
"wms_auth_type" "ntlm"
```

END

- **“wms_connectiontimeout” metadata** - the maximum time to wait for a remote WMS layer to load, set in seconds (default is 30 seconds). This metadata can be added at the layer level so that it affects only that layer, or it can be added at the map level (in the web object) so that it affects all of the layers. Note that wms_connectiontimeout at the layer level has priority over the map level.

METADATA

```
...
"wms_connectiontimeout" "60"
...
END
```

- **“wms_exceptions_format” metadata** - set the format for exceptions (as of MapServer 4.6). MapServer defaults to *application/vnd.ogc.se_inimage* (the exception will be in a picture format). You can check the GetCapabilities of the server to see what formats are available for exceptions. The *application/vnd.ogc.se_inimage* exception format is actually a non-required exception format in the WMS 1.1.1 spec, so there are servers out there which don't support this format. In that case you would use:

LAYER

```
...
METADATA
"wms_exceptions_format" "application/vnd.ogc.se_xml"
END
...
END
```

Which would return this xml exception in the MS_ERRORFILE:

```
Tue Jan 17 18:05:13 2006 - msDrawWMSLayerLow(): WMS server error. WMS GetMap
request got XML exception for layer 'prov_bound': <?xml version='1.0'
encoding="ISO-8859-1" standalone="no" ?><!DOCTYPE ServiceExceptionReport
SYSTEM "http://schemas.opengespatial.net/wms/1.1.1/exception_1_1_1.dtd">
<ServiceExceptionReport version="1.1.1"><ServiceException code="LayerNotDefined">
msWMSLoadGetMapParams(): WMS server error. Invalid layer(s) given in the LAYERS parameter.
</ServiceException>
</ServiceExceptionReport>
```

- **“wms_force_separate_request” metadata** - set this to “1” to force this WMS layer to be requested using its own separate GetMap request. By default MapServer will try to merge multiple adjacent WMS layers from the same server into a single multi-layer GetMap request to reduce the load on remote servers and improve response time. This metadata is used to bypass that behavior.
- **“wms_formatlist” metadata** - comma-separated list of image formats supported by the remote WMS server. Note that wms_formatlist is used only if wms_format is not set. If both wms_format and wms_formatlist are provided then wms_format takes precedence.
- **“wms_latlonboundingbox” metadata** - the bounding box of this layer in geographic coordinates in the format “lon_min lat_min lon_max lat_max”. If it is set then MapServer will request the layer only when the map view overlaps that bounding box. You normally get this from the server's capabilities output.

METADATA

```
"wms_latlonboundingbox" "-124 48 -123 49"
END
```

- **“wms_proxy_auth_type” metadata** - the authorization type to use for a proxy connection. Supported types include:
 - basic
 - digest
 - ntlm
 - any (the underlying http library picks the best among the options supported by the remote server)

- anysafe (the underlying http library picks only safe methods among the options supported by the remote server)

METADATA

```
"wms_proxy_auth_type" "ntlm"  
END
```

- **“wms_proxy_host” metadata** - the hostname of the proxy to use, in “dot-quad” format, with an optional port component (e.g. ‘192.168.2.10:8080’).

METADATA

```
"wms_proxy_host" "192.168.2.10"  
END
```

- **“wms_proxy_port” metadata** - the port to use for a proxy connection.

METADATA

```
"wms_proxy_port" "8080"  
END
```

- **“wms_proxy_type” metadata** - the type of the proxy connection. Valid values are ‘http’ and ‘socks5’, which are case sensitive.

METADATA

```
"wms_proxy_type" "http"  
END
```

- **“wms_proxy_username” metadata** - msEncrypt-style string for a proxy connection. Empty strings are also accepted.

METADATA

```
"wms_proxy_username" "foo"  
"wms_proxy_password" "{FF88CFDAAE1A5E33}"  
END
```

- **“wms_sld_body” metadata** - can be used to specify an inline SLD document.
- **“wms_sld_url” metadata** - can be used to specify a link to an SLD document.
- **“wms_style” metadata** - name of style to use for the STYLES parameter in GetMap requests for this layer.
- **“wms_style_<stylename>_sld” metadata** URL of a SLD to use in GetMap requests. Replace <stylename> in the metadata name with the name of the style to which the SLD applies.

METADATA

```
...  
"wms_style" "mystyle"  
"wms_style_mystyle_sld" "http://my.host.com/mysld.xml"  
...  
END
```

For more information on SLDs in MapServer see the *SLD HowTo document*.

- **“wms_time” metadata** - value to use for the TIME parameter in GetMap requests for this layer. Please see the *WMS Time HowTo* for more information.
- **“wms_bgcolor” metadata** - specifies the color to be used as the background of the map. The general format of BGCOLOR is a hexadecimal encoding of an RGB value where two hexadecimal characters are used for each of Red, Green, and Blue color values. The values can range between 00 and FF for each (0 and 255, base 10). The

format is 0xRRGGBB; either upper or lower case characters are allowed for RR, GG, and BB values. The “0x” prefix shall have a lower case “x”.

- **“wms_transparent” metadata** - specifies whether the map background is to be made transparent or not. TRANSPARENT can take on two values, “TRUE” or “FALSE”. If not specified, MapServer sets default to “TRUE”

Note: Note that each of the above metadata can also be referred to as ‘ows_’ instead of ‘wms_’. MapServer tries the ‘wms_’ metadata first, and if not found it tries the corresponding ‘ows_’ name. Using this reduces the amount of duplication in mapfiles that support multiple OGC interfaces since “ows_” metadata can be used almost everywhere for common metadata items shared by multiple OGC interfaces.

Old CONNECTION parameter format from version 3.5 and 3.6 (deprecated)

In MapServer version 3.5 and 3.6, the CONNECTION parameter had to include at a minimum the VERSION, LAYERS, FORMAT and TRANSPARENT WMS parameters. This mode of operation is still supported but is deprecated and you are encouraged to use metadata items for those parameters as documented in the previous section above.

Here is an example of a layer definition using this deprecated CONNECTION parameter format:

```
LAYER
NAME "bathymetry"
METADATA
  "wms_title" "Elevation/Bathymetry"
  "wms_srs" "EPSG:4269 EPSG:4326"
END
TYPE RASTER
STATUS ON
CONNECTIONTYPE WMS
CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?VERSION=1.1.0&LAYERS=bathymetry&FORMAT=im
PROJECTION
  "init=epsg:4236"
END
END
```

9.2.4 Limitations/TODO

1. GetFeatureInfo is not fully supported since the output of getFeatureInfo is left to the discretion of the remote server. A method layer.getWMSFeatureInfoURL() has been added to MapScript for applications that want to access featureInfo results and handle them directly.
2. MapServer does not attempt to fetch the layer’s capabilities. Doing so at every map draw would be extremely inefficient. And caching that information does not belong in the core of MapServer. This is better done at the application level, in a script, and only the necessary information is passed to the MapServer core via the CONNECTION string and metadata.

9.3 WMS Time

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8295 \$

Date \$Date: 2008-12-26 21:08:04 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2006/06/26

Table of Contents

- WMS Time
 - Introduction
 - Enabling Time Support in MapServer
 - Future Additions
 - Limitations and Known Bugs

9.3.1 Introduction

A WMS server can provide support to temporal requests. This is done by providing a TIME parameter with a time value in the request. MapServer 4.4 and above provides support to interpret the TIME parameter and transform the resulting values into appropriate requests.

Links to WMS-Related Information

- [MapServer WMS Server HowTo](#)
- [MapServer WMS Client HowTo](#)
- [WMS 1.1.1 specification](#)
- [MapServer OGC Web Services Workshop](#)

9.3.2 Enabling Time Support in MapServer

Time Patterns

WMS specifies that the basic format used for TIME requests is based on the ISO 8601:1988(E) “extended” format. MapServer supports a limited set of patterns that are defined in the ISO 8601 specifications, as well as few other patterns that are useful but not compliant to ISO 8601. Here is a list of patterns currently supported:

Table 1. Supported Time Patterns

Time Patterns	Examples
YYYYMMDD	20041012
YYYY-MM-DDTHH:MM:SSZ	2004-10-12T13:55:20Z
YYYY-MM-DDTHH:MM:SS	2004-10-12T13:55:20
YYYY-MM-DD HH:MM:SS	2004-10-12 13:55:20
YYYY-MM-DDTHH:MM	2004-10-12T13:55
YYYY-MM-DD HH:MM	2004-10-12 13:55
YYYY-MM-DDTHH	2004-10-12T13
YYYY-MM-DD HH	2004-10-12 13
YYYY-MM-DD	2004-10-12
YYYY-MM	2004-10
YYYY	2004
THH:MM:SSZ	T13:55:20Z
THH:MM:SS	T13:55:20

Setting Up a WMS Layer with Time Support

To have a valid WMS layer with time support, the user has to define the following metadata at the layer level:

- *wms_timeextent*: (Mandatory) this is used in the capabilities document to return the valid time values for the layer. The value defined here should be a valid time range. (more on this in ‘Specifying Time Extents’ below)
- *wms_timeitem*: (Mandatory) this is the name of the field in the DB that contains the time values.
- *wms_timedefault*: (Optional) this value is used if it is defined and the TIME value is missing in the request.

Specifying Time Extents

Time Extents can be declared with the following syntax for the *wms_timeextent* metadata (see Annex C.3 in the [WMS 1.1.1 specification](#) document for a full description):

1. *value* - a single value. This is not directly supported in MapServer but there is an easy workwound by specifying the same value as min and max.
2. *value1,value2,value3,...* - a list of multiple values.
3. *min/max/resolution* - an interval defined by its lower and upper bounds and its resolution. This is supported in MapServer (note that the resolution is not supported however).
4. *min1/max1/res1,min2/max2/res2,...* - a list of multiple intervals.

Example WMS-Server Layer

```
LAYER
  NAME "earthquakes"
  METADATA
    "wms_title"      "Earthquakes"
    "wms_timeextent" "2004-01-01/2004-02-01"
    "wms_timeitem"  "TIME"
    "wms_timedefault" "2004-01-01 14:10:00"
  END
  TYPE POINT
  STATUS ON
  DATA "quakes"
  CLASS
    ..
  END
END
```

GetCapabilities Output

If your layer is set up properly, requesting the capabilities on the server outputs a Dimension element. Here is an example of a GetCapabilities result for a layer configured for time support:

```
<Layer queryable="0" opaque="0" cascaded="0">
  <Name>earthquakes</Name>
  <Title>Earthquakes</Title>
  <SRS>EPSG:4326</SRS>
  <LatLonBoundingBox minx="-131.02" miny="24.84" maxx="-66.59" maxy="48.39" />
  <BoundingBox SRS="EPSG:4326"
```

```
        minx="-131.02" miny="24.84" maxx="-66.59" maxy="48.39" />
<Dimension name="time" units="ISO8601"/>
<Extent name="time" default="2004-01-01 14:10:00" nearestValue="0">2004-01-01/2004-02-01</Extent>
</Layer>
```

Supported Time Requests

When sending a request with the TIME parameter, different types of time values can be specified. The following are supported by MapServer:

- *single value*: for example: ...&TIME=2004-10-12&...
- *multiple values*: for example: ...&TIME=2004-10-12, 2004-10-13, 2004-10-14&...
- *single range value*: for example: ...&TIME=2004-10-12/2004-10-13&...
- *multiple range values*: for example: ...&TIME=2004-10-12/2004-10-13, 2004-10-15/2004-10-16&...

Interpreting Time Values

When MapServer receives a request with a TIME parameter, it transforms the time requests into valid expressions that are assigned to the filter parameter on layers that are time-aware. Here are some examples of how different types of requests are treated (wms_timeitem is defined here as being "time_field"):

- single value (2004-10-12) *transforms to* ('[time_field]' eq '2004-10-12')
- multiple values (2004-10-12, 2004-10-13) *transform to* ('[time_field]' eq '2004-10-12' OR '[time_field]' eq '2004-10-13')
- single range : 2004-10-12/2004-10-13 *transforms to* (('[time_field]' ge '2004-10-12') AND ('[time_field]' le '2004-10-13'))
- multiple ranges (2004-10-12/2004-10-13, 2004-10-15/2004-10-16) *transform to* (('[time_field]' ge '2004-10-12' AND '[time_field]' le '2004-10-13') OR ('[time_field]' ge '2004-10-15' AND '[time_field]' le '2004-10-16'))

As shown in the above examples, all fields and values are written inside back tics (') - this is the general way of specifying time expressions inside MapServer.

Exceptions to this rule:

1. When dealing with layers that are not Shapefiles nor through OGR, the expression built has slightly different syntax. For example, the expression set in the filter for the first example above would be ([time_field] = '2004-10-12').
2. For *PostGIS/PostgreSQL* layers, the time expression built uses the *date_trunc* function available in PostgreSQL. For example, if the user passes a time value of '2004-10-12', the expression set in the filter is *date_trunc('day', time_field) = '2004-10-12'*. The use of the *date_trunc* function allows requests to use the concept of time resolution. In the example above, for a request of '2004-10-12', MapServer determines that the resolution is "day" by parsing the time string and the result gives all records matching the date 2004-10-12 regardless of the values set for Hours/Minutes/Seconds in the database. For more information on the *date_trunc* function, please refer to the [PostgreSQL documentation](#).

Limiting the Time Formats to Use

The user has the ability to define the time format(s) to be used when a request is sent, in metadata at the WEB level. For example, the user can define the following two formats:

```
"wms_timeformat" "YYYY-MM-DDTHH, YYYY-MM-DDTHH:MM"
```

Another example is for a WMS layer that is based on time data that contains precise time values taken every minute (e.g., 2004-10-12T13:55, 2004-10-12T13:56, 2004-10-12 T13:57, ...). Normally, a valid request on such a layer would require the time value to be as complete as the data underneath. By defining a set of patterns to use, MapServer introduces the notion of resolution to be used when doing a query. Using the example above, a request TIME= 2004-10-12T13:55 would be valid and a request TIME= 2004-10-12T13 would also be valid and would return all elements taken for that hour.

Note that this functionality is only available on layers based on Shapefiles and OGR.

Example of WMS-T with PostGIS Tile Index for Raster Imagery

This example currently requires latest 4.9 CVS build!

Here is an example mapfile snippet for a raster WMS-T instance using a PostGIS tileindex. This example shows US Nexrad Base Reflectivity running at Iowa State U at <http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi?SERVICE=WMS&request=GetCapabilities>

```

1 # Tile Index
2 LAYER
3   STATUS ON
4   NAME "time_idx"
5   TYPE POLYGON
6   DATA "the_geom from nexrad_n0r_tindex"
7   METADATA
8     "wms_title" "TIME INDEX"
9     "wms_srs" "EPSG:4326"
10    "wms_extent" "-126 24 -66 50"
11    "wms_timeextent" "2003-08-01/2006-12-31/P5M"
12    "wms_timeitem" "datetime" #column in postgis table of type timestamp
13    "wms_timedefault" "2006-06-23T03:10:00Z"
14  END
15  CONNECTION "dbname=postgis host=10.10.10.20"
16  CONNECTIONTYPE postgis
17 END
18
19 # raster layer
20 LAYER
21   NAME "nexrad-n0r-wmst"
22   TYPE RASTER
23   STATUS ON
24   DEBUG ON
25   DUMP TRUE
26   PROJECTION
27     "init=epsg:4326"
28  END
29  METADATA
30    "wms_title" "NEXRAD BASE REF WMS-T"
31    "wms_srs" "EPSG:4326"
32    "wms_extent" "-126 24 -66 50"
33    "wms_timeextent" "2003-08-01/2006-12-31/P5M"
34    "wms_timeitem" "datetime" #datetime is a column in postgis table of type timestamp
35    "wms_timedefault" "2006-06-23T03:10:00Z"
36  END
37  OFFSITE 0 0 0

```

```
38 TILEITEM "filepath" #filepath is a column in postgis table with varchar of the filepath to each image
39 TILEINDEX "time_idx"
40 END
```

You can find more information on Time and tileindexes in the *WCS documentation*.

9.3.3 Future Additions

- Support for a special time value: “current”.

9.3.4 Limitations and Known Bugs

- Pattern “YYYYMMDD” does not work on Windows. (Bug#970)

9.4 Map Context

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8493 \$

Date \$Date: 2009-02-03 15:22:33 -0800 (Tue, 03 Feb 2009) \$

Last Updated 2008/03/20

Contents

- Map Context
 - Introduction
 - Implementing a Web Map Context

9.4.1 Introduction

The term ‘map context’ comes from the Open Geospatial Consortium’s (OGC) [Web Map Context Specification v1.0.0](#), which coincides with the OGC [Web Map Server Specification \(WMS\) v1.1.1](#). A map context is a XML document that describes the appearance of layers from one or more WMS servers, and can be transferred between clients while maintaining startup views, the state of the view (and its layers), and storing additional layer information.

Support for OGC Web Map Context was added to MapServer in version 3.7/4.0. This allows client applications to load and save a map configuration in a standard XML format. MapServer can read context documents of versions 0.1.2, 0.1.4, 0.1.7, 1.0.0, 1.1.0 and can export contents in versions 0.1.4, 0.1.7, 1.0.0, 1.1.0. Web Map Context 1.1.0 support was added to MapServer 4.10

This document assumes that you are already familiar with certain aspects of MapServer:

- MapServer application development and setting up *mapfiles*.
- Familiarity with the WMS spec would be an asset. Please see the following section for links to associated sources.

Links to WMS / Map Context Related Information

- [MapServer WMS Client HowTo](#)
- [Open Geospatial Consortium \(OGC\) home page](#)
- [WMS 1.1.1 specification](#)
- [Map Context 1.0.0 specification](#)
- [MapServer OGC Web Services Workshop](#)

9.4.2 Implementing a Web Map Context

Special Build Considerations

Map Context support requires PROJ4, GDAL/OGR and PHP support libraries.

Build/install the above libraries on your system and then build MapServer with the `'-with-wmsclient -with-proj -with-ogr -with-gdal -with-php'` configure options. Also make sure that your build uses the `USE_WMS_LYR` and `USE_OGR` flags. For more details on MapServer compilation see the appropriate HowTo: [Unix / Windows](#)

Windows users can use [MS4W](#), which is ready for Map Context use.

Map Context Mapfile

A map context document can ONLY contain WMS layers (e.g. CONNECTIONTYPE WMS). Please refer to the [MapServer WMS Client HowTo](#) for more information on declaring WMS layers.

MapFile Metadata

The following mapfile metadata are used by MapServer to handle map context information:

(Note that some parameters have width, height, format, and href, and some only have format and href. This is because width and height are only used for images and parameters that do not have them are text or html. For consistency with the spec MapServer supports height and width for all parameters, but they should only be used for images)

Web Object Metadata

- `ows_schemas_location` : Location of XML schema document. Default is <http://schemas.opengespatial.net>. See <http://ogc.dmsolutions.ca> for an example of a valid schema tree.
- `wms_abstract` : A blurb of text providing more information about the WMS server.
- `wms_address` : If provided must also then provide `wms_addresstype`, `wms_city`, `wms_stateorprovince`, `wms_postcode`, and `wms_country`)
- `wms_addresstype` : If provided must also then provide `wms_address`, `wms_city`, `wms_stateorprovince`, `wms_postcode`, and `wms_country`)
- `wms_city` : If provided must also then provide `wms_address`, `wms_addresstype`, `wms_stateorprovince`, `wms_postcode`, and `wms_country`)
- `wms_contactelectronicmailaddress` : contact Email address.
- `wms_contactfacsimiletelephone` : contact facsimile telephone number.

- *wms_contactorganization* :
- *wms_contactperson* :
- *wms_contactposition* :
- *wms_contactvoicetelephone* : contact voice telephone number.
- *wms_context_fid* : the feature id of the context. Set to 0 when saving if not specified.
- *wms_context_version* : the version of the map context specification.
- *wms_country* : If provided must also then provide *wms_address*, *wms_city*, *wms_stateorprovince*, *wms_postcode*, and *wms_addresstype*.
- *wms_descriptionurl_format* : Format of the webpage which contains relevant information to the view.
- *wms_descriptionurl_href* : Reference to a webpage which contains relevant information to the view.
- *wms_keywordlist* : A comma-separated list of keywords or keyword phrases to help catalog searching.
- *wms_logourl_width* : Width of the context logo.
- *wms_logourl_height* : Height of the context logo.
- *wms_logourl_format* : Format of the context logo.
- *wms_logourl_href* : Location of the context logo.
- *wms_postcode* : If provided must also then provide *wms_address*, *wms_city*, *wms_stateorprovince*, *wms_addresstype*, and *wms_country*.
- *wms_stateorprovince* : If provided must also then provide *wms_address*, *wms_city*, *wms_addresstype*, *wms_postcode*, and *wms_country*.
- *wms_title* : **(Required)** A human-readable name for this Layer (this metadata does not exist beyond version 0.1.4)

Layer Object Metadata

- *wms_abstract* : A blurb of text providing more information about the WMS server.
- *wms_dataurl_href* : Link to an online resource where data corresponding to the layer can be found.
- *wms_dataurl_format* : Format of the online resource where data corresponding to the layer can be found.
- *wms_dimension* : Current dimension used. New in version 4.10.
- *wms_dimensionlist* : List of available dimensions. New in version 4.10.
- *wms_dimension_%s_default* : Default dimension value. MapServer will check for *wms_time* and *wms_timedefault* metadata when this is not specified. %s = the name of the dimension. New in version 4.10.
- *wms_dimension_%s_multiplevalues* : Multiple dimension values. %s = the name of the dimension. New in version 4.10.
- *wms_dimension_%s_nearestvalue* : Nearest dimension value. The default value is 0. %s = the name of the dimension. New in version 4.10.
- *wms_dimension_%s_units* : Units for the dimension values. The default value is ISO8601. %s = the name of the dimension. New in version 4.10.
- *wms_dimension_%s_unitsymbol* : Symbol for dimension units. The default value is t. %s = the name of the dimension. New in version 4.10.

- *wms_dimension_%s_uservalue* : User dimension value. MapServer will check for *wms_time* and *wms_timedefault* metadata when this is not specified. %s = the name of the dimension. New in version 4.10.
- *wms_format* : Current format used.
- *wms_formatlist* : List of available formats for this layer.
- *wms_metadataurl_href* : Link to an online resource where descriptive metadata of the corresponding layer can be found.
- *wms_metadataurl_format* : Format of the online resource where descriptive metadata of the corresponding layer can be found.
- *wms_name* : Name of the WMS layer on the server.
- *wms_onlineresource* : **Required** URL to access the server.
- *wms_server_version* : The version of the web map server specification.
- *wms_server_title* : The title of the web map server.
- *wms_stylelist* : Current style used.
- *wms_style_%s_legendurl_width* : Width of an image describing the style. %s = the name of the style.
- *wms_style_%s_legendurl_height* : Height of an image describing the style. %s = the name of the style.
- *wms_style_%s_legendurl_format* : Format of an image describing the style. %s = the name of the style.
- *wms_style_%s_legendurl_href* : Location of an image describing the style. %s = the name of the style.
- *wms_style_%s_sld* : URL to the SLD document of this style. %s = the name of the style.
- *wms_style_%s_sld_body* : SLD_BODY document of this style. %s = the name of the style.
- *wms_style_%s_title* : Title of the layer. %s = the name of the style.
- *wms_title* : **(Required)** A human-readable name for this Layer.

Sample Map Context Mapfile

```

1  NAME W M S _ C O N T E X T
2  STATUS ON
3  SIZE 400 300
4  SYMBOLSET ../etc/symbols.sym
5  EXTENT -2200000 -712631 3072800 3840000
6  UNITS METERS
7  SHAPEPATH "../data"
8  IMAGECOLOR 255 255 255
9  FONTSET ../etc/fonts.txt
10
11 WEB
12  IMAGEPATH "/ms4w/tmp/ms_tmp/"
13  IMAGEURL "/ms_tmp/"
14  METADATA
15    "wms_abstract" "Demo for map context document. Blah blah..."
16    "wms_title" "Map Context demo" #### REQUIRED
17  END
18 END
19
20 PROJECTION
21  "init=epsg:42304"

```

```
22 END
23
24 LAYER
25     NAME "prov_bound"
26     TYPE RASTER
27     STATUS ON
28     CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?"
29     CONNECTIONTYPE WMS
30     METADATA
31         "wms_name" "prov_bound"
32         "wms_server_version" "1.1.1"
33         "wms_server_title" "GMap WMS Demo Server"
34         "wms_format" "image/gif"
35         "wms_srs" "EPSG:42304"
36         "wms_title" "Canadian boundaries" ##### REQUIRED
37         "wms_onlineresource" "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?" ##### REQUIRED
38         "wms_dimensionlist" "time,width"
39         "wms_dimension" "time"
40         "wms_dimension_time_unitsymbol" "t"
41         "wms_dimension_time_units" "ISO8601"
42         "wms_dimension_time_uservalue" "1310"
43         "wms_dimension_time_default" "1310"
44         "wms_dimension_time_multiplevalues" "1310,1410"
45         "wms_dimension_time_nearestvalue" "0"
46     END
47 END
48
49 LAYER
50     NAME p o p p l a c e
51     TYPE RASTER
52     STATUS ON
53     CONNECTIONTYPE WMS
54     CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?"
55     METADATA
56         "wms_name" "popplace"
57         "wms_server_version" "1.1.1"
58         "wms_server_title" "GMap WMS Demo Server"
59         "wms_format" "image/png"
60         "wms_srs" "EPSG:42304"
61         "wms_title" "Canadian Cities" ##### REQUIRED
62         "wms_onlineresource" "http://www2.dmsolutions.ca/cgi-bin/mswms_gmap?" ##### REQUIRED
63     END
64 END # Layer
65
66 END # Map File
```

Testing Map Context Support

1. The first thing to do is to save your mapfile using the `saveMapContext` function available from the *PHP/MapScript* library. An example script is shown below:

```
<?php
    dl("php_mapscript.dll");
    $oMap = ms_newMapObj("gmap_wms_context.map");
    $oMap->saveMapContext("gmap_wms_context_output.xml");
?>
```

2. Scan the XML output to look for `<!-- WARNING: ... -->` comments. Then make the necessary changes to fix every warning that you encounter. At the end of this you should have a mapfile compatible with the Map Context specification.
3. Now you can load your new Map Context document into an application using the `loadMapContext` function from the PHP/MapScript library.

Sample Map Context Document

The following is a sample Map Context document:

```

1  <?xml version='1.0' encoding="ISO-8859-1" standalone="no" ?>
2  <ViewContext version="1.1.0" id="WMS_CONTEXT" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  <General>
4  <Window width="400" height="300"/>
5  <!-- Bounding box corners and spatial reference system -->
6  <BoundingBox SRS="EPSG:42304" minx="-2200000.000000" miny="-712631.000000" maxx="3072800.000000"
7  <!-- Title of Context -->
8  <Title>Map Context demo</Title>
9  <Abstract>Demo for map context document. Blah blah...</Abstract>
10 <ContactInformation>
11 </ContactInformation>
12 </General>
13 <LayerList>
14 <Layer queryable="0" hidden="0">
15 <Server service="OGC:WMS" version="1.1.1" title="Canadian boundaries">
16 <OnlineResource xlink:type="simple" xlink:href="http://www2.dmsolutions.ca/cgi-bin/mswms.cgi?mapfile=canadian_boundaries.mxd"
17 </Server>
18 <Name>prov_bound</Name>
19 <Title>Canadian boundaries</Title>
20 <SRS>EPSG:42304</SRS>
21 <FormatList>
22 <Format current="1">image/gif</Format>
23 </FormatList>
24 <DimensionList>
25 <Dimension name="time" units="ISO8601" unitSymbol="t" userValue="1310" default="1310" multiplier="1"
26 </DimensionList>
27 </Layer>
28 <Layer queryable="0" hidden="0">
29 <Server service="OGC:WMS" version="1.1.1" title="Canadian Cities">
30 <OnlineResource xlink:type="simple" xlink:href="http://www2.dmsolutions.ca/cgi-bin/mswms.cgi?mapfile=canadian_cities.mxd"
31 </Server>
32 <Name>popplace</Name>
33 <Title>Canadian Cities</Title>
34 <SRS>EPSG:42304</SRS>
35 <FormatList>
36 <Format current="1">image/png</Format>
37 </FormatList>
38 </DimensionList>
39 </Layer>
40 </LayerList>
41 </ViewContext>

```

Map Context Support Through CGI

MapServer CGI allows you to load a map context through the use of a `CONTEXT` parameter, and you can point this parameter to a locally stored context file or a context file accessible through a URL. For more information on MapServer CGI see the *CGI Reference*.

Support for Local Map Context Files

There is a new cgi parameter called `CONTEXT` that is used to specify a local context file. The user can then use MapServer to request a map using the following syntax:

```
http://localhost/mapserver.cgi?MODE=map&MAP=/path/to/mapfile.map&CONTEXT=
/path/to/contextfile.xml&LAYERS=layer_name1 layer_name2
```

Note: All layers created from a context file have their status set to ON. To be able to display layers, the user needs to add the `LAYERS` argument in the URL.

Support for Context Files Accessed Through a URL

The syntax of using a web accessible context file would be similar to accessing a local context file:

```
http://localhost/mapserver.cgi?MODE=map&MAP=/path/to/mapfile.map&CONTEXT=
http://URL/path/to/contextfile.xml&LAYERS=layer_name1 layer_name2
```

Due to security concerns loading a file from a URL is disabled by default. To enable this functionality, the user needs to set a `CONFIG` parameter called `CGI_CONTEXT_URL` in the default mapfile that will allow this functionality. Here is an example of a map file with the `CONFIG` parameter:

```
# Start of map file
NAME "map-context"
STATUS ON
SIZE 400 300
EXTENT -2200000 -712631 3072800 3840000
UNITS METERS
IMAGECOLOR 255 255 255
IMAGETYPE png
CONFIG "CGI_CONTEXT_URL" "1"
...
WEB
...
END
LAYER
...
END
END
```

Default Mapfile

To smoothly run a MapServer CGI application with a Map Context, the application administrator needs to provide a default mapfile with at least the basic required parameters that will be used with the Context file. This default mapfile

can contain as little information as the `imagepath` and `imageurl` or contain a list of layers. Information coming from the context (e.g.: layers, width, height, ...) would either be appended or will replace values found in the mapfile.

Here is an example of a default map file containing the minimum required parameters:

```

1  NAME "CGI-CONTEXT-DEMO"
2  STATUS ON
3  SIZE 400 300
4  EXTENT -2200000 -712631 3072800 3840000
5  UNITS METERS
6  IMAGECOLOR 255 255 255
7  IMAGETYPE png
8  #
9  # Start of web interface definition
10 #
11 WEB
12   MINSCALE 2000000
13   MAXSCALE 50000000
14 #
15 # On Windows systems, /tmp and /tmp/ms_tmp/ should be created at the root
16 # of the drive where the .MAP file resides.
17 #
18   IMAGEPATH "/ms4w/tmp/ms_tmp/"
19   IMAGEURL "/ms_tmp/"
20 END
21 END # Map File

```

Map Context Support Through WMS

MapServer can also output your WMS layers as a Context document. MapServer extends the WMS standard by adding a `request=GetContext` operation that allows you to retrieve a context for a WMS-based mapfile with a call like:

```
http://localhost/mapserver.cgi?map=/path/to/mapfile.map&service=WMS&
request=GetContext&version=1.1.0
```

The `VERSION` parameter controls the version of context document to return.

`GetContext` is disabled by default because it could be considered a security issue: it could publicly expose the URLs of WMS layers used (cascaded) by a mapfile.

To enable it, set the `"wms_getcontext_enabled"` web metadata to `"1"` in your WMS server's mapfile.

9.5 WFS Server

Author Jean-François Doyon

Contact jdoyon at nrcan.gc.ca

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8376 \$

Date \$Date: 2009-01-02 07:13:17 -0800 (Fri, 02 Jan 2009) \$

Last Updated 2006/10/13

Contents

- WFS Server
 - Introduction
 - Configuring your MapFile to Serve WFS layers
 - Reference Section
 - To-do Items and Known Limitations

9.5.1 Introduction

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 OGC 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.

WFS-Related Information

Here are some WFS related links (including a newly added OGC services workshop with MapServer). Since these are highly detailed technical specifications, there is no need to read through them in their entirety to get a MapServer WFS up and running. It is still recommended however to read them over and get familiar with the basics of each of them, in order to understand how it all works:

- [The OGC Web Feature Service Implementation Specification.](#)
- [The Geography Markup Language Implementation Specification.](#)
- [MapServer OGC Web Services Workshop package.](#)

Working knowledge of MapServer is of course also required.

Software Requirements

In order to enable MapServer to serve WFS, it MUST be compiled against certain libraries:

- PROJ.4: The reprojection library. Version 4.4.3 or greater is required.
- GDAL/OGR: I/O support libraries. Version 1.1.8 or greater is required.

Please see the MapServer *UNIX Compilation and Installation HowTo* for detailed instructions on compiling mapserver with support for these libraries and features. For Windows users, the [MS4W](#) installer comes ready to serve both WFS and WMS.

Version of GML Supported

MapServer can output both GML2 and GML3. By default MapServer serves GML2. You can test this by adding an 'OUTPUTFORMAT' parameter to a GetFeature request, such as:

- [GML2 request output](#)
- [GML3 request output](#)

For a detailed discussion on the versions supported, see [bug#884](#).

9.5.2 Configuring your MapFile to Serve WFS layers

Much as in the WMS support, WFS publishing is enabled by adding certain magic METADATA keyword/value pairs to a MapFile.

MapServer will serve and include in its WFS capabilities only the layers that meet the following conditions:

- Data source is of vector type (Shapefile, OGR, PostGIS, SDE, SDO, ...)
- LAYER NAME must be set. Layer names must start with a letter when setting up a WFS server (layer names should not start with a digit or have spaces in them).
- LAYER TYPE is one of: LINE, POINT, POLYGON
- LAYER DUMP parameter set to TRUE
- The “wfs_onlineresource” metadata:

The wfs_onlineresource metadata is set in the map’s web object metadata and specifies the URL that should be used to access your server. This is required for the GetCapabilities output. If wfs_onlineresource is not provided then MapServer will try to provide a default one using the script name and hostname, but you shouldn’t count on that too much. It is strongly recommended that you provide the wfs_onlineresource metadata.

See section 12.3.3 of the [WFS 1.0.0 specification](#) for the whole story about the online resource URL. Basically, what you need is a complete HTTP URL including the `http://` prefix, hostname, script name, potentially a “map=” parameter, and terminated by “?” or “&”.

Here is a valid online resource URL:

```
http://my.host.com/cgi-bin/mapserv?map=mywfs.map&
```

By creating a wrapper script on the server it is possible to hide the “map=” parameter from the URL and then your server’s online resource URL could be something like:

```
http://my.host.com/cgi-bin/mywfs?
```

This is covered in more detail in the “More About the Online Resource URL” section of the *WMS Server* document.

Example WFS Server Mapfile

The following is an example of a bare minimum WFS Server mapfile. Note the comments for the required parameters.

```
NAME "WFS_server"
STATUS ON
SIZE 400 300
SYMBOLSET ../etc/symbols.sym
EXTENT -2200000 -712631 3072800 3840000
UNITS METERS
SHAPEPATH "../data"
IMAGECOLOR 255 255 255
FONTSET ../etc/fonts.txt

WEB
  IMAGEPATH "/ms4w/tmp/ms_tmp/"
  IMAGEURL "/ms_tmp/"
  METADATA
    "wfs_title"          "GMap WFS Demo Server"  ## REQUIRED
    "wfs_onlineresource" "http://127.0.0.1/cgi-bin/mapserv.exe?"  ## Recommended
```

```
"wfs_srs"          "EPSG:42304 EPSG:42101 EPSG:4269 EPSG:4326"  ## Recommended
"ows_schemas_location" "http://ogc.dmsolutions.ca"  ## Optional
END
END

PROJECTION
  "init=epsg:42304"
END

LAYER
  NAME "province"
  METADATA
    "wfs_title"      "Provinces"  ## REQUIRED
    "gml_featureid" "ID"  ## REQUIRED
    "gml_include_items" "all"  ## Optional (serves all attributes for layer)
  END
  TYPE POLYGON
  STATUS ON
  DATA p r o v i n c e
  PROJECTION
    "init=epsg:42304"
  END
  DUMP TRUE          ## REQUIRED
  CLASS
    NAME "Canada"
    STYLE
      COLOR 200 255 0
      OUTLINECOLOR 120 120 120
    END
    TEMPLATE "ttt_query.html"
  END
END # Layer

END # Map File
```

Rules for Handling SRS in MapServer WFS

Contrary to WMS, the OGC WFS specification doesn't allow a layer (feature type) to be advertised in more than one SRS. Also, there is no default SRS that applies to all layers by default in the OGC WFS spec. However, it is possible to have every layer in a WFS server advertised in a different SRS.

Here is how MapServer decides the SRS to advertise and use for each layer in your WFS:

- If a top-level map SRS is defined* then this SRS is used and applies to all layers (feature types) in this WFS. In this case the SRS of individual layers is simply ignored even if it's set.
- If there is no top-level map SRS defined* then each layer is advertised in its own SRS in the capabilities.
- By "SRS is defined", we mean either the presence of a PROJECTION object defined using an EPSG code, or of a "wfs_srs" metadata at this level.

Note that at the map top-level the "wfs_srs" metadata value takes precedence over the contents of the *PROJECTION* block.

At the layer level, if both the wfs_srs metadata and the PROJECTION object are set to different values, then the wfs_srs metadata defines the projection to use in advertising this layer (assuming there is no top-level map SRS), and the PROJECTION value is assumed to be the projection of the data. So this means that the data would be reprojected from the PROJECTION SRS to the one defined in the wfs_srs metadata before being served to WFS clients.

Confusing? As a rule of thumb, simply set the `wfs_srs` at the map level (in web metadata) and never set the `wfs_srs` metadata at the layer level and things will work fine for most cases.

Test Your WFS Server

Validate the Capabilities Metadata

OK, now that we've got a mapfile, we have to check the XML capabilities returned by our server to make sure nothing is missing.

Using a web browser, access your server's online resource URL to which you add the parameter "REQUEST=GetCapabilities" to the end, e.g.

http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?SERVICE=WFS&VERSION=1.0.0&REQUEST=getcapabilities

If everything went well, you should have a complete XML capabilities document. Search it for the word "WARNING"... MapServer inserts XML comments starting with "<!--WARNING: " in the XML output if it detects missing mapfile parameters or metadata items. If you notice any warning in your XML output then you have to fix all of them before you can register your server with a WFS client, otherwise things are likely not going to work.

Note that the `SERVICE` parameter is required for all WFS requests. When a request happens, it is passed through WMS, WFS, and WCS in MapServer (in that order) until one of the services respond to it.

Test With a GetFeature Request

OK, now that we know that our server can produce a valid XML GetCapabilities response we should test the GetFeature request. Simply adding "SERVICE=WFS&VERSION=1.0.0&REQUEST=GetFeature&TYPENAME=yourlayername1,yourlayername2" to your server's URL should return the GML associated with those layers.

http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?SERVICE=WFS&VERSION=1.0.0&REQUEST=getfeature&TYPENAME=park

Test with a Real Client

If you have access to a WFS client, then register your new server's online resource with it and you should be off and running.

If you don't have your own WFS client installed already, here are a few pointers:

- MapServer itself can be used as a WFS client, see the *WFS Client HowTo*.
- [Deegree](#) provides a WFS client.
- [uDig](#) can add layers from WMS/WFS servers.
- The [owsview](#) Viewer Client Generator is an online application that allows users to validate WFS Capabilities XML (it does not allow you to view WFS data).

Support for GET and POST Requests

Starting from version 4.2 MapServer supports XML-encoded POST requests and GET requests. The default in MapServer is POST.

Support for Filter Encoding

Starting from version 4.2 MapServer supports Filter Encoding (FE) in WFS GetFeature requests. For more information on the server side of Filter Encoding see the *Filter Encoding HowTo*.

9.5.3 Reference Section

The following metadata are available in the setup of the WFS Server mapfile:

Note

Each of the metadata below can also be referred to as ‘ows_*

Web Object Metadata

ows_updatesequence

- *Description:* (Optional) The updateSequence parameter can be used for maintaining the consistency of a client cache of the contents of a service metadata document. The parameter value can be an integer, a timestamp in [ISO 8601:2000] format, or any other number or string.

ows_schemas_location

- *Description:* (Optional) (Note the name ows_schemas_location and not wfs/_... this is because all OGC Web Services (OWS) use the same metadata) Root of the web tree where the family of OGC WFS XMLSchema files are located. This must be a valid URL where the actual .xsd files are located if you want your WFS output to validate in a validating XML parser. Default is <http://schemas.opengespatial.net>. See <http://ogc.dmsolutions.ca> for an example of a valid schema tree.

wfs_abstract

- *WFS TAG Name:* Abstract (WFS 1.0.0, sect. 12.3.3)
- *Description:* (Optional) Descriptive narrative for more information about the server.

wfs_accessconstraints

- *WFS TAG Name:* Accessconstraints (WFS 1.0.0, sect. 12.3.3)
- *Description:* (Optional) Text describing any access constraints imposed by the service provider on the WFS or data retrieved from this service.

wfs_encoding

- *Description:* (Optional) XML encoding for all XML documents returned by the server. The default is ISO-8859-1.

wfs_fees

- *WFS TAG Name:* Fees (WFS 1.0.0, sect. 12.3.3)
- *Description:* (Optional) Any fees imposed by the service provider for usage of this service or for data retrieved from the WFS.

wfs_keywordlist

- *WFS TAG Name:* Keyword (WFS 1.0.0, sect. 12.3.3)
- *Description:* (Optional) List of words to aid catalog searching.

wfs_maxfeatures

- *Description:* (Optional) The number of elements to be returned by the WFS server. This has priority over the 'maxfeatures' parameter passed by the user. If not set the current behaviour is not changed.

wfs_namespace_prefix

- *Description:* (Optional) User defined namespace prefix to be used in the response of a WFS GetFeature request. e.g. "wfs_namespace_prefix" "someprefix".

wfs_namespace_uri

- *Description:* (Optional) User defined namespace URI to be used in the response of a WFS GetFeature request. e.g. "wfs_namespace_uri" "http://somehost/someurl".

wfs_onlineresource

- *WFS TAG Name:* Onlineresource (WFS 1.0.0, sect. 12.3.3)
- *Description:* (Recommended) The URL prefix for HTTP GET requests.

wfs_service_onlineresource

- *Description:* (Optional) Top-level onlineresource URL. MapServer uses the onlineresource metadata (if provided) in the following order:
 1. wfs_service_onlineresource
 2. ows_service_onlineresource
 3. wfs_onlineresource (or automatically generated URL, see the onlineresource section of this document)

wfs_title

- *WFS TAG Name:* Title (WFS 1.0.0, sect. 12.3.3)
- *Description:* (Required) Human readable title to identify server.

wfs_srs

- *Description:* (Recommended) The SRS to use for all layers in this server. (e.g. EPSG:4326) See the notes below about the SRS rules in the WFS.

wfs_feature_collection

- *Description:* Replaces the default name of the feature-containing element (<msFeatureCollection>) with a user-defined value.

Layer Object**gml_exclude_items**

- *Description:* (Optional) A comma delimited list of items to exclude. As of MapServer 4.6, you can control how many attributes (fields) you expose for your data layer with metadata. The previous behaviour was simply to expose all attributes all of the time. The default is to expose no attributes at all. An example excluding a specific field would be:

```
"gml_include_items" "all"
"gml_exclude_items" "Phonenumber"
```

gml_featureid

- *Description:* (Required for MapServer 4.10) Field to be used for the ID of the feature in the output GML. wfs_featureid or ows_feature_id can be specified instead.

gml_groups

- *Description:* (Optional) A comma delimited list of group names for the layer.

gml_[group name]_group

- *Description:* (Optional) A comma delimited list of attributes in the group. Here is an example:

```
"gml_include_items" "all"  
"gml_groups" "display"  
"gml_display_group" "Name_e,Name_f"
```

gml_include_items

- *Description:* (Optional) A comma delimited list of items to include, or keyword “all”. As of MapServer 4.6, you can control how many attributes (fields) you expose for your data layer with this metadata. The previous behaviour was simply to expose all attributes all of the time. You can enable full exposure by using the keyword “all”, such as:

```
"gml_include_items" "all"
```

You can specify a list of attributes (fields) for partial exposure, such as:

```
"gml_include_items" "Name, ID"
```

The new default behaviour is to expose no attributes at all.

gml_[item name]_alias

- *Description:* (Optional) An alias for an attribute’s name. The served GML will refer to this attribute by the alias. Here is an example:

```
"gml_province_alias" "prov"
```

gml_[item name]_type

- *Description:* (Optional) Specifies the type of the attribute. Valid values are Integer|Real|Character|Date|Boolean.

gml_xml_items

- *Description:* (Optional) A comma delimited list of items that should not be XML-encoded.

gml_geometries

- *Description:* provides a name other than the default “msGeometry” for geometry elements. The value is specified as a string to be used for geometry element names.

gml_[name]_type

- *Description:* When employing `gml_geometries`, it is also necessary to specify the geometry type of the layer. This is accomplished by providing a value for `gml_[name]_type`, where `[name]` is the string value specified for `gml_geometries`, and a value which is one of:

- point
- multipoint
- line
- multiline
- polygon
- multipolygon

gml_[name]_occurrences

- *Description:* MapServer applies default values of 0 and 1, respectively, to the “minOccurs” and “maxOccurs” attributes of geometry elements, as can be seen in the preceding examples. To override these defaults, a value is assigned to a gml_[name]_occurrences layer metadata item, where again [name] is the string value specified for gml_geometries, and the value is a comma-delimited pair containing the respective lower and upper bounds.

wfs_abstract

- Same as wfs_abstract in the Web Object.

wfs_extent

- *Description:* (Optional) Used for the layer’s BoundingBox tag for cases where it is impossible (or very inefficient) for MapServer to probe the data source to figure its extents. The value for this metadata is “minx miny maxx maxy” separated by spaces, with the values in the layer’s projection units. If wfs_extent is provided then it has priority and MapServer will NOT try to read the source file’s extents.

wfs_featureid

- *Description:* (Required for MapServer 4.10) Field to be used for the ID of the feature in the output GML. gml_featureid or ows_feature_id can be specified instead.

wfs_keywordlist

- Same as wfs_keywordlist in the Web Object.

wfs_metadataurl_format

- *Description:* (Optional) The file format of the metadata record. Valid values are “XML”, “SGML”, or “HTML”. The layer metadata wfs_metadataurl_type and wfs_metadataurl_href must also be specified.
- refer to section 12.3.5 of the [WFS 1.0.0 spec](#).

wfs_metadataurl_href

- *Description:* (Optional) The URL to the layer’s metadata. The layer metadata wfs_metadataurl_type and wfs_metadataurl_format must also be specified.
- refer to section 12.3.5 of the [WFS 1.0.0 spec](#).

wfs_metadataurl_type

- *Description:* (Optional) The standard to which the metadata complies. Currently only two types are valid: “TC211” which refers to [ISO 19115], and “FGDC” which refers to [FGDC CSDGM]. The layer metadata wfs_metadataurl_format and wfs_metadataurl_href must also be specified.
- refer to section 12.3.5 of the [WFS 1.0.0 spec](#).

wfs_srs

- *Description:* If there is no SRS defined at the top-level in the mapfile then this SRS will be used to advertize this feature type (layer) in the capabilities. See the note below about the SRS rules in the WFS.

wfs_title

- Same as wfs_title in the Web Object.

9.5.4 To-do Items and Known Limitations

- This is just a basic WFS (read-only): transaction requests are not supported and probably never will given the nature of MapServer. [GeoServer](#) is recommended for those needing WFS-T support.

- WFS spec. seems to require that features of a given feature type must all be of the same geometry type (point, line, polygon). This works fine for shapefiles, but some data source formats supported by MapServer allow mixed geometry types in a single layer and this goes against the WFS spec. Suggestions on how to handle this are welcome (send suggestions to the MapServer-dev list, which you can join at <http://lists.umn.edu/archives/mapserver-dev.html>).

9.6 WFS Client

Author Jean-François Doyon

Contact jdoyon at nrcan.gc.ca

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8371 \$

Date \$Date: 2008-12-31 12:44:04 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2006/05/02

Contents

- WFS Client
 - Introduction
 - Setting up a WFS-client Mapfile
 - TODO / Known Limitations

9.6.1 Introduction

MapServer can retrieve and display data from a WFS server. The following document explains how to display data from a WFS server using MapServer.

A WFS (Web Feature Service) publishes feature-level geospatial data to the web. This means that it is possible to use this data as a data source to render a map. In effect, this is not unlike having a shapefile accessible over the web, only it's not a shapefile, it's XML-Encoded geospatial data (GML to be exact), including both geometry AND attribute information.

WFS-Related Information

Although in-depth understanding of WFS and GML is neither necessary nor required in order to implement a MapServer application that reads remote WFS data, it is recommended to at least get acquainted with the concepts and basic functionality of both. Here are the official references (including a newly added OGC workshop with MapServer):

- [OGC Web Feature Service Implementation Specification.](#)
- [Geography Markup Language Implementation Specification.](#)
- [MapServer OGC Web Services Workshop package.](#)

Software Requirements

In order to enable MapServer to serve WFS, it MUST be compiled against certain libraries:

- PROJ.4: The reprojection library. Version 4.4.3 or greater is required.
- GDAL/OGR: I/O support libraries. Version 1.1.8 or greater is required.
- LibCURL: Used to help MapServer act as an HTTP client. Version 7.10 or greater is required.

Please see the MapServer UNIX Compilation and Installation HOWTO for detailed instructions on compiling mapserver with support for these libraries and features. For Windows users, look on the MapServer website to see if there are any binaries available that meet these requirements.

9.6.2 Setting up a WFS-client Mapfile

Storing Temporary Files

You must set the *IMAGEPATH* parameter in your mapfile since MapServer uses this directory to store temporary files downloaded from the remote WFS server.

```
MAP
...
WEB
  IMAGEPATH "/tmp/ms_tmp/"
  IMAGEURL ...
END
...
END
```

WFS Layer

A WFS layer is a regular mapfile layer, which can use CLASS objects, with expressions, etc.

As of MapServer 4.4, the suggested method to define a WFS Client layer is through the CONNECTION parameter and the layer's METADATA. The necessary mapfile parameters are defined below:

- *CONNECTIONTYPE*: must be "wfs"
- *CONNECTION*: The URL to the WFS Server. e.g. http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap? The path to the mapfile on the WFS server is required if it was required in the GetCapabilities request e.g. you would have to specify the MAP parameter in the CONNECTION for the following server: <http://map.ns.ec.gc.ca/MapServer/mapserv.exe?MAP=/mapserver/services/envdat/config.map &SERVICE=WFS&VERSION=1.0.0&REQUEST=GetCapabilities>
- *METADATA*: The LAYER's must contain a METADATA object with the following parameters:
 - *wfs_connectiontimeout* (optional): The maximum time to wait for a remote WFS layer to load, set in seconds (default is 30 seconds). This metadata can be added at the layer level so that it affects only that layer, or it can be added at the map level (in the web object) so that it affects all of the layers. Note that *wfs_connectiontimeout* at the layer level has priority over the map level.
 - *wfs_filter*: This can be included to include a filter encoding parameter in the getFeature request (see the [Filter Encoding Howto](#) for more information on filtering). The content of the *wfs_filter* is a valid filter encoding element.

```

...
METADATA
  "wfs_filter"      "<PropertyIsGreaterThan><PropertyName>POP_RANGE</PropertyName><Literal>4
                  </Literal></PropertyIsGreaterThan>"
END
...

```

- *wfs_latlongboundingbox* (optional): The bounding box of this layer in geographic coordinates in the format “lon_min lat_min lon_max lat_max”. If it is set then MapServer will request the layer only when the map view overlaps that bounding box. You normally get this from the server’s capabilities output.
- *wfs_maxfeatures* (optional): Limit the number of GML features to return.
- *wfs_request_method* (optional): Can be set to “GET” to do a Get request to WFS servers that do not support Post requests. The default method in MapServer is Post.

```

...
METADATA
  "wfs_filter"      "GET"
END
...

```

- *wfs_typename* (required): the <Name> of the layer found in the GetCapabilities. An example GetCapabilities request is: http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?SERVICE=WFS&VERSION=1.0.0&REQUEST=getcapabilities
- *wfs_version* (required): WFS version, currently “1.0.0”

Note: Each of the above metadata can also be referred to as ‘ows_*’ instead of ‘wfs_*’. MapServer tries the ‘wfs_*’ metadata first, and if not found it tries the corresponding ‘ows_*’ name. Using this reduces the amount of duplication in mapfiles that support multiple OGC interfaces since “ows_*” metadata can be used almost everywhere for common metadata items shared by multiple OGC interfaces.

Example WFS Layer

```

LAYER
  NAME "park"
  TYPE POLYGON
  STATUS ON
  CONNECTIONTYPE WFS
  CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?"
  METADATA
    "wfs_typename"      "park"
    "wfs_version"       "1.0.0"
    "wfs_request_method" "GET"
    "wfs_connectiontimeout" "60"
    "wfs_maxfeatures"   "1"
  END
  PROJECTION
    "init=epsg:42304"
  END
  CLASS
    NAME "Parks"
    STYLE
      COLOR 200 255 0
      OUTLINECOLOR 120 120 120
    END
  END

```



```
END
END # Layer
```

Connection - deprecated

As of MapServer v4.4 the method of specifying all of the connection information in the CONNECTION parameter has been deprecated. The preferred method is mentioned above. If the metadata is not provided, VERSION, SERVICE, and TYPENAME will be fetched from the CONNECTION, as shown below

```
CONNECTION "http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap?SERVICE=WFS&VERSION=1.0.0&TYPENAME=park"
```

9.6.3 TODO / Known Limitations

1. Temporary WFS (gml) files are written to the IMAGEPATH directory, but this could become a security concern since it makes the raw GML data downloadable by someone who could guess the gml filename. We should consider having a “wfs_cache_dir” metadata that, if it is set will define a directory where temp files should be written. The default would still be to use the value of IMAGEPATH if “wfs_tmpdir” is not set.
2. Xerces is an annoying dependency for GML support in OGR. There have been discussions about modifying the OGR GML driver to use an internal XML parser by default. This would save us from the Xerces dependency. There is no formal plan for when this would happen yet.

9.7 WFS Filter Encoding

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Yewondwossen Assefa

Contact assefa at dmsolutions.ca

Revision \$Revision: 8479 \$

Date \$Date: 2009-01-29 11:56:34 -0800 (Thu, 29 Jan 2009) \$

Last Updated 2005/09/23

Table of Contents

- WFS Filter Encoding
 - Introduction
 - Currently Supported Features
 - Get and Post Requests
 - Use of Filter Encoding in MapServer
 - Limitations
 - Tests

9.7.1 Introduction

This document describes the procedures for taking advantage of the Filter Encoding (FE) support in WFS GetFeature requests, which was added to MapServer in version 4.2.

This document assumes that you are already familiar with the following aspects of MapServer:

- MapServer application development and setting up .map files.
- Familiarity with the WFS specification would be an asset. Links to the MapServer WFS documents are included in the next section.

Links to SLD-related Information

- [Filter Encoding Implementation Specification](#).
- [MapServer WFS Client Howto](#).
- [MapServer WFS Server Howto](#).
- [MapServer OGC Web Services Workshop](#).
- [Open GIS Consortium \(OGC\) home page](#).

9.7.2 Currently Supported Features

The following table lists the currently supported features for FE.

Table 1. Currently Supported Features

Feature Set	Feature
Spatial Capabilities	Equals Disjoint Touches Within Overlaps Crosses Intersects Contains DWithin BBOX
Scalar Capabilities	
Logical Operators	And Or Not
Comparison Operators	PropertyIsEqualTo (=) PropertyIsNotEqualTo (<>) PropertyIsLessThan (<) PropertyIsGreaterThan (>) PropertyIsLessThanOrEqualTo (<=) PropertyIsGreaterThanOrEqualTo (>=) PropertyIsLike PropertyIsBetween (range)

9.7.3 Get and Post Requests

MapServer already has the capability to receive and parse Get requests and URL-encoded Post requests. The ability for MapServer to be able to receive Post requests with XML-encoded information sent in the body of the request has been added. Also, the ability to generate XML-encoded Post requests for WFS layers has been added.

Both Get and Post request are now supported for all WFS requests:

- GetCapabilities
- GetFeatures
- DescribeFeatureType

Supporting these WFS requests in Post was implemented to keep consistency between all supported WFS requests.

When sending requests, the default request method used is Post. To change this behavior, we have introduced a layer level meta data, `wfs_request_method`, which can be set to “GET”.

9.7.4 Use of Filter Encoding in MapServer

This section describes how to use FE on both the server and client sides.

Server Side

To be able to use Filter Encoding, you need to create a valid WFS server using MapServer. Please refer to the *WFS Server HOWTO* for specifics.

There is nothing special that should be added to a WFS server for Filter Encoding, but you should note that, when requesting the capabilities of your WFS server, the document returned should contain the supported filters. Here is part of a Capabilities document as well as the Post support:

```

1  <?xml version='1.0' encoding="ISO-8859-1" ?>
2  <WFS_Capabilities
3      version="1.0.0"
4      updateSequence="0"
5      xmlns="http://www.opengis.net/wfs"
6      xmlns:ogc="http://www.opengis.net/ogc"
7      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
8      xsi:schemaLocation="http://www.opengis.net/wfs http://ogc.dmsolutions.ca/wfs/1.0.0/WFS-capabilities"
9
10 <!-- MapServer version 4.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
11 OUTPUT=PDF OUTPUT=SWF SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER
12 SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT INPUT=POSTGIS
13 INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE -->
14
15 <Service>
16   <Name>MapServer WFS</Name>
17   <Title>GMap WMS Demo Server</Title>
18   <OnlineResource>http://localhost/cgi-bin/mapserv.exe?map=
19     c:/msapps/wfs_filter/htdocs/ns_wfsserver.map&</OnlineResource>
20 </Service>
21
22 <Capability>
23   <Request>
24     <GetCapabilities>
25       <DCPType>
26         <HTTP>

```

```

27     <Get onlineResource="http://localhost/cgi-bin/mapserv.exe?map=
28       c:/msapps/wfs_filter/htdocs/ns_wfssserver.map&" />
29   </HTTP>
30 </DCPType>
31 <DCPType>
32   <HTTP>
33     <Post onlineResource="http://localhost/cgi-bin/mapserv.exe?map=
34       c:/msapps/wfs_filter/htdocs/ns_wfssserver.map&" />
35   </HTTP>
36 </DCPType>
37 </GetCapabilities>
38   ...
39 </Request>
40 </Capability>
41   ...
42 <Filter_Capabilities>
43   <Spatial_Capabilities>
44     <Spatial_Operators>
45       <Intersect/>
46       <DWithin/>
47       <BBOX/>
48     </Spatial_Operators>
49   </Spatial_Capabilities>
50   <Scalar_Capabilities>
51     <Logical_Operators />
52     <Comparison_Operators>
53       <Simple_Comparisons />
54       <Like />
55       <Between />
56     </Comparison_Operators>
57   </Scalar_Capabilities>
58 </Filter_Capabilities>
59
60 </WFS_Capabilities>

```

Client Side

To be able to generate a Filter to a WFS server, a layer level metadata called *wfs_filter* has been added, which should contain the filter to be sent to the server. Following is an example of a valid WFS client layer with a filter:

```

LAYER
NAME popplace
METADATA
  "wfs_version"      "1.0.0"
  "wfs_typename"    "popplace"
  "wfs_filter"       "<PropertyIsGreaterThan><PropertyName>POP_RANGE</PropertyName>
                    <Literal>4</Literal></PropertyIsC
END
TYPE POINT
STATUS ON
PROJECTION
  "init=epsg:42304"
END
CONNECTIONTYPE WFS
CONNECTION "http://www2.dmsolutions.ca:8090/cgi-bin/mswfs_filter?"
LabelItem "Name"

```

```

CLASSITEM "Capital"
CLASS
  SYMBOL 2
  SIZE 8
  NAME "Cities"
  LABEL
    COLOR 255 0 0
    FONT f r i t q a t - i t a l i c
    TYPE truetype
    SIZE 8
    POSITION AUTO
    OUTLINECOLOR 255 255 255
    FORCE TRUE
  END
  COLOR 0 0 0
END
END # Layer

```

Note:

- The filter given as a value of the `wfs_filter` metadata should not contain `<Filter>` start and end tags.
- The `CONNECTION` points to a valid WFS server supporting filters
- The returned shapes will be drawn using the class defined in the layer.

9.7.5 Limitations

- A limited set of spatial operators are supported.

9.7.6 Tests

Here are some test URLs for the different supported filters:

- `PropertyIsEqualTo`

```

http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><PropertyIsEqualTo><PropertyName>NAME</PropertyName>
<Literal>Digby</Literal></PropertyIsEqualTo></Filter>

```

- `PropertyIsNotEqualTo`

```

http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><PropertyIsNotEqualTo><PropertyName>NAME</PropertyName>
<Literal>Digby</Literal></PropertyIsNotEqualTo></Filter>

```

- `PropertyIsLessThan`

```

http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><PropertyIsLessThan><PropertyName>POP_RANGE</PropertyName>
<Literal>2</Literal></PropertyIsLessThan></Filter>

```

- `PropertyIsGreaterThan`

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><PropertyIsGreaterThan><PropertyName>POP_RANGE</PropertyName>
<Literal>5</Literal></PropertyIsGreaterThan></Filter>
```

- [PropertyIsLessThanOrEqualTo](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><PropertyIsLessThanOrEqualTo><PropertyName>POP_RANGE</PropertyName>
<Literal>2</Literal></PropertyIsLessThanOrEqualTo></Filter>
```

- [PropertyIsGreaterThanOrEqualTo](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><PropertyIsGreaterThanOrEqualTo><PropertyName>POP_RANGE</PropertyName>
<Literal>5</Literal></PropertyIsGreaterThanOrEqualTo></Filter>
```

- [PropertyIsBetween](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><PropertyIsBetween><PropertyName>POP_RANGE</PropertyName>
<LowerBoundary>5</LowerBoundary><UpperBoundary>6</UpperBoundary></PropertyIsBetween></Filter>
```

- [PropertyIsLike](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><PropertyIsLike wildcard='*' singleChar='.' escape='!'>
<PropertyName>NAME</PropertyName><Literal>Syd*</Literal></PropertyIsLike></Filter>
```

- [Logical operator OR](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><OR><PropertyIsEqualTo><PropertyName>NAME</PropertyName>
<Literal>Sydney</Literal></PropertyIsEqualTo><PropertyIsEqualTo><PropertyName>NAME</PropertyName>
<Literal>Digby</Literal></PropertyIsEqualTo></OR></Filter>
```

- [Logical operator AND](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><AND><PropertyIsLike wildcard='*' singleChar='.' escape='!'>
<PropertyName>NAME</PropertyName><Literal>Syd*</Literal></PropertyIsLike><PropertyIsEqualTo>
<PropertyName>POP_RANGE</PropertyName><Literal>4</Literal></PropertyIsEqualTo></AND></Filter>
```

- [Logical operator NOT](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&
TYPENAME=popplace&Filter=<Filter><AND><NOT><PropertyIsEqualTo><PropertyName>POP_RANGE</PropertyName>
<Literal>2</Literal></PropertyIsEqualTo></NOT><NOT><PropertyIsEqualTo><PropertyName>POP_RANGE</PropertyIsEqualTo>
<Literal>3</Literal></PropertyIsEqualTo></NOT></AND></Filter>
```

- [Spatial operator BBOX](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&TYPENAME=popplace&Filter=<Filter><BBOX><PropertyName>Name</PropertyName><Box%20srsName=' EPSG:42304' >
<coordinates>2518000,150000 2918000,450000</coordinates></Box></BBOX></Filter>
```

- [Spatial operator Dwithin](#)

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&TYPENAME=
<Filter><DWithin><PropertyName>Geometry</PropertyName><gml:Point><gml:coordinates>2587598.000000,404
</gml:Point><Distance units='m'>1000</Distance></DWithin></Filter>
```

- Spatial operator Intersects

```
http://www2.dmsolutions.ca/cgi-bin/mswfs_filter?&VERSION=1.0.0&SERVICE=WFS&REQUEST=GetFeature&TYPENAME=
<Intersect><PropertyName>Geometry</PropertyName><gml:Polygon><gml:outerBoundaryIs><gml:LinearRing>
<gml:coordinates>2488789,289552 2588789,289552 2588789,389552 2488789,389552 2488789,289552</gml:coor
</gml:LinearRing></gml:outerBoundaryIs></gml:Polygon></Intersect></Filter>
```

- Tests using MapServer as a WFS client and a WFS server are available at http://www2.dmsolutions.ca/msapps/wfs_fe/.
- The OGC conformance tests (<http://cite.occamlab.com/tsOGC/>) have been run on the FE support. The following table and notes reflect the current status.

Table 2. WFS OGC test suite (over the HTTP Get and Post method)

Test #	Description	# of Tests	# of Failed Tests
1	Basic WFS tests over the HTTP Get and Post method	402	281
1.1	GetCapabilities	16	0
1.2	DescribeFeatureType	18	0
1.3	GetFeature	368	281
1.3.1	Basic WFS tests	20	1
1.3.2	Complex WFS tests	18	18
1.3.3	Arithmetic filter WFS tests	8	8
1.3.4	Comparison WFS tests	50	26
1.3.4.1	GetFeature PropertyIsGreaterThanOrEqualTo filter	2	0
1.3.4.2	GetFeature PropertyIsBetween filter	6	2
1.3.4.3	GetFeature PropertyIsEqualTo filter	4	0
1.3.4.4	GetFeature PropertyIsGreaterThan filter	4	2
1.3.4.5	GetFeature PropertyIsGreaterThanOrEqualTo filter	6	6
1.3.4.6	GetFeature PropertyIsLessThan filter	6	4
1.3.4.7	GetFeature PropertyIsLessThanOrEqualTo filter	6	4
1.3.4.8	GetFeature PropertyIsLike filter	2	0
1.3.4.9	GetFeature PropertyIsNotEqualTo filter	6	0
1.3.4.10	GetFeature PropertyIsNull filter	8	8
1.3.5	Logical WFS test	20	0
1.3.5.1	GetFeature AND PropertyIsEqualTo PropertyIsEqualTo filter	8	0
1.3.5.2	GetFeature OR PropertyIsEqualTo PropertyIsEqualTo filter	8	0
1.3.5.3	GetFeature NOT PropertyIsNotEqualTo filter	4	0
1.3.6	Spatial operator WFS test	252	228
1.3.6.1	GetFeature BBOX filter	36	12
1.3.6.2	GetFeature with other filter types	216	216
2	Transactional WFS test	69	69

The OGC Cite WFS test suite can be found on the [OGC Cite portal](#).

Following are some MapServer specific notes on this test suite:

1. *Test number 1.3.1:*

- There is a contradiction between the wfs/1.0.0/basic/getfeature/post/3 assertion and the XPath expected value of the test. The assertion says: “Test that a GetFeature request with no output format defined returns a wfs:FeatureCollection with GML data.” and the expected XPath value for this request: “boolean(/ogc:ServiceExceptionReport)” is supposed to be true. So, the assertion means that when a WFS

server receives a request which contains an undefined output format or no output format at all, the WFS server must return a WFS collection containing GML data. The XPath expected value means that when a WFS server receives a request with an undefined output format or no output format at all, the WFS server must return a service exception report.

2. *Tests number 1.3.2 and 1.3.3:*

- Not supported.

3. *Tests number 1.3.4.2, 1.3.4.4 to 1.3.4.7:*

- The string comparison is not supported using >, <, >=, <=.
- The date comparison is not supported.

See Also:

[bug 461](#)

4. *Test number 1.3.4.10:*

- This property is not supported in MapServer.

5. *Test number 1.3.6.1:*

- The returned feature xml won't validate because the validation is done against a specific xsd (geometry.xsd).
- The data conversion on multipoints and multilayers are not supported within GDAL.

See Also:

[bug 461](#)

6. *Test number 2:*

- The transaction requests are not supported.

9.8 SLD

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Yewondwossen Assefa

Contact assefa at dmsolutions.ca

Last Updated 2008/07/18

Revision \$Revision: 8583 \$

Date \$Date: 2009-02-18 15:02:30 -0800 (Wed, 18 Feb 2009) \$

Contents

- SLD
 - Introduction
 - Server Side Support
 - Client Side Support
 - Named Styles support
 - Other Items Implemented
 - Issues Found During Implementation

9.8.1 Introduction

This document describes the procedures for taking advantage of the Styled Layer Descriptor (SLD) support in WMS GetMap requests with MapServer. SLD support exists for the server side (ability to read an SLD and apply it with a GetMap request) and for the client side (includes sending SLD requests to server and generate SLD files on the fly from MapServer map file). SLD support was added to MapServer in version 4.2.

This document assumes that you are already familiar with the following aspects of MapServer:

- MapServer application development and setting up *.map* files.
- Familiarity with the WMS specification would be an asset. Links to the MapServer WMS documents are included in the next section.

Links to SLD-related Information

- [Styled Layer Descriptor Implementation Specification](#).
- [MapServer WMS Client HowTo](#).
- [MapServer WMS Server HowTo](#).
- [MapServer OGC Web Services Workshop](#).
- [Open GIS Consortium \(OGC\) home page](#).

9.8.2 Server Side Support

General Information

There are two ways a WMS request can pass an SLD document with a GetMap request to MapServer:

- SLD parameter pointing to remote SLD (SLD=http://URL_TO_SLD).
- SLD_BODY parameter to send the SLD definition in the URL.

These two methods are both available through MapServer. An example of a request would be:

```
http://www2.dmsolutions.ca/cgi-bin/mswms_world?SERVICE=WMS&VERSION=1.1.1&Request=GetMap
&LAYERS=WorldGen_Outline&SLD=http://www2.dmsolutions.ca/msapps/world_testdata/tests
/sld_tests/sld_line_simple.xml
```

Test the [remote SLD request](#).

The SLD in the above request follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor version="1.0.0"
  xmlns="http://www.opengis.net/sld"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sld
http://schemas.opengis.net/sld/1.0.0/StyledLayerDescriptor.xsd">
  <NamedLayer>
    <Name>WorldGen_Outline</Name>
    <UserStyle>
      <Title>xxx</Title>
      <FeatureTypeStyle>
        <Rule>
          <LineSymbolizer>
            <Geometry>
              <ogc:PropertyName>center-line</ogc:PropertyName>
            </Geometry>
            <Stroke>
              <CssParameter name="stroke">#0000ff</CssParameter>
            </Stroke>
          </LineSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

Version 1.1.0 of the same SLD

```
<?xml version="1.0" encoding="UTF-8"?>
<StyledLayerDescriptor version="1.1.0"
  xmlns="http://www.opengis.net/sld"
  xmlns:se="http://www.opengis.net/se"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sld
http://schemas.opengis.net/sld/1.1.0/StyledLayerDescriptor.xsd">
  <NamedLayer>
    <se:Name>WorldGen_Outline</se:Name>
    <UserStyle>
      <se:Name>xxx</se:Name>
      <se:FeatureTypeStyle>
        <se:Rule>
          <se:LineSymbolizer>
            <se:Geometry>
              <ogc:PropertyName>center-line</ogc:PropertyName>
            </se:Geometry>
            <se:Stroke>
              <se:SvgParameter name="stroke">#0000ff</se:SvgParameter>
            </se:Stroke>
          </se:LineSymbolizer>
        </se:Rule>
      </se:FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
```

```
</StyledLayerDescriptor>
```

When MapServer gets a valid SLD through a request, it parses this SLD to extract all the styles attached to the NamedLayers, and then it applies these styles to the map before it is returned to the client. When applying the SLD, MapServer compares the <Name> parameter of the NamedLayers in the SLD document to the WMS layer names (WMS layer names are available in a *GetCapabilities* request).

Note: All the examples given in this document are live uses of valid SLDs and a MapServer installation with SLD support.

Additional WMS features related to SLDs have also been developed:

Table1. Additional WMS Features

Features	Supported	Notes
Method GET : SLD URL	Yes	Additional item
Method GET : SLD_BODY	Yes	
Describer Layer	Yes	
GetLegendGraphic	Yes	
GetStyles	Yes	
		Uses MapScript to get the SLD

Note: As of MapServer version 4.2.3, the GetLegendGraphic request (see section 12 of the *Styled Layer Descriptor Implementation Specification*) works as follows: if the RULE keyword is absent from the request, an image containing the entire legend for the specified layer will be returned. This image consists of the layer name and a symbolization graphic and label for each class.

Specific SLD Elements Supported

The following tables give a lot of additional details about SLD support in MapServer.

Table2. Named Layers and User Layers

Features	Supported	Notes
Named Layers	Yes	
User Layers	No	

Table3. Named Styles and User Styles

Features	Supported	Notes
Named Styles	Yes	
User Styles	Yes	

Table 4. User Styles

Features	Supported	Notes
Name	No	This was removed at implementation time, since it does not fit with MapServer
Title	No	No use in the MapServer environment
Abstract	No	No use in the MapServer environment
IsDefault	No	Only one style is available per layer
FeatureType-Style	Yes	MapServer has a concept of one feature type style per layer (either point, line, polygon, or raster)

Table 5. FeatureTypeStyle

Features	Supported	Notes
Name	No	No use in the MapServer environment
Title	No	No use in the MapServer environment
Abstract	No	No use in the MapServer environment
FeatureTypeName	No	No use in the MapServer environment
SemanticTypeIdentifier	No	Still an experimental element in the SLD specifications
Rule	Yes	

Table 6. Rule

Features	Supported	Notes
Name	Yes	
Title	Yes	
Abstract	No	No use in the MapServer environment
LegendGraphic	Yes	
Filter	Yes	
ElseFilter	Yes	
MinScaleDenominator	Yes	
MaxScaleDenominator	Yes	
LineSymbolizer	Yes	
PolygonSymbolizer	Yes	
PointSymbolizer	Yes	
TextSymbolizer	Yes	
RasterSymbolizer	Yes	Applies for 8-bit rasters

- Filter and ElseFilter

For each rule containing a filter, there is a class created with the class expression set to reflect that filter. Available filters that can be used are Comparison Filters and Logical Filters (see the *Filter Encoding HowTo*). The ElseFilter parameters are converted into a class in MapServer and placed at the end of the class list with no expression set. They are used to render elements that did not fit into any other classes.

- MinScaleDenomibator and MaxScaleDenominator are translated in minscale and maxscale in MapServer.

The following are examples of valid requests using the Filters:

- line with one filter: [sld 6a / full request 6a](#)
- line with multiple filters: [sld 6b / full request 6b](#)
- line with one filter and an else filter: [sld 6c / full request 6c](#)
- spatial filter using BBOX: [sld 6d/ full request 6d](#)
- The above example enables spatial filtering using the BBOX parameter as a Filter for a selected area (Africa). Note that an ElseFilter will not work with a spatial filter.

Table 7. LineSymbolizer

Features	Supported	Notes
Geometry	No	MapServer uses the data geometry to do the rendering
Stroke: GraphicFill	No	Solid color is used
Stroke: GraphicStroke	Yes	Draws the symbol along the line
Stroke (CssParameter): stroke	Yes	RGB colors are supported
Stroke (CssParameter): width	Yes	
Stroke (CssParameter): opacity	No	Not supported in MapServer
Stroke (CssParameter): linejoin and linecap	No	Not supported in MapServer
Stroke (CssParameter): dasharray	Yes	
Stroke (CssParameter): dashoffset	No	
PerpendicularOffset (only in SLD 1.1.0)	Yes	Offset values of the style object will be set
InitialGap(GraphicStroke SLD 1.1.0)	No	
Gap (GraphicStroke parameter SLD 1.1.0)	No	

Note that SvgParameter instead of CssParameter are required for SLD 1.1.0.

The following are examples of valid requests using the LineSymbolizer:

- simple line: [sld 7a](#) / [full request 7a](#)
- line with width: [sld 7b](#) / [full request 7b](#)
- dashed line: [sld 7c](#) / [full request 7c](#)

Table 8. PolygonSymbolizer

A Fill can be a solid fill or be a Graphic Fill, which is either a well-known Mark symbol (e.g., square, circle, triangle, star, cross, x) or an ExternalGraphic element (e.g., gif, png) available through a URL. When a Mark symbol is used in an SLD, MapServer creates a corresponding symbol in the map file and uses it to render the symbols. When a ExternalGraphic is used, the file is saved locally and a pixmap symbol is created in the mapfile referring to the this file. Note that the Web object IMAGEPATH is used to save the file.

The following are examples of valid requests using the PolygonSymbolizer:

- simple solid fill: [sld 8a](#) / [full request 8a](#)
- solid fill with outline: [sld 8b](#) / [full request 8b](#)
- fill with mark symbol: [sld 8c](#) / [full request 8c](#)
- fill with external symbol: [sld 8d](#) / [full request 8d](#)

Table 9. PointSymbolizer

Features	Supported	Notes
Geometry	No	
Graphic: Mark symbol	Yes	Well-known names (square, circle, triangle, star, cross, X) are supported
Graphic: ExternalGraphic	Yes	Was developed to support symbol fill polygons in addition to solid fill
Opacity	Yes	Support added in Mapserver 5.4
Size	Yes	
Rotation	Yes	Support added in MapServer 5.4
Displacement	Yes	SLD 1.1.0 Paramater. Support added in MapServer 5.4
AnchorPoint	No	

Note: refer to the PolygonSymbolizer notes for how the Mark and ExternalGraphic symbols are applied in MapServer.

The following are examples of valid requests using the PointSymbolizer:

- filled mark symbol: [sld 9a](#) / [full request 9a](#)
- default settings (square, size 6, color 128/128/128): [sld 9b](#) / [full request 9b](#)
- external symbol: [sld 9c](#) / [full request 9c](#)

Table 10. TextSymbolizer

Features	Supported	Notes
Geometry	No	
Label	Yes	
Font(font-family)	Yes	Font names used are those available in MapServer font file. If no fonts are available there, default bitmap fonts are used
Font-style (Italic, ...)	Yes	
Font-weight	Yes	
Font-size	Yes	
LabelPlacement	Yes	If true-type fonts are not used, default bitmap sizes are given PointPlacement is supported. LinePlacement is supported for versions >=5.2.1. Only PerpendicularOffset and IsAligned are supported for LinePlacement.
Halo	Yes	Supported (fill converted to outlinecolor, and radius is converted to outlinewidth. Note that outlinewidth is only available for AGG in >=5.2)
Fill	Yes	Only solid color is available

Notes on the TextSymbolizer:

- *Font names*: when converting Font parameters to MapServer, the following rule is applied to get the font name: FontFamily-FontStyle-FontWeight. For example, if there is an SLD with a Font Family of arial, a Font Style of italic, and a Font weight equal to bold, the resulting MapServer font name is arial-bold-italic. Font Style and Weight are not mandatory and, if not available, they are not used in building the font name. When a Font Style or a Font Weight is set to normal in an SLD, it is also ignored in building the name. For example, if there is an SLD with a Font Family of arial, a Font Style of normal and a Font weight equals to bold, the resulting MapServer font name is arial-bold.
- A TextSymbolizer can be used in MapServer either on an Annotation layer or on a Point, Line, or Polygon layer - in addition to other symbolizers used for these layers.
- PointPlacement: a point placement includes AnchorPoint (which is translated to Position in MapServer) Displacement (which is translated to Offset) and Angle (which is translated to Angle).
- Angle setting: by default the angle parameter is set to AUTO. For point features, users can use the PointPlacement to alter the value. For line features, the user can add a LinePlacement: If an 'empty' LinePlacement is part of the SLD, the angle will be set to FOLLOW, If a LinePlacement contains the PerpendicularOffset parameter, the angle will be set to 0 and the PerpendicularOffset will be used to set the offset values in the label object. SLD 1.1.0 introduces the IsAligned parameter for LinePlacement: if this parameter is set to false, the angle will be set to 0.

The following are examples of valid requests using the TextSymbolizer:

- annotation layer : test for label, font, point placement, color, angle: [sld 10a](#) / [full request 10a](#)
- annotation layer with text and symbols using 2 symbolizers: [sld 10b](#) / [full request 10b](#)
- line layer with text using 2 symbolizers: [sld 10c](#) / [full request 10c](#)

Table 11. RasterSymbolizer

Features	Supported	Notes
Geometry	No	
Opacity	Yes	
ChannelSelection	No	
OverlapBehaviour	No	
ColorMap	Yes	
ContrastEnhancement	No	
ShadedRelief	No	
ImageOutline	No	

The current support in MapServer includes only ColorMap parameter support. It can be used to classify 8-bit rasters. Inside the ColorMap parameters, the color and quantity parameters are extracted and used to do the classification.

Table 12. ColorMap

The following Features are available in SLD 1.0

Features	Supported	Notes
Color	Yes	
Opacity	No	
Quantity	Yes	
Label	No	

The following is an example of ColorMap usage for SLD 1.0.

If we have following ColorMap in an SLD:

```
<ColorMap>
  <ColorMapEntry color="#00ff00" quantity="22"/>
  <ColorMapEntry color="#00bf3f" quantity="30"/>
  <ColorMapEntry color="#007f7f" quantity="37"/>
  <ColorMapEntry color="#003fbf" quantity="45"/>
  <ColorMapEntry color="#0000ff" quantity="52"/>
  <ColorMapEntry color="#000000" quantity="60"/>
</ColorMap>
```

The six classes that are created are:

```
class 1: [pixel] >= 22 AND [pixel] < 30 with color 00ff00
class 2: [pixel] >= 30 AND [pixel] < 37 with color 00bf3f
class 3: [pixel] >= 37 AND [pixel] < 45 with color 007f7f
class 4: [pixel] >= 45 AND [pixel] < 52 with color 003fbf
class 5: [pixel] >= 52 AND [pixel] < 60 with color 0000ff
class 6: [pixel] = 60 with color 000000
```

Note that the ColorMapEntry quantity parameters should be in increasing order.

The following Features are available in SLD 1.1

The following is an example of and SLD 1.1.0 with a raster symbolizer

```
<StyledLayerDescriptor version="1.1.0" xsi:schemaLocation="http://www.opengis.net/sld http://schemas
<NamedLayer>
<se:Name>landsat</se:Name>
<UserStyle>
<se:Name>xxx</se:Name>
<se:FeatureTypeStyle>
<se:Rule>
<se:RasterSymbolizer>
```

```
<se:Opacity>0.7</se:Opacity>
<se:ColorMap>
<se:Categorize fallbackValue="#78c818">
<se:LookupValue>Rasterdata</se:LookupValue>
<se:Value>#ffffff</se:Value>
<se:Threshold>22</se:Threshold>
<se:Value>#00ff00</se:Value>
<se:Threshold>30</se:Threshold>
<se:Value>#00bf3f</se:Value>
<se:Threshold>37</se:Threshold>
<se:Value>#007f7f</se:Value>
<se:Threshold>45</se:Threshold>
<se:Value>#003fbf</se:Value>
<se:Threshold>52</se:Threshold>
<se:Value>#0000ff</se:Value>
<se:Threshold>60</se:Threshold>
<se:Value>#000000</se:Value>
</se:Categorize>
</se:ColorMap>
</se:RasterSymbolizer>
</se:Rule>
</se:FeatureTypeStyle>
</UserStyle>
</NamedLayer>
</StyledLayerDescriptor>
```

The classes that are created are:

```
class 1: [pixel] < 22 with color ffffff
class 2: [pixel] >= 22 AND [pixel] < 30 with color 00ff00
class 3: [pixel] >= 30 AND [pixel] < 37 with color 00bf3f
class 4: [pixel] >= 37 AND [pixel] < 45 with color 007f7f
class 5: [pixel] >= 45 AND [pixel] < 52 with color 003fbf
class 6: [pixel] >= 52 AND [pixel] < 60 with color 0000ff
class 7: [pixel] >= 60 with color 000000
```

Examples using 8 bits and 16 bits rasters can be seen at:

- [example 1](#)
- [example 2](#)

9.8.3 Client Side Support

Client side support of the SLD consists of two parts:

- The first part is using MapServer as a WMS client to send a GetMap request with an SLD. This is done using two metadata that can be placed at a layer level in a MapServer mapfile. These two metadata are:
 - `wms_sld_url`, which takes a valid URL as a value and appends `SLD=xxx` to the GetMap request.
 - `wms_sld_body`, which takes a valid SLD string and appends `SLD_BODY=xxx` to the GetMap request. If the value of `wms_sld_body` is set to `AUTO`, MapServer generates an SLD based on the classes found in the layer and send this SLD as the value of the `SLD_BODY` parameter in the GetMap request.
- The other major item is the generation of an SLD document from MapServer classes. These functions are currently available through MapServer/MapScript interface. Here are the functions available:
 - on a map object: `generatesld`

- on a layer object: generateSLD

Additional MapScript functions have been added or will be added to complement these functions:

- on a map object: applySLD
- on a layer object: applySLD

Note: When generating an SLD from MapServer classes, if there is a pixmap symbol you need to have this symbol available through a URL so it can be converted as an ExternalGraphic symbol in the SLD. To do this, you need to define the URL through a web object level metadata called WMS_SLD_SYMBOL_URL in your map file. The SLD generated uses this URL and concatenates the name of the pixmap symbol file to get the value that is generated as the ExternalGraphic URL.

PHP/MapScript Example that Generates an SLD from a Mapfile

The following is a small script that calls the generateSLD() function to create an SLD for a specific layer in a mapfile:

```

1  <?php
2
3  // define variables
4  define( "MAPFILE", "D:/ms4w/apps/cadastra/map/cadastra.map" );
5  define( "MODULE", "php_mapscript.dll" );
6
7  // load the mapscript module
8  if (!extension_loaded("MapScript")) dl(MODULE);
9
10 // open map
11 $oMap = ms_newMapObj( MAPFILE );
12
13 // get the parcel layer
14 $oLayer = $oMap->getLayerByName("parcel");
15
16 // generate the sld for that layer
17 $SLD = $oLayer->generateSLD();
18
19 // save sld to a file
20 $fp = fopen("parcel-sld.xml", "a");
21 fputs( $fp, $SLD );
22 fclose($fp);
23
24 ?>

```

9.8.4 Named Styles support

Named styles support are introduced in MapServer 5.2. The support is based on *MS RFC 39: Support of WMS/SLD Named Styles*

MapServer 5.2 introduces the possibility to assign a group to a series of classes defined on a layer object using two new non-mandatory keywords CLASSGROUP (at the layer level) and GROUP at the class level:

```

LAYER
...
  CLASSGROUP "group1"
...
  CLASS
    NAME "name1"

```

```
    GROUP "group1"
    ...
END
CLASS
  NAME "name2"
  GROUP "group2"
  ...
END
CLASS
  NAME "name3"
  GROUP "group1"
  ...
END
...
```

At rendering time, if the CLASSGROUP is defined, only classes that have the same group name would be used. Based on this concept, WMS/SLD support uses the class groups as named styles. Each group of classes is considered equivalent to a named style:

- The GetCapabilities request will output all the styles that are available
- The GetMap request can use the STYLES parameter to specify a named style
- The GetLegendGraphic can use the STYLES parameter to specify a named style

9.8.5 Other Items Implemented

- Support of filled polygons with Mark and ExternalGraphic symbols.
- MapScript functions to parse and apply SLD.
- SLD_BODY request support on client and server side.

9.8.6 Issues Found During Implementation

- Limitation of the FilterEncoding to comparison and logical filters. The spatial filters were not made available since it required major changes in MapServer WMS support.

9.9 WCS Server

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8451 \$

Date \$Date: 2009-01-23 10:42:36 -0800 (Fri, 23 Jan 2009) \$

Last Updated 2008/04/18

Table of Contents

- WCS Server
 - Introduction
 - Configuring Your Mapfile to Serve WCS Layers
 - Test Your WCS 1.0 Server
 - WCS 1.1.0+ Issues
 - Reference Section
 - Rules for handling SRS in a MapServer WCS
 - Spatio/Temporal Indexes
 - To-do Items and Known Limitations

9.9.1 Introduction

A WCS (or Web Coverage Service) allows for the publication of “coverages”- digital geospatial information representing space-varying phenomena. In the MapServer world it allows for unfiltered access to raster data. Conceptually it is easy think of WCS as a raster equivalent of WFS. The following documentation is based on the Open GIS Consortium’s (OGC) [Web Coverage Server Interfaces Implementation Specification version 1.0.0](#).

Links to WCS-Related Information

- [WCS 1.0.0 specification](#)
- [WCS 1.1.1c1 specification](#)
- [WMS Server HowTo](#)

Software Requirements

In order to enable MapServer to serve WCS data, it **MUST** be compiled against certain libraries:

- PROJ.4: The reprojection library. Version 4.4.3 or greater is required.
- GDAL: raster support library.
- MapServer: version ≥ 4.4 (tested with 5.0.2 while updating this document)

For WCS 1.1.x support (MapServer 5.2) there is an additional requirement:

- libxml2: An xml parser and generation library.

Please see the [MapServer UNIX Compilation and Installation HowTo](#) for detailed instructions on compiling MapServer with support for these libraries and features. For Windows users, [MapServer for Windows \(MS4W\)](#) comes WCS Server support.

9.9.2 Configuring Your Mapfile to Serve WCS Layers

Much as in the WMS and WFS support, WCS publishing is enabled by adding certain magic METADATA keyword/value pairs to a .map file.

MapServer will serve and include in its WCS capabilities only the layers that meet the following conditions:

- Data source is a raster, which is processed using GDAL (e.g GeoTIFF, Erdas Imagine, ...)
- LAYER NAME must be set

- LAYER TYPE is set to RASTER
- LAYER DUMP parameter set to TRUE
- WEB metadata “wcs_label” must be set
- LAYER metadata “wcs_label” must be set
- LAYER metadata “wcs_rangeset_name” must be set
- LAYER metadata “wcs_rangeset_label” must be set

Example WCS Server Mapfile

The following is an example of a simple WCS Server mapfile. Note the comments for the required parameters.

```
NAME WCS_server
STATUS ON
SIZE 400 300
SYMBOLSET ../etc/symbols.sym
EXTENT -2200000 -712631 3072800 3840000
UNITS METERS
SHAPEPATH "../data"
IMAGECOLOR 255 255 255
FONTSET "../etc/fonts.txt"

#
# Start of web interface definition
#
WEB
  IMAGEPATH "/ms4w/tmp/ms_tmp/"
  IMAGEURL "/ms_tmp/"
  METADATA
    "wcs_label" "GMap WCS Demo Server" ### required
    "wcs_description" "Some text description of the service"
    "wcs_onlineresource" "http://127.0.0.1/cgi-bin/mapserv.exe?" ### recommended
    "wcs_fees" "none"
    "wcs_accessconstraints" "none"
    "wcs_keywordlist" "wcs,test"
    "wcs_metadatalink_type" "TC211"
    "wcs_metadatalink_format" "text/plain"
    "wcs_metadatalink_href" "http://someurl.com"
    "wcs_address" "124 Gilmour Street"
    "wcs_city" "Ottawa"
    "wcs_stateorprovince" "ON"
    "wcs_postcode" "90210"
    "wcs_country" "Canada"
    "wcs_contactelectronicmailaddress" "blah@blah"
    "wcs_contactperson" "me"
    "wcs_contactorganization" "unemployed"
    "wcs_contactposition" "manager"
    "wcs_contactvoicetelephone" "613-555-1234"
    "wcs_contactfacimiletelephone" "613-555-1235"
    "wcs_service_onlineresource" "http://127.0.0.1/cgi-bin/mapserv.exe?"
  END
END

PROJECTION
```

```

"init=epsg:42304"
END

LAYER
  NAME bathymetry
  METADATA
    "wcs_label"          "Elevation/Bathymetry"    ### required
    "wcs_rangeset_name"  "Range 1"          ### required to support DescribeCoverage request
    "wcs_rangeset_label" "My Label"          ### required to support DescribeCoverage request
  END
  TYPE RASTER ### required
  STATUS ON
  DATA bathymapservers.tif
  PROJECTION
    "init=epsg:42304"
  END
  DUMP TRUE ### required
END

END # Map File

```

9.9.3 Test Your WCS 1.0 Server

Validate the Capabilities Metadata

OK, now that we've got a mapfile, we have to check the XML capabilities returned by our server to make sure nothing is missing.

Using a web browser, access your server's online resource URL to which you add the parameters "SERVICE=WCS&VERSION=1.0.0&REQUEST=GetCapabilities" to the end, e.g.

```
http://my.host.com/cgi-bin/mapserv?map=mywcs.map&SERVICE=WCS&VERSION=1.0.0&REQUEST=GetCapabilities
```

If you get an error message in the XML output then take necessary actions. Common problems and solutions are listed in the FAQ at the end of this document.

If everything went well, you should have a complete XML capabilities document. Search it for the word "WARNING"... MapServer inserts XML comments starting with "<!--WARNING: " in the XML output if it detects missing mapfile parameters or metadata items.

Note that when a request happens, it is passed through WMS, WFS, and WCS in MapServer (in that order) until one of the services respond to it.

Here is a working example of a GetCapabilities request:

```
http://maps.dnr.state.mn.us/cgi-bin/mapserv50?MAP=/usr/local/www/docs_maps/mapserver_demos/wcs/demo.map&SERVICE=wcs&
```

Test With a DescribeCoverage Request

OK, now that we know that our server can produce a valid XML GetCapabilities response we should test the DescribeCoverage request. The DescribeCoverage request lists more information about specific coverage offerings.

Using a web browser, access your server's online resource URL to which you add the parameters "SERVICE=WCS&VERSION=1.0.0&REQUEST=DescribeCoverage&COVERAGE=layername" to the end, e.g.

<http://my.host.com/cgi-bin/mapserv?map=mywcs.map&SERVICE=WCS&VERSION=1.0.0&REQUEST=DescribeCoverage&>

Here is a working example of a DescribeCoverage request:

http://maps.dnr.state.mn.us/cgi-bin/mapserv50?MAP=/usr/local/www/docs_maps/mapserver_demos/wcs/demo.map&SERVICE=wcs&

Test With a GetCoverage Request

The GetCoverage request allows for the retrieval of coverages in a specified output format to the client.

The following is a list of the required GetCoverage parameters according to the WCS spec:

VERSION=version: Request version

REQUEST=GetCoverage: Request name

COVERAGE=coverage_name: Name of an available coverage, as stated in the GetCapabilities

CRS=epsg_code: Coordinate Reference System in which the request is expressed.

BBOX=minx,miny,maxx,maxy: Bounding box corners (lower left, upper right) in CRS units. One of BBOX or TIME is required.

TIME=time1,time2: Request a subset corresponding to a time. One of BBOX or TIME is required..

WIDTH=output_width: Width in pixels of map picture. One of WIDTH/HEIGHT or RESX/Y is required.

HEIGHT=output_height: Height in pixels of map picture. One of WIDTH/HEIGHT or RESX/Y is required.

RESX=x: When requesting a georectified grid coverage, this requests a subset with a specific spatial resolution. One of WIDTH/HEIGHT or RESX/Y is required.

RESY=y: When requesting a georectified grid coverage, this requests a subset with a specific spatial resolution. One of WIDTH/HEIGHT or RESX/Y is required.

FORMAT=output_format: Output format of map, as stated in the GetCapabilities.

The following are optional GetCoverage parameters according to the WCS spec:

RESPONSE_CRS=epsg_code: Coordinate Reference System in which to express coverage responses.

So to follow our above examples, a valid DescribeCoverage request would look like:

```
http://my.host.com/cgi-bin/mapserv?map=mywcs.map&SERVICE=wcs&VERSION=1.0.0&REQUEST=GetCoverage
&coverage=bathymetry&CRS=EPSG:42304&BBOX=-2200000,-712631,3072800,3840000
&WIDTH=3199&HEIGHT=2833&FORMAT=Gtiff
```

Here is a working example of a GetCoverage request (note that a 3MB tif is being requested, so this may take a few seconds):

```
http://gws2.pcigeomatics.com/wcs1.0.0/wcs?SERVICE=wcs&VERSION=1.0.0&REQUEST=GetCoverage&COVERAGE=demo/wcs_
117.88341239280106,33.707704191028995,-117.65485697866967,33.89850474983795&WIDTH=700&HEIGHT=700&FORMAT=C
```

9.9.4 WCS 1.1.0+ Issues

WCS 1.1.0 and later versions of the WCS protocol are supported by MapServer 5.2. For the most part the map file setup for WCS 1.1.0 is similar to WCS 1.0.0, but the actual protocol is substantially changed.

GetCapabilities

The GetCapabilities request is the same as WCS 1.0 but with a different VERSION value:

```
SERVICE=WCS&VERSION=1.1.0&REQUEST=GetCapabilities
```

The format of the returned capabilities document is substantially altered from WCS 1.0, and makes use of OWS Common for service descriptions.

DescribeCoverage

The DescribeCoverage request is similar to WCS 1.0, but the IDENTIFIER keyword is used instead of COVERAGE to name the coverage being requested:

```
SERVICE=WCS&VERSION=1.1.0&REQUEST=DescribeCoverage&IDENTIFIER=spaceimaging
```

GetCoverage

The format for GetCoverage is substantially changed from 1.0. The following is a list of GetCoverage required parameters:

VERSION=version: Request version

REQUEST=GetCoverage: Request name

IDENTIFIER=coverage_name: Name of an available coverage, as stated in the GetCapabilities

BOUNDINGBOX=minx,miny,maxx,maxy,crs: Bounding box corners (lower left, upper right), and the CRS they are in. The CRS is described using a URN.

FORMAT=output_format: Output format (mime type) of grid product, as stated in the GetCapabilities.

If an alternate spatial resolution is desired, then the following set of keywords must be used to specify the sample origin and step size of the output grid to be produced. The produced grid will be of a number of pixels and lines as can be fit in the BOUNDINGBOX starting at GridOrigin, at GridOffsets resolution.

GRIDBASECRS=crs: The grid base CRS (URN).

GRIDCS=crs: The grid CRS (URN).

GridType=urn:ogc:def:method:WCS:1.1:2dGridIn2dCrS: This is the only supported value for MapServer.

GridOrigin=x_origin,y_origin: The sample point for the top left pixel.

GridOffsets=xstep,ystep: The x and y step size for grid sampling (resolution). Both are positive.

As well, the following optional parameters are available.

RangeSubset=selection: Selects a range subset, and interpolation method. Currently only subsetting on bands are allowed. Depending on rangeset names, this might take the form “BandsName[bands[1]]” to select band 1, or “BandsName:bilinear[bands[1]]” to select band 1 with bilinear interpolation.

So a simple GetCoverage might look like:

```
SERVICE=WCS&VERSION=1.1.0&REQUEST=GetCoverage&IDENTIFIER=dem&FORMAT=image/tiff
&BOUNDINGBOX=43,33,44,34,urn:ogc:def:crs:EPSG::4326
```

A more complex request might look like:

```
SERVICE=WCS&VERSION=1.1.0&REQUEST=GetCoverage&IDENTIFIER=dem&FORMAT=image/tiff
&BOUNDINGBOX=33,43,34,44,urn:ogc:def:crs:EPSG::4326
&GridBaseCRS=urn:ogc:def:crs:EPSG::4326&GridCS=urn:ogc:def:crs:EPSG::4326
&GridType=urn:ogc:def:method:WCS:1.1:2dGridIn2dCrs
&GridOrigin=33,44&GridOffsets=0.01,0.01
&RangeSubset=BandsName:bilinear[bands[1]]
```

It should also be noted that return results from WCS 1.1 GetCoverage requests are in multi-part mime format. Typically this consists of a first part with an xml document referencing the other parts of the message, and an image file part. However, for output formats that return multiple files, each will be a separate part. For instance, this means it is possible to return a jpeg file with a world file, the OUTPUTFORMAT is appropriately configured.

URNs

In WCS 1.1 protocol coordinate systems are referenced by URN. Some typical URNs are:

```
urn:ogc:def:crs:EPSG::4326
urn:ogc:def:crs:EPSG:27700
urn:ogc:def:crs:OGC::CRS84
```

The first two are roughly equivalent to EPSG:4326, and EPSG:27700 while the third is a CRS defined by OGC (essentially WGS84). One critical thing to note is that WCS 1.1 follows EPSG defined axis/tuple ordering for geographic coordinate systems. This means that coordinates reported, or provided in urn:ogc:def:EPSG::4326 (WGS84) are actually handled as lat, long, not long,lat. So, for instance the BOUNDINGBOX for an area in California might look like:

```
BOUNDINGBOX=34,-117,35,-116,urn:ogc:def:crs:EPSG::4326
```

And, likewise the bounds reported by GetCapabilities, and DescribeCoverage will be in this ordering as appropriate.

9.9.5 Reference Section

WCS support tries to leverage existing WMS or OWS metadata definitions. The rationale being that many values will be the same regardless of the type of service. The module will look for the following metadata prefixes (in order): wcs, ows and finally wms unless otherwise noted.

The following metadata are available in the setup of the mapfile:

Web Object Metadata

wcs_abstract

- *Description:* (Optional) A brief description of the service, maps to ows:Abstract (WCS 1.1+ only).

wcs_accessconstraints

- *Description:* (Optional) A list of codes describing any access constraints imposed by the service provider. The keyword NONE is reserved to mean no access constraints are imposed.

**wcs_address, wcs_city, wcs_contactelectronicmailaddress, wcs_contactfacilelphone,
wcs_contactorganization, wcs_contactperson, wcs_contactposition, wcs_contactvoicetelephone, wcs_country,
wcs_postcode, wcs_service_onlineresource, wcs_stateorprovince**

- *Description:* (Optional) Contact address information. If provided then all twelve metadata items are required. You can also use the responsibleparty* metadata instead.

wcs_description

- *Description:* (Optional) A description of the server.

wcs_fees

- *Description:* (Optional) A text string indicating any fees imposed by the service provider.

wcs_keywords

- *Description:* (Optional) Short words for catalog searching.

wcs_label

- *Description:* (Required) A human-readable label for the server.

wcs_metadatalink_format

- *Description:* (Optional) The file format MIME type of the metadata record (e.g. “text/plain”). The web metadata `wcs_metadatalink_type` and `wcs_metadatalink_href` must also be specified.

wcs_metadatalink_href

- *Description:* (Optional) The URL to the server’s metadata. The web metadata `wcs_metadatalink_format` and `wcs_metadatalink_type` must also be specified.

wcs_metadatalink_type

- *Description:* (Optional) The standard to which the metadata complies. Currently only two types are valid: “TC211” which refers to [ISO 19115], and “FGDC” which refers to [FGDC-STD-001-1988]. The web metadata `wcs_metadatalink_format` and `wcs_metadatalink_href` must also be specified.

wcs_name

- *Description:* (Optional) A name for the server.

`wcs_responsibleparty_address_administrativearea,` `wcs_responsibleparty_address_city,`
`wcs_responsibleparty_address_country,` `wcs_responsibleparty_address_deliverypoint,`
`wcs_responsibleparty_address_electronicmailaddress,` `wcs_responsibleparty_address_postalcode,`
`wcs_responsibleparty_individualname, wcs_responsibleparty_onlineresource, wcs_responsibleparty_organizationname,`
`wcs_responsibleparty_phone_facsimile, wcs_responsibleparty_phone_voice, wcs_responsibleparty_postionname`

- *Description:* (Optional) Contact address information. If provided then all twelve metadata items are required. You can also use the `address*` metadata instead.

Layer Object Metadata**wcs_abstract**

- *Description:* (Optional) A brief description of the service, maps to `ows:Abstract` (WCS 1.1+ only).

wcs_description

- *Description:* (Optional) A description of the layer.

wcs_extent

- *Description:* (Optional) Bounding box of layer, which must be provided for tiled data. Comma-delimited, in the format of: `minx,miny,maxx,maxy`

wcs_formats

- *Description:* (Optional) The formats which may be requested for this layer, in a comma-delimited list. (e.g. “GTiff,MrSID”)

wcs_keywords

- *Description:* (Optional) Short words for catalog searching.

wcs_label

- *Description:* (Required) A human-readable label for the layer.

wcs_metadatalink_format

- *Description:* (Optional) The file format MIME type of the metadata record (e.g. “text/plain”). The web metadata `wcs_metadatalink_type` and `wcs_metadatalink_href` must also be specified.

wcs_metadatalink_href

- *Description:* (Optional) The URL to the layer’s metadata. The web metadata `wcs_metadatalink_format` and `wcs_metadatalink_type` must also be specified.

wcs_metadatalink_type

- *Description:* (Optional) The standard to which the metadata complies. Currently only two types are valid: “TC211” which refers to [ISO 19115], and “FGDC” which refers to [FGDC-STD-001-1988]. The web metadata `wcs_metadatalink_format` and `wcs_metadatalink_href` must also be specified.

wcs_name

- *Description:* (Optional) A name for the layer.

wcs_nativeformat

- *Description:* (Optional) The current format of the served raster layer. (e.g. “GTiff”)

Axes Descriptions

MapServer allows you define a number of these for a layer. Individual axis are identified by name when defining specific metadata (e.g. description). All defined axes must be listed in the `rangeset_axes` metadata tag so MapServer knows in advance what to expect. A special rangeset for multiband data is automatically generated by adding the name “bands” to the `rangeset_axes` list. If found MapServer will automatically generate metadata for the image bands. You may of course extend that basic support using the naming conventions below.

wcs_rangeset_axes

- *Description:* (Optional) Delimited list of defined range sets. If defined, you can also use the following nine metadata items, where *rangeset axis* matches the axis name provided in this `wcs_rangeset_axes` metadata:

- `{rangeset axis}_semantic`
- `{rangeset axis}_refsys`
- `{rangeset axis}_refsyslabel`
- `{rangeset axis}_description`
- `{rangeset axis}_label`
- `{rangeset axis}_values`
- `{rangeset axis}_values_semantic`
- `{rangeset axis}_values_type`
- `{rangeset axis}_interval`

wcs_rangeset_label

- *Description:* (Required for DescribeCoverage request)

wcs_rangeset_name

- *Description:* (Required for DescribeCoverage request)

wcs_srs

- *Description:* (Optional) Spatial reference system of the layer, in the form of: EPSG:code (e.g. EPSG:42304)

wcs_timeitem

- *Description:* (Optional) The attribute in the spatio/temporal index that contains time values.

wcs_timeposition

- *Description:* (Optional) A list of the start and end time of a given coverage (i.e. “2000-11-11T11:11:11Z,2001-11-11T11:11:11Z”), used when advertising GetCapabilities.

9.9.6 Rules for handling SRS in a MapServer WCS

TODO!

9.9.7 Spatio/Temporal Indexes

MapServer has long supported a method of breaking a dataset into smaller, more manageable pieces or tiles. In this case a shapefile is used to store the boundary of each tile, and an attribute holds the location of the actual data. Within a MapServer mapfile the layer keywords `TILEINDEX` and `TILEITEM` are used to activate tiling.

Consider the example where an organization wants to serve hundreds or even thousands of MODIS scenes. Five images cover the spatial extent and each group of five varies by date of acquisition. This turns out to be a fairly common scenario for organizations interested in WCS, one that the existing tiling support does not adequately address. In previous versions of MapServer a developer would have to create one tile index and one layer definition for each group of five images. This could result in configuration files that are prohibitively long and difficult to manage.

In order to more efficiently support the WCS specification a new tiling scheme has been implemented within MapServer. One that supports spatial sub-setting, but also ad hoc sub-setting based on any attributes found within tile index. In many cases a temporal attribute could be used, but sub-setting is not limited to that case. The new scheme introduces the concept of tile index layers, that is, a separate layer definition is used to describe the tile index dataset. With this we get all the benefits of any MapServer layer, most importantly we can apply MapServer filters to the data. Filters can be defined at runtime using MapServer CGI, MapScript or via the WCS server interface. The syntax for the layer using the index remains unchanged except that the value for *Tile Indexes* refers to the index layer instead of an external shapefile.

So, looking at the example above again we can reduce our MapServer configuration to two layer definitions, one for the tile index and one for the imagery itself. Extracting a single dates worth of imagery is now a matter of setting the appropriate filter within the tile index layer.

Building Spatio-Temporal Tile Indexes

Developing these tile indexes is more difficult than basic indexes simply because there are no ready-made tools to do so. Fortunately we can leverage existing tool available within MapServer or supporting libraries such as GDAL by post processing their output.

Taking the above example, building an index is relatively simple task if you are willing to roll up your sleeves and write a bit of code. First, the basic spatial index needs to be built. The GDAL utility `gdaltindex` already does this. Simply point `gdaltindex` at the directory containing the collection of MODIS images and it will build a shapefile index suitable for use with MapServer. The next step would be to add the temporal information. The pseudo code would look something like:

- open the index `.dbf` file for reading
- create a new column to hold the image acquisition date

- for each image; 1) extract the image acquisition date and 2) insert it into the new column
- close the index .dbf file

This general approach could be used for many cases. A scripting language such as Perl, PHP or Python works well since they all have readily available modules for manipulating .dbf files. A worst case would involve hand editing the resulting .dbf file using a desktop tool such as Microsoft Access or ESRI Arcview.

9.9.8 To-do Items and Known Limitations

- For now we support only GET requests. The spec also talks about XML-encoded POST requests but that is not supported at this time. It is anticipated that a general XML request solution will be created for all OWS services.
- MapServer does not derive all of the metadata it could from a given dataset. For example, you must explicitly list time periods covered by a layer. This should get better with time.
- Only spatial, simple temporal and radiometric band subsetting is possible with the current implementation. Future enhancements should allow for arbitrary subsets based on pixel values or tile/image attributes.

9.10 WCS Use Cases

Author Norman Barker

Contact nbarker at ittvis.com

Author Gail Millin

Contact nbarker at ittvis.com

Revision \$Revision: 8295 \$

Date \$Date: 2008-12-26 21:08:04 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2005/12/12

Contents

- WCS Use Cases
 - Landsat
 - SPOT
 - DEM
 - NetCDF

This document explains how to use MapServer to deliver Landsat, SPOT, DEM, and NetCDF temporal/banded data through the MapServer WCS interface. Thanks go to Steve Lime and Frank Warmerdam for their assistance with these projects

9.10.1 Landsat

To serve Landsat imagery through the MapServer Web Coverage Service specify the *OUTPUTFORMAT* object. For format support install the GDAL library and from the command prompt and cd to where GDAL is installed and use the command, `gdalinfo -formats`. A list of all supported formats will appear and will specify if the format is read only `<ro>` or read and write `<rw>` for WCS the format needs to be supported for read and write (except for *GDAL's* own WCS format, however).

For the example below the Landsat 7 15m resolution mosaic is in a Enhanced Compressed Wavelets format (ECW). By running the gdalinfo.exe program I could verify that the ECW format has write permissions, therefore the format can be specified in the MapFile and requested using the GetCoverage request.

```

OUTPUTFORMAT
  NAME "ECW"
  DRIVER "GDAL/ECW"
  MIMETYPE "image/ecw"
  IMAGEMODE "BYTE"
  EXTENSION "ecw"
END

```

The keyword "DUMP" must be set to true to allow the data to download during a GetCoverage request.

```

LAYER
  NAME "Landsat7"
  STATUS OFF
  TYPE RASTER
  PROCESSING "SCALE=AUTO"
  UNITS Meters
  DUMP TRUE
  TILEINDEX "MapServer/wcs/landsat7/17mosaic15m.shp"
  TILEITEM "Location"
  METADATA
    "wcs_description" "Landsat 7 15m resolution mosaic"
    "wcs_name" "Landsat7"
    "wcs_label" "Landsat 7 15m resolution mosaic"
    "ows_srs" "EPSG:27700"
    "ows_extent" "0 0 700005 1050000"
    "wcs_resolution" "75 75"
    "wcs_bandcount" "3"
    "wcs_formats" "ECW"
  END
END

```

A GetCoverage request can then be requested (using the parameters set in the MapFile) by creating a URL with the elements: - Your Server, MapServer Program, Location of MapFile, Type of Service (WCS), Request (GetCoverage), Coverage (Landsat7), BBOX (0,0,700005,1050000), CRS (EPSG:27700), ResX (75) ResY (75), Format (ECW).

9.10.2 SPOT

SPOT imagery can be delivered through MapServer Web Coverage Service similarly to the Landsat example above. The main difference is that as SPOT is a greyscale image the wcs_bandcount = 1 rather than a Landsat image which consists of 3 bands. For this example the well known GeoTiff format will be used to demonstrate what to specify in a MapFile for SPOT data.

```

OUTPUTFORMAT
  NAME "GEOTIFF"
  DRIVER "GDAL/GTiff"
  MIMETYPE "image/tiff"
  IMAGEMODE "BYTE"
  EXTENSION "tif"
END
LAYER
  NAME "SPOT"
  STATUS OFF

```

```
TYPE RASTER
PROCESSING "SCALE=AUTO"
UNITS Meters
DUMP TRUE
TILEINDEX "MapServer/wcs/orthospot/spot.shp"
TILEITEM "Location"
METADATA
  "wcs_description" "Orthospot mosaic"
  "wcs_name" "SPOT"
  "wcs_label" "Orthospot mosaic"
  "ows_srs" "EPSG:27700"
  "ows_extent" "375960 64480 497410 200590"
  "wcs_resolution" "100 100"
  "wcs_bandcount" "1"
  "wcs_formats" "GEOTIFF"
  "wcs_nativeformat" "8-bit GeoTIF"
END
END
```

The key parameters to specify in the WCS MapFile for any data layer and format are:

- Layer Name = Create a short name for the data
- Layer Type = Raster
- Layer Dump = True

The following examples further demonstrate how WCS can be implemented and also how to create WCS containing layers with a temporal dimension (see NetCDF example).

9.10.3 DEM

It is possible to deliver 16 bit DEM data through the MapServer Web Coverage Service.

Firstly it is necessary to specify the output format in the map file

```
OUTPUTFORMAT
  NAME "GEOTIFFINT16"
  DRIVER "GDAL/GTiff"
  MIMETYPE "image/tiff"
  IMAGEMODE "INT16"
  EXTENSION "tif"
END
```

and in the corresponding layer to specify the keyword “DUMP” to be true

```
LAYER
  NAME "srtm"
  STATUS OFF
  TYPE RASTER
  DUMP TRUE
  DATA "srtm.tif"
  PROJECTION
    "init=epsg:4326"
  END
  METADATA
    "wcs_label" "SRTM WCS TIF Server"
    "ows_extent" "-180 -90 180 90"
```

```
"wcs_resolution" "0.00083 -0.00083"
"ows_srs" "EPSG:4326"
"wcs_formats" "GEOTIFFINT16"
"wcs_nativeformat" "geotiff"
```

END

END

Performance gains can be made by using the gdaladdo utility described at http://www.gdal.org/gdal_utilities.html#gdaladdo

9.10.4 NetCDF

Firstly GDAL doesn't support all versions of netCDF (there are a lot, it is a generic format), so for stability it may be necessary to convert the files into GeoTiff format first. This can be achieved using the netCDF libraries here <http://my.unidata.ucar.edu/content/software/netcdf/index.html>. Denis Nadeau and Frank Warmerdam have added netCDF CF as a read only format within GDAL, so it now possible to read the CF convention netCDF files directly from disk.

We placed the Z-levels in the bands of the GDAL data file (either GeoTiff or netCDF), and created a shape index for the time levels. GDAL data is a 2-D format (x,y) and bands. netCDF is an N-D file format, supporting time, x,y,z, and experiment parameters. By using a set of GDAL netCDF / geoTiff files it is possible to represent this, and to store the z-level (height) as bands within the data file. Although a hack, it is possible for a custom client to receive important metadata from the describeCoverage operation of a WCS about the which z-level a band of a geotiff represents by encoding this in the returned axes description tag.

To create the shape file for the temporal dimension we had to do some hacking with Java code, but we also got it to work with Steve Lime's perl script in the MODIS MapServer demo download (which doesn't seem to be available now).

The perl script used in Modis demo by Steve Lime is as follows, and I have placed inline comments below. The script assumes that gdaltindex has already been run in this directory to create a tile index shape and dbf file. It assumes that the filenames of your data files have the date in the filename, for example myfileYYYYMMDDHH.tif

```
1  #!/usr/bin/perl
2  use XBase;
3  opendir(DIR, '.'); # open the current directory
4  foreach $file (readdir(DIR)) {
5      next if !($file =~ /\.dbf$/); # read the dbf file in this directory created by gdaltindex
6      print "Working on $file...\n";
7      $tfile = 'temporary.dbf';
8      system("mv $file $tfile");
9      $oldtable = new XBase $tfile or die XBase->errstr;
10     print join("\t", $oldtable->field_names) ."\n";
11     print join("\t", $oldtable->field_types) ."\n";
12     print join("\t", $oldtable->field_lengths) ."\n";
13     print join("\t", $oldtable->field_decimals) ."\n";
14     $newtable = XBase->create("name" => $file,
15                             "field_names" => [$oldtable->field_names, "IMGDATE"], # this is the FILTERITEM in
16                             "field_types" => [$oldtable->field_types, "C"], # character column type
17                             "field_lengths" => [$oldtable->field_lengths, 13], # length of the date string
18                             "field_decimals" => [$oldtable->field_decimals, undef]) or die "Error creating new
19     foreach (0 .. $oldtable->last_record) {
20         ($deleted, @data) = $oldtable->get_record($_);
21         print "  ...record $data[0]\n";
22         # extract the date
23         $year = substr $data[0], 8, 4; # year is at position 8 in the filename string
```

```

24     $month = substr $data[0], 12, 2; # month is at position 12 in the filename string
25     $day = substr $data[0], 14, 2; # day is at position 14 in the filename string
26     $hour = substr $data[0], 16, 2; # hour is at position 16 in the filename string
27     $date = "$year-$month-$day" . "T" . "$hour\n"; # format is YYYY-MM-DDTHH, or any ISO format
28     print "$date";
29     push @data, "$date";
30     $newtable->set_record($_, @data);
31 }
32 $newtable->close();
33 $oldtable->close();
34 unlink($tfile);
35 }

```

If have used the perl script then skip to the layer definitions below, if you wish to code your own the description is here.

The DBF file has to have the column 'location' that indicates the location of the data file (either absolute path or relative to the map file location, and the second column that can be called whatever you want but indexes time. In our case we called it 'time' :-)

The corresponding shapefile then has to contain Polygons with the bounding boxes of the tif file for each time. So OGRInfo timeIndex.shp looks something like:

```

OGRFeature(timeIndex):116
  location(String) = mytime.tif
  time(String) = 2001-01-31T18:00:00
  POLYGON ((xxx,xxxx,.....))

```

Define your output format as

```

OUTPUTFORMAT
  NAME "GEOTIFF_FLOAT"
  DRIVER 'GDAL/GTiff'
  MIMETYPE 'image/tiff'
  IMAGEMODE FLOAT32
  EXTENSION 'tiff'
END

```

Then you need to define your tile index within the map file

```

LAYER
  NAME 'time_idx'
  TYPE TILEINDEX
  DATA 'timeIndex'
  FILTERITEM 'time'
  FILTER '%time%'
END

```

and the actual layer

```

LAYER
  NAME 'TempData'
  STATUS OFF
  TYPE RASTER
  DUMP TRUE
  TILEINDEX 'time_idx'
  PROJECTION
    "init=epsg:4326"

```


END**METADATA**

```

"wcs_label" 'Temperature data'
"ows_extent" '-180 -90 180 90'
"wcs_resolution" '1.125 -1.125'
"ows_srs" 'EPSG:4326'
"wcs_formats" 'GEOTIFF_FLOAT'
"wcs_nativeformat" 'netCDF'
"wcs_bandcount" '27'
"wcs_rangeset_axes" 'bands'
"wcs_rangeset_label" 'Pressure (hPa units) Levels'
"wcs_rangeset_name" 'bands'
"wcs_rangeset_description" 'Z levels '
"wcs_timeposition" '2001-01-01T06:00:00,2001-01-01T12:00:00,2001-01-01T18:00:00,2001-01-02T00:00:00'
"wcs_timeitem" 'time'

```

END**END**

The TempData coverage layer will now let you subset with the `&bands=...` `&time=...` subset parameters!

To do a coordinate reprojection specify in the request `&Response_CRS=ESPG:xxxx`

When you start doing temporal subsetting with WCS and MapServer you can see the need for an automatic way of generating map files such as using an XSL stylesheet!

For a tile-index layer you need to provide the following extra metadata in order to use it for WCS:

```

"OWS_EXTENT" "10050 299950 280050 619650"
"WCS_RESOLUTION" "100 100"
"WCS_SIZE" "2700 3197"
"WCS_BANDCOUNT" "3"

```

If your image has a colortable and only one band, it will come out greyscale unless you set the `IMAGEMODE` to `PC256` instead of `BYTE`.

9.11 SOS Server

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 9025 \$

Date \$Date: 2009-05-19 07:17:03 -0700 (Tue, 19 May 2009) \$

Last Updated 2007/12/06

Table of Contents

- SOS Server
 - Introduction
 - Setting Up an SOS Server Using MapServer
 - Limitations / TODO
 - Reference Section
 - Use of `sos_procedure` and `sos_procedure_item`

9.11.1 Introduction

SOS (Sensor Observation Service), currently an OGC discussion paper, is part of the OGC's SensorWeb Enablement (SWE) group of specifications. These specifications describe how applications and services will be able to access sensors of all types over the Web. Specifically, SOS provides an API for managing deployed sensors and retrieving sensor data.

SOS support is **available in MapServer 4.10.0 or more recent**. Note that no client tools currently exist in MapServer for SOS. More SWE based software is available at <http://www.52north.org/>

SOS support was implemented in MapServer to the guidelines of MapServer *MS RFC 13: Support of Sensor Observation Service in MapServer*.

This document assumes that you are already familiar with certain aspects of MapServer:

- MapServer application development and setting up .map files.

Links to SOS-Related Information

- [SOS discussion paper](#)
- [Sensor Web Enablement and OpenGIS SensorWeb](#)

Relevant Definitions

The following is taken from the SOS discussion paper:

Observation An observation is an event with a result which has a value describing some phenomenon.

Observation Offering An observation offering is a logical grouping of observations offered by a service that are related in some way.

Observed Value A value describing a natural phenomenon, which may use one of a variety of scales including nominal, ordinal, ratio and interval.

Sensor An entity capable of observing a phenomenon and returning an observed value. A sensor can be an instrument or a living organism (e.g. a person).

9.11.2 Setting Up an SOS Server Using MapServer

Install the Required Software

SOS requests are handled by the “*mapserv*” CGI program. The first step is to check that your mapserv executable includes SOS support. One way to verify this is to use the “-v” command-line switch and look for “SUPPORTS=SOS_SERVER”.

Example 1. On Unix:

```
$ ./mapserv -v
MapServer version 4.9 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP OUTPUT=SVG S
UPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT SUPPORTS=
WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER SUPPORTS=SOS_SERVER SUPPORTS=
THREADS INPUT=JPEG INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE DEBUG=MSDEBUG
```

Example 2. On Windows:

```
C:\Apache\cgi-bin> mapserv -v
MapServer version 4.9 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP OUTPUT=SVG S
UPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT SUPPORTS=
WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER SUPPORTS=SOS_SERVER SUPPORTS=
THREADS INPUT=JPEG INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE DEBUG=MSDEBUG
```

If you don't have SOS support in your MapServer build, then you must compile MapServer with the following in mind:

- flag `-DUSE_SOS_SVR` is required
- requires either `-DUSE_WMS_SVR` or `-DUSE_WFS_SVR` flags to be enabled
- requires libxml2 and proj libraries
- requires ICONV support (`-DUSE_ICONV`) on Windows

For more help with MapServer compilation see the appropriate HowTo: [Unix / Windows](#)

Configure a Mapfile For SOS

Each instance of SOS server that you setup needs to have its own mapfile. It is just a regular MapServer mapfile in which some parameters and some metadata entries are mandatory. Most of the metadata is required in order to produce a valid GetCapabilities output.

Here is the list of parameters and metadata items that usually optional with MapServer, but are **required (or strongly recommended) for a SOS configuration**:

MAP level:

- Map NAME
- Map PROJECTION
- Map Metadata (in the WEB Object):
 - `sos_title`
 - `sos_onlineresource`
 - `sos_srs`
 - see the Reference Section of this document for a full list of metadata and descriptions

LAYER level:

- Layer NAME
- Layer PROJECTION
- Layer METADATA
 - `sos_offering_id`
 - `sos_observedproperty_id`
 - `sos_observedproperty_id`
 - `sos_describesensor_url`
 - see the Reference Section of this document for a full list of metadata and descriptions

Onlineresource URL

The `sos_onlineresource` metadata is set in the map's web object metadata and specifies the URL that should be used to access your server. This is required for the `GetCapabilities` output. If `sos_onlineresource` is not provided then MapServer will try to provide a default one using the script name and hostname, but you shouldn't count on that too much. It is strongly recommended that you provide the `wfs_onlineresource` metadata.

Here is a valid online resource URL:

```
http://my.host.com/cgi-bin/mapserv?map=mysos.map&
```

By creating a wrapper script on the server it is possible to hide the "map=" parameter from the URL and then your server's online resource URL could be something like:

```
http://my.host.com/cgi-bin/mapserv?
```

This is covered in more detail in the "More About the Online Resource URL" section of the *WMS Server* document.

Example SOS Server Mapfile

The following is an example of a bare minimum SOS Server mapfile. Note the comments for the required parameters.

```
NAME "SOS_DEMO"
STATUS ON
SIZE 300 300
EXTENT -66 44 -62 45
UNITS METERS
SHAPEPATH "./data/"
IMAGECOLOR 255 255 0
SYMBOLSET "./etc/symbols.sym"

IMAGETYPE png

WEB
  IMAGEPATH "/ms4w/tmp/ms_tmp/"
  IMAGEURL "/ms_tmp/"

METADATA
  "sos_onlineresource" "http://127.0.0.1/cgi-bin/mapserv.exe?map=D:/ms4w/apps/sos/sos_test.map" ## REQUIRED
  "sos_title"          "My SOS Demo Server" ## Recommended
  "sos_srs"            "EPSG:4326" ## REQUIRED
END
END

PROJECTION
  "init=epsg:4326"
END

LAYER
  NAME "test_sos_layer"
  METADATA
    "sos_procedure" "NS01EE0014" ## REQUIRED
    "sos_offering_id" "WQ1289" ## REQUIRED
    "sos_observedproperty_id" "Water Quality" ## REQUIRED
    "sos_describesensor_url" "http://some/url/NS01EE0014.xml" ## REQUIRED
  END
```

```

TYPE POINT
STATUS ON
DATA "sos_test"

PROJECTION
  "init=epsg:4326"
END

CLASS
  NAME "water quality"
  STYLE
    COLOR 255 0 0
    SYMBOL "circle"
    SIZE 8
  END
END
END #map

```

Test Your SOS Server

GetCapabilities Request

The GetCapabilities request allows the clients to retrieve service metadata about a specific service instance. For an SOS service, it allows to identify such things as offerings and observed property available, as well as information on sensors that are used.

Using a web browser, access your server's online resource URL to which you add the parameters "SERVICE=SOS&REQUEST=GetCapabilities" to the end, e.g.

```
http://my.host.com/cgi-bin/mapserv?MAP=mysos.map&SERVICE=SOS&REQUEST=GetCapabilities
```

If everything went well, you should have a complete XML capabilities document. Search it for the word "WARNING"... MapServer inserts XML comments starting with "<!--WARNING: " in the XML output if it detects missing mapfile parameters or metadata items. If you notice any warning in your XML output then you have to fix all of them before you can try your server with an SOS client, otherwise things are likely not going to work.

Note: The SERVICE parameter is required for all SOS requests.

GetObservation Request

The GetObservation request is designed to query sensor systems to retrieve observation data in the form defined in the Observation and Measurement specification (O&M), and more information on this O&M spec can be found at <http://www.opengeospatial.org/functional/?page=swe>. Upon receiving a GetObservation request, a SOS shall either satisfy the request or return an exception report.

The following is a list of the possible parameters for a GetObservation request:

- request:** (Required) value must be "GetObservation".
- service:** (Required) value must be "SOS".
- version:** (Required) value must be "1.0.0".
- offering:** (Required) The Offering identified in the capabilities document.

observedProperty: (Required) The property identified in the capabilities document.

responseFormat: (Required) The format / encoding to be returned by the response.

eventTime (Optional) Specifies the time period for which observations are requested.

procedure: (Optional) The procedure specifies the sensor system used. In this implementation, the procedure is equivalent to be the sensor id that will be used when doing a DescribeSensor request.

featureOfInterest: (Optional) In this implementation, this will be represented by a gml envelope defining the lower and upper corners.

Result: (Optional) The Result parameter provides a place to put OGC filter expressions based on property values.

resultModel: (Optional) Identifier of the result model to be used for the requested data. The resultModel values supported by a SOS server are listed in the contents section of the service metadata (GetCapabilities). MapServer currently supports om:Observation and om:Measurement. om:Measurement provides a flat model of the geometry and attributes, similar to WFS GetFeature output. om:Observations provides a more compact definition which includes an XML header of the field names and definitions, followed by a “DataBlock” of delimited records (default is CSV delimited output). The default output is om:Measurement.

srsName: (Optional) srs (EPSG code) of the output response.

Here are some valid examples:

Example 1:

```
http://127.0.0.1/cgi-bin/mapserv.exe?map=D:/ms4w/apps/sos/sos_test.map&Request=GetObservation&service=
Offering=WQ1289&observedproperty=Water Quality&version=1.0.0&responseFormat=text/xml; subtype="om/1.0.0"
```

Example 2:

```
http://127.0.0.1/cgi-bin/mapserv.exe?map=D:/ms4w/apps/sos/sos_test.map&Request=GetObservation&service=
Offering=WQ1289&observedproperty=Water Quality&eventtime=<ogc:TM_Equals><gml:TimePeriod><gml:beginPos
<PropertyName>COLOUR</PropertyName><Literal>180</Literal></PropertyIsEqualTo><PropertyIsEqualTo>
<PropertyName>COLOUR</PropertyName><Literal>200</Literal></PropertyIsEqualTo></or></Filter>&version=
```

Example 3:

```
http://127.0.0.1/cgi-bin/mapserv.exe?map=D:/ms4w/apps/sos/sos_test.map&Request=GetObservation&service=
Offering=WQ1289&observedproperty=Water Quality&featureofinterest=
<gml:Envelope><gml:lowerCorner srsName='EPSG:4326'>-66 43</gml:lowerCorner><gml:upperCorner srsName=
-64 45</gml:upperCorner></gml:Envelope>&version=1.0.0&responseFormat=text/xml; subtype="om/1.0.0"
```

Example 4:

```
http://127.0.0.1/cgi-bin/mapserv.exe?map=D:/ms4w/apps/sos/sos_test.map&Request=GetObservation&service=
Offering=WQ1289&observedproperty=Water Quality&version=1.0.0&responseFormat=text/xml; subtype="om/1.0.0"
```

DescribeSensor Request

The DescribeSensor request gives the client the ability to retrieve the characteristics of a particular sensor and return the information in a SensorML xml document. In this implementation, MapServer does not generate the SensorML document but only redirect the request to an existing SensorML document.

The following is a list of the possible parameters for a DescribeSensor request:

request: (Required) value must be “DescribeSensor”

service: (Required) value must be “SOS”.

version: (Required) value must be “1.0.0”.

procedure: (Required) This is the sensor id, which was specified in the “sos_procedure” metadata.

outputFormat: (Required) The format encoding to be returned by the response.

Here is a valid example:

```
http://127.0.0.1/cgi-bin/mapserv.exe?map=D:/ms4w/apps/sos/sos_test.map&Request=DescribeSensor&
procedure=urn:ogc:def:procedure:NS01EE0014&service=SOS&version=1.0.0&outputFormat=text/xml; subtype=
```

9.11.3 Limitations / TODO

1. Have MapServer generate the SensorML document, instead of redirecting the request to an existing SensorML document.

9.11.4 Reference Section

The following metadata are available in the setup of the SOS Server mapfile:

Note: Each of the metadata below can also be referred to as ‘ows_*

Web Object Metadata

ows_updatesequence

- *Description:* (Optional) The updateSequence parameter can be used for maintaining the consistency of a client cache of the contents of a service metadata document. The parameter value can be an integer, a timestamp in [ISO 8601:2000] format, or any other number or string.

ows_language

- *Description:* (Optional) Descriptive narrative for more information about the server. Identifier of the language used by all included exception text values. These language identifiers shall be as specified in IETF RFC 1766. When this attribute is omitted, the language used is not identified. Examples: “en-CA”, “fr-CA”, “en-US”. Default is “en-US”.

ows_schemas_location

- *Description:* (Optional) (Note the name ows_schemas_location and not sos/_... this is because all OGC Web Services (OWS) use the same metadata) Root of the web tree where the family of OGC SOS XMLSchema files are located. This must be a valid URL where the actual .xsd files are located if you want your SOS output to validate in a validating XML parser. Default is <http://www.opengeospatial.net/sos>. See <http://ogc.dmsolutions.ca> for an example of a valid schema tree.

sos_abstract

- *Description:* (Optional) Descriptive narrative for more information about the server.

sos_keywordlist

- *Description:* (Optional) A comma-separated list of keywords or keyword phrases to help catalog searching.

sos_accessconstraints

- *Description:* (Optional) Text describing any access constraints imposed by the service provider on the SOS or data retrieved from this service.

sos_addresstype, sos_address, sos_city, sos_stateorprovince, sos_postcode, sos_country

- *Description:* Optional contact address information. If provided then all six metadata items are required.

sos_contactelectronicmailaddress

- *Description:* Optional contact Email address.

sos_contactperson, sos_contactposition, sos_contactorganization

- *Description:* Optional contact information. If provided then all three metadata items are required.

sos_contactvoicetelephone

- *Description:* Optional contact voice telephone number.

sos_contactfacsimiletelephone - * *Description:* Optional contact facsimile telephone number.

sos_fees

- *Description:* (Optional) Fees information. Use the reserved word “none” if there are no fees.

sos_onlineresource

- *Description:* (Required) The URL that will be used to access this OGC server. This value is used in the GetCapabilities response.
- See the section “Onlineresource URL” above for more information.

sos_service_onlineresource

- *Description:* (Optional) Top-level onlineresource URL.

sos_srs

- *Description:* (Required) Contains a list of EPSG projection codes that should be advertized as being available for all layers in this server. The value can contain one or more EPSG:<code> pairs separated by spaces (e.g. “EPSG:4269 EPSG:4326”) This value should be upper case (EPSG:42304.....not epsg:42304) to avoid problems with case sensitive platforms.

sos_title

- *Description:* (Recommended) A human-readable name for this Layer.

sos_hoursofservice

- *Description:* (Optional) Time period (including time zone) when individuals can contact the organization or individual.

sos_contactinstructions

- *Description:* (Optional) Supplemental instructions on how or when to contact the individual or organization.

sos_role

- *Description:* (Optional) Function performed by the responsible party. Possible values of this Role shall include the values and the meanings listed in Subclause B.5.5 of ISO 19115:2003.

sos_maxfeatures

- *Description:* (Optional) The number of elements to be returned by the WFS server. If the not set all observations are returned

sos_encoding_blockSeparator

- *Description:* (Optional) For GetObservation requests using resultModel=om:Observation (SWE DataBlock encoding). Record separator to be used. Default is ‘\n’

sos_encoding_tokenSeparator

- *Description:* (Optional) For GetObservation requests using resultModel=om:Observation (SWE DataBlock encoding). Token (field) separator to be used. Default is ‘;’

Layer Object Metadata**sos_describesensor_url**

- *Description:* (Required) This metadata item is only a temporary measure until the describe sensor is generated from MapServer. Right now when a DescribeSensor request is sent with a procedure (sensorid), it will redirect it to the url defined by this metadata item.
- In MapServer 5.0, it is possible to use variable substitution on the url. For example “sos_describesensor_url” “http://foo/foo?mysensor=%procedure%” will substitute the %procedure% in the metadata with the procedure value coming from the request.

```
"sos_describesensor_url" "http://some/url/NS01EE0014.xml"
```

sos_[item name]_alias

- *Description:* (Optional) An alias for an attribute’s name that will be returned when executing a GetObservation request.

sos_[item name]_definition

- *Description:* (Optional) An associated definition (usually a URN) for a component, that will be returned when executing a GetObservation request. Default is “urn:ogc:object:definition”

sos_[item name]_uom

- *Description:* (Optional) An associated unit of measure URN) for a component, that will be returned when executing a GetObservation request. Default is “urn:ogc:object:uom”

sos_observedproperty_id

- *Description:* (Required) ID of observed property, possibly in number format.

sos_observedproperty_name

- *Description:* (Optional) Name of observed property, possibly in string format.

sos_observedproperty_authority

- *Description:* (Optional) An associated authority for a given component of an observed property

sos_observedproperty_version

- *Description:* (Optional) An associated version for a given component of an observed property

sos_offering_description

- *Description:* (Optional) Description of offering.

sos_offering_extent

- *Description:* (Optional) Spatial extents of offering, in *minx, miny, maxx, maxy* format:

```
"sos_offering_extent" "-66, 43, -62, 45"
```

The logic for the bounding box returned as part of the offering is the following:

- note that it is a mandatory element that needs an espg code and lower/upper corner coordinates
- looks for the espg parameter in the first layer of the offering (this could be an ows/sos_srs or a projection object with the espg code (mandatory))
- looks for sos_offering_extent. If the metadata is not available, the extents of all layers in the offering will be used to compute it.

Here is an example result from a GetCapabilities request:

```
<gml:boundedBy>
  <gml:Envelope>
    <gml:lowerCorner srsName="EPSG:4326">-66 43</gml:lowerCorner>
    <gml:upperCorner srsName="EPSG:4326">-62 45</gml:upperCorner>
  </gml:Envelope>
</gml:boundedBy>
```

sos_offering_id

- *Description:* (Required) ID of offering, possibly in number format.

sos_offering_intendedapplication

- *Description:* (Optional) The intended category of use for this offering.

sos_offering_name

- *Description:* (Optional) Name of offering, possibly in string format.

sos_offering_timeextent

- *Description:* (Optional) Time extent of offering, in the format of “begin/end”. Here is an example:

```
"sos_offering_timeextent" "1990/2006"
```

If end is not specified it will be set to now. Here is an example result from a GetCapabilities request:

```
<sos:eventTime>
  <gml:TimePeriod>
    <gml:beginPosition>1990</gml:beginPosition>
    <gml:endPosition>2006</gml:endPosition>
  </gml:TimePeriod>
</sos:eventTime>
```

sos_procedure

- *Description:* (Required) Normally a sensor unique id. One per layer:

```
"sos_procedure" "NS01EE0014"
```

Note: sos_procedure can also be a list, separated by spaces, i.e.:

```
"sos_procedure" "35 2147 604"
```

All *sos_procedure* links from layers in the offerings will be outputted together, such as the following taken from a GetCapabilities response:

```
<procedure xlink:href="urn:ogc:object:feature:Sensor:3eTI:csi-sensor-1"/>
<procedure xlink:href="urn:ogc:object:feature:Sensor:3eTI:csi-sensor-2"/>
```

sos_procedure_item

- *Description:* (Required if sos_procedure is not present): See section 5 for more details

```
"sos_procedure_item" "attribute_field_name"
```

sos_timeitem

- *Description:* (Optional) Name of the time field. It will be used for queries when a GetObservation request is called with an EVENTTIME parameter. It is layer specific and should be set on all layers.

```
"sos_timeitem" "TIME"
```

9.11.5 Use of sos_procedure and sos_procedure_item

In MapServer 5.0 SOS support has been upgraded to use a new metadata called sos_procedure_item. The value for sos_procedure_item is the field/attribute name containing the procedure values. The use of this metadata as well as the sos_procedure is described here per type of request (refer to <http://trac.osgeo.org/mapserver/ticket/2050> for more description):

It should be noted that, for very large datasets defined only with sos_procedure_item, this may result in costly processing, because MapServer has to process attribute data. It is advised to setup and manage datasets accordingly if dealing with large observation collections.

GetCapabilities

- if sos_procedure is defined, use it
- if not look for sos_procedure_item : procedure values are extracted from the layer's attribute specified by this metadata. Not that this can be time consuming for layers with a large number of features.
- if none is defined return an exception

DescribeSensor

- if sos_procedure is defined, use it
- if not look for sos_procedure_item : procedure values are extracted from the layer's attribute specified by this metadata
- if none is defined return an exception

GetObservation

Both sos_procedure and sos_procedure_item can be define. Here are the cases:

- **case 1** [only sos_procedure is defined.]
 - Use this metadata to match the layer with the procedure value sent in the request
 - When outputting the <member/procedure> output the value of the metadata

Note: If more than one procedure is defined per LAYER object, output observations will have incorrect sos:procedure values, because there is no way to map procedures to observations. This is where sos_procedure_item should be used (i.e. when more than one procedure makes up a LAYER object).

- **case 2: only procedure_item is defined.** – Use the sos_procedure_item and do a query on the layer to match the procedure with the layer.

- When outputting the <member/procedure> use the procedure_item as a way to only output the attribute value corresponding to the feature.
- **case 3: both are defined.** – check in sos_procedure to match the procedure with the layer.
- When outputting the <member/procedure> use the procedure_item as a way to only output the attribute value corresponding to the feature.

9.12 MapScript Wrappers for WxS Services

Author Frank Warmerdam

Contact warmerdam at pobox.com

Revision \$Revision: 8278 \$

Date \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Contents

- MapScript Wrappers for WxS Services
 - Introduction
 - Python Examples
 - Perl Example
 - Java Example
 - PHP Example
 - Use in Non-CGI Environments (mod_php, etc)
 - Post Processing Capabilities

9.12.1 Introduction

With the implementation of MapServer *MS RFC 16: MapScript WxS Services* in MapServer 4.9, MapScript now has the ability to invoke MapServer's ability to execute OGC Web Service requests such as WMS, WCS, and WFS as well as capturing the results of processing the requests.

This makes it possible to dynamically configure a map object based on information in the original request, and to capture the output of processing requests for further post-processing.

9.12.2 Python Examples

The following trivial example, in Python, demonstrates a script that internally provides the map name, but otherwise uses normal mapserver processing.

```
import mapscript

req = mapscript.OWSRequest()
req.loadParams()

map = mapscript.mapObj( '/u/www/maps/ukpoly/ukpoly.map' )
map.OWSDispatch( req )
```

The OWSRequest object is used to manage a parsed list of OWS processing options. In the above example they are loaded from the environment using the loadParams() call which fetches and parses them from QUERY_STRING in the same way the *mapserv* executable would.

Then we load a map, and invoke OWSDispatch with the given arguments on that map. By default the results of the dispatched request are written to stdout which returns them back to the client.

The following example ignores all passed in arguments, and manually constructs a request argument by argument. It is likely more useful for testing purposes than for deploying WxS services, but demonstrates direct manipulation of the request object.

```
import mapscript

req = mapscript.OWSRequest()
req.setParameter( 'SERVICE', 'WMS' )
req.setParameter( 'VERSION', '1.1.0' )
req.setParameter( 'REQUEST', 'GetCapabilities' )

map = mapscript.mapObj( '/u/www/maps/ukpoly/ukpoly.map' )
map.OWSDispatch( req )
```

The previous example have all let results be returned directly to the client. But in some cases we want to be able to capture, and perhaps modify the results of our requests in some custom way. In the following example we force the hated OGC required mime type for errors to simple text/xml (warning - non-standard!)

```
import mapscript

req = mapscript.OWSRequest()
req.loadParams()

map = mapscript.mapObj( '/u/www/maps/ukpoly/ukpoly.map' )

mapscript.msIO_installStdoutToBuffer()
map.OWSDispatch( req )

content_type = mapscript.msIO_stripStdoutBufferContentType()
content = mapscript.msIO_getStdoutBufferBytes()

if content_type == 'vnd.ogc.se_xml':
    content_type = 'text/xml'

print 'Content-type: ' + content_type
print
print content
```

This example demonstrates capture capturing output of OWSRequest to a buffer, capturing the “Content-type:” header value, and capturing the actual content as binary data. The msIO_getStdoutBufferBytes() function returns the stdout buffer as a byte array. If the result was known to be text, the msIO_getStdoutBufferString() function could have been used to fetch it as a string instead, for easier text manipulation.

9.12.3 Perl Example

Most of the same capabilities are accessible in all SWIG based mapscript languages. In perl, we could script creation of a request like this:

```
#!/usr/bin/perl

use MapScript;

$req = new MapScript::OWSRequest();
$req->setParameter( "SERVICE", "WMS" );
$req->setParameter( "VERSION", "1.1.0" );
$req->setParameter( "REQUEST", "GetCapabilities" );

$map = new MapScript::mapObj( "/u/www/maps/ukpoly/ukpoly.map" );

MapScript::msIO_installStdoutToBuffer();

$dispatch_out = $map->OWSDispatch( $req );

printf "%s\n", MapScript::msIO_getStdoutBufferString();
```

One issue in Perl is that there is currently no wrapping for binary buffers so you cannot call `msIO_getStdoutBufferBytes()`, and so cannot manipulate binary results.

More Perl example code

```
#!/usr/bin/perl
#####
#
# Name:      wxs.pl
# Project:   MapServer
# Purpose:   MapScript WxS example
#
# Author:    Tom Kralidis
#
#####
#
# Copyright (c) 2007, Tom Kralidis
#
# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies of this Software or works derived from this Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
# OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.
#####

use CGI::Carp qw(fatalsToBrowser);
use MapScript;
```

```

use strict;
use warnings;
use XML::LibXSLT;
use XML::LibXML;

my $dispatch;

# uber-trivial XSLT document, as a file
my $xsltfile = "/tmp/foo.xslt";

# here's the actual document inline for
# testing save and alter $xsltfile above

=comment
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:wfs="http://www.opengis.net/wfs">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <WFSLayers>
      <xsl:for-each select="//wfs:FeatureType">
        <wfs_layer>
          <name><xsl:value-of select="wfs:Name"/></name>
          <title><xsl:value-of select="wfs:Title"/></title>
        </wfs_layer>
      </xsl:for-each>
    </WFSLayers>
  </xsl:template>
</xsl:stylesheet>
=cut

my $mapfile = "/tmp/config.map";
# init OWSRequest object
my $req = new MapScript::OWSRequest();

# pick up CGI parameters passed
$req->loadParams();

# init mapfile
my $map = new MapScript::mapObj($mapfile);

# if this is a WFS GetCapabilities request, then intercept
# what is normally returned, process with an XSLT document
# and then return that to the client
if ($req->getValueByName('REQUEST') eq "GetCapabilities" && $req->getValueByName('SERVICE') eq "WFS") {

  # push STDOUT to a buffer and run the incoming request
  my $io = MapScript::msIO_installStdoutToBuffer();
  $dispatch = $map->OWSDispatch($req);

  # at this point, the client's request is sent

  # pull out the HTTP headers
  my $ct = MapScript::msIO_stripStdoutBufferContentType();

  # and then pick up the actual content of the response
  my $content = MapScript::msIO_getStdoutBufferString();

```

```
my $xml = XML::LibXML->new();
my $xslt = XML::LibXSLT->new();

# load XML content
my $source = $xml->parse_string($content);

# load XSLT document
my $style_doc = $xml->parse_file($xsltfile);
my $stylesheet = $xslt->parse_stylesheet($style_doc);

# invoke the XSLT transformation
my $results = $stylesheet->transform($source);
# print out the result (header + content)
print "Content-type: $ct\n\n";
print $stylesheet->output_string($results);
}

# else process as normal
else {
    $dispatch = $map->OWSDispatch($req);
}
```

9.12.4 Java Example

One benefit of redirection of output to a buffer is that it is thread-safe. Several threads in the same process can be actively processing requests and writing their results to distinct output buffers. This Java example, used to test multi-threaded access demonstrates that.

```
import edu.umn.gis.mapscript.mapObj;
import edu.umn.gis.mapscript.OWSRequest;
import edu.umn.gis.mapscript.mapscript;

class WxSTest_thread extends Thread {

    public String      mapName;
    public byte[]      resultBytes;

    public void run() {
        mapObj map = new mapObj(mapName);

        map.setMetaData( "ows_onlineresource", "http://dummy.org/" );

        OWSRequest req = new OWSRequest();

        req.setParameter( "SERVICE", "WMS" );
        req.setParameter( "VERSION", "1.1.0" );
        req.setParameter( "REQUEST", "GetCapabilities" );

        mapscript.msIO_installStdoutToBuffer();

        int owsResult = map.OWSDispatch( req );

        if( owsResult != 0 )
            System.out.println( "OWSDispatch Result (expect 0): " + owsResult );
    }
}
```



```

        resultBytes = mapsript.msIO_getStdoutBufferBytes();
    }
}

public class WxSTest {
    public static void main(String[] args) {
        try {
            WxSTest_thread tt[] = new WxSTest_thread[100];
            int i;
            int expectedLength=0, success = 0, failure=0;

            for( i = 0; i < tt.length; i++ )
            {
                tt[i] = new WxSTest_thread();
                tt[i].mapName = args[0];
            }

            for( i = 0; i < tt.length; i++ )
                tt[i].start();

            for( i = 0; i < tt.length; i++ )
            {
                tt[i].join();
                if( i == 0 )
                {
                    expectedLength = tt[i].resultBytes.length;
                    System.out.println( "Document Length: " + expectedLength + ", expecting somewhere a" );
                }
                else if( expectedLength != tt[i].resultBytes.length )
                {
                    System.out.println( "Document Length:" + tt[i].resultBytes.length + " Expected:" +
                        failure++ );
                }
                else
                    success++;
            }

            System.out.println( "Successes: " + success );
            System.out.println( "Failures: " + failure );

        } catch( Exception e ) {
            e.printStackTrace();
        }
    }
}

```

9.12.5 PHP Example

Most of the same capabilities are accessible in php mapsript. Here is an example displaying a *wms capabilities*.

Example1 : get the capabilities

This is for example what a url could look like :

`http://.../php/ows.php?service=WMS&version=1.1.1&Request=GetCapabilities`

```
<?php
```

```
dl("php_mapscript_4.10.0.dll");

$request = ms_newowsrequestobj();

$request->loadparams();

/*example on how to modify the parameters :
 forcing the version from 1.1.1 to 1.1.0 */
$request->setParameter("VeRsIoN", "1.1.0");

ms_ioinstallstdouttobuffer();

$oMap = ms_newMapobj(".././service/wms.map");

$oMap->owsdispatch($request);

$contenttype = ms_iostripstdoutbuffercontenttype();

$buffer = ms_iogetstdoutbufferstring();

header('Content-type: application/xml');
echo $buffer;

ms_ioresethandlers();

?>
```

Example2 : get the map
This is for example what a url could look like :

```
http://.../php/ows.php?SERVICE=WMS&VeRsIoN=1.1.1&Request=GetMap&
LAYERS=WorldGen_Outline
```

```
<?php

dl("php_mapscript_4.10.0.dll");

$request = ms_newowsrequestobj();

$request->loadparams();

ms_ioinstallstdouttobuffer();

$oMap = ms_newMapobj(".././service/wms.map");

$oMap->owsdispatch($request);

$contenttype = ms_iostripstdoutbuffercontenttype();

if ($contenttype == 'image/png')
    header('Content-type: image/png');

ms_iogetStdoutBufferBytes();

ms_ioresethandlers();
```

?>

9.12.6 Use in Non-CGI Environments (mod_php, etc)

The `loadParams()` call establish parses the cgi environment variables (`QUERY_STRING`, and `REQUEST_METHOD`) into parameters in the `OWSRequest` object. In non-cgi environments, such as when php, python and perl are used as “loaded modules” in Apache, or Java with Tomcat, the `loadParams()` call will not work - in fact in 4.10.x it will terminate the web server instance.

It is necessary in these circumstances for the calling script/application to parse the request url into keyword/value pairs and assign to the `OWSRequest` object by other means, as shown in some of the above examples explicitly setting the request parameters.

9.12.7 Post Processing Capabilities

In the following python example, we process any incoming WxS request, but if it is a `GetCapabilities` request we replace the `Service` section in the capabilities with a section read from a file, that is carefully tailored the way we want.

```
#!/usr/bin/env python

import sys
import elementtree.ElementTree as ET

import mapscript

req = mapscript.OWSRequest()
req.loadParams()

map = mapscript.mapObj( '/u/www/maps/ukpoly/ukpoly.map' )

#
# Handle anything but a GetCapabilities call normally.
#
if req.getValueByName('REQUEST') <> 'GetCapabilities':

    map.OWSDispatch( req )

#
# Do special processing for GetCapabilities
#
else:
    mapscript.msIO_installStdoutToBuffer()

    map.OWSDispatch( req )

    ct = mapscript.msIO_stripStdoutBufferContentType()
    content = mapscript.msIO_getStdoutBufferString()
    mapscript.msIO_resetHandlers()

    # Parse the capabilities.

    tree = ET.fromstring(content)

    # Strip out ordinary Service section, and replace from custom file.
```

```
tree.remove(tree.find('Service'))
tree.insert(0,ET.parse('./Service.xml').getroot())

# Stream out adjusted capabilities.

print 'Content-type: ' + ct
print
print ET.tostring(tree)
```

OPTIMIZATION

10.1 Mapfile

Author David Fawcett

Contact david.fawcett at gmail.com

Revision \$Revision: 8365 \$

Date \$Date: 2008-12-31 07:49:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2007/08/01

Table of Contents

- Mapfile
 - Introduction

10.1.1 Introduction

The contents of a Map File are used by MapServer for configuration, data access, projection, and more. Because the Map File is parsed every time a map image is requested, it is important to think about what you include in the file in order to optimize performance. The optimal Map File is one that doesn't include or reference anything that isn't needed.

1. Projections

There are two ways to define projections in a Map File. You can either use inline projection parameters or specify an EPSG code for that projection. If you use the *EPSG* code method, *Proj.4* looks up the projection parameters in the Proj4 database using the EPSG code as an ID. This database lookup takes significantly more resources than when the projection parameters are defined inline. This lookup takes place for each projection definition using EPSG codes in a Map File.

Projection defined using inline projection parameters

PROJECTION

```
"proj=utm"  
"ellps=GRS80"  
"datum=NAD83"  
"zone=15"
```

```
"units=m"  
"north"  
"no_defs"  
END
```

Projection defined using EPSG Code

```
PROJECTION  
  "init=epsg:26915"  
END
```

Optimization Suggestions

- Use inline projection parameter definitions in place of EPSG codes.
- If you want to use EPSG codes, remove all unneeded projection definition records from the Proj.4 *EPSG* database.

2. Layers

For every layer in a Map File that has a status of ON or DEFAULT, MapServer will load that layer and prepare it for display, even if that layer never gets displayed.

Optimization Suggestions

- Build lean Map Files, only include layers that you plan to use.
- Turn off unnecessary layers; the more layers MapServer is displaying, the more time it takes. Have your opening map view show only the minimum necessary to orient the user, and allow them to turn on additional layers as needed. This is particularly true of remote WMS or very large rasters.
- Related to turning off layers, is turning them on but using `MINSCALEDENOM` and `MAXSCALEDENOM` to determine at what zoomlevels the layer is available. If a map's display is outside of the layer's `MINSCALEDENOM` and `MAXSCALEDENOM` range, then MapServer can skip processing that layer. It also makes for a really cool effect, that the national boundaries magically change to state boundaries.
- If you have a complex application, consider using multiple simple and specific Map Files in place of one large 'do everything' Map File.
- In a similar vein, each class also supports `MINSCALEDENOM` and `MAXSCALEDENOM`. If your dataset has data that are relevant at different zoomlevels, then you may find this a very handy trick. For example, give a `MINSCALEDENOM` of 1:1000000, county roads a `MINSCALEDENOM` of 1:100000, and streets a `MAXSCALEDENOM` of 1:50000. You get the cool effect of new data magically appearing, but you don't have MapServer trying to draw the nation's roads when the entire nation is in view!
- Classes are processed in order, and a feature is assigned to the first class that matches. So try placing the most commonly-used classes at the top of the class list, so MapServer doesn't have to try as many classes before finding a match. For example, if you wanted to highlight the single state of Wyoming, you would probably do this:

```
CLASS  
  EXPRESSION ( '[NAME]' eq 'WY' )  
  STYLE  
    COLOR 255 0 0  
  END  
END  
CLASS  
  STYLE
```

```
COLOR 128 128 128
END
END
```

But it would be a lot more efficient to do this, since 98% of cases will be matched on the first try:

```
CLASS
  EXPRESSION ( '[NAME]' ne 'WY' )
  STYLE
    COLOR 128 128 128
  END
END
CLASS
  STYLE
    COLOR 255 0 0
  END
END
```

- Use *tile indexes* instead of multiple layers.

3. Symbols

When the Map File is loaded, each raster symbol listed in the symbols file is located on the filesystem and loaded.

Optimization Suggestions

- Only include raster symbols in your symbols file if you know that they will be used by your application.

4. Fonts

To load a font, MapServer opens up the fonts.list *FONTSET* file which contains an alias for the font and the path for that font file. If you have a fonts.list file with a long list of fonts, it will take more time for MapServer to locate and load the font that you need.

Optimization Suggestions

- Limit the entries in fonts.list to fonts that you actually use.

10.2 Vector

Author HostGIS

Revision \$Revision: 8365 \$

Date \$Date: 2008-12-31 07:49:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2008/08/08

Table of Contents

- Vector
 - Splitting your data
 - Shapefiles
 - PostGIS
 - Databases in General (PostGIS, Oracle, MySQL)

10.2.1 Splitting your data

If you find yourself making several layers, all of them using the same dataset but filtering to only use some of the records, you could probably do it better. If the criteria are static, one approach is to pre-split the data.

The *ogr2ogr* utility can select on certain features from a datasource, and save them to a new data source. Thus, you can split your dataset into several smaller ones that are already effectively filtered, and remove the `FILTER` statement.

10.2.2 Shapefiles

Use *shptree* to generate a spatial index on your shapefile. This is quick and easy (“*shptree foo.shp*”) and generates a `.qix` file. MapServer will automatically detect an index and use it.

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

Although shapefiles are a very fast data format, *PostGIS* is pretty speedy as well, especially if you use indexes well and have memory to throw at caching.

10.2.3 PostGIS

The single biggest boost to performance is indexing. Make sure that there’s a GIST index on the geometry column, and each record should also have an indexed primary key. If you used *shp2pgsql*, then these statements should create the necessary indexes:

```
ALTER TABLE table ADD PRIMARY KEY (gid);
CREATE INDEX table_the_geom ON table (the_geom) USING GIST;
```

PostgreSQL also supports reorganizing the data in a table, such that it’s physically sorted by the index. This allows PostgreSQL to be much more efficient in reading the indexed data. Use the `CLUSTER` command, e.g.

```
CLUSTER the_geom ON table;
```

Then there are numerous optimizations one can perform on the database server itself, aside from the geospatial component. The easiest is to increase *max_buffers* in the *postgresql.conf* file, which allows PostgreSQL to use more memory for caching. More information can be found at the [PostgreSQL website](#).

10.2.4 Databases in General (PostGIS, Oracle, MySQL)

By default, MapServer opens and closes a new database connection for each database-driven layer in the mapfile. If you have several layers reading from the same database, this doesn’t make a lot of sense. And with some databases (Oracle) establishing connections takes enough time that it can become significant.

Try adding this line to your database layers:

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

This causes MapServer to not close the database connection for each layer until after it has finished processing the mapfile and this may shave a few seconds off of map generation times.

10.3 Raster

Author HostGIS

Revision \$Revision: 8365 \$

Date \$Date: 2008-12-31 07:49:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2008/08/08

Table of Contents

- Raster
 - Overviews
 - Tileindexes and Internal Tiling
 - Image formats
 - Remote WMS

10.3.1 Overviews

TIFF supports the creation of “overviews” within the file, which is basically a downsampled version of the raster data suitable for use at lower resolutions. Use the “gdaladdo” program to add overviews to a TIFF, and MapServer (via GDAL) will automatically choose which downsampled layer to use. Note that overviews significantly increase the disk space required by a TIFF, and in some cases the extra disk reading may offset the performance gained by MapServer not having to resample the image. You’ll just have to try it for yourself and see how it works.

10.3.2 Tileindexes and Internal Tiling

Tiling is mostly effective for cases where one commonly requests only a very small area of the image.

A tileindex is how one creates an on-the-fly mosaic from many rasters. This is described in the *Tile Indexes*. That document describes common cases where a tileindex makes sense. In particular, if you have a very large raster and most requests are for a very small spatial area within it, you may want to consider slicing it and tileindexing it.

As an alternative to slicing and mosaicing, TIFFs do support a concept of internal tiling. Like a tileindex, this is mostly effective when the requests are for a small portion of the raster. Internal tiling is done by adding “-co TILED=YES” to `gdal_translate`, e.g.:

```
gdal_translate -co TILED=YES original.tif tiled.tif
```

10.3.3 Image formats

The TIFF image format is the fastest to “decipher”, but once you get beyond a certain point, the disk reading (since TIFF is very large) may become slow enough to make it worthwhile to consider other image formats.

For TIFFs larger than 1 GB, ECW images tend to render faster than TIFFs, since decompressing the data (CPU and RAM) is faster than reading the uncompressed data (disk). The downside is that ECW is not open-source, and the licensing is often prohibitive.

JPEG2000 is a very slow image format, as is JPEG.

10.3.4 Remote WMS

Remote WMS servers are often slow, especially the public ones such as TerraServer or NASA’s Landsat server. There’s nothing you can do about that, except to reconsider when the remote WMS layer should be used.

For example, there may be a different WMS server (or a different set of imagery, or even vector outline maps) suitable for drawing the countries or states to orient the user. You could then have the WMS layer come on at a certain scale, or have the layer always available but turned off so the user can choose when to turn it on.

See Also:

Raster Data

10.4 FastCGI

Author Frank Warmerdam

Contact warmerdam at pobox.com

Author Howard Butler

Contact hobu.inc at gmail.com

Revision \$Revision: 8295 \$

Date \$Date: 2008-12-26 21:08:04 -0800 (Fri, 26 Dec 2008) \$

Last Updated 2008/07/15

Table of Contents

- FastCGI
 - Introduction
 - Obtaining the necessary software
 - Configuration
 - Common Problems
 - FastCGI on Win32

10.4.1 Introduction

FastCGI is a protocol for keeping cgi-bin style web applications running as a daemon to take advantage of preserving memory caches, and amortizing other high startup costs (like heavy database connections) over many requests.

10.4.2 Obtaining the necessary software

1. There are three pieces to the MapServer FastCGI puzzle. First, you need the actual FastCGI library. This can be downloaded from <http://www.fastcgi.com/>. This library does the usual *configure, make, make install* dance. One added complication is that it installs by default in `/usr/local`, and you might give the *configure* command a `--prefix=/usr` to put it in a location that is already visible to `ldconfig`.
2. Assuming you are running Apache, the next piece you need is `mod_fcgi`. `Mod_fcgi` depends on the version of Apache you are running, so make sure to download the correct fork (Apache 1.3 vs. Apache 2).
3. The third and final piece is to compile MapServer with FastCGI support. This is pretty straightforward, and all you need to do is tell *configure* where the FastCGI library is installed. If you changed the prefix variable as described above, this would be:

```
./configure [other options] --with-fastcgi=/usr
```

With those pieces in place, the MapServer CGI (`mapserv`) should now be FastCGI-enabled. You can verify this by testing it on with the command line:

```
[hobu@kenyon mapserver-4.4.2]# ./mapserv -v
MapServer version 4.4.2 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=FASTCGI INPUT=EPPL7
INPUT=SDE INPUT=ORACLESPATIAL INPUT=OGR INPUT=GDAL
INPUT=SHAPEFILE DEBUG=MSDEBUG
```

10.4.3 Configuration

1. Modify `http.conf` to load the FastCGI module.

```
LoadModule fastcgi_module /usr/lib/apache/1.3/mod_fastcgi.so
```

2. Add an Apache handler for FastCGI applications.

```
AddHandler fastcgi-script fcgi
```

3. Set FastCGI processing information

```
FastCgiConfig -initial-env PROJ_LIB=/usr/local/share/proj
-initial-env LD_LIBRARY_PATH=/usr/local/lib:/usr/local/pgsql/lib:/usr3/pkg3/oracle9/lib
-appConnTimeout 60 -idle-timeout 60 -init-start-delay 1
-minProcesses 2 -maxClassProcesses 20 -startDelay 5
```

4. In your mapfile, set a `PROCESSING` directive to tell FastCGI to cache the connections and layer information on all layers for which connection caching is desired - ie. all slow layers.

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

10.4.4 Common Problems

File permissions

Fedora Core 3 doesn't allow FastCGI to write to the process logs (when you use RedHat's Apache 2 rather than your own). This is described [here](#).

Also, FastCGI needs to write its socket information somewhere. This can be directed with the *FastCgiIpcDir* directive.

10.4.5 FastCGI on Win32

MS4W Users

MS4W (MapServer for Windows) >= version 2.2.2 contains MapServer compiled with FastCGI support. MS4W version >= 2.2.8 also contains the required Apache module (mod_fcgid), and users must follow the [README instructions](#) to setup FastCGI with their application.

Building fcgi-2.4.0

I used libfcgi-2.4.0 for use with Apache2 from <http://www.fastcgi.com>.

Binary IO Patch

It is critical that stdio be in binary mode when PNG and other binary images are written to it. To accomplish this for stdio handled through the FastCGI library, I had to do the following hack to libfcgi/fcgi_stdio.c within the fcgi-2.4.0 distribution.

In FCGI_Accept() made he following change

```
if(isCGI) {
    FCGI_stdin->stdio_stream = stdin;
    FCGI_stdin->fcgx_stream = NULL;
    FCGI_stdout->stdio_stream = stdout;
    FCGI_stdout->fcgx_stream = NULL;
    FCGI_stderr->stdio_stream = stderr;
    FCGI_stderr->fcgx_stream = NULL;

    /* FrankWarmerdam: added so that returning PNG and other binary data
       will still work */
#ifdef _WIN32
    _setmode( _fileno(stdout), _O_BINARY);
    _setmode( _fileno(stdin), _O_BINARY);
#endif

} else {
```

Also, add the following just before the FCGI_Accept() function

```
#ifdef _WIN32
#include <fcntl.h>
#include <io.h>
#endif
```

I'm sure there is a better way of accomplishing this. If you know how, please let me know!

Building libfcgi

The makefile.nt should be fine. Just ensure you have run VCVARS32.BAT (as is needed for building MapServer) and then issue the command:

```
nmake /f makefile.nt
```

Then the .lib and .dll will be in libfcgi/Debug?. Make sure you copy the DLL somewhere appropriate (such as your cgi-bin directory).

Other Issues

1) FastCGI's receive a very limited environment on win32, seemingly even more restricted than normal cgi's started by apache. Make sure that all DLLs required are either in the fastcgi directory or in windowssystem32. Any missing DLLs will result in very cryptic errors in the error_log, including stuff about Overlapping read requests failing or something like that.

2) Make sure you use a libfcgi.dll built against the same runtime library as your mapserv.exe (and possibly libmap_fcgi.dll) or you will suffer a world of pain! Different runtime libraries have different "environ" variables (as well as their own stdio and heaps). You can check that everything is using MSVCRT.DLL (or all using MSVCRTD.DLL) using the MS SDK Dependency Walker.

10.5 Tile Indexes

Author HostGIS

Revision \$Revision: 8365 \$

Date \$Date: 2008-12-31 07:49:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2007/08/03

Table of Contents

- Tile Indexes
 - Introduction
 - What is a tileindex and how do I make one?
 - Using the tileindex in your mapfile
 - Tileindexes may make your map faster

10.5.1 Introduction

An introduction to tileindexes, MapServer's method for doing on-the-fly mosaicing.

10.5.2 What is a tileindex and how do I make one?

A tileindex is a shapefile that ties together several datasets into a single layer. Therefore, you don't need to create separate layers for each piece of imagery or each county's road data; make a tileindex and let MapServer piece the mosaic together on the fly.

Making a tileindex is easy using ‘gdaltindex’ for GDAL data sources (rasters) and ‘ogrindex’ for OGR data sources (vectors). You just run them, specifying the index file to create and the list of data sources to add to the index.

For example, to make a mosaic of several TIFFs:

```
gdaltindex imagery.shp imagery/*.tif
```

And to make a mosaic of vectors:

```
ogrindex streets.shp tiger/CA/*.shp tiger/NV/*.shp
```

Note: ogrindex and gdaltindex **add** the specified files to the index. Sometimes you’ll have to delete the index file to avoid creating duplicate entries.

10.5.3 Using the tileindex in your mapfile

Using a tileindex as a layer is best explained by an example:

```
LAYER
  NAME "Roads"
  STATUS ON
  TYPE LINE
  TILEINDEX "tiger/index.shp"
  TILEITEM "LOCATION"
END
```

There are two items of note here: TILEINDEX and TILEITEM. TILEINDEX is simply the path to the index file, and TILEITEM specifies the field in the shapefile which contains the filenames referenced by the index. The TILEITEM will usually be “LOCATION” unless you specified the *-tileindex* option when running gdaltindex or ogrindex.

Two important notes about the pathnames:

- The path to TILEINDEX follows the same conventions as for any other data source, e.g. using the SHAPEPATH or else being relative to the location of the mapfile.
- The filenames specified on the command line to gdaltindex or ogrindex will be used with the same conventions as well, following the SHAPEPATH or else being relative to the mapfile’s location. I find it very useful to change into the SHAPEPATH directory and then run ogrindex/gdaltindex from there; this ensures that I specify the correct pathnames.

10.5.4 Tileindexes may make your map faster

A tileindex is often a performance boost for two reasons:

- It’s more efficient than having several separate layers.
- MapServer will examine the tileindex to determine which datasets fall into the map’s view, and will open only those datasets. This can result in a great savings for a large dataset, especially for use cases where most of the time only a very small spatial region of the data is being used. (for example, citywide maps with nationwide street imagery)

A tileindex will not help in the case where all/most of the data sources will usually be opened anyway (e.g. street data by county, showing states or larger regions). In that case, it may even result in a decrease in performance, since it may be slower to open 100 files than to open one giant file.

The ideal case for a tileindex is when the most typically requested map areas fall into very few tiles. For example, if you're showing state and larger regions, try fitting your data into state-sized blocks instead of county-sized blocks; and if you're showing cities and counties, go for county-sized blocks.

You'll just have to experiment with it and see what works best for your use cases.

UTILITIES

11.1 legend

11.1.1 Purpose

Creates a legend from a mapfile. Output is either PNG or GIF depending on what version of the GD library used.

11.1.2 Syntax

```
legend [mapfile] [output image]
```

11.2 msencrypt

11.2.1 Purpose

Used to create an encryption key or to encrypt portions of connection strings for use in mapfiles (added in v4.10). Typically you might want to encrypt portions of the CONNECTION parameter for a database connection. The following CONNECTIONTYPES are supported for using this encryption method:

```
OGR  
Oracle Spatial  
PostGIS  
SDE
```

11.2.2 Syntax

To create a new encryption key:

```
msencrypt -keygen [key_filename]
```

To encrypt a string:

```
msencrypt -key [key_filename] [string_to_encrypt]
```

11.2.3 Use in Mapfile

The location of the encryption key can be specified by two mechanisms, either by setting the environment variable `MS_ENCRYPTION_KEY` or using a `CONFIG` directive in the `MAP` object of your mapfile. For example:

```
CONFIG MS_ENCRYPTION_KEY "/path/to/mykey.txt"
```

Use the `{` and `}` characters as delimiters for encrypted strings inside database `CONNECTION`s in your mapfile. For example:

```
CONNECTIONTYPE ORACLESPATIAL
CONNECTION "user/{MIIBugIBAAKBgQCP0Yj+Seh8==}@service"
```

Example

```
LAYER
  NAME "provinces"
  TYPE POLYGON
  CONNECTIONTYPE POSTGIS
  CONNECTION "host=127.0.0.1 dbname=gmap user=postgres password=iluvyou18 port=5432"
  DATA "the_geom FROM province using SRID=42304"
  STATUS DEFAULT
  CLASS
    NAME "Countries"
    COLOR 255 0 0
  END
END
```

Here are the steps to encrypt the password in the above connection:

1. Generate an encryption key (note that this key should not be stored anywhere within your web server's accessible directories):

```
msencrypt -keygen "E:\temp\mykey.txt"
```

And this generated key file might contain something like:

```
2137FEFDB5611448738D9FBB1DC59055
```

1. Encrypt the connection's password using that generated key:

```
msencrypt -key "E:\temp\mykey.txt" "iluvyou18"
```

Which returns the password encrypted, at the commandline (you'll use it in a second):

```
3656026A23DBAFC04C402EDFAB7CE714
```

1. Edit the mapfile to make sure the `'mykey.txt'` can be found, using the `"MS_ENCRYPTION_KEY"` environment variable. The `CONFIG` parameter inside the `MAP` object can be used to set an environment variable inside a mapfile:

```
MAP
...
CONFIG "MS_ENCRYPTION_KEY" "E:/temp/mykey.txt"
...
END #mapfile
```

1. Modify the layer's CONNECTION to use the generated password key, making sure to use the “{}” brackets around the key:

```
CONNECTION "host=127.0.0.1 dbname=gmap user=postgres password={3656026A23DBAFC04C402EDFAB7CE714} port=5432"
```

1. Done! Give your new encrypted mapfile a try with the *shp2img* utility!

11.3 scalebar

11.3.1 Purpose

Creates a scalebar from a mapfile. Output is either PNG or GIF depending on what version of the GD library used.

11.3.2 Syntax

```
scalebar [mapfile] [output image]
```

11.4 shp2img

11.4.1 Purpose

Creates a map image from a mapfile. Output is either PNG or GIF depending on what version of the GD library is used. This is a useful utility to test your mapfile. You can simply provide the path to your mapfile and the name of an output image, and an image should be returned. If an image cannot be created an error will be displayed at the command line that should refer to a line number in the mapfile.

11.4.2 Syntax

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

-p n: pause for n seconds after reading the map
-d layername datavalue: change DATA value for layer

Example #1

```
shp2img -m vector_blank.map -o test.png
```

Result A file named 'test.png' is created, that you can drag into your browser to view.

Example #2

```
shp2img -m gmap75.map -o test2.png -map_debug 3
```

Result A file named 'test2.png' is created, and layer draw speeds are returned such as:

```
msDrawRasterLayerLow(bathymetry): entering
msDrawMap(): Layer 0 (bathymetry), 0.601s
msDrawMap(): Layer 3 (drain_fn), 0.200s
msDrawMap(): Layer 4 (drainage), 0.300s
msDrawMap(): Layer 5 (prov_bound), 0.191s
msDrawMap(): Layer 6 (fedlimit), 0.030s
msDrawMap(): Layer 9 (popplace), 0.080s
msDrawMap(): Drawing Label Cache, 0.300s
msDrawMap() total time: 1.702s
msSaveImage() total time: 0.040s
```

Example #3

```
shp2img -m gmap75.map -o test3.png -all_debug 3
```

Result A file named 'test3.png' is created, layer draw speeds are returned, and some warnings that index qix files are not found, such as:

```
msLoadMap(): 0.671s
msDrawRasterLayerLow(bathymetry): entering.
msDrawGDAL(): src=72,417,3077,2308, dst=0,0,400,300
msDrawGDAL(): red,green,blue,alpha bands = 1,0,0,0
msDrawMap(): Layer 0 (bathymetry), 0.090s
msSearchDiskTree(): Search returned no results. Unable to open spatial index
for D:\ms4w\apps\gmap\htdocs\..\data\drain_fn.qix. In most cases you can
safely ignore this message, otherwise check file names and permissions.
msDrawMap(): Layer 3 (drain_fn), 0.010s
msDrawMap(): Layer 4 (drainage), 0.050s
msSearchDiskTree(): Search returned no results. Unable to open spatial index
for D:\ms4w\apps\gmap\htdocs\..\data\province.qix. In most cases you can
safely ignore this message, otherwise check file names and permissions.
msDrawMap(): Layer 5 (prov_bound), 0.030s
msSearchDiskTree(): Search returned no results. Unable to open spatial index
for D:\ms4w\apps\gmap\htdocs\..\data\fedlimit.qix. In most cases you can
safely ignore this message, otherwise check file names and permissions.
msDrawMap(): Layer 6 (fedlimit), 0.010s
```

```
msDrawMap(): Layer 9 (popplace), 0.010s
msDrawMap(): Drawing Label Cache, 0.201s
msDrawMap() total time: 0.401s
msSaveImage() total time: 0.010s
shp2img total time: 1.082s
```

11.5 shptree

11.5.1 Purpose

Creates a quadtree-based spatial index for a shapefile. The default tree depth is calculated so that each tree node (quadtree cell) contains 8 shapes. Do not use the default with point files, a value between 6 and 10 seems to work ok. Your millage may vary and you'll need to do some experimenting.

The [shptree wiki page](#) may also contain information on this utility.

11.5.2 Description

This utility is a must for any MapServer application that uses shapefiles. Shptree creates a spatial index of your shapefile, using a quadtree method. This means that MapServer will use this index to quickly find the appropriate shapes to draw. It creates a file of the same name as your shapefile, with a *.qix* file extension. The quadtree method breaks the file into 4 quadrants, recursively until only a few shapes are contained in each quadrant. This minimum number can be set with the `<depth>` parameter of the command.

11.5.3 Syntax

```
shptree <shpfile> [<depth>] [<index_format>]
Where:
<shpfile> is the name of the .shp file to index.
<depth> (optional) is the maximum depth of the index
to create, default is 0 meaning that shptree
will calculate a reasonable default depth.
<index_format> (optional) is one of:
NL: LSB byte order, using new index format
NM: MSB byte order, using new index format
The following old format options are deprecated
N: Native byte order
L: LSB (intel) byte order
M: MSB byte order
The default index_format on this system is: NL
```

Example

```
shptree us_states.shp
creating index of new LSB format
```

Result A file named 'us_states.qix' is created in the same location. (note that you can use the `shptreevis` utility, described next, to view the actual quadtree quadrants that are used by MapServer in this qix file)

11.5.4 Mapfile Notes

Shapefiles are native to MapServer, and therefore do not require the *.shp* extension in the DATA path of the LAYER. In fact, in order for MapServer to use the *.qix* extension you MUST NOT specify the extension, for example:

LAYER

```
...  
  DATA "us_states"    #MapServer will search for us_states.qix and will use it  
  ...
```

END

LAYER

```
...  
  DATA "us_states.shp" #MapServer will search for us_states.shp.qix and won't find it  
  ...
```

END

Note: As of MapServer 5.2 the *qix* will be used even when the *.shp* extension is specified

11.6 shptreevis

11.6.1 Purpose

This utility can be used to view the quadtree quadrants that are part of a *.qix* file (that was created with the *shptree* utility).

11.6.2 Syntax

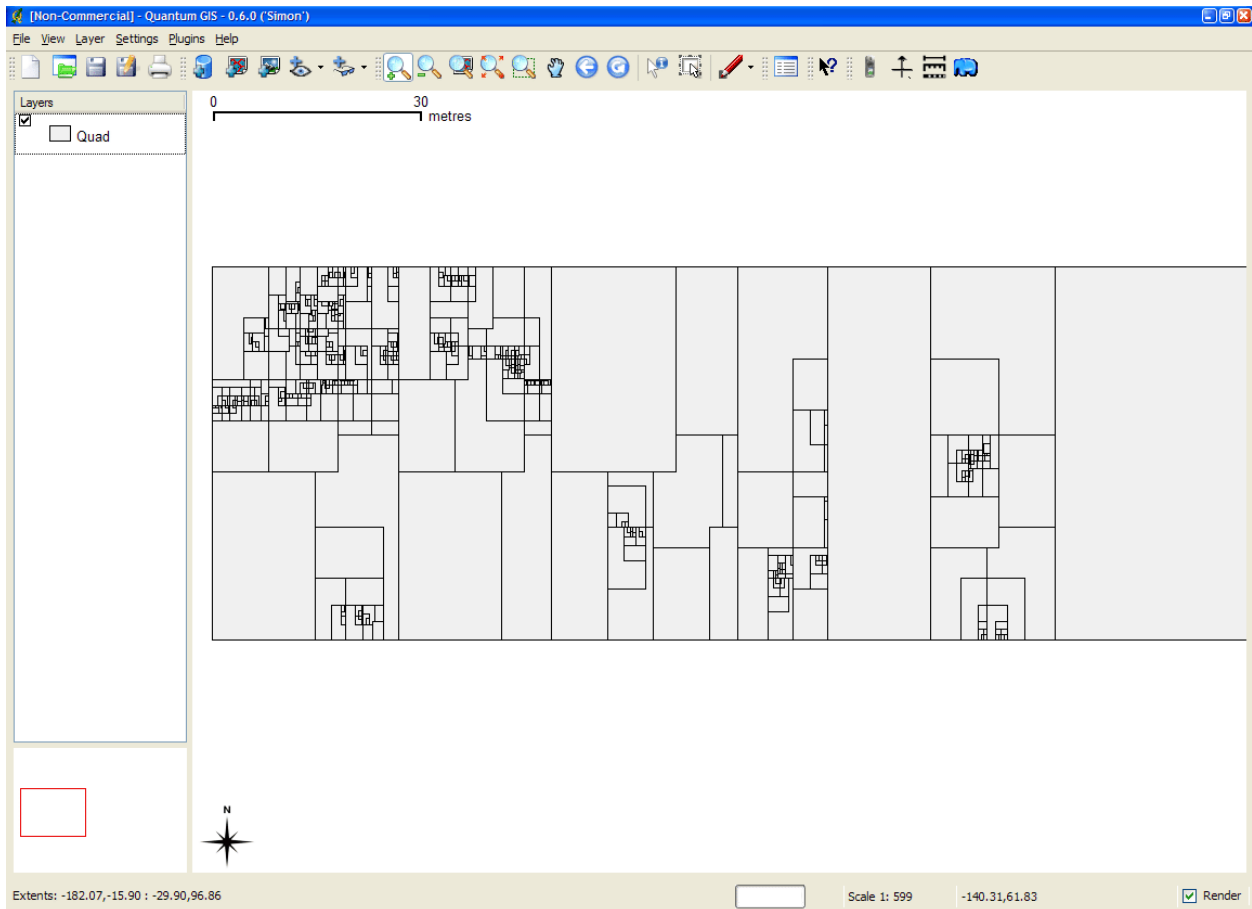
```
shptreevis shapefile new_shapefile
```

Example

```
shptreevis us_states.shp quad.shp  
This new LSB index supports a shapefile with 2895 shapes, 10 depth
```

Result A shapefile named 'quad.shp' is created. You can now view this shapefile in a desktop GIS (such as [QGIS](#) for example) to see the quadtrees that were created with the *shptree* command, as shown below.

Figure: shptreevis result displayed in QGIS



11.7 sortshp

Purpose Sorts a shapefile based on a single column in ascending or descending order. Supports INTEGER, DOUBLE and STRING column types. Useful for prioritizing shapes for rendering and/or labeling.

Description The idea here is that if you know that you need to display a certain attribute classed by a certain value, it will be faster for MapServer to access that value if it is at the beginning of the attribute file.

Syntax `sortshp [infile] [outfile] [item] [ascending|descending]`

Example This example uses a roads file ('roads_ugl') that has a field with road classes in integer format ('class1').

```
sortshp roads_ugl roads-sort class1 ascending
```

Result A new shapefile named 'roads-sort.shp' is created with shapes sorted in ascending order, according to the values in the 'class1' field, as shown below.

Figure1: Attributes Before sortshp

	median	class	class1	sign
1	2	Urban Principa	2	
2	0	Urban Principa	2	
3	2		4	42
4	2		4	42
5	2		4	
6	2	Urban Principa	3	28
7	2	Urban Principa	3	42
8	2		4	14
9	0	Urban Principa	2	
10	2		4	
11	2		4	
12	2	Urban Principa	2	51
13	2	Urban Principa	2	
14	2		2	169

Figure2: Attributes After sortshp

	median	class	class1	sign	
1		2	Urban Interstat	1	496
2		2	Urban Interstat	1	496
3		2	Urban Interstat	1	94
4		2	Urban Interstat	1	94
5		2	Urban Interstat	1	475
6		1	Rural Interstate	1	96
7		1	Rural Interstate	1	96
8		2	Urban Interstat	1	475
9		1	Urban Interstat	1	475
10		1	Urban Interstat	1	696
11		2	Urban Interstat	1	196
12		2	Urban Interstat	1	196
13		2	Urban Interstat	1	35E
14		2	Urban Interstat	1	35F

11.8 sym2img

11.8.1 Purpose

Creates a graphic dump of a symbol file. Output is either PNG or GIF depending on what version of the GD library used. (this utility is not currently included in pre-compiled packages, due to issues mentioned in [bug#506](#))

11.8.2 Syntax

```
sym2img [symbolfile] [outfile]
```

11.9 tile4ms

11.9.1 Purpose

Creates a tile index shapefile for use with MapServer's TILEINDEX feature. The program creates a shapefile of rectangles from extents of all the shapefiles listed in [metafile] (one shapefile name per line) and the associated DBF with the filename for each shape tile in a column called LOCATION as required by mapserv.

11.9.2 Description

This utility creates a shapefile containing the MBR (minimum bounding rectangle) of all shapes in the files provided, which can then be used in the LAYER object's TILEINDEX parameter of the mapfile. The new file created with this command is used by MapServer to only load the files associated with that extent (or tile).

11.9.3 Syntax

```
tile4ms <meta-file> <tile-file> [-tile-path-only]
<meta-file>      INPUT  file containing list of shapefile names
                  (complete paths 255 chars max, no extension)
<tile-file>      OUTPUT shape file of extent rectangles and names
                  of tiles in <tile-file>.dbf
-tile-path-only  Optional flag.  If specified then only the path to the
                  shape files will be stored in the LOCATION field
                  instead of storing the full filename.
```

11.9.4 Short Example

create tileindex.shp for all tiles under the /path/to/data directory:

```
<on Unix>

cd /path/to/data
find . -name "/*.shp" -print > metafile.txt
tile4ms metafile.txt tileindex

<on Windows>

dir /b /s *.shp > metafile.txt
tile4ms metafile.txt tileindex
```

11.9.5 Long Example

This example uses TIGER Census data, where the data contains files divided up by county (in fact there are over 3200 counties, a very large dataset indeed). In this example we will show how to display all lakes for the state of Minnesota. (note that here we have already converted the TIGER data into shapefile format, but you could keep the

data in TIGER format and use the ogrindex utility instead) The TIGER Census data for Minnesota is made up of 87 different counties, each containing its own lakes file ('wp.shp').

1. We need to create the 'meta-file' for the tile4ms command. This is a text file of the paths to all 'wp.shp' files for the MN state. To create this file we can use a few simple commands:

```
DOS: dir wp.shp /b /s > wp_list.txt
      (this includes full paths to the data, you might want to edit the txt
      file to remove the full path)

UNIX: find -name *wp.shp -print > wp_list.txt
```

The newly created file might look like the following (after removing the full path):

```
001\wp.shp
003\wp.shp
005\wp.shp
007\wp.shp
009\wp.shp
011\wp.shp
013\wp.shp
015\wp.shp
017\wp.shp
019\wp.shp
. . .
```

1. Execute the tile4ms command with the newly created meta-file to create the index file:

```
tile4ms wp_list.txt index
      Processed 87 of 87 files
```

1. A new file named 'index.shp' is created. This is the index file with the MBRs of all 'wp.shp' files for the entire state, as shown in Figure1. The attribute table of this file contains a field named 'LOCATION', that contains the path to each 'wp.shp file', as shown in Figure2.

Figure 1: Index file created by tile4ms utility

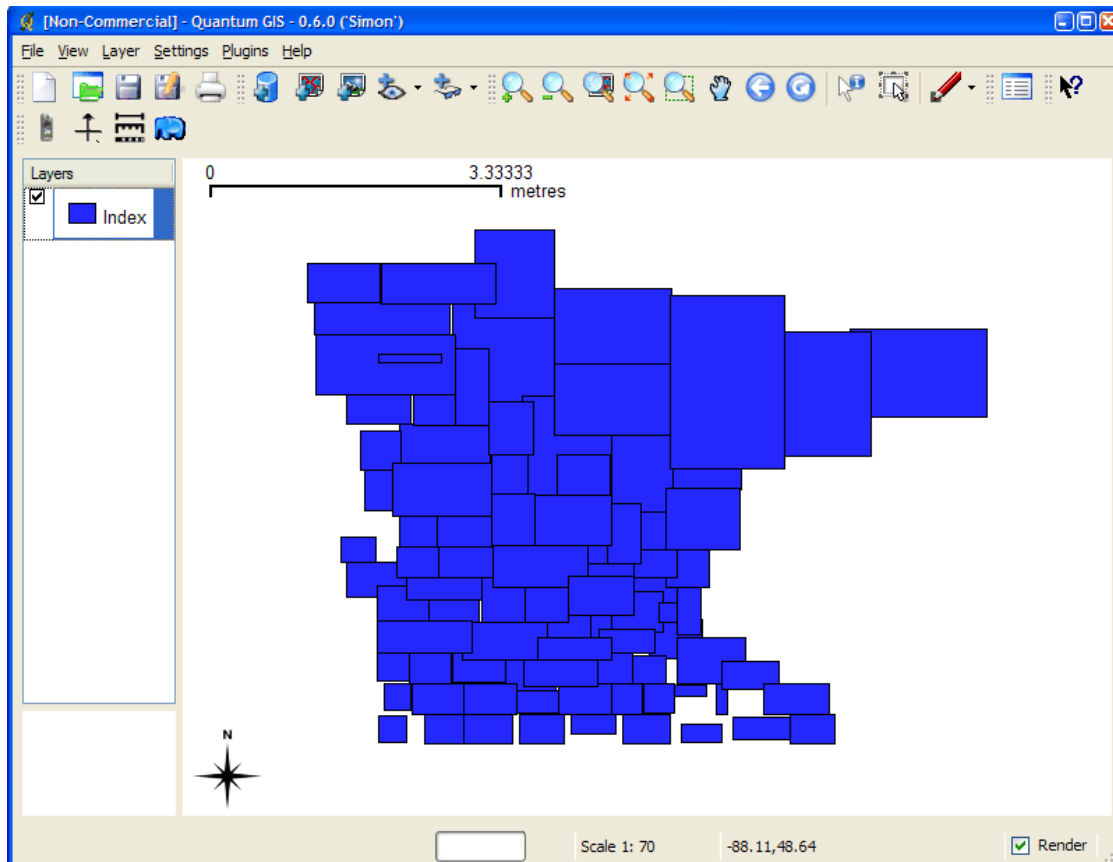
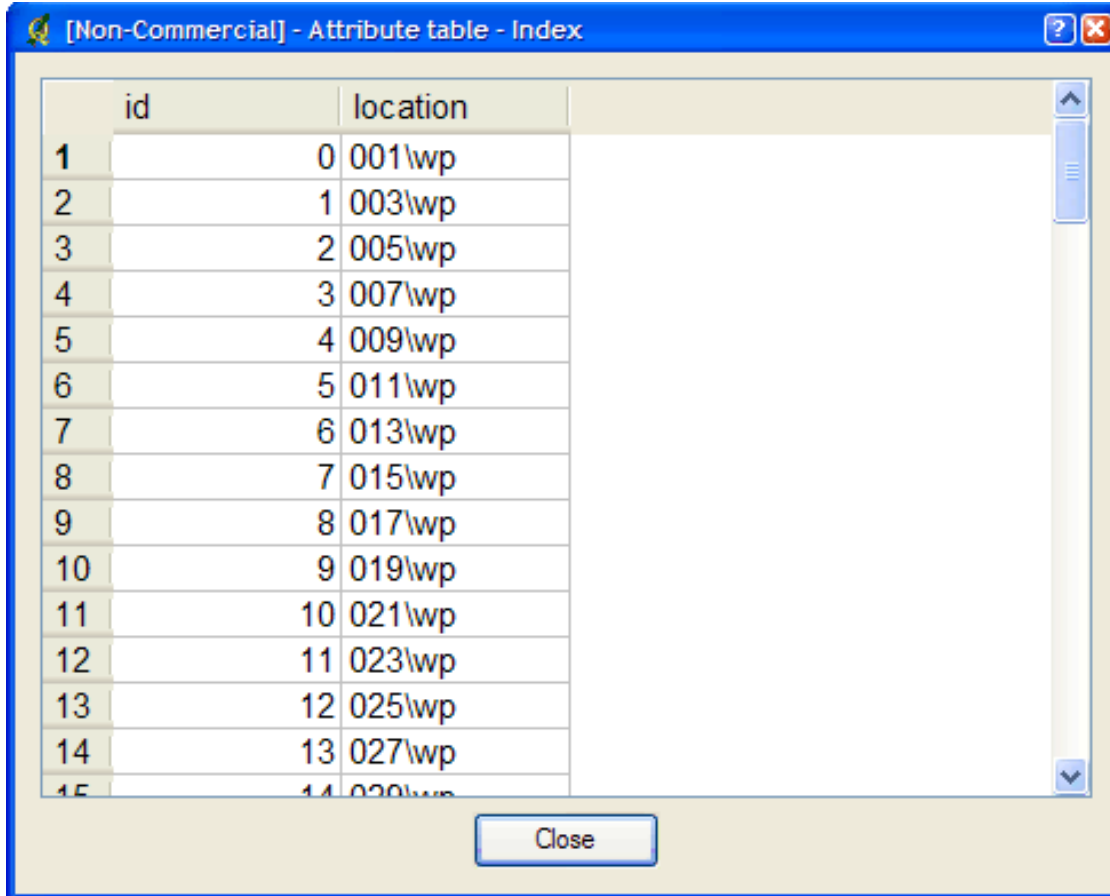


Figure 2: Attributes of index file created by tile4ms utility



	id	location
1	0	001\wp
2	1	003\wp
3	2	005\wp
4	3	007\wp
5	4	009\wp
6	5	011\wp
7	6	013\wp
8	7	015\wp
9	8	017\wp
10	9	019\wp
11	10	021\wp
12	11	023\wp
13	12	025\wp
14	13	027\wp
15	14	029\wp

1. **The final step is to use this in your mapfile.**
 - LAYER object's TILEINDEX - must point to the location of the index file
 - LAYER object's TILEITEM - specify the name of the field in the index file containing the paths (default is 'location')
 - do not need to use the LAYER's DATA parameter

For example:

```
LAYER
  NAME 'mn-lakes'
  STATUS ON
  TILEINDEX "index"
  TILEITEM "location"
  TYPE POLYGON
  CLASS
    NAME "mn-lakes"
    STYLE
      COLOR 0 0 255
    END
  END
END
```

When you view the layer in a MapServer application, you will notice that when you are zoomed into a small area of the state only those lakes layers are loaded, which speeds up the application.

11.10 Batch Scripting

If you need to run the utilities on multiple files/folders, here are some commands that will help you:

11.10.1 Windows

type the following at the command prompt:

```
for %f in (*.shp) do shptree %f
```

or to run recursively (throughout all subfolders):

```
for /R %f in (*.shp) do shptree %f
```

11.10.2 Linux

```
find /path/to/data -name "*.shp" -exec shptree {} \;
```

11.11 File Management

11.11.1 File Placement

MapServer requires a number of different files to execute. Except for graphics that are referenced in output templates (i.e. web pages) none of the data or configuration files need be accessible via a web server. File naming for MapServer follows two rules:

1. Files may be given using their full system path. *or*
2. Files may be given using a relative path where the path is relative to the location of the file they are being referenced from.

So, for files referenced in the Mapfile they can be given relative to the location of the Mapfile. Same holds true for symbol sets and font sets.

11.11.2 Temporary Files

MapServer also can produce a number of files (i.e. maps, legends, scalebars, etc...). These files **must** be accessible using a web server. To accomplish this MapServer creates these files in a scratch directory. The location of that directory is given using the `IMAGEPATH` and `IMAGEURL` parameters in the web section of a Mapfile. The scratch directory must be writable by the user that the web server runs under, usually *nobody*. It is recommended for security reasons that the web user own the scratch directory rather than making it world writable. The scratch area will need to be cleaned periodically. On busy sites this may need to happen several times an hour. Here's an example shell script that could be run using *cron*:

```
#!/bin/csh
```

```
find /usr/local/www/docs/tmp -follow -name "*.gif" -exec rm {} \;
```

Windows

The following *.bat* file can be used in 'Scheduled Tasks' to remove these temporary images daily:

```
REM this script deletes the contents of the ms_tmp directory
REM (i.e. the MapServer-created gifs)

cd D:\ms4w\tmp\ms_tmp
echo Y | del *.*
```


Date 2008/09/09

Author Daniel Morissette

Contact dmorissette at mapgears.com

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Frank Koormann

12.1 MapServer CGI Introduction

12.1.1 Notes

- Variable names are not case sensitive.
- In cases where multiple values are associated with a variable (eg. mapext), the values must be separated by spaces (or their escaped equivalents for GET requests).
- Variable contents are checked for appropriate data types and magnitude as they are loaded.
- Any CGI Variable not listed below is simply stored and can be referenced within a template file.

12.1.2 Changes

From MapServer version 4.x to version 5.x

- Modifying map parameters through a URL has changed to allow for chunks of a mapfile to be modified at once. The syntax has changed accordingly, so please see the *Changing map file parameters via a form or a URL* section.

From MapServer version 3.x to version 4.x

- New way to perform attribute queries: No longer do you set a layer filter, but rather you pass a query string (and optionally an query item) to the query function. To do this 2 new CGI parameters were added to MapServer: QSTRING and QITEM.
- SAVEMAP is switched off: The SAVEMAP functionality is considered insecure, since the saved files are accessible by everyone.

- TEMPLATE has been removed, since the map_web_template syntax can be used to alter a template file. Simplifies security maintenance by only having to deal with this option in a single place. Note that the TEMPLATEPATTERN of the mapfile has to be used to enable this feature.

12.2 mapserv

The CGI interface can be tested at the commandline by using the “QUERY_STRING” switch, such as:

```
mapserv "QUERY_STRING=map=/ms4w/apps/gmap/htdocs/gmap75.map&mode=map"
```

To suppress the HTTP headers, you can use the “-nh” switch, such as:

```
mapserv -nh "QUERY_STRING=map=/ms4w/apps/gmap/htdocs/gmap75.map&mode=map"
```

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

```
mapserv -nh "QUERY_STRING=map=/ms4w/apps/gmap/htdocs/gmap75.map&mode=map" > test.png
```

12.3 Map Context Files

12.3.1 Support for Local Map Context Files

There is a CGI parameter called CONTEXT that is used to specify a local context file. The user can then use MapServer to request a map using the following syntax:

```
http://localhost/mapserver.cgi?MODE=map&MAP=/path/to/mapfile.map&CONTEXT=
/path/to/contextfile.xml&LAYERS=layer_name1 layer_name2
```

Note: All layers created from a context file have their status set to ON. To be able to display layers, the user needs to add the LAYERS argument in the URL.

12.3.2 Support for Context Files Accessed Through a URL

The syntax of using a web accessible context file would be similar to accessing a local context file:

```
http://localhost/mapserver.cgi?MODE=map&MAP=/path/to/mapfile.map&CONTEXT=
http://URL/path/to/contextfile.xml&LAYERS=layer_name1 layer_name2
```

Due to security concerns loading a file from a URL is disabled by default. To enable this functionality, the user needs to set a CONFIG parameter called CGI_CONTEXT_URL in the default map file that will allow this functionality. Here is an example of a map file with the CONFIG parameter:

```
# Start of map file
NAME DEMO
STATUS ON
SIZE 400 300
EXTENT -2200000 -712631 3072800 3840000
UNITS METERS
IMAGECOLOR 255 255 255
IMAGETYPE png
```

```
CONFIG "CGI_CONTEXT_URL" "1"
...
```

12.3.3 Default Map File

To smoothly run a MapServer CGI application with a Map Context, the application administrator needs to provide a default map file with at least the basic required parameters that will be used with the Context file. This default map file can contain as little information as the imagepath and imageurl or contain a list of layers. Information coming from the context (e.g.: layers, width, height, etc.) would either be appended or will replace values found in the map file.

Here is an example of a default map file containing the minimum required parameters:

```
NAME CGI-CONTEXT-DEMO
STATUS ON
SIZE 400 300
EXTENT -2200000 -712631 3072800 3840000
UNITS METERS
IMAGECOLOR 255 255 255
IMAGETYPE png
#
# Start of web interface definition
#
WEB
  MINSCALE 2000000
  MAXSCALE 50000000
#
# On Windows systems, /tmp and /tmp/ms_tmp/ should be created at the root
# of the drive where the .MAP file resides.
#
  IMAGEPATH "/ms4w/tmp/ms_tmp/"
  IMAGEURL "/ms_tmp/"
END
END # Map File
```

12.4 MapServer CGI Controls

12.4.1 Variables

BUFFER [distance] A distance, in the same coordinate system as the map file, used in conjunction with MAPXY to create a new map extent.

CONTEXT [filename] Path to a context file. Path is relative to the map file to be used, or can also be a URL path (See the section “Map Context Support Through CGI” below for more details).

ICON [layername],[classindex] Used in MODE=legendicon to generate a legend icon for a layer. The class index value is optional and defaults to 0.

ID [id-string] By default MapServer generates a uniq session id based on system time and process id. This parameter overwrites the default.

IMG The name associated with the inline map image used to record user clicks. What actually is passed are two variables, img.x and img.y.

For the CGI Applications this is an essential variable, see the examples for sample usage.

IMGBOX [x1] [y1] [x2] [y2] Coordinates (in pixels) of a box drag in the image. Most often used in conjunction with Java based front ends to the MapServer.

IMGEXT [minx] [miny] [maxx] [maxy] The spatial extent of the existing inline image, that is, the image the users can see in their browser.

IMGSHAPE [x1 y1 x2 y2 x3 y3 ...] Shape given in image coordinates.

An arbitrary polygon shape to be used for query purposes. Multiple instances simply add parts to the shape so it is possible to construct a shape with holes. Used with modes NQUERY and NQUERYMAP.

IMGSIZE [cols] [rows] The size (in pixels) of the exiting inline image.

IMGXY [x] [y] Coordinates (in pixels) of a single mouse click. Used most often in conjunction with Java based front ends to the MapServer.

LAYER [name] The name of a layer as it appears in the map file. Sending mapserv a layer name sets that layer's STATUS to ON.

LAYERS [name name ...] The names of the layers to be turned on. Layer names must be seperated by spaces.

Version 4.4 and above: passing 'LAYERS=all' will automatically turn on all layers.

MAP [filename] Path, relative to the CGI directory, of the map file to be used.

MAPEXT [minx] [miny] [maxx] [maxy] , **MAPEXT** (shape) The spatial extent of the map to be created.

Can be set to shape as an alternative option. In this case mapextent is set to the extent of a selected shape. Used with queries.

MAPSIZE [cols] [rows] The size (in pixels) of the image to be created. Useful for allowing users to change the resolution of the output map dynamically.

MAPSHAPE [x1 y1 x2 y2 x3 y3 ...] Shape in map coordinates.

An arbitrary polygon shape to be used for query purposes. Multiple instances simply add parts to the shape so it is possible to construct a shape with holes. Used with modes NQUERY and NQUERYMAP.

MAPXY [x] [y] , **MAPXY** (shape) A point, in the same coordinate system as the shapefiles, to be used in conjunction with a buffer or a scale to construct a map extent.

Can be set to shape as an alternative option. In this case mapextent is set to the extent of a selected shape. Used with queries.

MINX | **MINY** | **MAXX** | **MAXY** [number] Minimum/Maximum x/y coordinate of the spatial extent for a new map/query. This set of parameters are the pieces of MAPEXT.

MODE [value] Mode of operation. The following options are supported (note that all of the query modes also support map-only modes, e.g. ITEMQUERYMAP, which for brevity are not all listed):

BROWSE Fully interactive interface where maps (and interactive pages) are created. This is the default mode.

QUERY A spatial search (finds closest) is triggered by a click in a map.

NQUERY A spatial search (finds all) is triggered by a click in a map or by user-define selection box.

ITEMQUERY A text search of attribute data is triggered using a QSTRING. Returns 1st match.

ITEMNQUERY A text search of attribute data is triggered using a QSTRING. Returns all matches.

FEATUREQUERY A spatial search that uses one feature from SLAYER to query other layers.

FEATURENQUERY A spatial search that uses multiple features from SLAYER to query other layers.

ITEMFEATUREQUERY A text search of attribute data is triggered using a QSTRING. Returns first match. Layer to be searched is defined using slayer parameter. The results of this search are applied to other searchable layers (which can be limited using the QLAYER parameter).

ITEMFEATURENQUERY A text search of attribute data is triggered using a QSTRING. Returns all matches. Layer to be searched is defined using slayer parameter. The results of this search are applied to other searchable layers (which can be limited using the QLAYER parameter).

LEGENDICON A legend icon is returned. The ICON parameter must also be used to specify the layername and a class index. Class index value is optional and defaults to 0. For example:

```
mapserv.exe?map=/ms4w/apps/gmap/htdocs/gmap75.map&MODE=legendicon&ICON=popplace,0
```

MAP The created map is returned. Used within an tag.

REFERENCE The created reference map is returned. Used within an tag.

SCALEBAR The created scalebar is returned. Used within an tag.

LEGEND The created legend is returned. Used within an tag.

ZOOMIN Switch to mode BROWSE with ZOOMDIR=1

ZOOMOUT Switch to mode BROWSE with ZOOMDIR=-1

INDEXQUERY Looks up a feature based on the values of SHAPEINDEX and TILEINDEX parameters. SHAPEINDEX is required, TILEINDEX is optional and is only used with tiled shapefile layers.

COORDINATE To be clarified.

QLAYER [name] Query layer. The name of the layer to be queried as it appears in the map file. If not specified then all layers are searched in turn.

QITEM [name] (optional) The name of an attribute in a layer attribute table to query on. The parameter is optional and used in conjunction with the QSTRING for attribute queries.

QSTRING [expression] Attribute queries: Query string passed to the query function.

QUERYFILE [filename] Used with BROWSE or NQUERY mode. This option identifies a query file to load before any regular processing. In BROWSE mode this result in a query map being produced instead of a regular map. This is useful when you want to hilite a feature while still in a pan/zoom mode. In NQUERY mode you'd gain access to any of the templates used in normally presenting the query, so you have access to query maps AND attribute information. See the SAVEQUERY option.

REF The name associated with the inline reference map image used to record user clicks. What actually is passed are two variables, ref.x and ref.y.

For the CGI Applications this is an essential variable when reference maps are used, see the examples for sample usage.

REFXY [x] [y] Coordinates (in pixels) of a single mouse click in the reference image. Used in conjunction with Java based front ends to the MapServer.

SAVEQUERY When used with any of the query modes this tells the MapServer to save the query results to a temporary file for use in subsequent operations (see QUERYFILE). Useful for making queries persistent.

SCALEDENOM [number] Scale to create a new map at. Used with mapxy. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000. Implemented in MapServer 5.0, to replace the deprecated SCALE parameter.

SCALE [number] - deprecated Since MapServer 5.0 the proper parameter to use is SCALEDENOM instead. The deprecated SCALE is the scale to create a new map at. Used with mapxy. Scale is given as the denominator of the actual scale fraction, for example for a map at a scale of 1:24,000 use 24000.

SEARCHMAP It is possible to do pan/zoom interfaces using querymaps. In these cases you will likely want information about the contents of the new map rather than the previous map which is the normal way queries

work. When searchmap is specified the new map is created and it's extent is used to query layers. Useful with NQUERY mode only.

SHAPEINDEX [index] Used for index queries (in conjunction with INDEXQUERY).

SLAYER [name] Select layer. The name of the layer to be used for any of the feature (i.e. staged) query modes. The select layer must be a polygon layer. The selection feature(s) are available for presentation to the user.

TILEINDEX [index] Used for index queries (in conjunction with INDEXQUERY), used with tiled shapefile layers.

ZOOM [number] Zoom scaling to apply to the creation of the new map. Values greater than 0 resulting in zooming in, 0 is a pan, and values less than zero are for zooming out. A value of 2 means "zoom in twice".

ZOOM can be used as a shortcut for the combination ZOOMDIR/ZOOMSIZE. The zoom is limited by the MINZOOM/MAXZOOM settings compiled into the MapServer (-25/25) by default.

ZOOMDIR [1 | 0 | -1] Direction to zoom. See above.

ZOOMSIZE [number] Absolute magnitude of a zoom. Used with ZOOMDIR.

ZOOMDIR is limited to MAXZOOM compiled into the MapServer (25 by default).

12.4.2 Changing map file parameters via a form or a URL

Beginning with version 3.3 it is possible to change virtually any map file value from a form or a URL. The syntax for this is fairly straight forward, and depends on what version of MapServer you are using. One potentially very powerful use of this ability to change mapfile parameters through a URL involves changing class expressions on-the-fly. Try it out.

Using MapServer version >= 5

Previous versions of the MapServer CGI program allowed certain parameters to be changed via a URL using a cumbersome syntax such as `map_layer_0_class_0_color=255+0+0` which changes the color in one classObj. So, in the past you have to change parameters one-at-a-time. Now you can pass chunks of mapfiles (with security restrictions) to the CGI interface. The `map_object` notation is still necessary to identify which object you want to modify but you can change multiple properties at one time. Note that you can use either a `'_'` or a `'.'` to separate identifiers.

Example 1, changing a scalebar object:

```
...&map.scalebar=UNITS+MILES+COLOR+121+121+121+SIZE+300+2&...
```

Example 2, changing a presentation style:

```
...&map.layer[lakes].class[0].style[0]=SYMBOL+crosshatch+COLOR+151+51+151+SIZE+15&...
```

Example 3, creating a new feature

```
...&map_layer[3]=FEATURE+POINTS+500000+1000000+END+TEXT+'A+test+point'+END&...
```

Example 4, set multiple web object parameters

```
...&map_web=imagepath+/ms4w/tmp/ms_tmp/+imageurl+/ms_tmp/
```

Example 5, set the map size

```
...&map_size=800+400
```

The variable identifies an object uniquely (by name or index in the case of layerObj's and classObj's). The value is a snippet of a mapfile. You cannot create new objects other than inline features at this point.

Using MapServer version < 5

For MapServer version < 5, any value can be expressed using the hierarchy used in a map file. A map contains a layer, which contains a class, which contains a label, which has a color. This hierarchy is expressed as a sequence of MapServer keywords separated by underscores. For example to change the color of a layer called "lakes" with only one class defined you would use a form variable named "map_lakes_class_color" and could assign it a color like "0 0 255". Layers can be referenced by index (i.e. map_layer_0...) or by name as shown above. Layer classes are referenced by index value (i.e. map_layer_0_class_2). If there is only 1 class for a layer then the index should be omitted. These variables must always begin with the sequence "map_". Values assigned must conform to the syntax of a map file.

It is also possible to define inline features using this mechanism. This is the only case where you can add on to the map file. You can edit/change layer parameters but you cannot create a new layer. With inline features you have to first create a feature and then build upon it, however, the layer the feature belongs to must exist. Here's a snippet from a GET request that adds a feature to a webuser layer:

```
. . . &map_webuser_feature=new&map_webuser_feature_points=12345.6789+12345.6789&map_webuser_feature_t
```

The "map_webuser_feature=new" creates a new feature for the webuser layer. All subsequent calls to the feature object for that layer will modify the new feature. You can repeat the process to create additional features. This is really intended for very small (point, rectangle) amounts of data.

12.4.3 ROSA-Applet Controls

note: Active development and maintenance of the ROSA Applet has stopped

The ROSA Applet parameters were added to the CGI MapServer in version 3.6. This Java Applet provides a more intuitive user interface to MapServer. The MapTools site provides detailed information on the ROSA Applet.

The parameters can also be used by other interfaces/tools, if set to the right values. Please note that the two parameters have to be handed over to the CGI application in the order identified below.

INPUT_TYPE (auto_rect | auto_point) The INPUT_TYPE parameter is needed to identify if the coordinates handed over to the mapserver have to be interpreted as rectangular or point data.

INPUT_COORD [minx,miny;maxx,maxy] The ROSA-Applet always fills the pair of coordinates. In case of a point (input_type=auto_point) min and max coordinate are equal (MapServer uses the min value).

12.5 Run-time Substitution

Author Steve Lime

Contact steve.lime at DNR.STATE.MN.US

Revision \$Revision: 9263 \$

Date \$Date: 2009-08-21 07:34:47 -0700 (Fri, 21 Aug 2009) \$

Last Updated 2005/12/12

Table of Contents

- Run-time Substitution
 - Introduction
 - Basic Example
 - Parameters Supported

12.5.1 Introduction

Run-time substitution for the MapServer CGI has been around since version 4.0 and it's use has continued to expand. In short, it allows you to alter portions of a mapfile based on data passed via a CGI request. This functionality is only available via the standard CGI application. Within MapScript this is easy to do since the developer has complete control over how input is handled.

12.5.2 Basic Example

Let's say you'd like the user to dynamically set a portion of an expression so they could highlight a certain land cover class, and you have a form element (called `ctype`) that allows them to choose between: forest, water, wetland and developed. You could then set up a layer like so:

```
LAYER
  NAME 'covertypes'
  ...
  CLASSITEM 'type'
  CLASS # highlighted presentation
    EXPRESSION '%ctype%'
    ...
  END
  CLASS # default presentation
    ...
  END
END
```

When a request is processed the value for `ctype` is substituted for the string `%ctype%` and the mapfile is processed as normal. If no `ctype` is passed in the EXPRESSION will never be true so it doesn't really hurt anything except for a slight performance hit. Often you would set a default class to draw features that don't match, but that is not required.

12.5.3 Parameters Supported

Not every mapfile parameter supports run-time substitution and care has been taken to try and support those that make the most sense. Remember, you also can do run-time configuration using the `map_object_property` type syntax detailed elsewhere (todo- add link). Below is a list of properties that do allow run-time substitution (todo- add MapServer version):

- LAYER: DATA (must validate against DATAPATTERN)
- LAYER: TILEINDEX
- LAYER: CONNECTION
- LAYER: FILTER
- CLASS EXPRESSION

FILTERs

You can use runtime substitutions to change values within a FILTER as you go. For example your FILTER could be written like so:

```
FILTER ("multimedia='[multimedia]' and seats >= [nseats] and Sound= '[sound]')
```

Then (assuming you're using the CGI interface) you could pass in variables named multimedia, nseats and sound with values defined by the user in an HTML form.

You should also define validation expressions on these variables to guard against unintentional SQL being submitted to postgres. Within the layer metadata you'd do the following:

METADATA

```
'multimedia_validation_pattern' '^yes|no$'
'sound_validation_pattern' '^yes|no$'
'nseats_validation_pattern' '^[0-9]{1,2}$'
'... more metadata ...'
```

END

The validation strings are regular expressions that are applied against the appropriate variable value before being added to the FILTER. The first two limit the value of multimedia and sound to yes or no. The third limits the value for nseats to a 2 digit integer.

12.6 A Simple CGI Wrapper Script

Author Steven Monai

Revision \$Revision: 8278 \$

Date \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Last Updated 2006/01/26

Table of Contents

- A Simple CGI Wrapper Script
 - Introduction
 - Script Information

12.6.1 Introduction

This document presents a simple shell script that can be used to “wrap” the MapServer CGI, in order to avoid having to specify the ‘map’ parameter (or any other chosen parameters) in your MapServer URLs.

12.6.2 Script Information

If you want to avoid having to specify the ‘map’ parameter in your MapServer URLs, one solution is to use a “wrapper”. Basically, a wrapper is a CGI program that receives an incoming CGI request, modifies the request parameters in some way, and then hands off the actual processing to another CGI program (e.g. MapServer).

The following shell script is a wrapper for CGI GET requests that should be generic enough to run on any OS with /bin/sh:

```
#!/bin/sh
MAPSERV="/path/to/my/mapserv"
MAPFILE="/path/to/my/mapfile.map"
if [ "${REQUEST_METHOD}" = "GET" ]; then
  if [ -z "${QUERY_STRING}" ]; then
    QUERY_STRING="map=${MAPFILE}"
  else
    QUERY_STRING="map=${MAPFILE}&${QUERY_STRING}"
  fi
  exec ${MAPSERV}
else
  echo "Sorry, I only understand GET requests."
fi
exit 1
# End of Script
```

You should set the `MAPSERV` and `MAPFILE` variables as appropriate for your configuration. `MAPSERV` points to your MapServer executable, and `MAPFILE` points to the mapfile you want MapServer to use. Both variables should be absolute file paths that your webserver has permission to access, although they need not (and probably should not) be in web-accessible locations. Then put the script in your web server's `cgi-bin` directory, and make it executable.

Although this script only sets the 'map' parameter, it is easily modified to set any number of other MapServer parameters as well. For example, if you want to force your MapServer to 'map' mode, you can simply add 'mode=map' to the front of the `QUERY_STRING` variable. Just remember to separate your parameters with ampersands ('&').

Finally, note that the script only works for GET requests.

COMMUNITY ACTIVITIES

13.1 IRC

Some of the development of MapServer is coordinated through IRC. This page describes how you log on to chat, ask questions, and hack around with the developers.

13.1.1 Server and Channel Information

```
Server: irc.freenode.net  
Channel: #mapserver  
Logs: http://logs.qgis.org/mapserver/
```

13.1.2 Why IRC?

IRC is a primary medium where Open Source GIS hackers congregate, collaborate, and hack. It makes it easy to communicate things like compilation issues, where immediate, iterative feedback allows folks to make a lot of progress. Something that might take days of heavily-quoted emails through a maillist might only take fifteen minutes on IRC.

IRC is a great way to coordinate on-line meetings. Much of the discussion about the development of the new MapServer website was coordinated through IRC.

Some folks find IRC distracting and do not normally participate except for on-line meetings.

13.1.3 How do I join?

[Chatzilla](#) is probably the easiest way to get going. Chatzilla works with Mozilla or Firefox, and once you have it installed, you can log on to the channel by pointing your browser at:

```
irc://irc.freenode.net/#mapserver
```

There are many other IRC clients available. [This page](#) provides a good listing for many different platforms.

13.2 Mailing Lists

13.2.1 mapserver-announce

The mapserver-announce listserv is used to announce MapServer software updates or security issues. It is a very low-traffic volume list.

- **Subscribing to mapserver-announce**

To subscribe to the mapserver-announce listserv visit <http://lists.osgeo.org/mailman/listinfo/mapserver-announce> and enter your subscription information. You can later change your subscription information or leave the list at this website.

Note: If you are subscribed to the mapserver-users listserv, you need not subscribe to mapserver-announce: all messages sent to mapserver-announce are also copied to mapserver-users.

- **Submitting Questions to mapserver-announce**

The mapserver-announce mailing list is “read only” - subscribers are not permitted to post to this list.

13.2.2 mapserver-users

The mapserver-users is the primary means for MapServer users and developers to exchange application ideas, discuss potential software improvements, and ask questions.

- **Subscribing to mapserver-users**

To subscribe to the mapserver-users listserv visit <http://lists.osgeo.org/mailman/listinfo/mapserver-users>. You can later change your subscription information or leave the list at this website.

- **Submitting Questions to mapserver-users**

To submit questions to the mapserver-users listserv, first join the list by following the subscription procedure above. Then post questions to the list by sending an email message to mapserver-users@lists.osgeo.org.

- **Searching the Archives**

All MapServer-Users archives are located in <http://lists.osgeo.org/pipermail/mapserver-users/>. Searching the archives is best done with [Nabble](#).

13.2.3 mapserver-dev

A separate listserv is available for MapServer developers. It is meant to be used by individuals working on MapServer source code and related libraries to discuss issues that would not be of interest to the entire mapserver-users listserv.

- **Subscribing to mapserver-dev**

To subscribe to the mapserver-dev listserv visit <http://lists.osgeo.org/mailman/listinfo/mapserver-dev>, enter your e-mail address, name, and create a password and click subscribe. You can later change your subscription information or leave the list at this website.

- **Submitting Questions to mapserver-dev**

To submit questions to the mapserver-dev listserv, first join the list by following the subscription procedure above. Then post questions to the list by sending an email message to mapserver-dev@lists.osgeo.org.

- **Searching the Archives**

All MapServer-Dev archives are located in <http://lists.osgeo.org/pipermail/mapserver-dev/>. Searching the archives is best done with [Nabble](#).

13.2.4 MapServer mailing lists in languages other than English

- MapServer-DE (German)

Note: Please send a note to the mapserver-dev list if you know of any other MapServer mailing lists in additional languages.

13.2.5 Downloading list archives

The MapServer-Users and MapServer-Dev mailing lists have been migrated to the new OSGeo list server. One of the benefits of this migration is that you no longer need an account to download or read the archives. To access the MapServer archives, simply point your browser to <http://lists.osgeo.org/pipermail/mapserver-users/> and download the zipped file or click on the link. For MapServer-Dev archives, go to <http://lists.osgeo.org/pipermail/mapserver-dev/>.

13.3 MapServer Wiki Pages

- Conferences
- Hosting Providers
- Presentations Archive
- User Groups
- Users Directory
- Sites Using MapServer

13.4 MapServer Service Providers

- search for MapServer service providers with the OSGeo Search Tool

DEVELOPMENT

14.1 Sponsors

DM Solutions • OGC support (2001-2005)

- PHP-MapScript implementation and Support
- MapServer Release Management
- Windows Binaries
- MS4W and FGS installers
- Product Documentation

Environment Canada • *RFC 13*: SOS Server support (2006-2007)

i-cubed • OUTPUTFORMAT / 24-bit color raster support (2002)

RealGo • SWIG mapscript improvements (2003-2004)

Refractions Research • FastCGI implementation (2004).

TMC Technologies • *RFC 5*: MapServer Horizon Reprojection Improvements (2005).

Tydac AG • Rotated Maps (2004)

- *RFC 4*: MapServer Raster Resampling (2005)

Note: The MapServer project relies on its developers to keep this page up to date and accurate. If you feel that due credit is overlooked, please contact the developer that completed the work for your company or organization.

14.2 Bug Submission

Bugs should be submitted into Trac. You will first need to login using an OSGeo userid.

http://www.osgeo.org/osgeo_userid

<http://trac.osgeo.org/mapserver/>

Please keep the following issues in mind when submitting a bug:

1. Please set the *Component* carefully. It will determine who the bug is assigned to by default.
2. Set a meaningful yet reasonably brief description.
3. Set the version of MapServer you are using.

4. In your description please indicate whether you built from source or got it from an prepared binary build.

The most important thing when reporting a bug is to boil down a minimum example that is needed to reproduce the bug. That means a minimal mapfile + any data files it depends on. Remove everything from the map file that isn't needed to reproduce the bug.

The developers often dislike having to spend the first 30 minutes working on a bug, having to fix paths, remove unnecessary layers, removing references to external symbols or fonts that were not included or even needed and otherwise doctoring your test case to get it to a point when they can actually use it.

If the bug is easily demonstrated with “*shp2img*“, without the need to setup a proper web service and test it through http, then please show it that way. If a standalone *MapScript* script can demonstrate a problem without it having to be a web service, likewise submit it that way.

The chances of a bug being addressed in a timely manner is directly related to the speed with which the developer can reproduce the bug. If you make that hard for the developer, chances are the bug will be given up on or ignored for quite a while.

14.3 Subversion

14.3.1 Code Developer's Subversion Access

MapServer's source code and documentation are under Subversion control. Subversion access is mostly intended for use by developers, but users can also access the MapServer source between releases as it is being developed. CVS access is only recommended for those who need the absolute latest and greatest code, if they are not afraid of getting their hands dirty building the source. The Subversion version does not contain **GD** or any of the support libraries, and it requires flex and bison to build it. Building MapScript will also require **SWIG** be installed on your machine. Here's how to access the read-only source:

1. Install a Subversion client, see [Subversion Homepage](#) for more information. TortoiseSVN is a good solution if you are on Windows, and most Unixes should have a client available.
2. Issue `svn co https://svn.osgeo.org/mapservers/trunk/mapservers mapservers` to check out a copy of the current trunk into your working directory.

14.3.2 Support Libraries

Information about supporting libraries and how to compile MapServer from source can be found in the [Compiling on Unix](#).

14.3.3 How to Obtain Commit Access

If you find yourself submitting a lot of patches to [Trac](#), or you would like to be an active developer that picks up the maintenance of a portion of MapServer, contact Steve Lime or one of the other developers. To obtain Subversion commit access, an individual must:

- demonstrate expertise about a specific area
- be willing to put more time into the project than just the short term (dumping code into the project and providing no way to maintain it is almost as bad as having no code at all)
- be active instead of casual about the project.
- election of Subversion committers is covered in [MS RFC 7.1: MapServer SVN Commit Management](#)

14.3.4 Subversion Web View

You can find an html viewer for the Subversion repository at <http://trac.osgeo.org/mapserver/browser>

14.4 Documentation Development Guide

Author Howard Butler

Contact hobu.inc at gmail.com

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 8659 \$

Date \$Date: 2009-03-08 07:31:00 -0800 (Sun, 08 Mar 2009) \$

Table of Contents

- Documentation Development Guide
 - Background
 - General Guidelines
 - reStructuredText Reference Guides
 - reStructuredText Formatting
 - Installing and Using Sphinx for rst-html Generation
 - Reference Labels

14.4.1 Background

The current structure of the MapServer documentation process is for developers with *Subversion* commit access to maintain their documents in reStructuredText format, and therefore all documents live in the /docs directory in SVN. The *Sphinx* documentation generator is used to convert the reStructuredText files to html, and the live website is then updated on an hourly basis.

14.4.2 General Guidelines

- MapServer instead of mapserver, map server, Map Server, mapServer or map Server.
- MapScript instead of mapscript, Mapscript, or map script.
- PostGIS instead of postgis.
- HowTo instead of howto or HOWTO.
- Email addresses should be manually spam protected:

hobu.inc at gmail.com instead of hobu.inc@gmail.com

14.4.3 reStructuredText Reference Guides

- Quick reStructuredText

14.4.4 reStructuredText Formatting

- All text should be hard breaks at or around the 80 column mark, just as the source code.
- No `.. sectnum::` in the contents directives
- All external links should live at the bottom of your document, under the heading:

```
.. ##### rST Link Section #####
```

- Always include the `:Revision:` and `:Date:` lines (as-is) at the top of your document, such as:

```
:Revision: $Revision: 8659 $  
:Date: $Date: 2009-03-08 07:31:00 -0800 (Sun, 08 Mar 2009) $
```

14.4.5 Installing and Using Sphinx for rst-html Generation

On Windows:

1. install [Python 2.X](#)
2. download [setuptools](#)
3. make sure that the 'C:Python2XScripts' directory is your path
4. execute the following at commandline:

```
easy_install Sphinx
```

...you should see message: "Finished processing dependencies for Sphinx"

5. inside the MapServer /docs directory, create /build/doctrees and /build/html directories
6. inside the /docs directory, execute:

```
sphinx-build -b html -d build\doctrees . build\html
```

...you should see a message: "build succeeded, ..."

On Linux:

1. make sure you have the Python dev and setuptools packages installed. On Ubuntu:

```
sudo apt-get install python-dev  
sudo apt-get install python-setuptools
```

2. install sphinx using `easy_install`:

```
sudo easy_install sphinx
```

3. to process the docs, from the MapServer /docs directory, run:

```
make html
```

or

```
make latex
```

the HTML output will be written to the build/html sub-directory.

14.4.6 Reference Labels

Table 14.1: :ref: reference labels

Label	Title
about	<i>About</i>
agg	<i>AGG Rendering Specifics</i>
antialias	<i>AntiAliasing with MapServer</i>
arcinfo	<i>ArcInfo</i>
arcsde	<i>ArcSDE</i>
autotest	<i>Regression Testing</i>
background	<i>Tutorial Timeframe</i>
batch_utilities	<i>Batch Scripting</i>
bugs	<i>Bug Submission</i>
cgi	<i>CGI</i>
cgi_controls	<i>MapServer CGI Controls</i>
cgi_introduction	<i>MapServer CGI Introduction</i>
class	<i>CLASS</i>
community	<i>Community Activities</i>
development	<i>Development</i>
dgn	<i>DGN</i>
documentation	<i>MapServer 5.4.0 Documentation</i>
documentation_development	<i>Documentation Development Guide</i>
dotnet_compile	<i>.NET MapScript Compilation</i>
download	<i>Download</i>
dynamic_charting	<i>Dynamic Charting</i>
editing	<i>Mapfile Editing</i>
errors	<i>Errors</i>
example1-1	<i>Example 1.1</i>
example1-1-map	<i>Example1-1.map</i>
example1-2	<i>Example 1.2</i>
example1-2-map	<i>Example1-2.map</i>
example1-3	<i>Example 1.3</i>
example1-3-map	<i>Example1-3.map</i>
example1-4	<i>Example 1.4</i>
example1-4-map	<i>Example1-4.map</i>
example1-5	<i>Example 1.5</i>
example1-5-map	<i>Example1-5.map</i>
example1-6	<i>Example 1.6</i>
example1-6-map	<i>Example1-6.map</i>
example1-7	<i>Example 1.7</i>
example1-7-map	<i>Example1-7.map</i>
example1-8	<i>Example 1.8</i>
example1-8-map	<i>Example1-8.map</i>
expressions	<i>Expressions</i>
faq	<i>FAQ</i>
fastcgi	<i>FastCGI</i>
feature	<i>FEATURE</i>
filter_encoding	<i>WFS Filter Encoding</i>
flash	<i>Flash Output</i>
fontset	<i>FONTSET</i>
format_types	<i>Data Format Types</i>
genindex	<i>Index</i>

Continued on next page

Table 14.1 – continued from previous page

gloss	<i>Glossary</i>
gml	<i>GML</i>
gpx	<i>GPS Exchange Format (GPX)</i>
grid	<i>GRID</i>
html_legend	<i>HTML Legends with MapServer</i>
iis	<i>IIS Setup for MapServer</i>
imagemaps	<i>HTML Imagemaps</i>
include	<i>INCLUDE</i>
inline	<i>Inline</i>
input	<i>Data Input</i>
input_postgis	<i>PostGIS/PostgreSQL</i>
installation	<i>Installation</i>
introduction	<i>An Introduction to MapServer</i>
irc	<i>IRC</i>
join	<i>JOIN</i>
kml	<i>KML - Keyhole Markup Language</i>
label	<i>LABEL</i>
layer	<i>LAYER</i>
legend	<i>LEGEND</i>
legend_utility	<i>legend</i>
license	<i>License</i>
linux	<i>Linux</i>
lists	<i>Mailing Lists</i>
management	<i>File Management</i>
map	<i>MAP</i>
map_context	<i>Map Context</i>
mapcontext_cgi	<i>Map Context Files</i>
mapfile	<i>Mapfile</i>
mapfile_tuning	<i>Mapfile</i>
mapinfo	<i>MapInfo</i>
mapscript	<i>MapScript</i>
mapscript_introduction	<i>Introduction</i>
mapscript_ows	<i>MapScript Wrappers for WxS Services</i>
mapscript_tests	<i>MapScript Unit Testing</i>
mapserv	<i>mapserv</i>
modindex	<i>Module Index</i>
msencrypt	<i>msencrypt</i>
mysql	<i>MySQL</i>
ntf	<i>NTF</i>
oci	<i>Oracle Spatial</i>
oci_install	<i>Oracle Installation</i>
ogc	<i>OGC Support and Configuration</i>
ogr	<i>OGR</i>
online_resource_wms	<i>More About the Online Resource URL</i>
optimization	<i>Optimization</i>
osx	<i>Mac OS X</i>
output	<i>Output Generation</i>
outputformat	<i>OUTPUTFORMAT</i>
pdf	<i>PDF Output</i>
pgeo	<i>ESRI Personal Geodatabase (MDB)</i>
php	<i>PHP MapScript</i>
php_example	<i>By Example</i>

Continued on next page

Table 14.1 – continued from previous page

php_install	<i>PHP MapScript Installation</i>
php_install_example_steps	<i>Example Steps of a Full Windows Installation</i>
projection	<i>PROJECTION</i>
python	<i>Python MapScript Appendix</i>
querying	<i>Querying</i>
querymap	<i>QUERYMAP</i>
raster	<i>Raster Data</i>
raster_optimization	<i>Raster</i>
reference	<i>REFERENCE</i>
rfc1	<i>MS RFC 1: Technical Steering Committee Guidelines</i>
rfc10	<i>MS RFC 10: Joining the Open Source Geospatial Foundation</i>
rfc11	<i>MS RFC 11: Support for Curved Labels</i>
rfc12	<i>MS RFC 12: C code Unit tests</i>
rfc13	<i>MS RFC 13: Support of Sensor Observation Service in MapServer</i>
rfc14	<i>MS RFC 14: Relative Coordinates for INLINE features</i>
rfc15	<i>MS RFC 15: Support for thread neutral operation of MapServer/MapScript</i>
rfc16	<i>MS RFC 16: MapScript WxS Services</i>
rfc17	<i>MS RFC 17: Dynamic Allocation of layers, styles, classes and symbols</i>
rfc18	<i>MS RFC 18: Encryption of passwords in mapfiles</i>
rfc19	<i>MS RFC 19: Style & Label attribute binding</i>
rfc2	<i>MS RFC 2: Creating line features and/or shapes using WKT</i>
rfc21	<i>MS RFC 21: MapServer Raster Color Correction</i>
rfc22a	<i>MS RFC 22a: Feature cache for long running processes and query processing</i>
rfc23	<i>MS RFC 23: Technical Steering Committee Guidelines</i>
rfc24	<i>MS RFC 24: Mapscript memory management</i>
rfc24first	<i>MS RFC 24: Mapscript memory management</i>
rfc25	<i>MS RFC 25: Align MapServer pixel and extent models with OGC models</i>
rfc26	<i>MS RFC 26: Version 5 Terminology Cleanup</i>
rfc27	<i>MS RFC 27: Label Priority</i>
rfc28	<i>MS RFC 28: Redesign of LOG/DEBUG output mechanisms</i>
rfc29	<i>MS RFC 29: Dynamic Charting Capability</i>
rfc3	<i>MS RFC 3: Feature Layer Plug-in Architecture</i>
rfc30	<i>MS RFC 30: Support for WMS 1.3.0</i>
rfc31	<i>MS RFC 31: Loading MapServer Objects from Strings</i>
rfc32	<i>MS RFC 32: Support for Anti-Grain Geometry (AGG) Rendering Engine</i>
rfc33	<i>MS RFC 33: Removing msLayerWhichItems() from maplayer.c</i>
rfc39	<i>MS RFC 39: Support of WMS/SLD Named Styles</i>
rfc4	<i>MS RFC 4: MapServer Raster Resampling</i>
rfc40	<i>MS RFC 40: Support Label Text Transformations</i>
rfc41	<i>MS RFC 41: Support of WCS 1.1.x Protocol</i>
rfc42	<i>MS RFC 42: Support of Cookies Forwarding</i>
rfc43	<i>MS RFC 43: Direct tile generation for Google Maps and Virtual Earth API</i>
rfc44	<i>MS RFC 44: Restore URL modification of mapfiles to pre-5.0 levels</i>
rfc45	<i>MS RFC 45: Symbology, Labeling, and Cartography Improvements</i>
rfc46	<i>MS RFC 46: Migrate Website to OSGeo</i>
rfc47	<i>MS RFC 47: Move IGNORE_MISSING_DATA to run-time configuration</i>
rfc48	<i>MS RFC 48: GEOTRANSFORM Geometry operations</i>
rfc49	<i>MS RFC 49: Symbology, Labeling, and Cartography Improvements</i>
rfc5	<i>MS RFC 5: MapServer Horizon Reprojection Improvements</i>
rfc50	<i>MS RFC 50: OpenGL Rendering Support</i>
rfc51	<i>MS RFC 51: XML Mapfile Format</i>
rfc53	<i>MS RFC 53: Guidelines for MapScript method return values</i>

Continued on next page

Table 14.1 – continued from previous page

rfc54	<i>MS RFC 54: Rendering Interface API</i>
rfc55	<i>MS RFC 55: Improve control of output resolution</i>
rfc56	<i>MS RFC 56: Tighten control of access to mapfiles and templates</i>
rfc6	<i>MS RFC 6: Color Range Mapping of Continuous Feature Values</i>
rfc7	<i>MS RFC 7: MapServer CVS Commit Management</i>
rfc7.1	<i>MS RFC 7.1: MapServer SVN Commit Management</i>
rfc8	<i>MS RFC 8: Pluggable External Feature Layer Providers</i>
rfc9	<i>MS RFC 9: Item tag for query templates</i>
rfcs	<i>Request for Comments</i>
runsub	<i>Run-time Substitution</i>
s57	<i>S57</i>
scalebar	<i>SCALEBAR</i>
scalebar_utility	<i>scalebar</i>
sdt	<i>SDTS</i>
search	<i>Search Page</i>
section1	<i>Section 1: Static Maps and the MapFile</i>
section2	<i>Section 2: CGI variables and the User Interface</i>
shapefiles	<i>ESRI Shapefiles (SHP)</i>
shp2img	<i>shp2img</i>
shptree	<i>shptree</i>
shptreevis	<i>shptreevis</i>
sld	<i>SLD</i>
sortshp	<i>sortshp</i>
sos_server	<i>SOS Server</i>
source	<i>Source</i>
sponsors	<i>Sponsors</i>
style	<i>STYLE</i>
styleitemauto-label-styles	<i>Accessing OGR STYLEITEMAUTO Label Styles Through MapScript</i>
svg	<i>SVG</i>
svn	<i>Subversion</i>
swig	<i>SWIG MapScript API Reference</i>
sym2img	<i>sym2img</i>
sym_construction	<i>Cartographic Symbol Construction with MapServer</i>
sym_examples	<i>Symbology Examples</i>
symbol	<i>SYMBOL</i>
template	<i>Templating</i>
testing	<i>Testing</i>
tiger	<i>USGS TIGER</i>
tile4ms	<i>tile4ms</i>
tile_mode	<i>Tile Mode</i>
tileindex	<i>Tile Indexes</i>
tutorial	<i>MapServer Tutorial</i>
unix	<i>Compiling on Unix</i>
utilities	<i>Utilities</i>
variable_sub	<i>Variable Substitution</i>
vector	<i>Vector Data</i>
vector_optimization	<i>Vector</i>
vim	<i>VIM Syntax</i>
virtual_vector	<i>Virtual Spatial Data</i>
wcs_format	<i>WCS Use Cases</i>
wcs_server	<i>WCS Server</i>
web	<i>WEB</i>

Continued on next page

Table 14.1 – continued from previous page

wfs	<i>WFS</i>
wfs_client	<i>WFS Client</i>
wfs_server	<i>WFS Server</i>
win32	<i>Compiling on Win32</i>
windows	<i>Windows</i>
wms_capabilities	<i>Validate the Capabilities Metadata</i>
wms_client	<i>WMS Client</i>
wms_server	<i>WMS Server</i>
wms_time	<i>WMS Time</i>
wrapper	<i>A Simple CGI Wrapper Script</i>

Regenerating the reference labels

You can regenerate the reference labels by issuing:

```
make labels
```

from the docs directory like when you are building the html or latex versions

14.5 Testing

MapServer provides testing in multiple forms. The *Regression Testing* framework uses *CGI* and other tools to test the generation of images and *OGC* output. The *MapScript Unit Testing* framework tests the MapScript and its object hierarchy.

14.5.1 Regression Testing

Author Frank Warmerdam

Contact warmerdam at pobox.com

Revision \$Revision: 8365 \$

Date \$Date: 2008-12-31 07:49:02 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2007/8/31

Table of Contents

- Regression Testing
 - Getting msautotest
 - Running msautotest
 - Checking Failures
 - PerceptualDiff
 - Background
 - TODO
 - Adding New Tests

The msautotest is a suite of test maps, data files, expected result images, and test scripts intended to make it easy to run an a set of automated regression tests on MapServer.

Getting msautotest

The autotest is available from SVN. On Unix it could be fetched something like:

```
% svn checkout http://svn.osgeo.org/mapserver/trunk/msautotest
```

This would create an msautotest subdirectory wherever you are. I normally put the autotest within my MapServer directory.

Running msautotest

The autotest requires python (but not python MapScript), so if you don't have python on your system - get and install it. More information on python is available at <http://www.python.org>. Most Linux system have some version already installed.

The autotest also requires that the executables built with MapServer, notably *shp2img*, *legend*, *mapserv* and *scalebar*, are available in the path. I generally accomplish this by adding the MapServer build directory to my path.

csh:

```
% setenv PATH $HOME/mapserver:$PATH
```

bash/sh:

```
% PATH=$HOME/mapserver:$PATH
```

Verify that you can run stuff by typing 'shp2img -v' in the autotest directory:

```
warmerda@gdal2200[152]% shp2img -v
MapServer version 3.7 (development) OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
SUPPORTS=PROJ SUPPORTS=TTF SUPPORTS=WMS_SERVER SUPPORTS=GD2_RGB
INPUT=TIFF INPUT=EPPL7 INPUT=JPEG INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

Now you are ready to run the tests. The tests are subdivided into categories, currently just "gdal", "misc", and "wxs" each as a subdirectory. To run the "gdal" tests cd into the gdal directory and run the run_test.py script.

Unix:

```
./run_test.py
```

Windows:

```
python.exe run_test.py
```

The results in the misc directory might look something like this:

```
warmerda@gdal2200[164]% run_test.py
version = MapServer version 3.7 (development) OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
SUPPORTS=PROJ SUPPORTS=TTF SUPPORTS=WMS_SERVER SUPPORTS=GD2_RGB INPUT=TIFF
INPUT=EPPL7 INPUT=JPEG INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

```
Processing: rgba_scalebar.map
  results match.
```

```
Processing: tr_scalebar.map
  results match.
```

```
Processing: tr_label_rgb.map
```



```
    results match.
Processing: ogr_direct.map
    results match.
Processing: ogr_select.map
    results match.
Processing: ogr_join.map
    results match.
Test done:
    0 tested skipped
    6 tests succeeded
    0 tests failed
    0 test results initialized
```

In general you are hoping to see that no tests failed.

Checking Failures

Because most msautotest tests are comparing generated images to expected images, the tests are very sensitive to subtle rounding differences on different systems, and subtle rendering changes in libraries like freetype and gd. So it is quite common to see some failures.

These failures then need to be reviewed manually to see if the differences are acceptable and just indicating differences in rounding/rendering or whether they are “real” bugs. This is normally accomplished by visually comparing files in the “result” directory with the corresponding file in the “expected” directory. It is best if this can be done in an application that allows images to be layers, and toggled on and off to visually highlight what is changing. OpenEV can be used for this.

PerceptualDiff

If you install the PerceptualDiff program (<http://pdiff.sourceforge.net/>) and it is in the path, then the autotest will attempt to use it as a last fallback when comparing images. If images are found to be “perceptually” the same the test will pass with the message “result images perceptually match, though files differ.” This can dramatically cut down the number of apparent failures that on close inspection are for all intents and purposes identical. Building PerceptualDiff is a bit of a hassle.

Background

The msautotest suite was initially developed by Frank Warmerdam (warmerdam at pobox.com), who can be contacted with questions it.

The msautotest suite is organized as a series of .map files. The python scripts basically scan the directory in which they are run for files ending in .map. They are then “run” with the result dumped into a file in the result directory. A binary comparison is then done to the corresponding file in the expected directory and differences are reported. The general principles for the test suite are that:

- The test data should be small so it can be easily stored and checked out of svn without big files needing to be downloaded.
- The test data should be completely contained within the test suite ... no dependencies on external datasets, or databases that require additional configuration. PostGIS and Oracle will require separate testing mechanisms.
- The tests should be able to run without a significant deal of user interaction. This is as distinct from the DNR test suite described in FunctionalityDemo.
- The testing mechanism should be suitable to test many detailed functions in relative isolation.

- The test suite is not dependent on any of the MapScript environments, though I think it would be valuable to extend the testsuite with some mapscript dependent components in the future (there is a start on this in the mspython directory).

Test Script Internal Functioning

Because MapServer can be built with many possible extensions, such as support for OGR, GDAL, and PROJ.4, it is desirable to have the testsuite automatically detect which tests should be run based on the configuration of MapServer. This is accomplished by capturing the version output of “shp2img -v” and using the various keys in that to decide which tests can be run. A directory can have a file called “all_require.txt” with a “REQUIRES:” line indicating components required for all tests in the directory. If any of these requirements are not met, no tests at all will be run in this directory. For instance, the gdal/all_require.txt lists:

```
REQUIRES: INPUT=GDAL OUTPUT=PNG
```

In addition, individual .map files can have additional requirements expressed as a REQUIRES: comment in the mapfile. If the requirements are not met the map will be skipped (and listed in the summary as a skipped test). For example gdal/256_overlay_res.map has the following line to indicate it requires projection support (in addition to the INPUT=GDAL and OUTPUT=PNG required by all files in the directory):

```
REQUIRES: SUPPORTS=PROJ
```

The output files generated by processing a map is put in the file results/<mapfilebasename>.png (regardless of whether it is PNG or not). So when gdal/256_overlay_res.map is processed, the output file is written to gdal/results/256_overlay_res.png. This is compared to gdal/expected/256_overlay_res.png. If they differ the test fails, and the “wrong” result file is left behind for investigation. If they match the result file is deleted. If there is no corresponding expected file the results file is moved to the expected directory (and reported as an “initialized” test) ready to be committed to CVS.

There is also a RUN_PARMS keyword that may be placed in map files to override a bunch of behaviour. The default behaviour is to process map files with shp2img, but other programs such as mapserv or scalebar can be requested, and various commandline arguments altered as well as the name of the output file. For instance, the following line in misc/tr_scalebar.map indicates that the output file should be called tr_scalebar.png, the commandline should look like “[SCALEBAR] [MAPFILE] [RESULT]” instead of the default “[SHP2IMG] -m [MAPFILE] -o [RESULT]”.

```
RUN_PARMS: tr_scalebar.png [SCALEBAR] [MAPFILE] [RESULT]
```

For testing things as they would work from an HTTP request, use the RUN_PARMS with the program [MAPSERV] and the QUERY_STRING argument, with results redirected to a file.

```
# RUN_PARMS: wcs_cap.xml [MAPSERV] QUERY_STRING='map=[MAPFILE]&SERVICE=WCS&VERSION=1.0.0&REQUEST=GetCapabilities'
```

For web services that generate images that would normally be prefixed with the Content-type header, use [RESULT_NOMIME] to instruct the test harness to script off any http headers before doing the comparison.

```
# Generate simple PNG.  
# RUN_PARMS: wcs_simple.png [MAPSERV] QUERY_STRING='map=[MAPFILE]&SERVICE=WCS&VERSION=1.0.0&REQUEST=GetCapabilities'
```

What If A Test Fails?

When running the test suite, it is common for some tests to fail depending on vagaries of floating point on the platform, or harmless changes in MapServer. To identify these compare the results in result with the file in expected and

determine what the differences are. If there is just a slight shift in text or other features it is likely due to floating point differences on different platforms. These can be ignored. If something has gone seriously wrong, then track down the problem!

It is also prudent to avoid using output image formats that are platform specific. For instance, if you produce TIFF it will generate big endian on big endian systems and therefore be different at the binary level from what was expected. PNG should be pretty safe.

TODO

- Add lots of tests for different stuff! Very little vector testing done yet.
- Add a high level script in the msautotest directory that runs the subscripts in all the subdirectories and produces a summary report.

Adding New Tests

- Pick an appropriate directory to put the test in. Feel free to start a new one for new families of testing functionality.
- Create a minimal map file to test a particular issue. I would discourage starting from a “real” mapfile and cutting down as it is hard to reduce this to the minimum.
- Give the new mapfile a name that hints at what it is testing without making the name too long. For instance “ogr_join.map” tests OGR joins. “rgb_overlay_res_to8bit.map” tests RGB overlay layers with resampling and converting to 8bit output.
- Put any MapServer functionality options in a # REQUIRES: item in the header as described in the internal functioning topic above.
- Write some comments at the top of the .map file on what this test is intended to check.
- Add any required datasets within the data directory beneath the test directory. These test datasets should be as small as possible! Reuse existing datasets if at all possible.
- run the “run_tests.py” script.
- verify that the newly created expected/<testname>.png file produces the results you expect. If not, revise the map and rerun the test, now checking the results/<testname>.png file. Move the results/<testname>.png file into the expected directory when you are happy with it.
- add the .map file, and the expected/<testname>.png file to CVS when you are happy with them. Make sure the .png file (and any supporting data files) are marked as binary files. For example,

```
% svn add mynewtest.map expected/mynewtest.png
% svn commit -m "new" mynewtest.map expected/mynewtest.png
```

You're done!

14.5.2 MapScript Unit Testing

Date 2005/11/20

Author Sean Gillies

Test Driven Development

In 2003, I began to commit to test driven development of the mapscript module. TDD simply means development through repetition of two activities:

1. add a test, cause failure, and write code to pass the test
2. remove duplication

Test Driven Development is also a book by Kent Beck.

New features that I develop for MapServer begin as test expressions. There are a bazillion good reasons for working this way. The most obvious are

1. accumulation of automated unit tests
2. accumulation of excellent usage examples
3. that i'm prevented from starting work on flaky ideas that can't be tested

About the tests

Tests are committed to the MapServer CVS under mapscript/python/tests. They are written in Python using the JUnit inspired unittest module. A good introduction to unit testing with Python is found at http://diveintopython.org/unit_testing/index.html.

The test framework imports mapscript from python/tests/cases/testing.py. This allows us to test the module before installation

```
[sean@lenny python]$ python setup.py build
[sean@lenny python]$ python tests/runtests.py -v
```

Test cases are implemented as Python classes, and individual tests as class methods named beginning with test*. The special setUp() and tearDown() methods are for test fixtures and are called before and after every individual test.

Since version 4.2, MapServer includes a very lightweight testing dataset under mapserver/tests. The set consists of symbols, fonts, three single-feature shapefiles, and a test.map mapfile. This is the only data used by the unit tests.

Many tests that require a mapObj derive from testing.MapTestCase:

```
class MapTestCase (MapPrimitivesTestCase) :
    """Base class for testing with a map fixture"""
    def setUp(self) :
        self.map = mapscript.mapObj(TESTMAPFILE)
    def tearDown(self) :
        self.map = None
```

One example is the MapSymbolSetTestCase, the test case I used for development of the expanded symbolset functionality present in the 4.2 release:

```
class MapSymbolSetTestCase (MapTestCase) :
    def testGetNumSymbols(self) :
        """expect getNumSymbols == 2 from test fixture test.map"""
        num = self.map.getNumSymbols()
        assert num == 2, num

    ...
```

Status

This unit testing framework only covers functionality that is exposed to the Python mapscript module. It can help to check on pieces of the core MapServer code, but is no guarantor of the *mapserv* program or of the *PHP MapScript* module. As of this writing, there are 159 tests in the suite. These are tests of features added since mid-2003. Much of MapServer's older stuff remains untested and it is doubtful that we'll make the time to go back and fill in.

14.6 Request for Comments

A MapServer RFC describes a major change in the technological underpinnings of MapServer, major additions to functionality, or changes in the direction of the project.

14.6.1 MS RFC 1: Technical Steering Committee Guidelines

Date 2005/06/24

Author Frank Warmerdam, Independent

Contact warmerdam at pobox.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Superseded by *MS RFC 23: Technical Steering Committee Guidelines*

Id \$Id: ms-rfc-1.txt 8278 2008-12-23 21:34:31Z hobu \$

Summary

This document describes how the MapServer Technical Steering Committee determines membership, and makes decisions on MapServer technical issues.

In brief the technical team votes on proposals on mapserver-dev. Proposals are available for review for at least two days, and a single veto is sufficient delay progress though ultimately a majority of members can pass a proposal.

Detailed Process

1. Proposals are written up and submitted on the mapserver-dev mailing list for discussion and voting, by any interested party, not just committee members.
2. Proposals need to be available for review for at least two business days before a final decision can be made.
3. Respondents may vote "+1" to indicate support for the proposal and a willingness to support implementation.
4. Respondents may vote "-1" to veto a proposal, but must provide clear reasoning and alternate approaches to resolving the problem within the two days.
5. A vote of -0 indicates mild disagreement, but has no effect. A 0 indicates no opinion. A +0 indicate mild support, but has no effect.
6. Anyone may comment on proposals on the list, but only members of the Technical Steering Committee's votes will be counted.
7. A proposal will be accepted if it receives +2 (including the proposer) and no vetos (-1).
8. If a proposal is vetoed, and it cannot be revised to satisfy all parties, then it can be resubmitted for an override vote in which a majority of all eligible voters indicating +1 is sufficient to pass it. Note that this is a majority of all committee members, not just those who actively vote.

9. Upon completion of discussion and voting the proposer should announce whether they are proceeding (proposal accepted) or are withdrawing their proposal (vetoed).
10. The Chair gets a vote.
11. The Chair is responsible for keeping track of who is a member of the Technical Steering Committee (perhaps as part of a TSC file in CVS).
12. Addition and removal of members from the committee, as well as selection of a Chair should be handled as a proposal to the committee.
13. The Chair adjudicates in cases of disputes about voting.

When is Vote Required?

- Anything that could cause backward compatibility issues.
- Adding substantial amounts of new code.
- Changing inter-subsystem APIs, or objects.
- Issues of procedure.
- When releases should take place.
- Anything that might be controversial.

Boundaries of “Technical”

- If it relates to changes in the code, it is technical.
- If it relates to how the developers cooperate, it is technical.
- If it relates to legal issues around code ownership it is technical.
- If it relates to documentation, or the web site it is not technical.
- If it relates to events such as conferences, it is not technical.

Observations

- The Chair is the ultimate adjudicator if things break down.
- The absolute majority rule can be used to override an obstructionist veto, but it is intended that in normal circumstances vetoers need to be convinced to withdraw their veto. We are trying to reach consensus.
- It is anticipated that separate “committees” will exist to manage conferences, documentation and web sites.

Bootstrapping

Steve Lime is declared initial Chair of the Technical Steering Committee.

Steve Lime, Daniel Morissette, Frank Warmerdam, Sean Gilles, Assefa Yewondwossen, Howard Butler and Perry Nacionales are declared to be the founding Technical Steering Committee.

14.6.2 MS RFC 2: Creating line features and/or shapes using WKT

Date 2005/07/13

Author Steve Lime

Contact steve.lime at DNR.STATE.MN.US

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Completed

Version MapServer 4.8

Id \$Id: ms-rfc-2.txt 8278 2008-12-23 21:34:31Z hobu \$

Description: Developing inline features or shapes within MapScript can be a bit cumbersome. One alternative would be to allow users to define feature using the Well-Known Text format. The proposed solution would allow users to use this format:

1. within a mapfile
2. via URL
3. via MapScript
4. via MapServer query template

Instead of writing a new WKT parser we would provide access to underlying GEOS or OGR functionality to do this. Notes about the actual implementation are included below.

Files affected

- mapfile.h => new constant, WKT
- maplexer.l => recognize the new constant
- mapfile.c => process new mapfile parameter with FEATURE block (WKT), and update URL parsing in a similar manner
- mapgeos.cpp => wrap GEOS WKT reading/writing code
- mapogr.cpp => wrap OGR WKT reading/writing code
- mapprimitive.c => wrap the GEOS and OGR WKT writer/reading code, this would be the “public” interface
- maptemplate.c => update the shpxy tag with a “-wkt” option so it would output the WKT version of a shape. Placing here would allow us to take advantage of the projection support already in place, plus any future options (thining, buffers and so on).
- mapscript/swiginc/shape.i => update constructor to pass a WKT string and to define a “toString” method that would output a WKT string. Similar modifications would have to be made within PHP/MapScript, patterned after the SWIG-based interface.

Backwards compatabilty issues

N/A, new functionality

Implementation Details

- The C API will take the form:

```
shapeObj *msShapeFromWKT( const char * );  
char *msShapeToWKT( shapeObj * );
```

These are contrary to some of the older code (e.g. `msLayerNextShape()`). However there are 2 places the WKT APIs will be used: 1) `MapScript` and 2) creating inline features via URL or `MapFiles`. In both cases the above functions would be preferred.

- In `MapScript`, creating a shape will take the form of an overloaded constructor, e.g.:

```
$shape = new shapeObj($mapscript::MS_SHAPE_LINE); OR $shape = new  
shapeObj('LINESTRING(0 0,1 1,1 2)');
```

- In `MapScript`, a `toWKT()` method will be added to `shapeObj` object.
- WKT is geometry only. The attributes, index, tileindex, classindex, and text fields do not exist in WKT and will have to be set “elsewhere”.
- There is no widely supported, or standardized approach to “measure” values in WKT though `Refractions` does support it in “EWKT”. For now it is assumed that measure values will not be preserved from WKT.
- There is a well defined way of including Z coordinates and these should be carried through if `MapServer` is built with Z and M support enabled.
- Development will be accompanied by a set of tests. Sean Gillies has already created a test case or two.

Transformations shapeObj to WKT

- `MS_SHAPE_POINT`: If `numlines` and `numpoints` are one, then this is converted to a `POINT` object in WKT. If there are more points, this is converted to a `MULTIPOINT` object.
- `MS_SHAPE_LINE`: if `numlines` is 1 then this will be translated to a `LINESTRING` otherwise it will be translated to a `MULTILINESTRING`.
- `MS_SHAPE_POLYGON`: `MapServer` does not keep track of interior and exterior rings in a shape, since the scanline rasterization mechanism of GD does not require this information. However, when converting to WKT we need to know whether we have a single polygon with holes, or multiple polygons (more than one exterior ring). If `numlines` is 1, we can directly translate to `POLYGON()`, otherwise the rings will need to be analysed to identify outer rings, and to associate inner rings with their outer ring. If more than one outer ring exist, a `MULTIPOLYGON` will be produced, otherwise a `POLYGON` will be produced.
- `MS_SHAPE_NULL`: This results in an empty WKT string.

Transformations WKT to shapeObj

- `POINT`: Translates to an `MS_SHAPE_POINT` object with one point and one line.
- `LINESTRING`: Translates to an `MS_SHAPE_LINE` object with one line.
- `POLYGON`: Translates to an `MS_SHAPE_POLYGON` object with one line for the outer ring and one line for each inner ring.
- `MULTIPOINT`: Translates to an `MS_SHAPE_POINT` object with one line, and one point for each line.

- **MULTILINESTRING**: Translates to an `MS_SHAPE_LINE` object with one line for each linestring in the container.
- **MULTIPOLYGON**: Translates into an `MS_SHAPE_POLYGON` with each ring of each polygon being a line in the resulting polygon object.
- **GEOMETRYCOLLECTION**: Treat as `MULTIPOINT`, `MULTILINESTRING` or `MULTIPOLYGON` if the contents are all compatible, otherwise throw an exception.

Bug ID

<http://trac.osgeo.org/mapserver/ticket/1466> (primary bug, addition entries for MapScript and OGR will follow)

Voting history

Vote proposed by Steve Lime on 9/4/2005, result was +4 (3 non-voting members).

Voting +1: Lime, Warmerdam, Morissette, Gillies

Proposal passes and will move forward.

14.6.3 MS RFC 3: Feature Layer Plug-in Architecture

Date 2005/08/22

Author Jani Averbach

Contact javerbach at extendthereach.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Adopted

Version MapServer 4.8

Id \$Id: ms-rfc-3.txt 8278 2008-12-23 21:34:31Z hobu \$

Abstract Solution

Implement a virtual table structure for `layerObj`. This structure will contain function pointers for all layer specific operations. This structure will be populated when the layer is opened or first time accessed. After that all back-end format specific layer operations will happen through this vtable which is cached in the `layerObj` struct.

Technical Solution

All file names and numbers are against released MapServer 4.6.0 source code.

1. Add new field to the `layerObj`. It will be pointer to the vtable, which will contain function pointers for this layer.
2. Add the virtual table architecture

The vtable will be initialized when the layer is accessed at the first time. The vtable will be populated with dummies when it is created and these dummies will implement feasible default actions if there is any (e.g. do nothing) or return error by default.

- 2.1. Standard functions which are found currently from `maplayer.c`

2.1.1. InitItemInfo

```
int (*LayerInitItemInfo)(layerObj *layer);
```

2.1.2. FreeItemInfo

```
void (*LayerFreeItemInfo)(layerObj *layer);
```

2.1.3. Open

```
int (*LayerOpen)(layerObj *layer);
```

Currently there are two layers which accept more than the generic layer arg for LayerOpen function:

```
int msOGRLayerOpen(layerObj *layer,  
                  const char *pszOverrideConnection);  
int msWFSLayerOpen(layerObj *lp,  
                  const char *pszGMLFilename,  
                  rectObj *defaultBBOX);
```

However, these are called from msLayerOpen with NULL args, so I think that proposed interface for this virtual table function should be fine.

2.1.4. IsOpen

```
int (*LayerIsOpen)(layerObj *layer);
```

2.1.5. WhichShapes

```
int (*LayerWhichShapes)(layerObj *layer,  
                       rectObj rect);
```

2.1.6. NextShape

```
int (*LayerNextShape)(layerObj *layer, shapeObj *shape);
```

2.1.7. GetShape

```
int (*LayerGetShape)(layerObj *layer, shapeObj *shape,  
                    int tile, long record);
```

2.1.8. LayerClose

```
int (*LayerClose)(layerObj *layer);
```

2.1.9. LayerGetItems

```
int (*LayerGetItems)(layerObj *layer);
```

2.1.10. GetExtent

```
int (*LayerGetExtent)(layerObj *layer,  
                     rectObj *extent);
```

2.1.11. GetAutoStyle

```
int (*LayerGetAutoStyle)(mapObj *map, layerObj *layer,
                        classObj *c, int tile,
                        long record);
```

2.2. New functions and/or fields for vtable

2.2.1. CloseConnection

This function is used to actually close the connection, in case that layer implements some kind of connection pooling by its own. If layer doesn't use any connection pooling, this function should be implemented as no-op. Caller should first call layer's "close" function, and finally at the very end `CloseConnection`.

The signature of function is

```
int (*LayerCloseConnection)(layerObj *layer);
```

And the main place where this function will be called, is `mapfile.c: 4822`

```
void msCloseConnections(mapObj *map)
```

This function is needed because e.g. POSTGIS is implementing this usage pattern at the moment `maplayer.c:599`

```
void msLayerClose(layerObj *layer)
...
/*
 * Due to connection sharing, we need to close the results
 * and free the cursor, but not close the connection.
 */
msPOSTGISLayerResultClose(layer);
```

2.2.2. SetTimeFilter

This function is used to create a time filter for layer. At the moment we have three special cases (`maplayer.c: 1635`):

- (a) POSTGIS with it's own function
- (b) Layers with backticks delimited expressions
- (c) Layers without backticks

The idea is provide a generic helper function,

```
int makeTimeFilter(layerObj *lp,
                  const char *timestring,
                  const char *timefield,
                  const bool bBackTicks)
```

And the actual layer's `SetTimeFilter` could use the above, or implement something totally different as POSTGIS is doing at the moment.

The signature for layer's vtable function is

```
int (*LayerSetTimeFilter)(layerObj *lp,
                          const char *timestring,
                          const char *timefield);
```

2.3. Extra functions to add to the vtable

2.3.1. FLTApplyFilterToLayer (mapogcfilter.c: 1084)

This is the main filter interface for layers. We will provide two helper functions, one for “SQL” case and the another for “non-SQL” case, and set all layers, except POSTGIS, ORACLESPATIAL and OGR to call directly this “non-SQL” version of this helper function (else-branch of if).

ORACLESPATIAL, POSTGIS and OGR could use “SQL” version of the helper function (actual if-branch) if the conditions for this are met, otherwise they will use Non-SQL version of the function.

2.3.2. layerObj->items allocation

There will be vtable function for allocation `items` and initialization for `numitems`. If layer has some special needs for these objects it can override default function.

The signature for function will be:

```
int (*CreateItems)(layerObj *)
```

which does the allocation and set `numitems` to correct value.

2.3.3. msCheckConnection (mapfile.c: 4779)

This API is deprecated. It is called only from `msMYGISLayerOpen`. We will not provide this functionality through vtable.

2.4. Interface functions for internal layers

We have to add some new interface functions to access layers.

2.4.1 Function interface to initialize internal layer type

We need a per layer type a function which will be called when the layer’s vtable has to be initialized. These functions will be

```
int msXXXInitializeLayerVirtualTable(layerObj *)
```

where XXX will be name of the layer. This function is called anytime when the vtable isn’t initialized and the layer is accessed at the first time.

2.4.2 Function interface to change the connectiontype of layer

To change the connection type of layer, it has to be done by function interface. Accessing directly `connectiontype` field is not supported anymore.

To change the connectiontype and to connect to the new layer, there will be following interface function

```
int msConnectLayer(int connectiontype,  
                  const char *library_str)
```

where `connectiontype` is the type of new layer, and `library_str` is the name of library which will provide functionality for this new layer. For internal layer types this second argument is ignored.

3. Remove unwanted interfaces

Frank Warmerdam proposed [FW1] that we remove all layer specific interface functions from `map.h`.

I see each “built-in” module such as `mapsde.c` providing a registration function such as “`msSDEInitializeLayerVirtualTable`” so that none of the layer type specific definitions need to appear in `map.h` any more.

Files and objects affected

This proposal will affect at least following files and objects:

- `map.h`
 - `layerObj` will contain new fields. There will be a new object `vtableObj` in the `map.h`.
- `maplayer.c`
 - Various changes, layer specific `switch`-statements will go away, `vtable` handling and layers `vtable` initialization will be added.
- `mapfile.c`
 - Cleaning of `msCheckConnection`
 - `Vtable` for `msCloseConnection`
- `mapogcfilter.c`
 - Remove layer-logic from `FLTApplyFilterToLayer`
- `mapXXX.c`, where `XXX` is the name of layer.
 - Add new initialization function
 - Add all new interface functions
 - Fix existing interface functions, if needed / add wrappers for `msOGRLayerOpen` and `msWFSLayerOpen`.

Backwards compatibility issues

This is binary and source code level backward incompatible change. The proposal will remove some previously public functions, and add new field(s) to the `layerObj` struct.

This proposal is not MapScript backward compatible, it will break scripts which change directly `connectiontype` field in `layerObj`.

The proposal is MAP-file backward compatible.

Implementation Issues

- Biggest problem is probably that the author has ignored or missed something by oversight which will show up during implementation. However, there is a prototype implementation of external plug-in architecture which works at the moment and is based on ideas presented in this proposal. So there is some real life experience that this architecture change is feasible thing to do.
- I also like to note that this proposal won't remove all layer specific code from MapServer e.g. WMF, WMS and GRATICULE are used as special cases from place to place.

Bug ID

Bug 1477

Voting history

Vote proposed by Jani Averbach on 9/19/2005, result was +4 (3 non-voting members).

Voting +1: Howard Butler, Frank Warmerdam, Yewondwossen Assefa, Daniel Morissette

Proposal passes and will move forward.

Open questions

- How do we like to expose layer's virtual table to the layers. We have at least two different routes to go:
 - expose it as a struct, layers will fill vtable pointers by accessing directly struct's field.
 - expose it as a complete opaque type, vtable pointers will be set by accessing functions `setLayerOpenFP`, `setLayerCloseFP` and so on.

The advance of second option is that this way we could easily add new functions to the struct if we refactor code more or found some logic which is ignored by oversight in this proposal.

- Are there any special issues with the raster query layer support which is handled via the layer API?

14.6.4 MS RFC 4: MapServer Raster Resampling

Date 2005/09/16

Author Frank Warmerdam

Contact warmerdam at pobox.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status adopted

Version MapServer 4.8

Id \$Id: ms-rfc-4.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

Additional resampling kernels will be implemented in the MapServer GDAL raster resampler code including “averaging”, “bilinear” and “cubic”.

Technical Details

- The new resampling options will be implemented in `mapresample.c` and will only be accessible for datasets processing through that mechanism (ie. GDAL raster formats, requires coordinate system information defined).
- The resampling kernel to use will be selected with one of the following. The default will be `RESAMPLE=NEAREST`, the current behavior.

```
PROCESSING "RESAMPLE=NEAREST"  
PROCESSING "RESAMPLE=AVERAGE"  
PROCESSING "RESAMPLE=BILINEAR"  
PROCESSING "RESAMPLE=BICUBIC"
```

- The `mapraster.c` code currently decides whether to invoke the “simple” GDAL renderer or the “resampling” GDAL renderer based on whether the projection seems to differ. It will also now check for the `RESAMPLE` processing option, and force use of `mapresample.c` if the resampling kernel select is other than `NEAREST`.
- Note that resampling kernels other than `NEAREST` can have a substantial effect on rendering performance. For this reason `NEAREST` will remain the default.

Mapfile Implications

All new options are selected via new `PROCESSING` options. There is no change in the mapfile syntax. There should be no compatibility problems with old mapfiles.

MapScript Implications

There are no additions or changes to the MapScript API. The new options are controlled via `PROCESSING` information on the layers which I believe is already manipulatable from MapScript.

Documentation Implications

The new processing options will need to be documented in the *Raster Data* (and possibly the *MAP* reference).

Test Plan

New test cases for each mode will be incorporated in `msautotest/gdal`.

Staffing / Timeline

The new feature will be implemented by Frank Warmerdam and completed by October 15th, 2005, in time for the MapServer 4.8 release. Implementation is generously funded by Tydac AG and managed by DM Solutions.

14.6.5 MS RFC 5: MapServer Horizon Reprojection Improvements

Date 2005/09/16

Author Frank Warmerdam

Contact warmerdam at pobox.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Adopted

Version MapServer 4.8

Id \$Id: ms-rfc-5.txt 8278 2008-12-23 21:34:31Z hobu \$

Purpose

To provide a practical solution to MapServer Bug 411 - Clipping Issue with Ortho Projection:

<http://trac.osgeo.org/mapserver/ticket/411>

Approach

The `msProjectShape()` function would be altered support reprojection of shapes where some, but not all, vertices fail to transform due to issues such as falling over the horizon. An iterative method would be used to preserve a reduced geometry approximating the portion that can be transformed.

Point Features

- Point features will be handled as they are now. They either transform successfully or not. Those that fail to transform are dropped.

Line Features

- Line strings would potentially be split into multiple line strings, if there were more than one discontinuous portion of a line string that transforms.
- Interpolated (horizon) points would be computed by transforming points along the line segment between the last successful point and the first unsuccessful point. A binary search algorithm will be used to find the closest point within our tolerance distance.
- Line strings where no vertices transform successfully will still be dropped.

Area Features

- Each ring will be treated distinctly, without regard to the other rings. This means there is no assurance that an inner ring will remain completely within the outer ring after clipping. In particular, where the horizon passes through an area and a hole in the area, it is likely that the hole edge will touch over even overlap the outer area edge a bit. Generally this should not matter much in the MapServer context since a simple scanline rasterization (winding number based) technique is used. So polygon “correctness” is not paramount.
- A similar technique to that for lines is used to interpolate the edge vertices when clipping is to be applied.
- For polygons there will be two (or possibly 4, 6, 8...) interpolated horizon points. These will be connected with a straight line segment. For very large polygons spanning a significant portion of the horizon (ie. dozens of degrees of arc of the horizon) this will result in part of the circle of the horizon getting clipped off. Optionally an iterative method could be applied to address this curvature issue (not priced into this proposal).

Tolerances

MapServer will need make a decision on how detailed to get when doing the iterative horizon determination. The tolerance will be fixed as half the size of a pixel in the current map rendering. In normal map drawing this should mean that the final rendering is effectively exact.

However, communicating the tolerance into `msProjectShape()` will require substantial work as the `msProjectShape()` function does not normally know anything about the map or current rendering information. To communicate this information to `msProjectShape()`, I will add a transformation options structure with this information (and possibly other flags controlling transformation) which will be passed into `msProjectShape()`. If NULL particular defaults will be used.

All appropriate uses of `msProjectShape()` will need to be updated to reflect this change, and to setup the transformation options information ahead of time. `layerObj` level PROCESSING options will also be provided to control (override) the default transformation options.

Caveats

1. msProjectShape() may produce slightly invalid polygons from a GIS point of view, though it shouldn't matter much for rendering purposes.
2. The straight line segment for "area horizon edges" will result in visible "clipping" of the horizon for very large polygons (ie. the Soviet Union).
3. This change is too disruptive to back port to the stable 4.6.x release, so it would only be available as part of CVS based MapServer builds (aka 4.7) until such time as the 4.8 release is made, likely in December or January.

Mapfile Implications

There will be no changes to the mapfile.

MapScript Implications

There will be no changes to MapScript.

Backward Compatibility Issues

Some maps that previously dropped incompletely reprojected features will now produce clipped features. I don't foresee this causing obvious problems other than a slight change (improvement!) in appearance.

Staffing and Timeline

The new feature will be implemented by October 15th, 2005, in time for inclusion in MapServer 4.8. It will be implemented by Frank Warmerdam with generous funding from TMC Technologies.

14.6.6 MS RFC 6: Color Range Mapping of Continuous Feature Values

Date 2005/09/27

Author Bill Binko

Contact bill at binko.net

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Proposed

Id \$Id: ms-rfc-6.txt 8278 2008-12-23 21:34:31Z hobu \$

Description: This proposal addresses the need to be able to easily map continuous feature values to a continuous range of colors. This RFC is the result of (and my interpretation of) the discussion that surrounded Bug #1305.

A preliminary patch has already been applied to Mapserver 4.6+ (before the RFC process was in place), however, there is little consensus on the format being used and there is no support for proper display of legends for classes using ColorRanges.

Background

This work started as a patch that I created to be able to quickly visualize data with a large range of values. In particular, I was wishing to map property values, and various ratios that could take on a large range of values. To me, the natural way to do this was to set a max value, a min value and what colors those mapped to. The patch I wrote had Mapserver do a linear interpolation of the value for each feature on to that color range.

The initial syntax for this feature simply added 5 new keywords to the STYLE block and looked as follows:

```
STYLE
  COLOR 60 60 60
  MINCOLOR 0 0 0
  MAXCOLOR 255 255 0
  MINVALUE 0.0
  MAXVALUE 300000.0
  GRADIENTITEM "sale_price"
END
```

After some discussion, the term Gradient was shown to be problematic. Also, the number of new keywords seemed high. After some discussion, the syntax was changed to this format:

```
STYLE
  COLORRANGE 0 0 0 255 255 0 # black to yellow
  DATARANGE 0.0 100.0
  RANGEITEM "foobar"
END
```

While this is still just a set of keywords under a Style, it seemed simpler and is now working in the Mapserver 4.6 branch.

Current Syntax Problems

Several people pointed out that the current syntax could be improved by:

1. Moving the new keywords into a block
2. Adding a METHOD keyword with the type of interpolation used ('linear' being the first defined type, 'logarithmic' being a potential second type, etc.
3. Adding an INTERVALS keyword that would limit the number of colors actually used by rounding values before interpolation.
4. Moving all of the keywords out of the Style block.
5. Allowing the ColorRange to be defined separately so that it can be reused

Proposed New Syntax

To meet the above needs, I propose the following new Block Syntax:

```
COLORRANGE
  RANGEITEM 'itemname' #required
  MINCOLOR 0 0 0 #optional - default = Black
  MAXCOLOR 255 255 0 #optional -default = White
  MINVALUE 0.0 #optional - default = 0
  MAXVALUE 100.0 #optional - default = 1
  INTERVALS 10 #optional - default = 0 (unlimited)
```

```
METHOD LINEAR #optional - default = 'LINEAR'
END
```

I propose that this block lives at the CLASS level. My reasons for putting it at the CLASS, rather than the LAYER (or above) level are as follows:

1. CLASS is the lowest level that can define a Legend entry (by using a named class)
2. Allows multiple COLORANGES to be applied to a single layer (i.e. Red->Yellow and Yellow->Green to make a contiguous Reg->Yellow->Green).
3. Allows “out of bounds” values to be highlighted separately (with a different CLASS).

(If we wish to provide this capability on the OUTLINECOLOR, then I would suggest we create a OUTLINECOLOR-RANGE block with identical format.)

Note: I am (and have always been) flexible on all of the keyword names and formats here. However, given the discussion that’s gone on around (and around) this, I thought I’d put a straw man up and start here.

Proposed Legend Format

I have posted a mockup of how I believe legends should look at

<http://trac.osgeo.org/mapserver/ticket/324>

The format only changes the height of the legend (which is already dynamic) so it should not have major layout implications. (Nit-picking about how tall the colorbar should be can be worked out during implementation.)

I have not started the legend support, and would like some help in this area. However, I do believe it is straightforward, and I will do it on my own if there are no volunteers.

MapScript Issues

As Sean Gillies mentioned in the discussion, this should be encapsulated in MapScript as a class. While I disagree with his putting it on the LAYER (see above), the rest of his suggestions all seem right in line.

I Propose a class named ColorRamp with the following read/write attributes:

```
Color minColor
Color maxColor
double minValue
double maxValue
String rangeItem
String method
int intervals
```

and two methods

```
Color findColor(double value)
double findValue(Color color)
```

ColorRamps can be obtained through an appropriate constructor or through a new (read/write) attribute on the ClassObj object:

```
ColorRange colorRange
```

Note: I will need help in adding these to MapScript as I do not have the SWIG experience to do it well.

Files affected

I will update this list as the RFC evolves, but right now, these are what I see:

- `mapfile.h` => Change Keywords
- `maplexer.l` => Change Keywords
- `mapfile.c` => process new Keywords
- `mapscript/swiginc/*.i` => various interface modifications to fit the above requirements
- `maplegend.c` => Add new Legend support
- `mapdraw.c` => update existing `ColorRange` code to use new keywords & add intervals and LOGARITHMIC method

Backwards compatabilty issues

Right now, certain code `_requires_` that there is a `COLOR` attribute set on any layer that is going to be displayed. Either this will need to be changed, or we will have to decide what that means if both a `COLOR` and a `COLORRANGE` are defined. One option is to use the `COLOR` for any values that are outside of the range.

Multiple Mapping Methods

The system will allow new color mapping methods to be added with as little effort as possible. If a new color mapping method uses only the keywords defined by this RFC, it should be a simple as:

1. Implement a function with the signature

```
int mappingFn(colorRangeObj\* range, shapeObj\* shape, colorObj \*color)
```

This function should use the `shape` and `range` parameters to determine the shapes color, and modify the `color` parameter accordingly.

1. Choose a unique method name ('linear', 'logarithmic', 'discrete') modify the method `msMapColorRamp()` to call its method when its method name is found on a `ColorRange` definition.

Note: Whether the `msMapColorRamp()` method uses if/then logic or dispatches by function pointer can be determined later. For now, I believe the simplest approach would be to move all of the mapping logic + all current methods into a `mapColorRange.c` file.

Bug ID

Currently this is being tracked by <http://trac.osgeo.org/mapserver/ticket/1305>

Voting history

None

14.6.7 MS RFC 7: MapServer CVS Commit Management

Date 2005/09/22

Author Frank Warmerdam

Contact warmerdam at pobox.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Adopted

Id \$Id: ms-rfc-7.txt 8278 2008-12-23 21:34:31Z hobu \$

Note: This RFC is superseded by [RFC 7.1](#)

Purpose

To formalize CVS commit access, and specify some guidelines for CVS committers.

Election to CVS Commit Access

Permission for CVS commit access shall be provided to new developers only if accepted by the MapServer Technical Steering Committee. A proposal should be written to the TSC for new committers and voted on normally. It is not necessary to write an RFC document for these votes ... email to mapserver-dev is sufficient.

Removal of CVS commit access should be handled by the same process.

The new committer should have demonstrated commitment to MapServer and knowledge of the MapServer source code and processes to the committee's satisfaction, usually by reporting bugs, submitting patches, and/or actively participating in the various MapServer forums.

The new committer should also be prepared to support any new feature or changes that he/she commits to the MapServer source tree in future releases, or to find someone to which to delegate responsibility for them if he/she stops being available to support the portions of code that he/she is responsible for.

All committers should also be a member of mapserver-dev mailing list so they can stay informed on policies, technical developments and release preparation.

Committer Tracking

A list of all project committers will be kept in the main mapserver directory (called COMMITTERS) listing for each CVS committer:

- Userid: the id that will appear in the CVS logs for this person.
- Full name: the users actual name.
- Email address: A current email address at which the committer can be reached. It may be altered in normal ways to make it harder to auto-harvest.
- A brief indication of areas of responsibility.

CVS Administrator

One member of the Technical Steering Committee will be designed the CVS Administrator. That person will be responsible for giving CVS commit access to folks, updating the COMMITTERS file, and other CVS related management. That person will need login access on the CVS server of course.

Initially Steve Lime will be the CVS Administrator.

CVS Commit Practices

The following are considered good CVS commit practices for the MapServer project.

- Use meaningful descriptions for CVS commit log entries.
- Add a bug reference like “(bug 1232)” at the end of CVS commit log entries when committing changes related to a bug in bugzilla.
- Include an entry in the HISTORY file for any significant change or bug fix committed in the main MapServer source tree. Make sure it is placed under the correct version heading and include bug numbers in these messages too.
- Changes should not be committed in stable branches without a corresponding bug id and HISTORY entry. Any change worth pushing into the stable version is worth a bugzilla bug and good HISTORY notations.
- Never commit new features to a stable branch: only critical fixes. New features can only go in the main development trunk.
- Only bug fixes should be committed to the code during pre-release code freeze.
- Significant changes to the main development version should be discussed on the -dev list before you make them, and larger changes will require a RFC approved by the TSC.
- Do not create new branches without the approval of the TSC. Release managers are assumed to have permission to create a branch.
- All source code in CVS should be in Unix text format as opposed to DOS text mode.
- When committing new features or significant changes to existing source code, the committer should take reasonable measures to insure that the source code continues to build and work on the most commonly supported platforms (currently Linux and Windows), either by testing on those platforms directly, or by getting help from other developers working on those platforms. If new files or library dependencies are added, then the configure.in, Makefile.in, Makefile.vc and related documentations should be kept up to date.

14.6.8 MS RFC 7.1: MapServer SVN Commit Management

Date 2008/07/02

Author Frank Warmerdam and Tom Kralidis

Contact warmerdam at pobox.com and tomkralidis at hotmail.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Adopted

Id \$Id: ms-rfc-7.1.txt 8278 2008-12-23 21:34:31Z hobo \$

Note: This RFC obsoletes *MS RFC 7: MapServer CVS Commit Management*.

Purpose

To formalize SVN commit access, and specify some guidelines for SVN committers.

Election to SVN Commit Access

Permission for SVN commit access shall be provided to new developers only if accepted by the MapServer Project Steering Committee. A proposal should be written to the PSC for new committers and voted on normally. It is not necessary to write an RFC document for these votes ... email to mapserver-dev is sufficient.

Removal of SVN commit access should be handled by the same process.

The new committer should have demonstrated commitment to MapServer and knowledge of the MapServer source code and processes to the committee's satisfaction, usually by reporting tickets, submitting patches, and/or actively participating in the various MapServer forums.

The new committer should also be prepared to support any new feature or changes that he/she commits to the MapServer source tree in future releases, or to find someone to which to delegate responsibility for them if he/she stops being available to support the portions of code that he/she is responsible for.

All committers should also be a member of mapserver-dev mailing list so they can stay informed on policies, technical developments and release preparation.

Committer Tracking

A list of all project committers will be kept in the main mapserver directory (called COMMITTERS) listing for each SVN committer:

- Userid: the id that will appear in the SVN logs for this person.
- Full name: the users actual name.
- Email address: A current email address at which the committer can be reached. It may be altered in normal ways to make it harder to auto-harvest.
- A brief indication of areas of responsibility.

SVN Administrator

One member of the Project Steering Committee will be designed the SVN Administrator. That person will be responsible for giving SVN commit access to folks, updating the COMMITTERS file, and other SVN related management. Initially Steve Lime will be the SVN Administrator.

SVN Commit Practices

The following are considered good SVN commit practices for the MapServer project.

- Use meaningful descriptions for SVN commit log entries.
- Add a ticket reference like “(#1232)” at the end of SVN commit log entries when committing changes related to a ticket in Trac.
- Include changeset revision numbers like “r7622” in tickets when discussing relevant changes to the codebase.
- Include an entry in the HISTORY file for any significant change or fix committed in the main MapServer source tree. Make sure it is placed under the correct version heading and include ticket numbers in these messages too.
- Changes should not be committed in stable branches without a corresponding ticket and HISTORY entry. Any change worth pushing into the stable version is worth a Trac ticket and good HISTORY notations.
- Never commit new features to a stable branch: only critical fixes. New features can only go in the main development trunk.

- Only ticket defects should be committed to the code during pre-release code freeze.
- Significant changes to the main development version should be discussed on the -dev list before you make them, and larger changes will require an RFC approved by the PSC.
- Do not create new branches without the approval of the PSC. Release managers are assumed to have permission to create a branch.
- All source code in SVN should be in Unix text format as opposed to DOS text mode.
- When committing new features or significant changes to existing source code, the committer should take reasonable measures to insure that the source code continues to build and work on the most commonly supported platforms (currently Linux and Windows), either by testing on those platforms directly, or by getting help from other developers working on those platforms. If new files or library dependencies are added, then the configure.in, Makefile.in, Makefile.vc and related documentations should be kept up to date.

Legal

Committers are the front line gatekeepers to keep the code base clear of improperly contributed code. It is important to the MapServer users, developers and the OSGeo foundation to avoid contributing any code to the project without it being clearly licensed under the project license.

Generally speaking the key issues are that those providing code to be included in the repository understand that the code will be released under the MapServer License, and that the person providing the code has the right to contribute the code. For the committer themselves understanding about the license is hopefully clear. For other contributors, the committer should verify the understanding unless the committer is very comfortable that the contributor understands the license (for instance frequent contributors).

If the contribution was developed on behalf of an employer (on work time, as part of a work project, etc) then it is important that an appropriate representative of the employer understand that the code will be contributed under the MapServer License. The arrangement should be cleared with an authorized supervisor/manager, etc.

The code should be developed by the contributor, or the code should be from a source which can be rightfully contributed such as from the public domain, or from an open source project under a compatible license.

All unusual situations need to be discussed and/or documented.

Committers should adhere to the following guidelines, and may be personally legally liable for improperly contributing code to the source repository:

- Make sure the contributor (and possibly employer) is aware of the contribution terms.
- Code coming from a source other than the contributor (such as adapted from another project) should be clearly marked as to the original source, copyright holders, license terms and so forth. This information can be in the file headers, but should also be added to the project licensing file if not exactly matching normal project licensing (mapserver/LICENSE.txt).
- Existing copyright headers and license text should never be stripped from a file. If a copyright holder wishes to give up copyright they must do so in writing to the foundation before copyright messages are removed. If license terms are changed it has to be by agreement (written in email is ok) of the copyright holders.
- When substantial contributions are added to a file (such as substantial patches) the author/contributor should be added to the list of copyright holders for the file.
- If there is uncertainty about whether a change is proper to contribute to the code base, please seek more information from the project steering committee, or the foundation legal counsel.

Voting History

Adopted on 2008/07/02 with +1 from PericlesN, DanielM, TamasS, JeffM, UmbertoN, SteveW, AssefaY, FrankW, TomK

14.6.9 MS RFC 8: Pluggable External Feature Layer Providers

Date 2005/10/26

Author Jani Averbach

Contact javerbach at extendthereach.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status adopted

Version MapServer 4.8

Id \$Id: ms-rfc-8.txt 8278 2008-12-23 21:34:31Z hobu \$

Purpose

The purpose of this proposal is provide a way to user actually plug-in external third party feature layer providers to the MapServer at run time.

Abstract Solution

Provide a way to user to tell via map-file which library to load for layer, then load this library on demand and cache it for later use and bind it to the running MapServer by layer's virtual table.

Technical Solution

- New CONFIG option `MS_PLUGIN_DIR` which, if set, tells the base path for plugins
- New MAP-file keyword `PLUGIN` with string argument. This keywords tells which library dynamically to load for this layer.

The plugins are loaded based on the following algorithm:

- If plugin string is missing `.so` or `.dll` extension, append it to the plugin string
- If `MS_PLUGIN_DIR` is set and plugin string is not an absolute path, prefix plugin string with `MS_PLUGIN_DIR`
- otherwise use plugin string directly

In general, the dynamic library loader will use system paths to seek appropriate plugin to load, if the path is not absolute.

- New connection type `PLUGIN`
- New field `char* plugin_library` in `layerObj` structure, this is the name of library to load for this layer.
- Function to get virtual table for requested layer. If the library isn't already loaded, it will be loaded on demand.

```
static const layerVTableObj *
getCustomLayerVirtualTable(layerObj *layer)
```

where `layerVTableObj` is the virtual table and `layer` is a custom layer. In case of error, function will return `NULL`.

- Function to get a function pointer from dynamic loaded library. This function will also load the library.

```
msGetDynamicLibrarySymbol(const char *Library,
                          const char *SymbolName)
```

This is implemented by GDAL project, and I have planned to use their implementation of this (`CPLGetSymbol`).

- Cache structure for already loaded libraries. This cache structure will consist of a full name/path of the library (provided by user via mapfile), and a function pointer to the virtual table initialization function. The size of cache will be fixed and will be same as the maximum amount of layers (200 at the moment). This is the maximum number of different custom layers for MapServer which could be loaded at the same time. This cache implementation is internal, so if it has to be make dynamically allocated, it is possible to do later without breaking interface.
- New lock item (`TLOCK_LAYER_VTABLE`) to protect library cache structure.

Files and objects affected

This proposal will affect at least following files and objects:

- `map.h`
 - `layerObj` will contain a new field, `char *plugin_library`.
- New lock token `TLOCK_LAYER_VTABLE`
- New files and objects for custom layer handling.

Backwards compatibility issues

This change is binary incompatible, but mapfile backward compatible. It will add a new keyword which is unknown for old MapServers.

Implementation Issues

None

Bug ID

Bug 1477

Voting history

Vote proposed by Jani Averbach on 10/26/2005, the initial result was +3 and after amending RFC, got +4 (3 non-voting members).

Voting +1: Frank Warmerdam, Steve Lime, Yewondwossen Assefa, Daniel Morissette

Proposal passed and will move forward.

Open questions

None

Working Notes

Plugin library has to implement function `PluginInitializeVirtualTable`

```
int (*pfPluginInitVTable)(layerVTableObj *,
                          layerObj *);
```

which is called during library loading. This function is responsible to populate `layerVTableObj *` virtual table. If this function leaves some function pointers to NULL in this virtual table, then default actions are used for these missing functions. The defaults are visible in function `maplayer.c: populateVirtualTable(...)`. The function must not populate directly `layerObj->vtable`, it have to use `layerVTableObj *` argument for this. The MapServer is holding `TLOCK_LAYER_VTABLE` lock during this function call.

14.6.10 MS RFC 9: Item tag for query templates

Date 2006/01/05

Author Steve Lime

Contact steve.lime at DNR.STATE.MN.US

Status Implemented

Version MapServer 4.10

Id \$Id: ms-rfc-9.txt 8278 2008-12-23 21:34:31Z hobu \$

Description: A the moment processing of attributes exposed via query templates is limited to a couple forms of escaping. This is too limiting. It would be much nicer to be able to apply a format to an attribute (e.g. truncate decimals or add a prefix) or detect when it is NULL or empty and present an alternative representation. This RFC describes a simple scheme to allow for extended processing of attributes before sending to the a users.

This would involve adding a new “item” tag to the template processor.

Files affected

- `maptemplate.c` => add tag processor, update query template processor to use tag processor.

Backwards compatibility issues

N/A, new functionality. Existing methods for exposing an attribute (e.g. `[attribute name]`) would continue to function, although we might mark that as deprecated at some point.

Implementation Details

The “item” tag would have the following properties.

- `name`: the name of the attribute to be exposed
- `format`: a simple string containing the token(s) `$value`, default format is the result of any processing

- `nullformat`: string to return if value is NULL or empty (`length=0`)
- `uc`: convert string to upper case
- `lc`: convert string to lower case
- `substr`: perform a perl-like substring operation by providing offset, and optionally length (un-implemented as of 3/13/07)
- `commify`: add commas to a numeric string (12345 would become 12,345), note, supports only North American notation at the moment
- `escape`: what type of escaping to do (default is html), permissible values would be html, url, none
- `precision`: number of decimals to retain after the decimal point
- `pattern`: regular expression that must validate (against the column value) to process the tag

We could add simple case statement support at a later date but the above would more than meet my needs at the moment.

Examples

Display no decimal places, and commify:

```
[item name=foo commify=true precision=0] - 12345.6789 output as 12,345
```

Conditional display an item if its value contains 'bar':

```
[item name=foo pattern=bar]
```

Escape a value for inclusion in a URL, and convert to upper case:

```
[item name=foo uc=true escape=url] - "hello world" output as "HELLO%20WORLD"
```

Apply a custom format to an item, and display a message if the item is empty:

```
[item name=foo format="foo=$value" nullformat="foo is not found"]
```

Notes

Item tags are processed after other column tags (e.g. `[foo]`) so you can include substitution strings within the item tag itself.

Bug ID

1636

Voting history

+1: Lime, Morissette, Butler +0: Gillies, Warmerdam

14.6.11 MS RFC 10: Joining the Open Source Geospatial Foundation

Date 2006/02/06

Author Howard Butler

Contact hobu.inc at gmail.com

Author Frank Warmerdam

Contact warmerdam at pobox.com

Author Steve Lime

Contact sdlime at comcast.net

Status passed

Id \$Id: ms-rfc-10.txt 8278 2008-12-23 21:34:31Z hobu \$

Abstract

The Open Source Geospatial Foundation (OSGeo) presents a unique opportunity for MapServer in that it can provide an umbrella organization that can provide benefits and possibilities that a project by itself simply cannot provide. This RFC will outline what those benefits can be to the project, define MapServer's role in participating, and describe the unique process of how we propose to decide to join.

MapServer's Participation in the Foundation

The OSGeo bootstrap meeting of February 4th, 2006 was attended by Steve Lime, Frank Warmerdam, Pericles Nacionales, Howard Butler, Tom Burk, Dave McIlhagga and others from the MapServer community. By their participation in this meeting, these individuals are part of the initial group of members in the OSGeo foundation. A thorough, thoughtful, and stimulating discussion about how the OSGeo foundation can help to alleviate issues that an individual project cannot overcome took place. Many projects that were represented at the meeting, including GDAL, OSSIM, MapGuide Open Source, MapBuilder, and GRASS decided immediately to participate in the foundation. The folks represented by MapServer at the meeting stated that they would also like to participate, contingent on approval by both the MTSC and the community at large.

In this RFC we are proposing that MapServer join the foundation as one of the founding projects as this is a unique opportunity for MapServer to influence the direction that the foundation will take, and in the end get a foundation that will better suit its specific needs.

Expected Benefits of OSGeo to the MapServer Project

OSGeo provides a unique and necessary structure for the MapServer project to solve some of the issues an Open Source project by itself cannot. One of the most important ones is the idea of an umbrella organization that can take on issues like infrastructure, branding and visibility, outreach, solicitation of sponsorship, and more formal project organization. While the MapServer project has done some of these things by itself in some form over the years, a formalized structure that has its mission to do so provides a more attractive solution. Here are some expected benefits, described by [Schuyler Erle](#), who also attended the meeting:

The Open Source Geospatial Foundation, or OSGeo for short, will seek to provide roughly the same kind of function for the F/OSS GIS community that the Apache Software Foundation provides for the Apache development and user communities, with the primary difference being that, where Apache started as a single project and then branched out, the OSGeo Foundation is attempting to weld together the overlapping but sometimes disparate interests of different projects with different communities. The Foundation will hopefully serve as an outreach and advocacy organization for the community; a forum for improving cross-project collaboration; a unified professional front for large government and corporate users; a source of shared infrastructure, like code and documentation repositories; and as a legal entity to help protect developers and users of Open Source geospatial software against greedy patent lawsuits or unscrupulous license infringements. In general, the object of the Foundation will be, in the words of Mark Lucas of OSSIM, to "help us do what we love" – which, for most of us, is building useful tools for digital cartography and geospatial analysis, and solving interesting problems with them.

Deciding to Join

The MTSC's charter, specified in *MS RFC 1*, clearly states that it is only to concern itself with technical matters in the project. In our opinion, deciding to join OSGeo presents a special case in that it affects both the technical and non-technical. In this case we propose that the MTSC go through its decision process, and if approved, the community will be solicited to provide their input via a non-anonymous website poll. If both groups are in agreement - via the normal approval process from the MTSC and a majority from the community - the motion to join OSGeo will be considered passed.

Considerations

As MapServer joins the foundation some changes are anticipated in the project. There will need to be a copyright review of the existing code base, ensuring that it is all legitimately contributed under the existing license. All MapServer committers will need to sign some sort of committer agreement providing assurance they are not adding encumbered code. MapServer may have to consider moving its project infrastructure (CVS, website, lists, etc.) to the foundation at some point. We will also need to establish a MapServer Project Committee within the foundation. This may just be the MTSC or it may be broader, including other stakeholders.

For the time being the MTSC will continue to operate under *MS RFC 1: Technical Steering Committee Guidelines* if the motion to join OSGeo is passed. Additional RFC(s) will address any changes in process deemed necessary as a result of joining the foundation.

Voting history

Passed +7. February 6th, 2005.

14.6.12 MS RFC 11: Support for Curved Labels

Date 2006/02/09

Author Benj Carson

Contact benjcarson at digitaljunkies.ca

Author Stephen Lime

Contact sdlime at comcast.net

Status passed

Id \$Id: ms-rfc-11.txt 8278 2008-12-23 21:34:31Z hobo \$

Overview

One of the features most frequently asked for are labels that follow along linear features. This RFC describes an initial implementation of this feature.

Technical Details

The proposed solution has a couple of primary goals:

- isolate virtually all computations and data storage into a minimum number of functions and structures.
- integrates easily into the existing labelCacheObj structure and label cache processing routines.

A single, new function- `msPolylineLabelPath()` serves as the sole computational function for this new functionality. Like the existing `msPolylineLabelPoint()` function it takes an input feature and annotation string and computes a labeling position. However, instead of computing a single point (and optionally, angle) it computes a label point and angle for each character in the annotation string. The computation results are returned in a new structure called a “`labelPathObj`” that looks like:

```
typedef struct {
    multipointObj path;
    shapeObj bounds;
    double *angles;
} labelPathObj;
```

The function will return NULL if a curved label is not appropriate for the feature in question so traditional labeling can take place (for example, if the feature has only 2 points a curved label is not necessary). The curved label’s bounding polygon will be calculated in this function as well and stored in the “`bounds`” member of the `labelPathObj` structure.

In order to get the `labelPathObj` into the label cache it will be necessary to do 2 things:

- extend `labelCacheMemberObj` to optionally reference a `labelPathObj`
- extend the function `msAddLabel` to take a `labelPathObj` in addition to the parameters it already accepts

Since each `labelPathObj` will contain the boundary for the curved label, it will be ready to use with the existing label cache rendering code.

The only addition to the label cache rendering is code to detect when a text path should be rendered instead of a traditional label. Driver specific code to render a text path will have to be written but in general this is trivial and just involves calling the normal text rendering code once for each character in the path.

Mapfile Implications

It is proposed that we simply extend the `labelObj` `ANGLE` parameter. Currently it takes an angle (given in degrees) or the keyword `AUTO`. We suggest adding support for the keyword `FOLLOW`. This would set a new `labelObj` member, `anglefollow`, to `MS_TRUE` (and also `angleauto` to `MS_TRUE` as `ANGLE FOLLOW` implies `ANGLE AUTO` if a curved label is not appropriate).

Support for Non-GD Renderers

Presently all MapServer output renders use the contents of the label cache, which is basically render agnostic. This will not be the case any more. The placement computations necessary to support curved labels do leverage font metrics derived from the GD/Freetype interface. It may well be possible for the SWF, PDF and SVG renders to leverage even the GD-based curved labels, however it is probably best to consider this a raster-only output feature in this implementation. If font metrics support for other renderers is developed in the future then this feature can be easily extended to support them.

Bug Tracking

Bug 1620 has been setup to track this feature addition.

Voting History

+1: Lime, Assefa, Nacionales, Warmerdam, Morissette

14.6.13 MS RFC 12: C code Unit tests

Date 2006/02/28

Author Umberto Nicoletti

Contact umberto.nicoletti at gmail.com

Status Draft

Id \$Id: ms-rfc-12.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

Unit tests are a simple but effective way of checking progress and code correctness while programming. Mapserver already has unit tests for python MapScript and a functional testing suite but lacks unit tests for the C code which represents the core of its functionality.

This rfc outlines how unit tests can be (gradually) added to the usual development cycle of MapServer, without pretending that developers switch to extreme programming or adopt test driven development.

Issue #1664 on bugzilla and mapserver-dev will be used to track code samples and discussion.

Example

A patch has been attached to issue #1664 on bugzilla which implements the basic infrastructure and some unit tests to demonstrate functionality. The patch was created in under 6 hours by a single person, who is not a frequent C programmer. This is meant to prove that unit tests do not take away much time from developers and that in general the benefit is worth the cost.

The example already supports all the usual preprocessor flags (USE_FLAGS like #ifdef PROJ) so only the relevant tests will be compiled and run. This is necessary to avoid test failures due to a missing feature (as happens with Python unit tests).

Unit testing software

This rfc recommends the use of the CUnit unit testing library for the C language, version 2.0.x. The configure script must be updated to reflect this new dependency and the following or a similar text will be displayed when cunit is not found: "cunit not found, C unit testing disabled."

When cunit is available unit tests could be run by default at the end of the compile phase: this will give users some more confidence in the software they just compiled (most of them are linux newbies and have no experience with programming) and could help in catching obvious bugs (like a typo while committing to cvs).

Unit tests are placed in the cunit subdirectory of the MapServer source and distributed with all future releases. Unit tests files should be named with respect to the original MapServer source file they test plus the addition of the _tests suffix: for instance unit tests for the mappool.c file should be placed in a file called mappool_tests.c. The files can have an alphanumeric code added to the suffix to keep them conveniently short (like mappool_tests_1.c, mappool_tests_1b.c or mappool_tests_umberto.c).

Usage recommendations

Developers are not required to write unit tests but when they do we suggest to follow this checklist.

Developer implementing a new feature

- she will try to reproduce the bug with a unit test; if this is not possible or seems difficult she should consider asking on mapserver-dev. The unit test must fail at this stage (if written)
- she fixes the bug and verifies that the test written at the previous item now passes (if written)
- she verifies that also all other tests pass and only after she commits

Developer adding functionality

- she writes a test that verifies the new functionality (this can also be done afterward, as we have relaxed requirements)
- she writes the code implementing the desired functionality
- she now verifies that the test passes
- she verifies that all other tests pass and only after she commits

General

She will also use the usual preprocessor flags to ensure only the relevant tests for the current configuration context will be compiled and run. The example attached to issue #1664 already implements this feature.

Testing specific functionalities

Some aspects of MapServer like database connections and gd rendering are inherently difficult to test. This section provides guidance on how to deal with them in unit tests and will be expanded as new or better solutions are devised.

Database connections

Database connections generally require a specific setup so that expected tables, data and possibly other structures are in place. While this should be a long term goal in the short term unit tests could be limited to comparing the sql query against the expected one.

This quite likely requires a refactoring of current PostGIS code.

In the future a minimal setup script should be provided to create and populate the database for the user (if she desires to do so).

GD (and others) rendering engines

Taking inspiration from the perl GD module a set of images must be produced and then compared against those produced by MapServer. This kind of operation is already performed by the msautotest suite, so effort should rather go toward improving that instead of implementing yet another gd test suite.

WFS/WMS support

Note: The author is not an expert here

In this case unit tests can be used to verify that for certain requests (possibly using the test data supplied with MapServer) the string returned by MapServer equals the expected one.

A refactoring of existing code could be necessary.

Mapscript

Unit tests should be developed by the MapScript maintainers by following the guidelines given here.

Running unit tests and functional tests (Continuous integration)

Not part of unit tests for now, but useful in perspective

Effort should be put toward developing a build system capable of

1. testing the overall build/test of MapServer and of the various MapServers with different configure options
2. and integrating those results with the msautotest suite.

14.6.14 MS RFC 13: Support of Sensor Observation Service in MapServer

Date 2006/02/21

Author Yewondwossen Assefa

Contact yassefa at dmsolutions.ca

Status passed

Id \$Id: ms-rfc-13.txt 8280 2008-12-24 02:56:36Z hobu \$

Overview

This is the a first attempt to support part of the OGC Sensor Web Enablement (SWE) in MapServer. The different specifications developed around the SWE are :

- Sensor Observation Service (SOS) : provides an API for managing deployed sensors and retrieving sensor data.
- Observation & Measurement: Information model and encoding for observations and measurements.
- Sensor Alert Service : A service by which a client can register for and receive sensor alert messages. The service supports both pre-defined and custom alerts and covers the process of alert publication, subscription, and notification.
- Sensor Model Language : Information model and XML encoding for discovering, querying and controlling Web-resident sensors.
- Sensor Planning Service : A service by which a client can determine collection feasibility for a desired set of collection requests for one or more mobile sensors/platforms, or a client may submit collection requests directly to these sensors/platforms.
- Transducer markup Language (TML) : General characterizations of transducers (both receivers and transmitters)

- Web Notification Service : Executes and manages message dialogue between a client and Web service(s) for long duration asynchronous processes.

The intention here is to support the SOS mandatory operations. Please refer to the OGC site (<http://www.opengeospatial.org/functional/?page=swe>) for more details on the SWE initiatives.

User Interface

From the user perspective there will be an SOS interface will offer the three core operations (GetCapabilities, GetObservation and DescribeSensor). A full description of what could be available is presented in Annexe A : Sensor Observation System (SOS) support in MapServer.

Changes in MapServer

- All the development will be localized into a mapogcsos.c file.
- There will be additions to the mapows.c/h file to integrate the dispatch of the requests.
- In mapgml.c (function msGMLWriteWFSQuery), extract the loop that writes features (gml:featureMember) into a separate function so the the GetObservation can also use it to output the results.
- The SOS capability will be available when MapServer is built using the flag USE_SOS.

Mapscript implications

There are no special implications for the MapScript module

Additional libraries

There will be an attempt to use the libxml2 (<http://xmlsoft.org/>) library when generating the GetCapabilities document. Decision to go ahead will be based on ease to use and speed of output.

Testing

It is proposed that automatic tests with map/data/expected results be added into the msautotest project to test the GetCapabilities and GetObservation requests.

Bug Tracking

Bug 1710 : <http://trac.osgeo.org/mapserver/ticket/1710>

Voting History

- +1 : Assefa, Warmerdam, Nacionales
- +0 : Morissette
- -0 : Gillies

Note: discussions, concerns are available in the mapserver-dev list (Feb 2006 RFC 13 : SOS support)

Annexe A : Sensor Observation System (SOS) support in MapServer

This is a first attempt to define what will be supported in MapServer to be able to deploy a Sensor Observation System (SOS)

Specifications and useful links used :

- Sensor Observation Service (SOS) (OGC 05-088r1, Version 0.1.1)
- Observation and Measurement (OGC 05-087r1, toward Version 1.0)
- Sensor ML : Sensor Model Language (ML) OGC 04-019r2
- <http://www.opengeospatial.org/functional/?page=swe> [svn link](https://svn.opengeospatial.org:8443/ogc-projects/ows-3/schema4demo/) for members : <https://svn.opengeospatial.org:8443/ogc-projects/ows-3/schema4demo/>

SOS provides several operations divided into core mandatory operations (GetCapabilities, DescribeSensor and GetObservation) and optional transactional and enhanced operations. The first implementation of SOS in MapServer will only address the core operations

1. GetCapabilities Request

The GetCapabilities request will use the following parameters :

- Request : fixed at GetCapabilities
- Service : fixed at SOS

2. GetCapabilities returned document

Attached at the end of the document examples of a GetCapabilities document. The following elements are SOS items included in the capabilities document, with an equivalent MapServer implementation

- ServiceIdentification (all elements are extracted from metadata at the web level)
 - Title : extracted from a metadata at web level ows/sos_title. Same concept as wms/wfs
 - Abstract : metadata ows/sos abstract
 - ServiceType : Fixed to SOS
 - ServiceType version : Fixed to 0.3
 - Fees : metadata Ows/sos_fees
 - Access Constrains : metadata : Ows/sos_constrain
- ServiceProvider (all elements are extracted from metadata at the web level, using an equivalent name as the SOS element)
 - ProviderName :
 - ProviderSite
 - IndividualName
 - PositionName
 - Voice
 - Facsimile
 - DeliveryPoint
 - City
 - AdministrativeArea
 - PostalCode

- Country
- ElectronicmailAddress
- EndAddress
- OnlineResource
- HoursOfService
- ContactInstructions
- Operation Metadata : This part of the capabilities defines the operations that will be supported which are GetCapabilities and GetObservation. For more information refer to <https://svn.openeospatial.org:8443/projects/ows-3/schema4demo/ows/1.0.30/owsOperationsMetadata.xsd>
 - Operation : GetCapabilities
 - * Operation Name : Fixed at GetCapabilities
 - * HTTP : Connect point URL extracted. Only the Get request method is supported
 - * Parameter : includes name and version. We could use the parameters to propagate the name of the service SOS and the version. Other parameters may be added if needed.
 - Operation : GetObservation
 - * Operation Name : Fixed GetObservation
 - * DCP (HTTP) : extracted from a metadata
 - * Parameter : We could use this to propagate the parameters needed when doing a GetObservation request (Offering, eventTime)
 - Operation : DescribeSensor
- Filter Capabilities

The Filter Capabilities that will be supported are the same ones that are currently supported in MapServer (See *WFS Filter Encoding* for more info) : Spatial Capabilities, Logical Operators, Comparison Operators

There is a mention in the specifications of ogc filter temporal capabilities, but I could not locate the exact definition of it. In any case,

- SOS Contents (Observation Offerings)

As explained in SOS specifications (section 6.2) , “ ...An observation offering is also analogous to a layer in a Web Map Service because each offering is intended to be a non-overlapping group of related observations. Each Observation Offering is constrained by a number of parameters including sensor system that report observation, Time, phenomena, geographical region ... “

In MapServer an Offering will be represented by a group of layers using mapserver’s group parameter. The metadata associated with a group (offering will be taken from the first layer of the group)

The following properties would be set at a group level.

- Standard Properties
 - * id : Unique Offering Identifier. Mandatory
 - * name : name used with the offering. Optional
 - * description : description of the offering. Optional

- Bounded By : used to define the geographical boundaries of the Offering. It should be extracted from a metadata. Mandatory
- EventTime : used to define a valid time range for the offering. It should be extracted from a metadata. Mandatory
- Procedures : series of URLs reference to one or more systems that supply observations in the offering. It should be extracted from a metadata. Mandatory
- Observed property : Observables/Phenomena that can be requested in this offering.

Two specializations are identified in the specifications:

- Constrained : modifies base phenomenon by adding a single constrains (ex surface water temperature add a constrain that depth is between 0, -0.3)
- Compound which is either a composite (a set of component phenomena that may or may not relate to each other) or a phenomenonSeries that applies one or more constraintList to the base phenomenon)

There is no clear cut indication which representation would be the more natural for MapServer but if we consider the group/layer/attribute combination, we can see that a group of layers could represent an offering, a layer would be an observed property (or phenomenon) and the attributes would be the composite phenomenon defining the phenomenon.

The capabilities document will include CompositePhenomenonType element with an mandatory id element identifying the phenomenon and optional elements such as name and component.

- Feature Of Interest

The definition given in the specification is : This is a single feature or a collection of features that represents the object on which the sensor systems are making observations. In case of in-situ sensor this may be a station to which the sensor is attached or the atmosphere directly surrounding it. For remote sensors this may be the area or volume that is being sensor. It is represented by GML feature type and is expected to include bounding box extents.

In our case here, this would be equivalent to the “bounded by” element defined earlier.

Note that in the implementation of MapServer, It is assumed that geographical information used to represent the individual sensors represent the feature of interest of the sensor. This is a requirement to be able to do spatial queries.

- Result Format

MIME type of the result that will be returned to a GetObservation request. Fixed to text/xml;subtype=”om1.0.30”

3. GetObservation Request

The GetObservation request used to retrieve observation data will be supported using the Get method. Post method will not be supported in the first implementation.

Here are the parameters that will be supported and their definitions :

- Offering : Equivalent to the Offering Id identified in the capabilities (Mandatory)
- eventTime : single time or time period. This will be used as a Time filter to do the queries using an identified time attribute. (Optional)
- observedProperty : Identifies the layer in MapServer (Mandatory)
- featureOfInterest : Additional geographical filter using a bbox. (Optional)
- Result : Will be used to filter using the OGC Filter supported capabilities.

4. Get Observation Response

An observation response should contain the following information :

- Information describing the Offering
- Valid time (instances or period)
- Description of the phenomenon (like the offering name)
- location and feature of interest for the offering
- The result associated with the offering

In the case of MapServer implementation, what is proposed is to be returned an observation collection reflecting the query results. Here is the different elements returned :

- name : The offering unique identifier
- description : Description of the offering
- time : Valid time instance or period
- featureofinterest : Geographical extents covered of the offering
- Member : This is repeated for all the observations returned. The following are the parameters included for each member
 - observed property : the phenomenon observed
 - location : geographical coordinates
 - procedure
 - result : Result of the observation. In the first implementation It is proposed that the gml:feature member is returned. This gives the possibility to return one/more attribute values in an easily manipulated format. The will be equivalent to a gml:feature member returned in wfs.

5. Describe Sensor

The Describe sensor request uses two parameter that are SensorId (Mandatory) and an optional output-Format.

In this phase the DescribeSensor will use a metadata of URL type set on the layer and relay the request. There won't be any SensorML output generation done in MapServer in this implementation.

6. Examples

- <http://vast.uah.edu:8080/ows/weather?request=GetCapabilities>
- [http://vast.uah.edu:8080/ows/weather? request=GetObservation&offering=WEATHER_DATA&time=2004-04-01T05:00:00Z/2004-04-01T06:00:00Z&format=application/com-xml](http://vast.uah.edu:8080/ows/weather?request=GetObservation&offering=WEATHER_DATA&time=2004-04-01T05:00:00Z/2004-04-01T06:00:00Z&format=application/com-xml)

14.6.15 MS RFC 14: Relative Coordinates for INLINE features

Date 2006/04/10

Author Steve Lime

Contact steve.lime at DNR.STATE.MN.US

Status Complete

Version MapServer 4.10

Id \$Id: ms-rfc-14.txt 8278 2008-12-23 21:34:31Z hobu \$

Description: Current it is possible to have features with pixel coordinates and to draw them by setting TRANSFORM FALSE in a layer definition. However the coordinates are relative to the upper lefthand corner of the image (0,0) which makes it impossible to anchor things like copyright statements to the other corners of the images if the image size can change (e.g. via WMS).

The proposed solution extends the behavior of the LAYER TRANSFORM parameter that would tell MapServer to use an alternative origin that the UL corner of the image.

C Structural Changes

None. Existing structure, members and constants would be utilized. The position enumeration should probably be given a different starting value to avoid conflict with variables like MS_TRUE and MS_FALSE.

Mapfile Changes

This functionality is really geared towards inline features. However, I'd like to keep the door open to support features from any datasource. The proposed change would extend the use of the LAYER TRANSFORM parameter. Currently it takes values TRUE or FALSE (default is TRUE). I propose extending to also take any of the standard explicit position values. So for the typical inline feature use you'd see a layer like:

```
LAYER
  NAME 'copyright'
  TYPE POINT
  TRANSFORM LL
  FEATURE
    POINTS 10 -10 END
    TEXT 'Copyright &copy; MNDNR'
  END
  CLASS
    ...
  END
END
```

Within MapScript the syntax would be similarly simple:

```
$layer->{transform} = $mapscript::MS_LL;
... draw as normal ...
```

Files affected

- map.h => change starting value of the positions enumeration
- mapfile.c => add detection of the additional TRANSFORM values
- mapprimitive.c => add a new offset shape function that would take the map hight, width and shapeObj as input.
- mapdraw.c => update shape drawing code to use the new function (basically an else condition for all the if (layer->transform) checks).

Testing

- Python suite: none needed
- MsAutoTest suite: a mapfile testing all 9 positions would be developed

Backwards compatabilty issues

N/A, new functionality. A value for TRANSFORM of FALSE implies UL... It should be noted that by changing the starting value for the position enumeration there is the possibility of breaking scripts that refer position by integer (poor programming practice). I would expect this to be a remote possibility and worth the risk.

Bug ID

1547

Voting history

Passed

14.6.16 MS RFC 15: Support for thread neutral operation of MapServer/MapScript

Date 2006/05/01

Author Tamas Szekeres

Contact szekerest at gmail.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Draft

Version Not assigned yet

Id \$Id: ms-rfc-15.txt 8278 2008-12-23 21:34:31Z hobu \$

1. Overview

By this time the core MapServer code has become widely used by many of the different application models. The extended usage of MapScript causes the possibility that the execution of the code is carried out by multiple threads, standing a fair chance portions of the code being called by the different threads simultaneously. The threading model of the application may fall beyond the control of the MapServer code, at the worst case even the host process of the application is beyond the programmer's scope. For example a web mapping application may be executed by a pre-existing host process ensuring to serve the user requests and calling the MapServer code by different threads from a pool of threads. The usage of process-wide global variables and resources within MapServer may require the need to serialize of the access to the variable by the multiple threads by using locks. The current approach uses process-wide locks to ensure the safe access to these variables causing performance degradation in multithreaded environments. Currently this behavior may be controlled by the USE_THREAD compilation switch.

2. Purpose

The purpose of this change is to get a significant performance boost by removing "big locks" from the code. It would add a significant improvement of the applications having high number of multiple threads executed simultaneously. This activity could bring in a more feasible support for some application models not really aimed at now like Microsoft ASP.NET. May result in higher clarity of the code by determining the code segments affected by multiple threads.

2. General principles of the solution

This RFC is in a special situation since wide area of the code is to be reviewed and extensive developer support is needed. The proposal will reach the proposed state only if all of the open issues are classified and properly handled.

Changing of the modules of MapServer should be in accordance of the primary maintainer. The maintainer may reserve the right to commit the changes proposed by this RFC during the implementation phase. Otherwise the primary author Tamas Szekeres will be responsible to make the changes.

The problems of the current implementation and the proposed changes will be enumerated later. To handle these issues the following options will be considered ordering by the expediency. The latter is the less expedient.

2.1 Not changing the code (Considering as safe without locks)

Some of the process-wide global variables store invariant data - like enumerations - having been initialized at compilation time. Since these variables can be simultaneously read by multiple threads there is nothing to be done and even the locks should be removed - if exist - ensuring the thread isolation to be realized. Might be redeclared as “static const”.

2.2 Retaining the variable, but reconsidering the initialization code

Some of the global variables store invariant data but the initial value is assigned at run time during the module initialization phase. In this situation we should ensure that the initialization will be done before the subsequent access is receivable to prevent from the possibility of race conditions. `msSetup()` is designated as a suitable place to make module level initialization of the variables. Mapscript language bindings are encouraged to provide calling `msSetup` automatically at module startup. The current locking strategy should be reviewed and the locks should be restricted to the initialization code.

2.3 Rewriting the code to eliminate the need of the global variable

If the global variable could be eliminated by changing the code structure we will consider to make these changes. The locks related to these variables should be removed.

2.4 Using thread local variable instead of the global one

When the usage of the global variable cannot be eliminated easily we will consider to use thread local variables instead. To implement thread local variables, theoretically we have at least 2 options to do:

1. By modifying the variable declaration we could use storage specifiers like Microsoft specific “`__declspec(thread)`” or “`__thread`” on the POSIX environments.
2. By implementing the allocation of the thread local memory utilizing the platform dependent TLS APIs like the Windows TLS or the unix Pthreads implementation.

The first one is not applicable for Mapserver since the Microsoft implementation does not allow delay loading of the dll containing variables declared with `__declspec(thread)`.

The second approach is applicable, however rewriting the existing code may be more difficult. GDAL has a sample implementation for it might be taken over.

2.5 Not changing the code (Marking as safe with locks, will be reconsidered later)

Some of the MapServer code and/or the related external libraries may not be modified easily and will be protected by locks in this phase of the realization. These issues will be kept open and the solution for the thread isolation might be reconsidered later. The open issues will be enumerated and documented along with the MapServer source files and the thread safety FAQ

http://mapserver.gis.umn.edu/docs/faq/thread_safety

2.6 Not changing the code (Marking as unsafe, will be deprecated and unsupported)

Having lack of support of portions of the existing code may keep from changing the code easily. These portions of the code will not be modified in this phase of the realization. These issues will be kept open and might be reconsidered later. The open issues will be enumerated and documented along with the MapServer source files and the thread safety FAQ.

3. Issues of the Mapserver/Mapscript code

This chapter will enumerate the sections of the existing code should be reconsidered according to the options mentioned previously. The line numbers may slightly change according to the developers work.

```
epplib.c(47):static int REVERSE; /* set to 1 on bigendian machines */
```

This variable is set in eppreset using the following code { union { long i; char c[4]; } u; u.i=1; REVERSE=(u.c[0]==0); } should be set during the initialization or by the makefile as a predefined constant. Also the MapServer configure script may include byte order detection. This constant might be used by other modules.

SDL: This constant is limited to epplib.c...

```
mapcpl.c(57):static char      szStaticResult[MS_PATH_BUF_SIZE];
```

FrankW: In fact this code carried over from GDAL was later remodelled in GDAL. It looks like msGetBasename() is only used in a few places and we should remodel those to avoid using a static buffer.

```
maperror.c(110):static char *ms_errorCodes[MS_NUMERRORCODES] = {"",
maperror.c(154):    static errorObj ms_error = {MS_NOERR, "", "", NULL};
maperror.c(169):static te_info_t *error_list = NULL;
maperror.c(552): static char version[1024];
maperror.c(651):    static char nonblocking_set = 0;
```

TODO

```
mapfile.c(184):static char *msUnits[8]={ "INCHES", "FEET", "MILES", "METERS", "KILOMETERS", "DD", "P
mapfile.c(185):static char *msLayerTypes[8]={ "POINT", "LINE", "POLYGON", "RASTER", "ANNOTATION", "QU
mapfile.c(186):char *msPositionsText[MS_POSITIONS_LENGTH] = {"UL", "LR", "UR", "LL", "CR", "CL", "UC
mapfile.c(187):static char *msBitmapFontSizes[5]={ "TINY", "SMALL", "MEDIUM", "LARGE", "GIANT"};
mapfile.c(188):static char *msQueryMapStyles[4]={ "NORMAL", "HILITE", "SELECTED", "INVERTED"};
mapfile.c(189):static char *msStatus[5]={ "OFF", "ON", "DEFAULT", "EMBED"};
mapfile.c(191):static char *msTrueFalse[2]={ "FALSE", "TRUE"};
mapfile.c(193):static char *msJoinType[2]={ "ONE-TO-ONE", "ONE-TO-MANY"};
```

The above are all const static data, being handled as 2.1.

```
mapgd.c(239):static unsigned char PNGsig[8] = {137, 80, 78, 71, 13, 10, 26, 10}; /* 89 50 4E 47 0D 0A 1A 0A */
mapgd.c(240):static unsigned char JPEGsig[3] = {255, 216, 255}; /* FF D8 FF hex */
mapgd.c(911): static gdPoint points[38];
mapgd.c(2428): static double last_style_size;
mapgd.c(2719): static int styleIndex, styleVis;
mapgd.c(2720): static double styleSize=0, styleCoef=0, last_style_size=-1;
mapgd.c(2721): static int last_style_c=-1, last_style_stylelength=-1, last_styleVis=0;
```

TODO

```
mapgdal.c(141):static int bGDALInitialized = 0;
```

Will be handled as 2.2. (Startup initialization)

```
mapgeos.cpp(98):GeometryFactory *gf=NULL;
```

Will be handled as 2.2. (Startup initialization)

```
maphttp.c(140):static int gbCurlInitialized = MS_FALSE;
```

Will be handled as 2.2. (Startup initialization)

```
mapimagemap.c(141):static char *layerlist=NULL;
mapimagemap.c(142):static int layersize=0;
mapimagemap.c(143):static pString imgStr, layerStr={ &layerlist, &layersize, 0 };
mapimagemap.c(146):static const char *polyHrefFmt, *polyMOverFmt, *polyMOutFmt;
mapimagemap.c(147):static const char *symbolHrefFmt, *symbolMOverFmt, *symbolMOutFmt;
mapimagemap.c(148):static const char *mapName;
mapimagemap.c(150):static int suppressEmpty=0;
mapimagemap.c(229):static int lastcolor=-1;
mapimagemap.c(273):static char* lname;
mapimagemap.c(274):static int dxf;
```

Imagemap support is not widely used, will be handled as 2.6 for now. Still waiting for comments.

```
mapio.c(67):static int is_msIO_initialized = MS_FALSE;
mapio.c(69):static msIOContext default_stdin_context;
mapio.c(70):static msIOContext default_stdout_context;
mapio.c(71):static msIOContext default_stderr_context;
mapio.c(73):static msIOContext current_stdin_context;
mapio.c(74):static msIOContext current_stdout_context;
mapio.c(75):static msIOContext current_stderr_context;
```

FrankW: Currently there is only one process wide set of “io handlers” for io. This will almost certainly need to change at some point to be thread local in some fashion. I hope to address this when I work on the redirectable OWS services accessible from MapScript this spring.

```
maplexer.c(220):static YY_BUFFER_STATE yy_current_buffer = 0;
maplexer.c(230):static char yy_hold_char;
maplexer.c(232):static int yy_n_chars; /* number of characters read into yy_ch_buf */
maplexer.c(238):static char *yy_c_buf_p = (char *) 0;
maplexer.c(239):static int yy_init = 1; /* whether we need to initialize */
maplexer.c(240):static int yy_start = 0; /* start state number */
maplexer.c(245):static int yy_did_buffer_switch_on_eof;
maplexer.c(306):static yyconst short int yy_accept[2442] =
maplexer.c(579):static yyconst int yy_ec[256] =
```

```
maplexer.c(611):static yyconst int yy_meta[52] =
maplexer.c(621):static yyconst short int yy_base[2456] =
maplexer.c(895):static yyconst short int yy_def[2456] =
maplexer.c(1169):static yyconst short int yy_nxt[2776] =
maplexer.c(1478):static yyconst short int yy_chk[2776] =
maplexer.c(1787):static yy_state_type yy_last_accepting_state;
maplexer.c(1788):static char *yy_last_accepting_cpos;
maplexer.c(1886):static int yy_start_stack_ptr = 0;
maplexer.c(1887):static int yy_start_stack_depth = 0;
maplexer.c(1888):static int *yy_start_stack = 0;
```

According to Steve's suggestions we will address this with newer versions flex and bison invoked appropriately.

```
mapmygis.c(245):static int gBYTE_ORDER = 0;
```

TODO

```
mapogcsos.c(1974): static char *request=NULL, *service=NULL;
```

TODO

```
mapogr.cpp(840):static int bOGRDriversRegistered = MS_FALSE;
```

Will be handled as 2.2. (Startup initialization)

```
mapows.c(1647): static char epsgCode[20] ="";
```

TODO

```
mapparser.c(282):static const unsigned char yytranslate[] =
mapparser.c(317):static const unsigned char yyprhs[] =
mapparser.c(328):static const yysigned_char yyrhs[] =
mapparser.c(354):static const unsigned short yyrline[] =
mapparser.c(368):static const char *const yytname[] =
mapparser.c(381):static const unsigned short yytoknum[] =
mapparser.c(391):static const unsigned char yr1[] =
mapparser.c(402):static const unsigned char yr2[] =
mapparser.c(415):static const unsigned char yydefact[] =
mapparser.c(431):static const yysigned_char yydefgoto[] =
mapparser.c(439):static const short yypact[] =
mapparser.c(455):static const yysigned_char yypgoto[] =
mapparser.c(465):static const unsigned char yytable[] =
mapparser.c(491):static const yysigned_char yycheck[] =
mapparser.c(519):static const unsigned char yystos[] =
```

According to Steve's suggestions we will address this with newer versions flex and bison invoked appropriately.

```
mappluginlayer.c(46):static VTFactoryObj gVirtualTableFactory = {MS_MAXLAYERS, 0, {NULL}};
```

TODO

```
mappool.c(206):static int connectionCount = 0;
mappool.c(207):static int connectionMax = 0;
mappool.c(208):static connectionObj *connections = NULL;
```

TODO

```
mapproject.c(889):static char *ms_proj_lib = NULL;
mapproject.c(890):static char *last_filename = NULL;
mapproject.c(918):    static int finder_installed = 0;
```

TODO

```
mapscale.c(68):static char *unitText[5]={"in", "ft", "mi", "m", "km"};
mapscale.c(69):double inchesPerUnit[6]={1, 12, 63360.0, 39.3701, 39370.1, 4374754};
```

const static data, being handled as 2.1.

```
mapsde.c(216):static int lcacheCount = 0;
mapsde.c(217):static int lcacheMax = 0;
mapsde.c(218):static layerId *lcache = NULL;
```

TODO

```
mapserv.c(1196):    static int nRequestCounter = 1;
```

Considering safe without locks (2.1). The *mapserv* application does not involved in multiple threads.

```
mapshape.c(70):static int    bBigEndian;
```

Should be handled as `epplib.c(47):static int REVERSE;`

```
mapsvg.c(164):    static char epsgCode[20] = "";
mapsvg.c(165):    static char *value;
```

TODO

```
mapswf.c(97):static char gszFilename[128];
mapswf.c(98):static char gszAction[256];
mapswf.c(99):static char gszTmp[256];
```

TODO

```
mapsymbol.c(154):static char *msCapsJoinsCorners[7]={"NONE", "BEVEL", "BUTT", "MITER", "ROUND", "SQUA
```

const static data, being handled as 2.1.

```
mapthread.c(196):static int thread_debug = 0;
mapthread.c(198):static char *lock_names[] =
```

const static data, being handled as 2.1.

```
mapthread.c(213):static int mutexes_initialized = 0;
mapthread.c(214):static pthread_mutex_t mutex_locks[TLOCK_MAX];
mapthread.c(223):    static pthread_mutex_t core_lock = PTHREAD_MUTEX_INITIALIZER;
mapthread.c(294):static int mutexes_initialized = 0;
mapthread.c(295):static HANDLE mutex_locks[TLOCK_MAX];
mapthread.c(305):    static HANDLE core_lock = NULL;
```

Should be handled as 2.2. (Startup initialization). Initialized in `msThreadInit()`;

```
maputil.c(1068):static int tmpCount = 0;
maputil.c(1069):static char *ForcedTmpBase = NULL;
```

TODO

```
mapwms.c(252):static char *wms_exception_format=NULL;
```

TODO

```
md5c.c(58):static unsigned char PADDING[64] = {
```

const static data, being handled as 2.1.

```
strptime.c(43):static const char *abb_weekdays[] = {
strptime.c(54):static const char *full_weekdays[] = {
strptime.c(65):static const char *abb_month[] = {
strptime.c(81):static const char *full_month[] = {
strptime.c(97):static const char *ampm[] = {
```

const static data, being handled as 2.1.

```
gdft\gdkanji.c(103): static int whatcode;
gdft\gdkanji.c(403): static unsigned char tmp[BUFSIZ];
gdft\gdkanji.c(489): static unsigned char tmp_dest[BUFSIZ];
```

SDL: gdft and it's contents are not part of MapServer distributions.

```
gdft\gdttf.c(712): static gdCache_head_t *tweenColorCache=NULL; /***** set up tweenColorCache on fi
gdft\gdttf.c(852): static gdCache_head_t *fontCache=NULL; /***** initialize font engine on fi
gdft\gdttf.c(853): static TT_Engine engine;
```

SDL: gdft and it's contents are not part of MapServer distributions.

```
gdft\jisx0208.h(7):static unsigned short UnicodeTbl[][94] = {
```

SDL: gdft and it's contents are not part of MapServer distributions.

```
mapscript\csharp\csmodule.i(32): static $moduleHelper()
mapscript\csharp\csmodule.i(41): static $moduleHelper the$moduleHelper = new $moduleHelper();
```

Initialization startup code.

```
mapscript\php3\mapscript_i.c(223): static int i=0;
mapscript\php3\mapscript_i.c(1137): static char pszFieldName[1000];
```

TODO

```
mapscript\php3\php_mapscript.c(789):static int le_msmap;
mapscript\php3\php_mapscript.c(790):static int le_mslayer;
mapscript\php3\php_mapscript.c(791):static int le_msimg;
mapscript\php3\php_mapscript.c(792):static int le_msclass;
mapscript\php3\php_mapscript.c(793):static int le_mslabel;
mapscript\php3\php_mapscript.c(794):static int le_mscolor;
mapscript\php3\php_mapscript.c(795):static int le_msrect_new;
mapscript\php3\php_mapscript.c(796):static int le_msrect_ref;
```

```
mapscript\php3\php_mapscript.c(797):static int le_mspoint_new;
mapscript\php3\php_mapscript.c(798):static int le_mspoint_ref;
mapscript\php3\php_mapscript.c(799):static int le_msline_new;
mapscript\php3\php_mapscript.c(800):static int le_msline_ref;
mapscript\php3\php_mapscript.c(801):static int le_msshape_new;
mapscript\php3\php_mapscript.c(802):static int le_msshape_ref;
mapscript\php3\php_mapscript.c(803):static int le_msshapefile;
mapscript\php3\php_mapscript.c(804):static int le_msweb;
mapscript\php3\php_mapscript.c(805):static int le_msrefmap;
mapscript\php3\php_mapscript.c(806):static int le_msprojection_new;
mapscript\php3\php_mapscript.c(807):static int le_msprojection_ref;
mapscript\php3\php_mapscript.c(808):static int le_msscalebar;
mapscript\php3\php_mapscript.c(809):static int le_mslegend;
mapscript\php3\php_mapscript.c(810):static int le_msstyle;
mapscript\php3\php_mapscript.c(811):static int le_msoutputformat;
mapscript\php3\php_mapscript.c(812):static int le_msgrid;
mapscript\php3\php_mapscript.c(813):static int le_mserror_ref;
mapscript\php3\php_mapscript.c(814):static int le_mslabelcache;
mapscript\php3\php_mapscript.c(815):static int le_mssymbol;
mapscript\php3\php_mapscript.c(816):static int le_msquerymap;
mapscript\php3\php_mapscript.c(857):static zend_class_entry *map_class_entry_ptr;
mapscript\php3\php_mapscript.c(858):static zend_class_entry *img_class_entry_ptr;
mapscript\php3\php_mapscript.c(859):static zend_class_entry *rect_class_entry_ptr;
mapscript\php3\php_mapscript.c(860):static zend_class_entry *color_class_entry_ptr;
mapscript\php3\php_mapscript.c(861):static zend_class_entry *web_class_entry_ptr;
mapscript\php3\php_mapscript.c(862):static zend_class_entry *reference_class_entry_ptr;
mapscript\php3\php_mapscript.c(863):static zend_class_entry *layer_class_entry_ptr;
mapscript\php3\php_mapscript.c(864):static zend_class_entry *label_class_entry_ptr;
mapscript\php3\php_mapscript.c(865):static zend_class_entry *class_class_entry_ptr;
mapscript\php3\php_mapscript.c(866):static zend_class_entry *point_class_entry_ptr;
mapscript\php3\php_mapscript.c(867):static zend_class_entry *line_class_entry_ptr;
mapscript\php3\php_mapscript.c(868):static zend_class_entry *shape_class_entry_ptr;
mapscript\php3\php_mapscript.c(869):static zend_class_entry *shapefile_class_entry_ptr;
mapscript\php3\php_mapscript.c(870):static zend_class_entry *projection_class_entry_ptr;
mapscript\php3\php_mapscript.c(871):static zend_class_entry *scalebar_class_entry_ptr;
mapscript\php3\php_mapscript.c(872):static zend_class_entry *legend_class_entry_ptr;
mapscript\php3\php_mapscript.c(873):static zend_class_entry *style_class_entry_ptr;
mapscript\php3\php_mapscript.c(874):static zend_class_entry *outputformat_class_entry_ptr;
mapscript\php3\php_mapscript.c(875):static zend_class_entry *grid_class_entry_ptr;
mapscript\php3\php_mapscript.c(876):static zend_class_entry *error_class_entry_ptr;
mapscript\php3\php_mapscript.c(877):static zend_class_entry *labelcache_class_entry_ptr;
mapscript\php3\php_mapscript.c(878):static zend_class_entry *symbol_class_entry_ptr;
mapscript\php3\php_mapscript.c(879):static zend_class_entry *querymap_class_entry_ptr;
mapscript\php3\php_mapscript.c(891):static unsigned char one_arg_force_ref[] =
mapscript\php3\php_mapscript.c(893):static unsigned char two_args_first_arg_force_ref[] =
```

TODO

```
mapscript\php3\php_proj.c(196):static zend_class_entry *proj_class_entry_ptr;
mapscript\php3\php_proj.c(200):static int le_projobj;
mapscript\php3\php_proj.c(294):static long _php_proj_build_proj_object(PJ *pj,
```

TODO

```
mapscript\swiginc\dbfinfo.i(40):          static char pszFieldName[1000];
```

TODO


```
mapscript\swiginc\map.i(204):    static int i=0;
```

TODO

3. Issues of the related libraries

TODO

4. Considerations for the future development

Developers should keep in mind that their code may be called by multiple threads simultaneously. Using process-wide global variables modified at run time will be discouraged in the future. If the usage is inevitable the variables should be protected by using mutual exclusion properly depending on the `USE_THREAD` constant.

5. Backwards compatibility issues

This is binary and source code level backward incompatible change. The compatibility of the mapscript interface might be kept.

Bug ID

The following bug is added as a primary source for collecting the developer and user response. Nevertheless the MapServer users and developers list will also be monitored and the RFC will be updated accordingly by the primary author.

Bug 1764

Voting history

Still not proposed for voting

14.6.17 MS RFC 16: MapScript WxS Services

Date 2006/05/10

Author Frank Warmerdam

Contact warmerdam at pobox.com

Last Edited May 22, 2006

Status adopted and implemented

Version MapServer 4.10

Id \$Id: ms-rfc-16.txt 8278 2008-12-23 21:34:31Z hobu \$

Purpose

The general intention is that a WMS or WCS service should be able to be setup via MapScript. The web request would be turned over to a MapScript script (Python, Perl, Java, etc) which can then manipulate the request and a map as needed. Then it can invoke the low level MapServer request function (ie. GetMap, GetCapabilities), and if desired manipulate the returned information before returning to the client. This will provide a means to do customizations such as:

- Implement security policy.
- Use dynamically created maps from a database, etc.
- Flesh out capabilities documents with auxiliary information not settable via the .map METADATA tags.
- Adjust error behaviors.

Technical Solution

- GetCapabilities, GetFeatureInfo and GetMap calls for WMS callable from MapScript, and results capturable for processing.
- GetCapabilities, DescribeCoverage and GetCoverage calls for WCS callable from MapScript and results capturable for processing.
- GetCapabilities, DescribeFeatureType and GetFeature calls for WFS callable from MapScript and results capturable for processing.
- Any other OWS services dispatched through OWSDispatch (such as SOS) would also be accessible from MapScript.
- IO hooking to capture various output from MapServer services will be accomplished via mapio.c services, the same as is used to capture output for FastCGI services.
- All SWIG based MapScript languages will be supported (Perl, Python, Ruby, C#, Java). PHP (non SWIG) may be supported if the PHP MapScript maintainers do a similar implementation.
- A MapScript WxS HOWTO will be written, including simple examples of customized services.

WxS Functions

Add the following methods on the mapObj in mapsript/swiginc/map.i.

```
int OWSDispatch( OWSRequest *req );
```

We can't call the lower level functions, like msWMSGetCapabilities() directly very easily because these functions require some pre-processing done by msWMSDispatch().

- Note that the OWSDispatch() reconfigures the map it is invoked on to match the request.
- Note that the results of the OWSRequest are still written out via the normal stdout stream handling, so separate msIO hooking is needed to capture the results.

OWSRequest

This object is already defined to MapScript in mapsript/swiginc/owsrequest.i but it seems to lack a means of setting it from cgi arguments or directly by parsing a provided url. I propose to add the following method on the OWSRequest:

```
loadParams();
```

- Loads the parameters from the normal sources (QUERY_STRING env. variable for instance).

```
loadParams( const char * url );
```

- Loads the parameters from the given url portion as would have appeared in QUERY_STRING if REQUEST_METHOD was GET.

IO Hooking

Currently output from functions such as msWMSGetCapabilities() is directed through the msIO services (in mapio.c), such as msIO_printf(). In order to capture this output and make it available for further processing it will be necessary to provide a means for MapScript applications to insert their own IO handlers of some sort.

Additionally, currently the msIO layer has a concept of io handlers, but they are global to the application. For non-trivial use of WxS MapScript services in Java, c# and other multithreaded environments it is desirable for it to be possible to insert per-thread IO handlers.

- Need to make at least current_stdin/stdout/stderr_context variables thread local. Possibly using the same approach as maperror.c. A new mutex will be required for this.
- Consider thread safe output to shared stdin/stdout/stderr handles for all threads? ie. protect with a mutex.
- We need to provide a convenient way to install “to buffer” and “from buffer” io handlers.
- We need to always use msIO redirection. Currently in the default case of not using FastCGI, USE_MAPIO is not defined in mapio.h and msIO_printf() and similar functions are actually just #define’ed to printf(). But if we want to be able to capture output all the time for MapScript, we will actually always need the msIO layer. So USE_MAPIO will always have to be defined.

```
msIO_resetHandlers();
```

- Resets msIO handlers to defaults (using stdin, stdout, stderr).
- Clears buffer data if buffered handlers were installed.

```
msIO_installStdoutToBuffer();
```

- Install handler to send stdout to a buffer, clear buffer.

```
msIO_installStdinFromBuffer();
```

- Install handler to get stdin from a buffer, clear buffer.

```
msIO_setStdinBuffer( unsigned char *data, int length );
```

- Set data for stdin buffer.

```
gdBuffer msIO_getStdoutBufferBytes();
```

- Fetch stdout buffer pointer and length.
- gdBuffer already provides language specific bindings to get byte data.

```
const char *msIO_getStdoutBufferString();
```

- Fetch stdout buffer as a string. Appropriate for XML and HTML results for instance.

The installed “buffer” handlers will manage their own buffer and concept of current read/write position.

My objective is that folks should be able to do something like this in Python MapScript.

```
mapscript.msIO_installStdoutToBuffer()
if map.OWSDispatch( req ) == mapscript.MS_SUCCESS:
    result = mapscript.msIO_getStdoutBufferString()
    mapscript.msIO_resetHandlers()
```

Questions:

- 1) Should we be “pushing” handlers instead of installing them and losing track of the previous handler? Then we could just pop them off.
- 2) Should we make the whole msIOContext thing more visible to MapScript? It seems like it would be complicated.

gdBuffer

The msIO_getStdoutBufferBytes() returns a gdBuffer since most language bindings already have a way of using this as a “array of raw bytes” buffer. It is normally used for fetched gdImage buffers. But because the msIO function returns a gdBuffer referring to an internally memory array not “owned” by the gdBuffer we need to add a owns_data flag.

```
typedef struct {
    unsigned char *data;
    int size;
    int owns_data;
} gdBuffer;
```

Likewise, each of the language bindings needs to be modified to only call gdFree() on data if owns_data is true.

This:

```
%typemap(out) gdBuffer {
    $result = PyString_FromStringAndSize($1.data, $1.size);
    gdFree($1.data);
}
```

becomes this:

```
%typemap(out) gdBuffer {
    $result = PyString_FromStringAndSize($1.data, $1.size);
    if( $1.owns_data )
        gdFree($1.data);
}
```

And similarly for the other bindings.

Files and objects affected

```
mapio.c
mapio.h
mapscript/mapscript.i
mapscript/swiginc/owsrequest.i
mapscript/swiginc/image.i
mapscript/swiginc/msio.i (new)
mapscript/python/pymodule.i (gdBuffer)
```

Backwards compatibility issues

There are no apparent backward compatibility problems with existing MapScript scripts.

Implementation Issues

- the gdBuffer stuff likely ought to be generalized.
- some MapScript languages lack gdBuffer typemaps (ie. perl).
- some performance testing should be done to verify that USE_MAPIO isn't going to slow down normal operations significantly. This is specially a concern once the mapio.c statics are actually handled as thread local as each msIO call will need to search for the appropriate thread local context.
- the msIO “buffer” approach is predicated on streaming output results into a memory buffer. For very large return results this may use an unreasonable amount of memory. For instance a WFS request with a 250MB response document. But such results aren't necessarily reasonable in web services context anyways.
- The set of functions will need to be exposed separately in the PHP bindings.

Test suite

The msautotest/mspython and python unit tests will be extended with at least rudimentary testing of a few of these services.

As we have no automated tests for other MapScript languages, no automated tests will be added, but I will endeavour to prepare simple scripts to test things. Currently this has been done for Python and Perl MapScript.

Example

This shows a very simple Python MapScript script that invokes a passed in OWSRequest passed via normal cgi means, but adding a text/plain content type ahead of the regular content type so we can see the results. The script could easily have done extra manipulation on the URL parameters, and on the map object.

Example:

```
#!/usr/bin/env python

import sys
import mapscript

req = mapscript.OWSRequest()
req.loadParams()
```

```
mapscript.msIO_installStdoutToBuffer()

map.OWSDispatch( req )

print 'Content-type: text/plain'
print
print mapscript.msIO_getStdoutBufferString()
```

Bug ID

<http://trac.osgeo.org/mapserver/ticket/1788>

Voting history

+1: FrankW, SteveW.

Open questions

None

14.6.18 MS RFC 17: Dynamic Allocation of layers, styles, classes and symbols

Date 2006/05/12

Author Frank Warmerdam, with edits from Daniel Morissette and Umberto Nicoletti

Contact warmerdam at pobox.com, dmorissette at mapgears.com, umberto.nicoletti at gmail.com

Last Edited 2007/07/18

Status Adopted (2007/07/18) - Implementation completed (2007/07/23)

Version MapServer 5.0

Id \$Id: ms-rfc-17.txt 8278 2008-12-23 21:34:31Z hobu \$

Purpose

Modify the MapServer core libraries so that lists of layers, classes and styles are dynamic, not fixed to compile limits MS_MAXCLASSES, MS_MAXSTYLES, MS_MAXLAYERS and MS_MAXSYMBOLS.

MS_MAXSYMBOLS

Change symbolSetObj so that this:

```
int numsymbols;
symbolObj symbol[MS_MAXSYMBOLS];
```

becomes:

```
int numsymbols;
int maxsymbols;
symbolObj **symbol;
```

Add the following function to ensure there is at least one free entry in the symbol array in the symbolSetObj. This function will only grow the allocated array of pointers if needed (if maxsymbols == numsymbols) and will then allocate a new symbolObj if symbol[numsymbols] == NULL. The new entries in the array will be set to NULL and the new symbolObj[numsymbols] will be all set to zero bytes.

```
symbolObj *msGrowSymbolSet( symbolSetObj * );
```

- Modify all places that add new symbols to call msGrowSymbolsSet() to ensure there is space. These locations can be fairly easily identified by greping on MS_MAXSYMBOLS.
- Modify all places that access symbols by index to use the proper way to reference symbols in the new array, possibly using a GET_SYMBOL() macro similar to the GET_LAYER() added in MS-RFC-24 for layers.
- mapsymbol.c: Modify msInitSymbolSet() to initially setup the symbol set with an array of just one symbol.
- There should be no swig MapScript changes required as it already uses msAppendSymbol().

MS_MAXLAYERS

- map.h: Since *MS RFC 24: Mapscript memory management* already converted the array of layers in a mapObj to an array of pointers, we only need to add “int maxlayers;” to mapObj to indicate the current allocation size of layers array. Note that this also determines the size of the mapObj layerorder array.
- mapdraw.c: Change msDrawMap() to use map->numlayers in place of current MS_MAXLAYERS for asOWS-ReqInfo array.
- mapobject.c: Add msGrowMapLayers(mapObj*) function to ensure there is room for at least one more layer on the map. This grows layers, and layerorder arrays.
- mapobject.c: modify msInsertLayer() and MapScript layer constructor code (see last item) to use msGrowMapLayers().
- mapfile.c: set maxlayers to MS_MAXLAYERS and allocate layers accordingly in initMap(). Use msGrowMapLayers() before calling loadLayer() and loadLayerValue() when parsing.
- mappluginlayer.c: make dynamic. I’m not exactly sure what there is a static factory with entries dimensioned by the number of layers.
- It looks like swig MapScript already has an mapObj.insertLayer() method using msInsertLayer() which should be safe. Do MapScript applications sometimes just update the layers and numlayers directly? Answer: all layer manipulation is done by the layer constructor (when a not null mapObj is passed as only argument) or by msInsertLayer. Must modify the layer constructor as well.

MS_MAXCLASSES

- map.h: Add maxclasses field in layerObj (RFC-24 already converted the array of classes in a layerObj to an array of pointers).
- layerobject.c: Modify msInsertClass() and MapScript constructor code (like for layers) to use msGrowLayerClasses().
- layerobject.c: Add msGrowLayerClasses() to ensure there is at least one extra class in the classes list.

- `mapdraw.c`: `colorbuffer` and `mindistancebuffer` in `msDrawQueryLayer()` will need to be dynamically sized and allocated on the heap.
- `mapfile.c`: `initLayer()` will need to initialize `maxclasses` to `MS_MAXCLASSES`, and allocate class list accordingly.
- `mapogcsld.c`: modify to use `msGrowLayerClasses()` instead of checking limit.

MS_MAXSTYLES

- `map.h`: add `maxstyles` field to `classObj` (RFC-24 already converted the array of styles in a `classObj` to an array of pointers).
- `mapfile.c`: set `maxstyles` to `MS_MAXSTYLES` and allocate accordingly in `initClass()`. Use `msGrowClassStyles()` as appropriate.
- `classobj.c`: Add `msGrowClassStyles()` to ensure there is an unused class.
- `classobject.c`: Use `msGrowClassStyles()` in `msInsertStyle()` and in the style constructor.

Files and objects affected

```
map.h
mapfile.c
mapsymbol.c
mapdraw.c
mapobject.c
mappluginlayer.c
layerobject.c
mapogcsld.c
classobject.c
mapscript/swiginc/layer.i
mapscript/swiginc/class.i
mapscript/swiginc/style.i
```

Backwards compatibility issues

None.

Test suite

Python unit test entries will be added to exceed the builtin maximums for all of layers, classes, styles and symbols. An `msautotest` entry with a large number of classes will also be added.

Bug ID

- 302: <https://trac.osgeo.org/mapserver/ticket/302>

Voting history

Vote completed on 2007-07-19:

+1 from DanielM, SteveL, FrankW, SteveW, UmbertoN, TamasS, AssefaY

Comments/Questions from the review period

- Has the following issue already been dealt with by RFC-24 since the code already allocates less than `MS_MAX_LAYERS` in `initMap()`? Did RFC-24 make it a formal requirement to use the proper insert/add methods to add layers, styles and classes?

Answer: the implementation adopted with *MS RFC 24: Mapscript memory management* tries to save memory by only allocating the number of layers that are effectively needed. A map with 5 layers will allocate exactly 5 layers, a map with 50 layers will allocate 50, and so on up to the hard-coded limit of `MS_MAX_LAYERS`. This is also true for classes and style.

The arrays of pointers are for obvious reasons always allocated to `MS_MAX_LAYERS` size, but memory usage is reduced anyway because arrays of pointers are much smaller than arrays of structs. This represents a substantial change from before, when `MS_MAX_LAYERS` blank layers were always allocated causing a sure waste of memory in small maps.

MS RFC 17: Dynamic Allocation of layers, styles, classes and symbols should then invoke the various `msGrow*()` methods to grow the arrays when `numlayers == maxlayers-1`. I'd suggest to grow the array to a sensible size (like half of the current size) as the impact on memory allocation is going to be mitigated by the dynamic allocation approach introduced by *MS RFC 24: Mapscript memory management*

MS RFC 24: Mapscript memory management did not make a formal requirement to use the proper insert methods as those and the object constructors are the *only* way to add a layer, class or style to a map. I do not know if this is also true for symbols.

- “Because MapScript application often explicitly initialize “blank” layers, classes and styles directly, and then increment the count, we can't depend on all access going through the proper insert/add methods. For this reason we preserve the old `MS_MAX` values to establish the initial allocation. This should mean that existing applications will continue to work at some cost in unused memory. But well behaved MapScript applications using insert methods to increase sizes will be able to grow beyond the initial allocation.”

I'm not aware of any MapScript way to bypass the insert or the constructors (i.e. with direct manipulation of the arrays). If there was such way (which I doubt) I suggest that we forbid it by explicitly changing the `MS_MAX_*` names and making required fields immutable in swig.

- Comment from Tamas: The `mappluginlayer` stuff should be concretized a bit. That static repository of the `vtables` would prevent from loading the same library twice. That array should also be allocated dynamically since I don't think we will ever have `MS_MAXLAYERS` number of different plugin layers.

14.6.19 MS RFC 18: Encryption of passwords in mapfiles

Date 2006/05/26

Author Daniel Morissette

Contact dmorissette at mapgears.com

Last Edited 2006/08/11

Status Completed (2006/08/11)

Version MapServer 4.10

Overview

This proposal provides a mechanism to protect database connection passwords used inside *mapfiles* by encrypting them instead of including them in plain text.

Technical Solution

MapServer will be extended to allow the use of encrypted passwords as part of the CONNECTION string for the following layer types:

- *Oracle Spatial*
- *PostGIS*
- *ESRI SDE*
- *OGR*

The Tiny Encryption Algorithm (TEA) at <http://www.simonshepherd.supanet.com/tea.htm> will be used for the encryption/decryption functions.

The implementation details follow...

Encryption key

In order to safely protect the encrypted information, an encryption key will be required by this mechanism. The key will NOT be stored in the mapfile: it will be stored in a separate file on the server and should be kept in a safe area by the server administrator (especially outside of the web server's document directories).

The location of the encryption key can be specified by two mechanisms, either by setting the environment variable MS_ENCRYPTION_KEY or using a CONFIG directive:

```
CONFIG MS_ENCRYPTION_KEY "/path/to/mykey.txt"
```

New “msencrypt” command-line utility

A “msencrypt” command-line utility will be provided to create an encryption key and to encrypt passwords (or any string) for use in a mapfile.

To create an encryption key:

```
msencrypt -keygen /path/to/mykey.txt
```

To encrypt a password or any string:

```
msencrypt -key /path/to/mykey.txt <string_to_encrypt>
```

If the MS_ENCRYPTION_KEY environment variable is set then the -key argument does not need to be specified.

Encoding of encrypted strings

Since the result of encryption is binary data that is not suitable for inclusion directly in a MapServer mapfile, hex encoding will be used for the encrypted strings in the mapfile as well as for storing the encryption key to disk.

The { and } characters will be used as delimiters for encrypted strings inside database CONNECTIONs. This will allow the use of either plain text or encrypted passwords in mapfiles without any backwards compatibility issues.

e.g.

```
CONNECTIONTYPE ORACLESPATIAL  
CONNECTION "user/{MIIBugIBAAKBgQCP0Yj+Seh8==}@service"
```

Any part of a CONNECTION string can be encrypted and not just the password. This will allow protecting other information such as login name, hostname or port numbers if necessary.

For reference, here are examples of typical connection strings for the layer types that will be affected:

```
CONNECTIONTYPE POSTGIS
CONNECTION "host=yourhostname dbname=yourdatabasename user=yourdbusername password=yourdbpassword port=yourdbport"

CONNECTIONTYPE SDE
CONNECTION "sdemachine.iastate.edu,port:5151,sde,username,password"

CONNECTIONTYPE ORACLESPATIAL
CONNECTION "user/pwd@service"

CONNECTIONTYPE OGR
CONNECTION "OCI:user/pwd@service"
```

Modifications to the source code

A `msDecryptString()` function will be created, it will take a CONNECTION string as input and decrypt any encrypted component that it may find in it. This function will be called by the various `msXXXXLayerOpen()` methods before opening the connection to the database.

```
char *msDecryptString(mapObj *map, const char *string)
```

The first time that `msDecryptString()` is called for a given mapfile, it will load the encryption key from the file and store the key in a new private member of the `mapObj` (`char * encryption_key`).

To reduce the chances of false matches in long CONNECTION strings such as OGR VRT data sources, `msDecryptString()` function will look for a pair of { + }, and then verify that all chars in the block are valid hex encoding chars (0-9,A-F) before proceeding with decryption.

Note that the decrypted string will never be stored in the `layerObj`, it will be kept local to the function that opens the connection and destroyed as soon as the function is done with it. This is to prevent exposing the decrypted information in error messages or in calls to `msSaveMap()`.

Files affected

```
map.h
mapfile.c
maporaclespatial.c
mappostgis.c
mappostgresql.c
mapsde.c
mapogr.cpp
```

Backwards compatibility issues

None.

Bug ID

1792: <http://trac.osgeo.org/mapserver/ticket/1792>

Voting history

Adopted on 2006/06/01. +1: FrankW, DanielM, HowardB, YewondwossenA, SteveW

Comments from the review period

- There were concerns about the use of the {...} delimiter to indicate encrypted strings inside CONNECTIONs. Since there was not a better alternative we will stick to that.
- There was a suggestion to use an ENCRYPTION_KEY mapfile keyword instead of CONFIG MS_ENCRYPTION_KEY. Since there was no strong argument either way we decided to stick to CONFIG MS_ENCRYPTION_KEY.
- There was a suggestion to consider the Blowfish algorithm (<http://www.schneier.com/blowfish.html>) instead of TEA. The sample implementations of Blowfish would require much more work to integrate than TEA, and since TEA is public domain and so much simpler (simpler is better!), we'll stick to TEA for now and can always change the underlying algorithm at a later time if we find that TEA is too weak (which doesn't appear to be an issue).
- Will the encryption methods be made available to MapScript? No plan to do so at this time, but this could easily be added later on.
- Since the user running the web server (and MapServer) needs to have permissions to read the key, any web server process or user with permissions to read the key can decrypt the passwords using a trivial program. It should be made very clear in the documentation that this is just simple obfuscation and is by no means secure and that users should not place valuable passwords in mapfiles encrypted or not.

14.6.20 MS RFC 19: Style & Label attribute binding

Date 2006/06/07

Author Steve Lime

Contact steve.lime at DNR.STATE.MN.US

Status Passed (in process)

Version MapServer 5.0

Id \$Id: ms-rfc-19.txt 8278 2008-12-23 21:34:31Z hobu \$

Description: Presently MapServer supports binding of label and style properties for a few select attributes (e.g. angle). However, it is cumbersome to add new bindings which leads to unnecessarily complex structures and code that is difficult to maintain. This RFC presents ideas to bring a bit of order to that chaos and make expansion of this capability easier to achieve.

C Structural Changes

A new structure called attributeBindingObj would be added:

```
typedef struct {
    char *item;
    char index;
} attributeBindingObj;
```

Several new enumerations would then define what properties could be bound:

```
#define MS_STYLE_BINDING_LENGTH ...
#define MS_STYLE_BINDING_ENUM { MS_STYLE_BINDING_SIZE, MS_STYLE_BINDING_ANGLE, ... };

#define MS_LABEL_BINDING_LENGTH ...
#define MS_LABEL_BINDING_ENUM { MS_LABEL_BINDING_SIZE, MS_LABEL_BINDING_ANGLE, ... };
```

Several elements would be removed from layerObj and styleObj. For example, angleitem, angleitemindex and similar members would be removed. styleObj's and labelObj's would each take on 2 new members:

```
char hasBindings;
attributeBindingObj bindings[MS_STYLE_BINDING_LENGTH];
```

Mapfile/MapScript Changes

Options like SIZEITEM and ANGLEITEM would go away. Instead a more logical syntax such as:

```
STYLE
  SIZE [mySizeItem]
  ANGLE [myAngleItem]
  COLOR 255 0 0
  SYMBOL 'square'
END
```

Square brackets have been used in MapServer templates and expressions to bind to attributes so they are a natural choice to denote attribute bindings in this case.

Similarly MapScript would lose the ability to set/get the xxxITEM properties. Instead the style and label objects would get setBinding and deleteBinding methods:

```
(in Perl)
$style->setBinding($mapscript::MS_STYLE_BINDING_SIZE, 'mySizeItem');
$style->deleteBinding($mapscript::MS_STYLE_BINDING_COLOR);
```

Files Affected

- map.h => structure changes, enum and define additions
- maputil.c => add msStyleBindAttributes();

```
void msStyleBindAttributes(shapeObj *shape, styleObj *style)
{
  if(style->bindings[MS_STYLE_BINDING_SIZE].item)
    style->size = atoi(shape->values[style->bindings[MS_STYLE_BINDING_SIZE].index]);
  if(style->bindings[MS_STYLE_BINDING_ANGLE].item)
    style->angle = atoi(shape->values[style->bindings[MS_STYLE_BINDING_ANGLE].index]);
  ...
}
```

- maplabel.c => add msLabelBindAttributes();
- maplayer.c => fix msWhichItems() to populate the binding indexes
- mapdraw.c => remove references to the xxxITEM properties, if a style or label has bindings then call msStyleBindAttributes or msLabelBindAttributes

- `mapfile.c` => fix parsing to handle strings in addition to numbers for all supported bindings (symbols already allow this), if a binding is defined flip the `hasBindings` flag, remove all references to `xxxITEM` properties, alter style and label writing code to honor bindings as part of output
- `mapcopy.c` => fix style and labeling copying
- `maplexer.l/mapfile.h` => remove `xxxITEM` defines and lexer references, and to define a new raw type (similar to `MS_STRING`) called `MS_BINDING`
- `mapscript/swiginc/style.i` => add `setBinding` and `deleteBinding` methods (similar for PHP/MapScript)

Testing

- Python suite: need tests to set and delete bindings
- `MsAutoTest` suite: a mapfile testing various bindings would be developed (DNR tests have one example for labels)

Backwards compatabilty issues

This does affect a number of parameters, however they are lightly used so this is probably worth the risk. The primary risk is breaking mapfiles as opposed to scripts. Given that this will definitely break some stuff I think it is most appropriate as a 5.0 change.

Bug ID

2100

Voting history

+1 Lime, Morissette, Woodbridge

14.6.21 MS RFC 21: MapServer Raster Color Correction

Date 2006/10/13

Author Frank Warmerdam

Contact warmerdam at pobox.com

Status adopted

Version MapServer 5.0

Id \$Id: ms-rfc-21.txt 8278 2008-12-23 21:34:31Z hobo \$

Overview

To add support to MapServer to apply color correction curves (aka lookup tables) to raster images on the fly.

Technical Details

MapServer's raster rendering engine (`mapgdaldraw.c`) will be extended to support a `PROCESSING` option indicating a color correction file to be applied on the fly when raster data is being read from disk. The `PROCESSING` option would take the form:

```
PROCESSING "LUT=<lut_specification>"
```

for a single LUT applied to all bands equally or:

```
PROCESSING "LUT_<color#>=<lut_specification>"
```

for a LUT applied only to one band. The `<lut_specification>` can be the name of a file containing a LUT, or an inline lut definition. The inline lut definition looks like:

```
<lut_specification>=<in_value>:<out_value>,<in_value>:<out_value>[,<in_value>:<out_value>]*
```

The text file takes the same form, except that it may be multiline (lines will be implicitly joined by commas). So some common forms would be:

```
PROCESSING "LUT=0:0,128:150,255:255"
```

```
PROCESSING "LUT_2=green_lut.txt"
```

Note that the LUT specification files will be searched for relative to the map file if a relative path is provided.

The LUT will be applied in `LoadGDALImage()`, after any scaling to 8bit. So currently only 8bit (0-255) input and output values are supported. Optionally we could consider non-8bit inputs, and allow the LUT to apply scaling to 8bit but this would be somewhat complicated to do in a computationally efficient manner.

A 256 entry LUT will be created by linear interpolation between the LUT control points. Note this is different than a true "curve" based approach where some form of curve fitting is done to the control points. For a significant number of control points the difference will be very slight, but it could be noticeable for LUTs with only a few control points (ie. 3). Optionally we could compute a proper curve fit, but this will require extra research and development.

If no control points exist for input values 0 and 255 a mapping of "0:0" and "255:255" will be assumed.

Other Curve Formats

The GIMP's ascii curve format will also be supported directly. Exact details of this are to be determined, and since a GIMP curve file contains several curves (for different bands) it may be necessary to select which is to be used in the `PROCESSING` statement. Perhaps something like:

```
PROCESSING "LUT_2=GIMP_red:gimp.crv"
```

Mapfile Implications

All new options are selected via new `PROCESSING` options. There is no change in the mapfile syntax. There should be no compatibility problems with old mapfiles.

MapScript Implications

There are no additions or changes to the MapScript API. The new options are controlled via `PROCESSING` information on the layers which I believe is already manipulatable from MapScript.

Documentation Implications

The new processing options will need to be documented in the Raster Access HOWTO (and possibly the mapfile reference).

Test Plan

New test cases for each mode will be incorporated in msautotest/gdal.

Staffing / Timeline

The new feature will be implemented by Frank Warmerdam and completed by November 30th, 2006, for inclusion in the MapServer 5.0 release. Funding provided by the Information Center of the Ministry of Agriculture and Forestry (Finland).

Tracking Bug

<http://trac.osgeo.org/mapserver/ticket/1943>

14.6.22 MS RFC 22a: Feature cache for long running processes and query processing

Date 2007/06/24

Author Tamas Szekeres

Contact szekerest at gmail.com

Last Edited 2007/06/24

Status Discussion Draft

Version Mapserver 5.0

Id \$Id: ms-rfc-22a.txt 8278 2008-12-23 21:34:31Z hobu \$

1. Overview

Currently the various query operations involve multiple accesses to the data providers which may cause a significant performance impact depending on the providers. In the first phase all of the features in the given search area are retrieved and the index of the relevant shapes are stored in the result cache. In the second phase the features in the result cache are retrieved from the provider one by one. Retaining the shapes in the memory we could eliminate the need of the subsequent accesses to the providers and increase the overall performance of the query. Implementing the cache requires a transformation of the data between the data provider and the client. From this aspect it is desirable to provide a framework to implement this transformation in a higher level of abstraction.

2. Purpose

The main purpose of this RFC is to implement a feature cache and retain the shapes in the memory for the further retrievals during a query operation. However I would also want to create a mechanism so that a layer could use another layer as a data source. This outer layer could apply transformations on the shapes coming from the base layer or even retain the shapes in the memory for the subsequent fetches. Here are some other examples where this framework would provide a solution:

1. Constructing geometries based on feature attributes
2. Modifying the geometries or the feature attribute values
3. Geometry transformations (like GEOS operations)
4. Feature cache
5. Providing default layer style based on external style information, or attribute based styling
6. Providing custom data filters

Setting up a proper layer hierarchy one can solve pretty complex issues without the need of creating additional (eg. mapscript) code. Later on - as a live example - I'll show up the solution of the following scenario:

1. The user will select one or more shapes in one layer (by attribute selection in this case).
2. Cascaded transformations will be applied on the selected shapes (I'll use the GEOS convexhull and buffer operations)
3. In another layer the features will be selected based on the transformed shapes as the selection shapes.

In this proposal to ensure the better readability I'll avoid embedding much of the code inline. However most of the implementation patches are available at the tracking ticket of this RFC (see later).

3. General principles of the solution

This proposal involves writing additional data providers. These providers will use another layer as the source of the features. To set up this hierarchy of the layers, the CONNECTION parameter of these layers will contain the name of the source layer. In some cases the source layer doesn't participate in the renderings and should be kept internal to the original layer. Therefore we will establish the support for nesting the layers into each other. In this regard the CONNECTION parameter will contain the full path of the layer it refers to. The path contains the layer names in the hierarchy using the slash '/' as the separator between the names. We can specify the pathnames relative to the actual layer or relative to the map itself.

1. Specify a reference relative to the map:

```
LAYER
  NAME "roads"
  CONNECTIONTYPE OGR
  CONNECTION "roads.tab"
  ...
END
LAYER
  NAME "cache"
  CONNECTIONTYPE CACHE
  CONNECTION "/roads"
  ...
END
```

1. Specify a reference relative to the outer layer

```
LAYER
  NAME "cache"
  CONNECTIONTYPE CACHE
  CONNECTION "roads"
  ...
LAYER
  NAME "roads"
  CONNECTIONTYPE OGR
```

```
CONNECTION "roads.tab"  
...  
END  
END
```

The main difference between these 2 options is that in the first case the referred layer is contained by the layers collection of the map, so the layer will participate in the drawings. In the second case the referred layer is internal to the outer layer. It is supported that 2 or more layers connect to the same base layer so that the features will be served from the same cache repository. The base layer can reside in any place of the layer hierarchy.

In any case, the extension layer can also be implemented as a pluggable external layer. (CONNECTIONTYPE PLUGIN)

To achieve the desired functionality the following 3 providers will be implemented:

3.1 Feature caching provider

The purpose of this provider is to retain the features from the source layer in the memory so the subsequent fetches on the feature caching provider will be served from the internal cache. With the current implementation I'm planning to retain the shapes of last extent. When the subsequent shape retrieval refers to the same extent (or within that extent) the features will be served from the cache. At the moment I will not address caching features from multiple non overlapping extents and implement more sophisticated cache management (like size limit/expiration handling) but it might be the object of an enhancement in the future. All of the provider specific data will be placed in the layerinfo structure of the provider. The shapes in the cache will be stored in a hashtable. This provider will be implemented in a separate file (mapcache.c).

3.1.1 Shape retrieval options

This provider will be capable to populate the cache with all of the shapes in the given extent or only with the shapes in the resultcache of the source layer. The first one is the default option and the latter can be selected by the following PROCESSING directive:

```
PROCESSING "fetch_mode=selection"
```

The cache will be populated upon the WhichShapes call of the caching provider when the given rect falls outside of the previous one. We can also specify that the provider will retrieve all the shapes of the current map extent regardless to the search area using the following setting:

```
PROCESSING "spatial_selection_mode=mapextent"
```

3.1.2 Items selection

The feature caching provider will store all of the items available regardless of which items have been selected externally (by using the WhichItems call). However the iteminfo will contain the indexes only of the requested items. The GetShape and the NextShape operations will copy only the requested subset of the items to the caller. This solution will provide the compatibility with any other existing provider so there's no need to alter the common mapserver code from this aspect.

3.1.3 Support for the STYLEITEM “AUTO” option

By using the STYLEITEM “AUTO” option the provider can retrieve the classObj of every shapeObj from the data source. So as to keep on supporting this option the caching provider will be capable to cache these classObj-s as well as the shapeObj-s in a separate hashtable. When the STYLEITEM “AUTO” option is set on the feature caching provider the style cache will also be populated by calling the GetAutoStyle function to the source layer. The subsequent GetAutoStyle calls on the feature caching provider will retrieve the classObj-s from the style cache and provide a copy to the caller.

3.1.4 Support for the attribute filter

The feature caching provider will be compatible with the existing query operations (implemented in the msQueryBy[] functions in mapquery.c). For supporting the msQueryByAttributes the NextShape will use msEvalExpression before returning the shapes to the caller.

3.2 Geometry transformation provider

The geometry transformation provider (implemented in mapgeomtrans.c) will support transparent access to the source layer. Every vtable operations will be dispatched to corresponding vtable operation of the source layer. In addition some of the other layer properties might require to copy between the connected layers.

3.2.1 Items selection

This provider will serve the same subset of the items as the source layer provides. Therefore the InitItemInfo will copy the items and the numitems of the external layer to the source layer, like:

```
int msGeomTransLayerInitItemInfo( layerObj *layer )
{
    /* copying the item array and call the subsequent InitItemInfo*/
    return msLayerSetItems(((GeomTransLayerInfo*)layer->layerinfo)->srclayer, layer->items, layer->numitems);
}
```

And upon the GetItems call these values will be copied back to the external layer, like:

```
int msGeomTransLayerGetItems(layerObj *layer)
{
    /* copying the item array back */
    int result;
    GeomTransLayerInfo* layerinfo = (GeomTransLayerInfo*)layer->layerinfo;
    result = msLayerGetItems(layerinfo->srclayer);
    msLayerSetItems(layer, layerinfo->srclayer->items, layerinfo->srclayer->numitems);
    return result;
}
```

3.2.2 Applying the transformations

The proposed implementation will support most of the GEOS transformations supported by mapserver (buffer, convexhull, boundary, union, intersection, difference, symdifference). The transformations will be applied right before retrieving a shape to the caller. The desired transformation can be selected using a PROCESSING option, like:

```
PROCESSING "transformation=buffer"
```

Some of the transformations will accept further parameters:

```
PROCESSING "buffer_width=0.2"
```

Some of these operations will use 2 shapes to create the transformed shape. The reference shape of these transformations can be set externally using the WKT representation of a processing option:

```
PROCESSING "ref_shape=[WKT of the shape]"
```

I'm also planning to support retrieving this shape from the features collection of the layer and from an external layer as well.

3.3 Layer filter provider

The layer filter provider (implemented in `mapfilter.c`) will provide the shape retrieved from the source layer and will not apply any transformation on that. However some of the shapes will be skipped in the `NextShape` operations depending on the spatial and the attribute selection options. This provider ensures transparent access to the source layer (just as the previous one) but uses a cache for storing the selection shapes. The selection shapes will be retrieved from another layer which can be specified in the following processing option:

```
PROCESSING "selection_layer=[path to the layer]"
```

When populating the selection cache I'll support to retrieve all of the shapes or only the shapes in the result cache as for the caching provider mentioned before:

```
PROCESSING "fetch_mode=selection"
```

4. Putting the things together (example)

In this chapter I'll describe the solution of the scenario have been mentioned earlier. I'll start with a simple map file definition with 2 layers the counties and the citypoints of the country:

```
MAP
  NAME "Hungary"
  STATUS ON
  EXTENT 421543.362603 47885.103526 933973.563202 367180.479761
  SIZE 800 600
  IMAGETYPE PNG
  IMAGECOLOR 255 255 255

  PROJECTION
    "proj=somerc"
    "lat_0=47.14439372222222"
    "lon_0=19.04857177777778"
    "x_0=650000"
    "y_0=200000"
    "ellps=GRS67"
    "units=m"
    "no_defs"
  END

  SYMBOL
    Name 'point'
    Type ELLIPSE
```

```

        POINTS
          1 1
        END
        FILLED TRUE
      END

    LAYER
      PROJECTION
        "AUTO"
      END
      NAME "Hun_Counties"
      CONNECTIONTYPE OGR
      CONNECTION "Hun_Megye.TAB"
      STATUS default
      TYPE POLYGON
      LABELITEM "name"
      CLASS
        TEMPLATE "query.html"
        LABEL
          SIZE medium
          COLOR 64 128 64
        END
        STYLE
          COLOR 208 255 208
          OUTLINECOLOR 64 64 64
        END
      END
    END
  END
  LAYER
    PROJECTION
      "AUTO"
    END
    NAME "Hun_CityPoints"
    CONNECTIONTYPE OGR
    CONNECTION "Hun_CityPoints.TAB"
    STATUS default
    TYPE POINT
    CLASS
      STYLE
        COLOR 255 0 0
        SYMBOL "point"
        SIZE 2
      END
    END
  END
END
END

```

This map will look like this:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample.png>

4.1 Adding the feature cache for the layers

Any of the layers might be cached by using the CACHE layer provider. The original provider might be nested inside the cache. The parameters related to the drawing should be specified for the outer layer, like:

```
LAYER
    PROJECTION
        "AUTO"
    END
    NAME "Hun_Counties_cache"
    CONNECTIONTYPE CACHE
    CONNECTION "Hun_Counties"
    STATUS default
    TYPE POLYGON
    LABELITEM "name"
    CLASS
        TEMPLATE "query.html"
        LABEL
            SIZE medium
            COLOR 64 128 64
        END
        STYLE
            COLOR 208 255 208
            OUTLINECOLOR 64 64 64
        END
    END
LAYER
    PROJECTION
        "AUTO"
    END
    NAME "Hun_Counties"
    CONNECTIONTYPE OGR
    CONNECTION "Hun_Megye.TAB"
    TYPE POLYGON
END
END
```

The `Hun_Counties_cache` layer is queryable and all of the existing methods are available to populate the resultcache. In my example I'll use the mapscript C# drawquery example (implemented in `drawquery.cs`) to execute an attribute query and use the drawquery afterwards. The following command will be used along with this sample:

```
drawquery sample.map "('[Name]='Tolna')" sample.png
```

Which passes the (`'[Name]='Tolna'`) query string to the `queryByAttributes` of the first layer.

The result of the rendering will look like:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample1.png>

The corresponding mapfile can be found here:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample1.map>

4.2 Applying transformations on the counties

In the next step I'll apply a cascaded GEOS convexhull and buffer operations on the first layer. The results will be rendered in a separate layer using the following definition:

```
LAYER
    NAME "selectionshape"
    CONNECTIONTYPE GEOMTRANS
    CONNECTION "simplify"
```

```

STATUS default
TYPE POLYGON
TRANSPARENCY 50
CLASS
    STYLE
        COLOR 64 255 64
        OUTLINECOLOR 64 64 64
    END
END
PROCESSING "transformation=buffer"
PROCESSING "buffer_width=0.2"
LAYER
    NAME "simplify"
    TYPE POLYGON
    CONNECTIONTYPE GEOMTRANS
    CONNECTION "/Hun_Counties_cache"
    PROCESSING "transformation=convexhull"
END
END

```

The result of the rendering will look like:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample2.png>

The corresponding mapfile:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample2.map>

Which is not the desired result since all of the polygons were transformed. So as to transform only the selected shape an additional cache should be specified by using the “fetch_mode=selection” processing option:

```

LAYER
    NAME "selectionshape"
    CONNECTIONTYPE GEOMTRANS
    CONNECTION "simplify"
    STATUS default
    TYPE POLYGON
    TRANSPARENCY 50
    CLASS
        STYLE
            COLOR 64 255 64
            OUTLINECOLOR 64 64 64
        END
    END
    PROCESSING "transformation=buffer"
    PROCESSING "buffer_width=0.2"
    LAYER
        NAME "simplify"
        TYPE POLYGON
        CONNECTIONTYPE GEOMTRANS
        CONNECTION "selectioncache"
        PROCESSING "transformation=convexhull"
        LAYER
            NAME "selectioncache"
            TYPE POLYGON
            CONNECTIONTYPE CACHE
            CONNECTION "/Hun_Counties_cache"
            PROCESSING "fetch_mode=selection"
        END
    END
END

```

```
        END
    END
```

And here is the result of the drawing:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample3.png>

The corresponding mapfile:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample3.map>

4.3 Using the transformed shape as the selection shape

My final goal is to select the features of the point layer using the transformed shape as the selection shape. Therefore I'll have to use the layer filter provider on the point layer and setting the `selection_layer` to the transformed layer:

```
LAYER
    TYPE POINT
    CONNECTIONTYPE LAYERFILTER
    CONNECTION "Hun_CityPoints"
    NAME "selectedpoints"
    STATUS default
    PROCESSING "selection_layer=/selectionshape"
    CLASS
        STYLE
            COLOR 255 0 0
            SYMBOL "point"
            SIZE 2
        END
    END
END
LAYER
    PROJECTION
        "AUTO"
    END
    NAME "Hun_CityPoints"
    CONNECTIONTYPE OGR
    CONNECTION "Hun_CityPoints.TAB"
    TYPE POINT
END
END
```

Here is the result:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample4.png>

The corresponding mapfile:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample4.map>

Note1: Without altering the map configuration I can modify the selection externally and the rendered image will reflect the changes automatically. For example I can select 2 counties using the query string: (`'[Name]='Tolna'` or `'[Name]='Baranya'`)

The resulting image can be found here:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample4a.png>

Note2: If there's no need to render the selection layer I can specify that configuration inline:


```

LAYER
  TYPE POINT
  CONNECTIONTYPE LAYERFILTER
  CONNECTION "Hun_CityPoints"
  NAME "selectedpoints"
  STATUS default
  PROCESSING "selection_layer=selectionshape"
  CLASS
    STYLE
      COLOR 255 0 0
      SYMBOL "point"
      SIZE 2
    END
  END
END
LAYER
  PROJECTION
    "AUTO"
  END
  NAME "Hun_CityPoints"
  CONNECTIONTYPE OGR
  CONNECTION "Hun_CityPoints.TAB"
  TYPE POINT
END
LAYER
  NAME "selectionshape"
  CONNECTIONTYPE GEOMTRANS
  CONNECTION "simplify"
  STATUS default
  TYPE POLYGON
  TRANSPARENCY 50
  CLASS
    STYLE
      COLOR 64 255 64
      OUTLINECOLOR 64 64 64
    END
  END
  PROCESSING "transformation=buffer"
  PROCESSING "buffer_width=0.2"
  LAYER
    NAME "simplify"
    TYPE POLYGON
    CONNECTIONTYPE GEOMTRANS
    CONNECTION "selectioncache"
    PROCESSING "transformation=convexhull"
    LAYER
      NAME "selectioncache"
      TYPE POLYGON
      CONNECTIONTYPE CACHE
      CONNECTION "/Hun_Counties_cache"
      PROCESSING "fetch_mode=selection"
    END
  END
END
END
END

```

Here is the result:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample5.png>

And the corresponding mapfile:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/sample5.map>

And recall that because I've used a cache on the counties layer all of these renderings require to retrieve the shapes only once from the original provider. That was the primary objective of this RFC.

5. Modifying the mapserver core

To achieve the desired functionality the following major steps should be done in the mapserver core. The details of the proposed changes can be found here:

<http://trac.osgeo.org/mapserver/attachment/ticket/2128/common.patch>

5.1 Hashtable implementation

The current hashtable implementation (in `maphash.c`) will be generalized to be capable to store objects as well as strings. Currently the hashtable can store only string values. In addition we will provide support for specifying the hashsize upon the construction of the hashtable. The objects will be stored in the hashtable by reference and the destroy function of the objects can also be specified externally. The following functions will be added to `maphash.c`

```
int initHashTableEx( hashTableObj *table, int hashSize );
void msClearHashItems( hashTableObj *table );
struct hashObj *msInsertHashTablePtr(hashTableObj *table, const char *key, const char *value);
struct hashObj *msFirstItemFromHashTable( hashTableObj *table);
struct hashObj *msNextItemFromHashTable( hashTableObj *table, struct hashObj *lastItem );
int msGetHashTableItemCount(hashTableObj *table);
```

`initHashTableEx` can be called to specify the hash size externally. Actually the original `initHashTable` will be implemented as calling `initHashTableEx` with `MS_HASHSIZE`. `msClearHashItems` will clear all of the elements of the hashtable but does not clear the items array. `msInsertHashTablePtr` provides the support for adding object by reference to the hashtable. `msFirstItemFromHashTable` and `msNextItemFromHashTable` will provide the iteration of the elements efficiently and will be called by the `NextShape` of the feature caching provider. `msGetHashTableItemCount` will retrieve the actual number of the hashtable items.

5.2 Extending the layerObj structure to support nesting the layers (map.h)

The sublayers in the `layerObj` structure will be stored as an array of layers.

```
typedef struct layer_obj {
    ...
    #ifndef SWIG
    struct layer_obj **layers;
    int numlayers; /* number of sublayers in layer */
    #endif /* SWIG */
    ...
} layerObj;
```

The parser will take care of loading the nested layers:

```
int loadLayer(layerObj *layer, mapObj *map)
{
    ...
    case (LAYER):
        if(layer->numlayers == 0)
```

```

        layer->layers = (layerObj**) malloc(sizeof(layerObj*));
    else
        layer->layers = (layerObj**) realloc(layer->layers, sizeof(layerObj*) * (layer->numlayers));
    layer->layers[layer->numlayers]=(layerObj*)malloc(sizeof(layerObj*));
    if (layer->layers[layer->numlayers] == NULL) {
        msSetError(MS_MEMERR, "Malloc of a new layer failed.", "loadLayer()");
        return MS_FAILURE;
    }
    if(loadLayer(layer->layers[layer->numlayers], map) == -1) return MS_FAILURE;
    layer->layers[layer->numlayers]->index = layer->numlayers; /* save the index */
    ++layer->numlayers;
    break;
    ...
}

```

5.3 Adding a new built in data connection types (map.h)

A new data connection type is established for the new providers.

```
enum MS_CONNECTION_TYPE {MS_INLINE, MS_SHAPEFILE, MS_TILED_SHAPEFILE, MS_SDE, MS_OGR, MS_UNUSED_1, MS_...
```

The lexer will be modified to interpret the new connection types.

5.4 Support for destroying the persistent data of the providers (map.h, maplayer.c)

The providers would keep persistent data between the various Connection/Close operations on that layer so we should establish a mechanism to destroy this provider specific data by adding a new method to the layerVtbl:

```
void (*LayerDestroy)(layerObj *layer);
```

5.5 Vtable initialization for the new data providers (maplayer.c)

```

int
msInitializeVirtualTable(layerObj *layer)
{
    if (layer->baselayer)
        msInitializeVirtualTable(layer->baselayer);
    ...
    switch(layer->connectiontype) {
    ...
    case(MS_GEOMTRANS) :
        return(msGeomTransLayerInitializeVirtualTable(layer));
        break;
    case(MS_LAYERFILTER) :
        return(msFilterLayerInitializeVirtualTable(layer));
        break;
    case(MS_CACHE) :
        return(msCacheLayerInitializeVirtualTable(layer));
        break;
    ...
    }
    ..
}

```

6. Files affected

The following files will be affected by this RFC:

```
maphash.c
maphash.h
maplayer.c
maplexer.l
mapfile.c
map.h
Makefile.vc
Makefile.in
mapcache.c (new)
mapgeomtrans.c (new)
mapfilter.c (new)
```

7. Backwards compatibility issues

These changes will retain mapfile and mapscript backward compatibility.

8. Bug ID

The ticket for RFC-22a can be found [here](#).

Bug [2128](#)

9. Voting history

None

14.6.23 MS RFC 23: Technical Steering Committee Guidelines

Date 2006/05/12

Author Frank Warmerdam, Steve Lime

Contact warmerdam at pobox.com, sdlime at dnr.state.mn.us

Last Edited 2008/01/25

Status Adopted

Id \$Id: ms-rfc-23.txt 8278 2008-12-23 21:34:31Z hobo \$

Summary

This document describes how the MapServer Project Steering Committee determines membership, and makes decisions on all aspects of the MapServer project - both technical and non-technical.

Examples of PSC management responsibilities:

- setting the overall development road map
- developing technical standards and policies (e.g. coding standards, file naming conventions, etc...)
- ensuring regular releases (major and maintenance) of MapServer software

- reviewing RFC for technical enhancements to the software
- project infrastructure (e.g. CVS/SVN, Bugzilla, hosting options, etc...)
- formalization of affiliation with external entities such as OSGeo
- setting project priorities, especially with respect to project sponsorship
- creation and oversight of specialized sub-committees (e.g. project infrastructure, training)

In brief the project team votes on proposals on mapserver-dev. Proposals are available for review for at least two days, and a single veto is sufficient delay progress though ultimately a majority of members can pass a proposal.

Detailed Process

- Proposals are written up and submitted on the mapserver-dev mailing list for discussion and voting, by any interested party, not just committee members.
- Proposals need to be available for review for at least two business days before a final decision can be made.
- Respondents may vote “+1” to indicate support for the proposal and a willingness to support implementation.
- Respondents may vote “-1” to veto a proposal, but must provide clear reasoning and alternate approaches to resolving the problem within the two days.
- A vote of -0 indicates mild disagreement, but has no effect. A 0 indicates no opinion. A +0 indicate mild support, but has no effect.
- Anyone may comment on proposals on the list, but only members of the Project Steering Committee’s votes will be counted.
- A proposal will be accepted if it receives +2 (including the author) and no vetoes (-1).
- If a proposal is vetoed, and it cannot be revised to satisfy all parties, then it can be resubmitted for an override vote in which a majority of all eligible voters indicating +1 is sufficient to pass it. Note that this is a majority of all committee members, not just those who actively vote.
- Upon completion of discussion and voting the author should announce whether they are proceeding (proposal accepted) or are withdrawing their proposal (vetoed).
- The Chair gets a vote.
- The Chair is responsible for keeping track of who is a member of the Project Steering Committee (perhaps as part of a PSC file in CVS).
- Addition and removal of members from the committee, as well as selection of a Chair should be handled as a proposal to the committee.
- The Chair adjudicates in cases of disputes about voting.

When is Vote Required?

- Any change to committee membership (new members, removing inactive members)
- Changes to project infrastructure (e.g. tool, location or substantive configuration)
- Anything that could cause backward compatibility issues.
- Adding substantial amounts of new code.
- Changing inter-subsystem APIs, or objects.
- Issues of procedure.

- When releases should take place.
- Anything dealing with relationships with external entities such as OSGeo
- Anything that might be controversial.

Observations

- The Chair is the ultimate adjudicator if things break down.
- The absolute majority rule can be used to override an obstructionist veto, but it is intended that in normal circumstances vetoers need to be convinced to withdraw their veto. We are trying to reach consensus.
- It is anticipated that separate “committees” will exist to manage conferences, documentation and web sites. That said, it is expected that the PSC will be the entity largely responsible for creating any such committees.

Committee Membership

The PSC is made up of individuals consisting of technical contributors (e.g. developers) and prominent members of the MapServer user community. There is no set number of members for the PSC although the initial desire is to set the membership at 9.

Adding Members

Any member of the mapserver-dev mailing list may nominate someone for committee membership at any time. Only existing PSC committee members may vote on new members. Nominees must receive a majority vote from existing members to be added to the PSC.

Stepping Down

If for any reason a PSC member is not able to fully participate then they certainly are free to step down. If a member is not active (e.g. no voting, no IRC or email participation) for a period of two months then the committee reserves the right to seek nominations to fill that position. Should that person become active again (hey, it happens) then they would certainly be welcome, but would require a nomination.

Membership Responsibilities

Guiding Development

Members should take an active role guiding the development of new features they feel passionate about. Once a change request has been accepted and given a green light to proceed does not mean the members are free of their obligation. PSC members voting “+1” for a change request are expected to stay engaged and ensure the change is implemented and documented in a way that is most beneficial to users. Note that this applies not only to change requests that affect code, but also those that affect the web site, technical infrastructure, policies and standards.

IRC Meeting Attendance

PSC members are expected to participate in pre-scheduled IRC development meetings. If known in advance that a member cannot attend a meeting, the member should let the meeting organizer know via e-mail.

Mailing List Participation

PSC members are expected to be active on both the mapserver-users and mapserver-dev mailing lists, subject to open source mailing list etiquette. Non-developer members of the PSC are not expected to respond to coding level questions on the developer mailing list, however they are expected to provide their thoughts and opinions on user level requirements and compatibility issues when RFC discussions take place.

Bootstrapping

Prior to the TSC anointing itself the PSC this RFC must be distributed before the MapServer community via MapServer-Users for comment. Any and all substantive comments must be discussed (and hopefully, but not necessarily, addressed via MapServer-Dev.

All members of the existing Technical Steering Committee will form the initial Project Steering Committee. Steve Lime is declared initial Chair of the Project Steering Committee.

Initial members are:

- Steve Lime
- Daniel Morissette
- Frank Warmerdam
- Assefa Yewondwossen
- Howard Butler
- Steve Woodbridge
- Perry Nacionales

There are two open committee positions at the initial formation of the PSC.

Updates

29 May 2007

The following members were added to the PSC after discussion and consensus:

- Tom Kralidis
- Jeff McKenna
- Umberto Nicoletti
- Tamas Szekeres

14.6.24 MS RFC 25: Align MapServer pixel and extent models with OGC models

Date 2007/10/23

Author Steve Lime

Contact steve.lime at DNR.STATE.MN.US

Status Draft

Version

Id \$Id: ms-rfc-25.txt 8278 2008-12-23 21:34:31Z hobo \$

Overview

At present MapServer uses different pixel and extent model than defined by OGC services such as WCS and WMS. MapServer uses the center of a pixel to represent its unique coordinate value. An extent is interpreted as the bounding box that runs from the center of the UL pixel in an image to the center of the LR pixel in an image. Why? Well, it goes back to companion software that existed along side MapServer to display satellite data stored in ERDAS that used the center to center extent model. The math is simple and there is a certain logic in having the extent actually represent pixel values - that is, if you render the extent as a polygon you get the exact edge of the image as one might expect.

On the other hand, OGC service specifications define an extent (BBOX) to refer to the dimensions of the outside edges of the image being requested. This appears to be a far more common means of expressing an area of interest.

I've not been able to ascertain where the coordinate of an individual pixel is located from various OGC specifications. MapServer "could" retain a center-based pixel model. That does add complexity to the map \Leftrightarrow image coordinate transformations since you have to offset things by one-half cellsize. Since that computation is done many times I would expect a performance hit. We could optimize things by computing and storing the one-half cellsize value once (as cellsize is now), but that complicates the C APIs and requires huge amounts of change. I propose moving to a upper-left-based pixel model to simplify these conversions.

Note: In looking at the code there were past efforts to go to the OGC extent in 4.8 and 4.10, but it was not universally applied. This RFC would ensure that the same extent and pixel model is in use throughout MapServer and that it is consistent with OGC.

model diagram: http://maps.dnr.state.mn.us/mapserver_docs/rfc25_extent.pdf

Technical Details

Affected files (relative to what is in main development trunk):

- map.h: change definition of MS_CELLSIZE

```
#define MS_CELLSIZE(min,max,d) ((max - min)/d)
```

- map.h: change coordinate conversion macros (no change just add comments)

```
/*
** These macros work relative to the UL corner of the UL pixel of a map extent. Pixel
** model is (as of 5.0) the UL corner of a pixel. UL pixel = minx,maxy.
*/
#define MS_MAP2IMAGE_X(x,minx,cx) (MS_NINT((x - minx)/cx))
#define MS_MAP2IMAGE_Y(y,maxy,cy) (MS_NINT((maxy - y)/cy))
#define MS_IMAGE2MAP_X(x,minx,cx) (minx + cx*x)
#define MS_IMAGE2MAP_Y(y,maxy,cy) (maxy - cy*y)
```

- maputil.c: update msAdjustExtent()

```
ox = MS_MAX((width - (rect->maxx - rect->minx)/cellsize)/2,0);
oy = MS_MAX((height - (rect->maxy - rect->miny)/cellsize)/2,0);
```

- mapwcs.c: remove code to convert between MapServer extent and OWS extent
- mapwms.c: remove code to convert between MapServer extent and OWS extent
- mapwmsclient.c remove code to convert between MapServer extent and OWS extent
- various raster layer handlers: Need Frank's comments here.
- mapscript ...zoom functions: I don't believe these will need any change.

The affect of these changes on end-users should be minimal since it is unlikely they the are aware of differences in extent interpretation. The larger impact may be on 3rd party applications like dBox, Chameleon and QGIS that manage extents and call MapServer. Efforts must be made to make those folks aware of the change.

Mapfile Implications

None, these are internal changes only.

MapScript Implications

None, these are internal changes only.

Documentation Implications

The models for a pixel and an extent need to be documented in a couple of places: mapfile reference, the related OWS service how-to's and perhaps a new how-to pertaining to just this topic.

Test Plan

Need to develop some tests to somehow validate the math. The WCS interface or output drivers using GDAL are excellent candidates since they produce georeferenced output. Some testing has already been done to verify the existence of the issue (first discovered via WCS) and the proposed fix.

Mini-images (e.g. 7x7) can be used to verify rendering makes sense. For example, if you draw the extent as a polygon you would expect to see lines for the left and top edges, but not for the bottom and right. That is because the maxx,miny extent values won't represent a pixel in the output (rather the next tile to the right and/or below).

Staffing / Timeline

Changes to maputil.h, map.h, mapwcs.c and mapwms.c would be done by Steve Lime. Changes to other portions of MapServer would be coordinated with the various component owners. This RFC would be completed for the 5.0 release.

14.6.25 MS RFC 26: Version 5 Terminology Cleanup

Date 2007/04/21

Author Steve Lime, Havard Tveite

Contact steve.lime at dnr.state.mn.us, havard.tveite at umb.no

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Implemented

Version MapServer 5.0

Id \$Id: ms-rfc-26.txt 8278 2008-12-23 21:34:31Z hobo \$

Mapserver terminology is mostly good and consistent, with a few exceptions. The two that generate the most confusion are TRANSPARENCY (layerObj) and various scale referencing parameters (e.g. MINSCALE).

Another change would be changing the symbolObj STYLE parameter to PATTERN instead. This would be simply to avoid confusion with the classObj STYLE.

The purpose of this proposal is to make Mapserver even easier to use, removing confusion that can arise in cases of inconsistent terminology.

TRANSPARENCY

1. Overview

TRANSPARENCY is used in the Mapserver mapfile layer object for what is in fact opacity (as is also pointed out in the current documentation).

2) Technical details To achieve more consistent terminology, the following should be done:

Changing the keyword TRANSPARENCY to OPACITY

The old mapfile layer keyword “TRANSPARENCY” will be deprecated, but shall be supported in future versions of Mapserver as an alias for OPACITY.

The new mapfile layer keyword OPACITY should replace TRANSPARENCY. There will be no changes in type or semantics, only a change of keyword name.

The internal structure member name in the layerObj will change from transparency to opacity.

1. Mapfile Implications

The parser will have to support both OPACITY and TRANSPARENCY (for backward compatibility). The type will not change.

At debug level 1 a warning will be issued that the TRANSPARENCY parameter is deprecated and OPACITY should be used instead.

1. MapScript Implications

Since the layerObj will no longer contain a transparency member this will break old scripts. The fix will be evident.

1. Documentation Implications

Documentation should be updated, introducing OPACITY as a new layer keyword with documentation similar to the current transparency. The layer keyword TRANSPARENCY should be documented as deprecated. All other documentation that references TRANSPARENCY will have to be updated (simple search-replace with a manual check of all occurrences of the word transparency?)

SCALE

1. Overview

SCALE and MAXSCALE/MINSCALE is a case where Mapserver terminology is not in line with mainstream map terminology.

In proper usage, scale is a representative fraction. The scale 1:50000 tells us that one meter on the map corresponds to 50000 meters in the “terrain”. 1:1000 (0.001) is a larger scale than 1:50000 (0.00002). The current use of MAXSCALE and MINSCALE is therefore not consistent with proper terminology.

1. Technical details

To achieve more consistent terminology, the following could be done with limited consequences:

Change all occurrences of “SCALE” in keywords to “ScaleDenom”.

```
MinScale 10000 Maxscale 1000000
```

will become:

```
MinScaleDenom 10000 MaxScaleDenom 1000000
```

This will apply to the following occurrences of SCALE in mapserver keywords:

Map object: SCALE -> ScaleDenom

Web object: MAXSCALE/MINSCALE

Layer object: MAXSCALE/MINSCALE LABELMAXSCALE/LABELMINSCALE SYMBOLSCALE

Class object: MAXSCALE/MINSCALE

1. Mapfile Implications

The parser would have to accept the new keywords. The old keywords will be unique prefixes of the new keywords, and could be accepted by the parser (in a transition period or for eternity).

At debug level 1 a warning will be issued that the ...SCALE parameter is deprecated and ...SCALEDENOM should be used instead.

4) MapScript Implications For all scale variables, scale should be changed to scaledenom (for consistency reasons). The type (double) will not change. Like opacity above this could break old scripts.

1. Template Implications

The [scale] substitution should be deprecated (but still supported) and [scaledenom] added.

1. CGI Implications

The scale CGI parameter would continue to be supported (e.g. users often define a map extent using a center point and a scale value). Scaledenom would also be supported.

1. Documentation Implications

All occurrences of scale keywords in the documentation will have to be changed to reflect the new names. The old style keywords could be included, but should be marked as deprecated.

PATTERN

1. Overview

Currently both the symbolObj and classObj contain STYLE parameters. The symbolObj style stores dash patterns used for line symbols. The name is derived from GD terminology. To avoid confusion with the classObj STYLE the symbolObj STYLE should be renamed to more concisely reflect its purpose.

1. Technical details

In the symbolObj structure definition the member style will be renamed pattern. All references to the style in the code will also be updated.

1. Mapfile/symbol file Implications

The parser would have to accept the new keywords. The old keywords will be unique prefixes of the new keywords, and could be accepted by the parser (in a transition period or for eternity).

At debug level 1 a warning will be issued that the STYLE parameter is deprecated and PATTERN should be used instead.

1. MapScript Implications

As above the symbolObj structure would be altered and so MapScript scripts that set a symbol style programmatically would break. They would need to reference the patter parameter instead.

1. Template Implications

None

1. CGI Implications

None

1. Documentation Implications

The symbolObj reference guides would need to be updated to reflect the name change.

14.6.26 MS RFC 27: Label Priority

Date 2007/05/22

Author Daniel Morissette

Contact dmorissette at mapgears.com

Last Edited 2007/06/29

Status Adopted (2007/05/25) - Implementation completed (2007/07/05)

Version MapServer 5.0

Id \$Id: ms-rfc-27.txt 8278 2008-12-23 21:34:31Z hobo \$

Overview

MapServer 4.10 and older used a last in first out (LIFO) mechanism to plot labels on a map. This resulted in excessive use of ANNOTATION layers to make certain labels more prominent. This RFC introduces a new PRIORITY parameter on the LABEL object to control the order in which labels are rendered.

Technical Solution

PRIORITY is a new LABEL parameter that takes an integer value between 1 (lowest) and MS_MAX_LABEL_PRIORITY (highest). The default value is 1.

MS_MAX_LABEL_PRIORITY is defined and can be altered in map.h, its default value is 10.

The prioritization is handled by maintaining an array of MS_MAX_LABEL_PRIORITY cache lists in the label cache. When a label is added to the label cache, its priority index is used to decide in which cache list it should be added.

Then at rendering time, we loop through the cache lists, starting with the highest priority list.

Specifying an out of range PRIORITY value inside a map file will result in a parsing error. An out of range value set via MapScript or coming from a shape attribute will be clamped to the min/max values in msAddLabel().

There is no expected impact on performance for using label priorities.

Support for attribute binding

The PRIORITY parameter can also be bound to an attribute using the attribute bindings mechanism defined in RFC-19. This means two ways to set LABEL PRIORITY:

```
...
LABEL
  PRIORITY 5
  ...
END
...
```

or

```

...
LABEL
  PRIORITY [someattribute]
  ...
END
...

```

Modifications to the source code

- PRIORITY will be added to the LABEL object in map.h, in the mapfile parser/writer (mapfile.c) and in MapScript
- A MS_IS_VALID_LABEL_PRIORITY() macro will be defined to validate priority ranges in a consistent way everywhere.
- The label cache code (maplabel.c) will be modified to work with an array of MS_MAX_LABEL_PRIORITY cache lists instead of a single list
- The various msDrawLabelCacheXX() functions will be modified to replace the current loop on cache items with two nested loops: the outer loop will iterate on cache lists (from highest to lowest), and the inner loop will iterate on the cache items inside each list.
- msBindLayerToShape() will be updated to support binding PRIORITY to a shape attribute field.

MapScript Implications

The labelObj will have a new priority property of type integer.

Files affected

```

map.h
mapfile.c
maplabel.c
maputil.c
mapgd.c      (msDrawLabelCacheGD)
mapimagemap.c (msDrawLabelCacheIM)
mappdf.c     (msDrawLabelCachePDF)
mapsvg.c     (msDrawLabelCacheSVG)
mapswf.c     (msDrawLabelCacheSWF)
mapagg.cpp   (msDrawLabelCacheAGG)

```

Backwards compatibility issues

None.

Bug ID

- 1619: <https://trac.osgeo.org/mapserver/ticket/1619>
- Bug 206 also made mention of label priority but has been closed as duplicate of 1619: <https://trac.osgeo.org/mapserver/ticket/206>

Voting history

Vote completed on 2007-05-25:

+1 from DanielM, SteveW, SteveL, YAssefa, UmbertoN and FrankW

Questions/Comments from the review period

- Q: Why use an array of cache lists instead of doing a quicksort on all cache entries?

A: Mainly for performance reason, but it was also pointed out that quicksort is not stable and could result in different orderings depending on the set of labels.

14.6.27 MS RFC 28: Redesign of LOG/DEBUG output mechanisms

Date 2007/06/14

Author Daniel Morissette

Contact dmorissette at mapgears.com

Last Edited 2007/08/24

Status Adopted (2007-06-29) - Implementation completed (2007/07/06)

Version MapServer 5.0

Id \$Id: ms-rfc-28.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

MapServer 4.10 and older used to have multiple LOG/DEBUG output mechanisms that did not play well together. In order to improve the usability of the software this RFC proposes a new LOG/DEBUG output mechanism with more control on the output location (logfile or stderr) and that works under all supported web servers.

Inventory of existing mechanisms

MapServer 4.10 and older support the following LOG/DEBUG systems:

- LOG [filename] in the WEB object:

From the documentation: “File to log MapServer activity in. Must be writable by the user the web server is running as.”

More specifically, the function writeLog() in mapserv.c logs information about the mapserv CGI request results at the end of its execution. This option has no effect with MapScript.

- MS_ERRORFILE environment variable:

If set then all calls to msSetError() are logged to this file.

- msDebug() with DEBUG ON/OFF at the MAP and LAYER level:

There are msDebug() calls in various areas of the code that generate information that may be useful in tuning and troubleshooting apps. The DEBUG ON/OFF statements in the MAP and LAYER objects are used to enable/disable the msDebug() calls. The msDebug() mechanism is also disabled by default and enabled only by the -DENABLE_STDERR_DEBUG compile-time option. Users could benefit from easier access to this and more debug information.

The output of `msDebug()` is sent to `stderr` and cannot be redirected. In addition to not being very flexible, this is a problem under IIS where `stderr` output goes to `stdout` and the `msDebug()` output corrupts MapServer's output.

Questions

- Q: Is the `mapserv-specific LOG` option really used by anyone? Do we need to maintain it, should we get rid of it, or perhaps extend it? At a minimum this option should be better documented to avoid confusion with the other mechanisms described in this RFC.

A: The `LOG` option will be left untouched by this RFC.

Technical Solution

We will essentially merge the `MS_ERRORFILE` and `DEBUG/msDebug()` mechanisms. The variable `MS_ERRORFILE` will specify the location of the output, with possible values being either a file path on disk, or one of "stderr" or "stdout".

The current `DEBUG ON/OFF` mechanism that controls `msDebug()` output will also be extended to support multiple `DEBUG` levels.

In order to keep the implementation relatively simple and efficient, the output file handle will be kept open with a reference to it in a global variable in the context of the current request and will be closed only in `msCleanup()`. This means that if two mapfiles are loaded in the same request with different `MS_ERRORFILE` settings then errors/debug statements for both will go to the same file (and the output file will change when the second mapfile is read).

Note: the output file handle will be global in the context of the current request only and not global across threads, this will be done using the same mechanism currently used by `msGetErrorObj()` in `maperror.c`.

Setting `MS_ERRORFILE`

The variable `MS_ERRORFILE` will specify the location of the output, with possible values being either a file path on disk, or one of the following special values:

- "stderr" to send output to standard error. Under Apache `stderr` is the Apache `error_log` file. Under IIS `stderr` goes to `stdout` so its use is discouraged. With IIS it is recommended to direct output to a file on disk instead.
- "stdout" to send output to standard output, combined with the rest of MapServer's output
- "windowsdebug" to send output to the Windows `OutputDebugString` API, allowing the use of external programs like `SysInternals debugview` to display the debug output.

It will be possible to specify `MS_ERRORFILE` either as an environment variable or via a `CONFIG` directive inside a mapfile:

```
CONFIG "MS_ERRORFILE" "/tmp/mapserver.log"
```

or

```
CONFIG "MS_ERRORFILE" "stderr"
```

If both the `MS_ERRORFILE` environment variable is set and a `CONFIG MS_ERRORFILE` is also set, then the `CONFIG` directive takes precedence.

If `MS_ERRORFILE` is not set then error/debug logging is disabled. During parsing of a mapfile, error/debug logging may become available only *after* the `MS_ERRORFILE` directive has been parsed.

DEBUG levels

The current DEBUG ON/OFF mechanism, at the layer and map level will be extended to support multiple debug levels as follows. The default is DEBUG OFF (Level 0):

- Level 0: Errors only (DEBUG OFF, or DEBUG 0)

In level 0, only `msSetError()` calls are logged to `MS_ERRORFILE`. No `msDebug()` output at all. This is the default and corresponds to the original behavior of `MS_ERRORFILE` in MapServer 4.x

- Level 1: Errors and Notices (DEBUG ON, or DEBUG 1)

Level 1 includes all output from Level 0 plus `msDebug()` warnings about common pitfalls, failed assertions or non-fatal error situations (e.g. missing or invalid values for some parameters, missing shapefiles in tileindex, timeout error from remote WMS/WFS servers, etc.)

- Level 2: Map Tuning (DEBUG 2)

Level 2 includes all output from Level 1 plus notices and timing information useful for tuning mapfiles and applications

- Level 3: Verbose Debug (DEBUG 3)

All of Level 2 plus some debug output useful in troubleshooting problems such as WMS connection URLs being called, database connection calls, etc.

- Level 4: Very Verbose Debug (DEBUG 4)

Level 3 plus even more details...

- Level 5: Very Very Verbose Debug (DEBUG 5)

Level 4 plus any `msDebug()` output that might be more useful to the developers than to the users.

The MS_DEBUGLEVEL environment variable

Debug level can also be set using the (optional) `MS_DEBUGLEVEL` environment variable.

When set, this value is used as the default debug level value for all map and layer objects as they are loaded by the mapfile parser. This option also sets the debug level for any `msDebug()` call located outside of the context of a map or layer object, for instance for debug statements relating to initialization before a map is loaded. If a DEBUG value is also specified in the mapfile in some map or layer objects then the local value (in the mapfile) takes precedence over the value of the environment variable.

This option is mostly useful when tuning applications by enabling timing/debug output before the map is loaded, to capture the full process initialization and map loading time, for instance.

MapScript Implications

No direct implication. Setting `MS_ERRORFILE` will enable debug output in MapScript as well.

Note that output to `stderr` or `stdout` may not work as expected in some scripting environments and we do not plan to make any special efforts to support those special cases as part of this RFC. However, logging to a file on disk will work with all MapScript flavours.

Files affected


```
map.h
mapfile.c
maperror.c
```

Several source files with `msDebug()` calls in them may also need to be edited to adjust the debug level at which the various `msDebug()` calls kick in.

Backwards compatibility issues

- The MapServer 4.x and older `MS_ERRORFILE` continues to work as before by default
- The DEBUG ON/OFF mechanism continues to work as before. However some `msDebug()` output from previous versions of MapServer may require a higher debug level in order to be enabled.

Bug ID

- 709: <https://trac.osgeo.org/mapserver/ticket/709>
- 1333: <https://trac.osgeo.org/mapserver/ticket/1333>
- 1783: <https://trac.osgeo.org/mapserver/ticket/1783>
- 2124: <http://trac.osgeo.org/mapserver/ticket/2124>

Voting history

Vote completed on 2007-06-29:

+1 from DanielM, TomK, FrankW, TamasS, JeffM, SteveL and AssefaY.

Questions/Comments from the review period

- Q: Can we catch the output of OGR/GDAL's `CPLDebug` calls, perhaps when the map-level debug level is ≥ 3 ?
A: Frank suggests that we keep this for a later phase.

14.6.28 MS RFC 29: Dynamic Charting Capability

Date 2007/06/15

Author Daniel Morissette

Contact dmorissette at mapgears.com

Last Edited 2007/07/31

Status Adopted (2007/07/05) - Completed (2007/07/09)

Version MapServer 5.0

Id \$Id: ms-rfc-29.txt 8278 2008-12-23 21:34:31Z hobo \$

Overview

This RFC proposes the addition of simple dynamic charting capability in MapServer 5.0 by integrating the patch proposed by Thomas Bonfort in ticket #1800.

Note: this is already implemented and working (as a patch in ticket 1800). This RFC is to document the new feature and (hopefully) include it in the MapServer 5.0 release.

Technical Solution

A new CHART layer type is created and the initial implementation would support pie and bar charts.

PROCESSING directives are used to set the layer's CHART_TYPE (pie or bar) and CHART_SIZE,

Then we have one class for each pie slice or bar from the chart. Inside each class, the STYLE COLOR defines the color of the pie slice or bar, and the STYLE SIZE is used to set the relative size (value) of each pie slice or bar. This is mostly useful in combination with attribute binding of course. e.g.

Example:

```
LAYER
  NAME "Ages"
  TYPE CHART
  CONNECTIONTYPE postgis
  CONNECTION "blabla"
  DATA "the_geom from demo"
  PROCESSING "CHART_TYPE=pie"
  PROCESSING "CHART_SIZE=30"
  STATUS ON
  CLASS
    NAME "Population Age 0-19"
    STYLE
      SIZE [v1006]
      COLOR 255 244 237
    END
  END
  CLASS
    NAME "Population Age 20-39"
    STYLE
      SIZE [v1007]
      COLOR 255 217 191
    END
  END
  CLASS
    NAME "Population Age 40-59"
    STYLE
      SIZE [v1008]
      COLOR 255 186 140
    END
  END
END
```

In the example above, if for a given shape we have $v1006=1000$, $v1007=600$ and $v1008=400$ then the actual pie slices for each class will be respectively 50%, 30% and 20% of the total pie size. If we produced bar charts then the values would represent the relative height of the bars with the largest value (highest bar) being 100% of the chart height.

The following attachment to ticket 1800 contains a sample map produced by the layer definition above:

<https://trac.osgeo.org/mapserver/attachment/ticket/1800/chart-test.jpg>

The layer's legend behaves as usual and produces one color sample per class.

Issues and limitations

- The initial implementation supports only GD output formats. However Thomas Bonfort has offered to implement an AGG version once support for it is available. This is considered a future enhancement outside of the scope of this RFC. Update 2007/09/06: chart rendering is supported by the AGG renderer too.
- Should we use special keywords instead of PROCESSING parameters to specify the chart type and size?
- The values of each class are taken from the SIZE of the corresponding STYLE, which is semantically awkward (but that saves us from creating new keywords)

MapScript Implications

The new CHART type (constant) would be exposed via MapScript. There are no other MapScript implications.

Files affected

```
map.h           (new MS_LAYER_CHART constant)
mapfile.c
maplexer.l     (new CHART keyword)
mapdraw.c      (hooks to call chart rendering code)
mapchart.c     (implementation of chart rendering)
maplegend.c    (add case for chart layer type)
Makefile.in    (addition of mapchart.o)
makefile.vc    (addition of mapchart.obj)
```

Note:

The patch from ticket 1800 also contains changes to mappostgis.c and mapmygis.c to treat layer type CHART the same way as POLYGON but I'm not exactly sure why or if the same needs to be done for other data sources. This will need further review before being released.

Backwards compatibility issues

None. This is a new feature.

Bug ID

- 1800: <https://trac.osgeo.org/mapserver/ticket/1800>

Future enhancements:

- 2136: <https://trac.osgeo.org/mapserver/ticket/2136>
- 2145: <https://trac.osgeo.org/mapserver/ticket/2145>

Documentation

- <http://mapserver.gis.umn.edu/docs/howto/dynamic-charting>

Voting history

Vote completed on 2007/07/05:

+1 from DanielM, SteveW, AssefaY

Questions/Comments from the review period

- Q: Does this assume a POLYGON data type input only? Can LINE and POINT data sources also be supported?

A: The current implementation works only for polygon data sources. I could extend it to work on point data sources as well fairly easily I think.

I'm not sure about line data sources though. I guess we could do like we do for ANNOTATION layers and use `msPolylineLabelPoint()` to determine the location of the chart.

Ticket #2145 has been opened to track this enhancement: <https://trac.osgeo.org/mapserver/ticket/2145>

- Q: Could we support attribute binding in `CHART_SIZE = [size]`? This would allow you to show the relative size of hits at a point and have the chart show the percentage contributions in the wedges.

A: While this would be a nice feature, we will keep it as a potential enhancement for a future release. See ticket #2136: <https://trac.osgeo.org/mapserver/ticket/2136>

14.6.29 MS RFC 30: Support for WMS 1.3.0

Date 2007/06/15

Author Daniel Morissette

Contact dmorissette at mapgears.com

Author Yewondwossen Assefa

Contact yassefa at dmsolutions.ca

Last Edited 2009/02/11

Status Draft

Version MapServer 5.4

Id \$Id: ms-rfc-30.txt 8883 2009-04-06 14:45:30Z assefa \$

Overview

This RFC documents the changes required in order to upgrade MapServer's OGC WMS support to version 1.3.0 of the specification.

MapServer already includes mechanisms to support multiple WMS versions (and already supports WMS versions 1.0.0, 1.1.0 and 1.1.1) so in theory this upgrade should be straightforward and shouldn't require a RFC. Unfortunately, WMS 1.3.0 contains some tricky changes that while they are not exactly backwards incompatible are likely to make the life of users of WMS miserable.

This RFC is mostly to document those changes and the way MapServer deals with them.

Coordinate Systems and Axis Orientation

The main issue introduced by WMS 1.3.0 is the change in the way it handles axis order for several SRS. This has an impact on the way the BBOX is specified in WMS requests and in Capabilities documents and in how the CONNECTIONTYPE WMS code interacts with remote servers.

In previous versions of WMS, for any SRS the first axis was the easting (x or lon) and the second axis was the northing (y or lat). Starting with WMS 1.3.0, some SRS such as the very popular EPSG:4326 have their axis reversed and the axis order becomes lon, lat instead of lat, lon. This change in WMS 1.3.0 was done in order to align with the definitions from the EPSG database (a requirement to make WMS an ISO specification).

This change is sure to confuse simple clients that used to treat all SRS the same way. MapServer and PROJ will need to be extended to carry information about the axis order of all EPSG SRS codes and treat them using the correct axis order.

New CRS codes such as CRS:xxxx and AUTO2:xxxx have also been added by WMS 1.3.0 that will need to be supported by MapServer. Note the two types of Layer CRS identifiers are supported by the specification: “label” and “URL” identifiers. The intention is to support at this point the label types CRS.

List of CRS planned to be supported are:

- CRS:84 (WGS 84 longitude-latitude)
- CRS:83 (NAD83 longitude-latitude)
- CRS:27 (NAD27 longitude-latitude).
- AUTO2:42001 AUTO2:42002 AUTO2:42003 AUTO2:42004 AUTO2:42005

CRS:1 (pixel coordinates) would not be supported at this point (there is a ticket discussing this issue <http://trac.osgeo.org/mapserver/ticket/485>)

EPSG codes: when advertising (such as BoundingBox for a layer element) or using a CRS element in a request such as GetMap/GetFeatureInfo, elements using epsg code ≥ 4000 and < 5000 will be assumed to have a reverse axes.

All the above need to be done in a way that allows continued support for older versions of the WMS specification (1.0.0 to 1.1.1) and will have the least impact on existing WMS services.

Implementation for the reverse axis will follow closely to what was already done for WCS1.1 support (<http://mapserver.org/development/rfc/ms-rfc-41.html>)

WMS and SLD

All references to SLD support has been removed from the WMS 1.3.0 specifications. It has been replaced by two specifications:

- Styled Layer Descriptor profile of the Web Map Service
- Symbology Encoding Implementation Specification

The Styled Layer Descriptor profile allows:

- to extend the WMS to support additional operations (DescribeLayer, GetLegendGraphic)
- additional parameters related to the SLD for the GetMap support
- advertise SLD support.

This specification will be supported in the current implementation. Note that the GetStyles operation (available in WMS 1.1.1) might not be supported in the first phase

The Symbology Encoding Implementation represent basically the definitions for the different symbolizers. We need to upgrade the current SLD support to support this specification.

HTTP Post support

HTTP Post support is optional and currently supported by Mapserver WMS 1.1.1. WMS 1.3.0 defines that if POST is supported, the the request message is formulated as an XML document. Although this is highly desirable, the first implementation of the WMS 1.3.0 might not support the XML Post requests.

OGC compliance tests

The OGC Compliance and Interoperability Testing Initiative (CITE) http://cite.opengeospatial.org/test_engine/wms/1.3.0/ provides automatic tests to validate the implementation. The short term intention is to use this service as a first validation tool. The longer term goal is to have Mapserver WMS 1.3.0 fully compliant.

Other Notes

- GetCapabilities advertises only the text/xml format. During a request we do not parse the optional Format parameter. It is always set to text/xml

MapScript Implications

None. This affects only the WMS server interface and WMS CONNECTION type.

Files affected

```
mapwms.c
mapwmslayer.c
mapfile.c
mapows.c/h
mapogcsld.c
```

Backwards compatibility issues

- The change in the way the axis order is handled is likely to cause lots of confusion.

Bug ID

- 473: <http://trac.osgeo.org/mapserver/ticket/473>

Voting history

Passed with +1 from: TomK, Woodbridge, Assefa, Morissette

Questions/Comments from the review period

- Q: Can libxml2 be used to generate XML responses to continue the work started in mapowscommon.c?
A: I'll keep libxml2 in mind during the implementation, but I do not plan to refactor and risk breaking any code to convert it to libxml2 as part of this upgrade.

Since WMS 1.3.0 doesn't implement OWS common, it won't benefit from any of the code that's already using libxml2. It will actually mostly reuse existing printf-based code that's already well tested and working. I think the right time to switch to libxml2 for WMS would be when it will support OWS common and then there will be real benefits by reusing functions from mapowscommon.c.

14.6.30 MS RFC 31: Loading MapServer Objects from Strings

Date 2007/06/19

Author Steve Lime

Contact Steve.Lime at DNR.State.MN.US

Version 5.0

Status Accepted (2007/06/22) Implemented

Id \$Id: ms-rfc-31.txt 8278 2008-12-23 21:34:31Z hobu \$

Description: This RFC addresses the ability of the MapServer tokenizer (in maplexer.l and mapfile.c) to work from strings as well as files. A mapfile-wide ability was added to 5.0 source and this RFC looks at loading MapServer objects (layers, scalebars, etc...) via MapScript and via URLs.

Current State

Presently MapServer can load entire mapfile's from a text block using `msLoadMapFromString`. This is a new capability in 5.0. MapServer has long been able to load/modify individual values via URL using a `map_object_property` syntax (e.g. `map_scalebar_units`).

The problem with the URL support is that it is cumbersome for the user and results in a ton of duplicative code in `mapfile.c` making maintenance difficult. Developers will often add a parameter but forget to add a URL equivalent. This proposal removes that redundant code and relies on a single tokenizing function for each object.

C API Changes

All major objects would get a new `...LoadFromString` function (e.g. `msLoadLayerFromString` and so on). These functions would be very simple and would take an existing reference to an object and a string snippet. They would:

1. establish lexer thread locks
2. set lexer state to `MS_TOKENIZE_STRING`
3. call `loadObject` (e.g. `loadLayer`)

In effect this would be a way to load an empty object or update a new one.

The `loadObject` functions would need minor changes:

1. Each function would need to remove restrictions for duplicate properties. That is setting a parameter twice should not generate an error as is does now.
2. Properties with allocated memory (e.g. `char *`) should be free'd if they already have values and are being updated.
3. the object main keyword (e.g. `LAYER` or `CLASS`) should be allowed as a token within that object loader. When parsing a file the object identifier (e.g. `LAYER`) is stripped off with the parent object. For example, a `CLASS` is recognized by `loadLayer` so that token never is encountered by `loadClass`. It makes the most sense to pass entire object definitions including the object identifier for ease of use.

MapScript

I'm open to suggestions but I think the easiest thing to do would be to add an `updateFromString` method to all major objects. It would simply take a string snippet and would wrap the `...LoadFromString` methods mentioned above. They would return `MS_SUCCESS` or `MS_FAILURE`. Might consider adding a "clear" method to (`freeObject` then `initObject`) so that users could clean things out and reload from a string. I'm not sure about the effects on reference counting here.

URL

I propose removing all the `loadObjectValue` (e.g. `loadLayerValue`) functions in favor of entire object loading. So, instead of doing something like:

```
...map_scalebar_units=meters&map_scalebar_intervals=5&map_scalebar_size=300+2...
```

You would do:

```
...map_scalebar=UNITS+METERS+INTERVALS+5+SIZE+300+2...
```

The major objects would still be referenced by `map_scalebar` or `map_legend` or `map_layername`, but all other properties would be loaded through snippets.

The function `msLoadMapParameter` would become `msUpdateMapFromURL` and it would set the lexer state, acquire a thread lock and then call the appropriate `loadObject` function.

One issue is that the `loadObject` functions have traditionally worked just from files so there are no limitations on what can be altered. Obviously from a URL you can't allow just anything to be altered (e.g. `CONNECTION`, `DUMP` and so on). So, we would create a new lexer state, `MS_TOKENIZE_URL`, that would only recognize the parameters that we want. In that state the lexer would not return tokens like `DUMP` or `CONNECTION` so the `loadObject` functions would not handle those cases. This is a simple addition to the lexer. Any parameter exposed to URL modification will have the relevant loading block examined so that there are no memory leaks or buffer overflow possibilities.

In addition, it was pointed out that URL configuration should not be a default behavior but should be enabled explicitly. Enabling this feature would happen by way of a new parameter within the `webObj-URLCONFIG [pattern]`, with a default of `NULL`. The pattern would be a regular expression that would be applied against any `map_*` variables. So, one could limit changes to just the scalebar object with `URLCONFIG 'scalebar'` or allow more with `URLCONFIG '.'`. The default would not to be allow any URL configuration.

Backwards Compatibility

The URL change will break backwards compatibility but I feel this is a relatively lightly used option and this change will be very beneficial.

Post Implementation Notes

Apparently a number of folks are having trouble with porting applications to use the new url configuration. Below are more examples and lists of supported keywords by object type. Rule of thumb one: when there is the opportunity for more than one of a particular object (e.g. layers, classes and styles) the syntax must uniquely identify the object in question in the variable name (e.g. `map.layer[lakes]`) and then the mapfile snippet to modify the object is given as the variable value. We have no way to modify 5 styles at once because the mapfile syntax is so freeform. Rule of thumb two: any parameters or objects that hang off the `mapObj` must be referenced in the variable name (e.g. `map.imagetype`).

Example 1, changing a scalebar object:


```
...&map.scalebar=UNITS+MILES+COLOR+121+121+121+SIZE+300+2&...
```

Example 2, changing a presentation style:

```
...&map.layer[lakes].class[0].style[0]=SYMBOL+crosshatch+COLOR+151+51+151+SIZE+15&...
```

Example 3, creating a new feature:

```
...&map_layer[3]=FEATURE+POINTS+500000+1000000+END+TEXT+'A+test+point'+END&...
```

Changeable objects/keywords by object type.

mapObj (example - ...&map.angle=50&map.imagecolor=255+0+0&...)

angle, config, extent, imagecolor, imagetype, layer, legend, projection, querymap, reference, resolution, scalebar, size, shapepath, transparent, units, web

layerObj (example - ...&map.layer[lakes].data=myTempShapefile&...)

class, data (subject to DATAPATTERN validation), feature, footer (subject to TEMPLATEPATTERN validation), header (subject to TEMPLATEPATTERN validation), labelitem, opacity, projection, status, template (subject to TEMPLATEPATTERN validation), tolerance, units

classObj (example - ...&map.layer[lakes].class[0].style[1]=COLOR+255+0+0...)

color, label, outlinecolor, overlaycolor, overlayoutlinecolor, overlaysize, overlaysymbolsize, size, status, style, symbol, text (note that setting of color etc... should really be done through a styleObj and not the class shortcuts)

labelObj (example - ...&map.scalebar=LABEL+COLOR+255+0+0+SIZE+15+END)

angle, antialias, backgroundcolor, backgroundshadowcolor, backgroundshadowsize, color, font, outlinecolor, position, shadowcolor, shadowsize, size

styleObj (example - ...&map.layer[lakes].class[0].style[0]=COLOR+255+0+0+ANGLE+50+SIZE+30...)

angle, backgroundcolor, color, outlinecolor, size, symbol, width

featureObj (example - ...&map_layer[3]=FEATURE+POINTS+500000+1000000+END+TEXT+'A+test+point'+END&...)

points, text, wkt

More to come...

Bug IDs

<http://trac.osgeo.org/mapserver/ticket/2143>

Voting history

+1 from SteveL, SteveW, TomK, FrankW, AssefaY, PericlesN

+0 from JeffM

14.6.31 MS RFC 32: Support for Anti-Grain Geometry (AGG) Rendering Engine

Date 2007/07/09

Author Steve Lime, John Novak

Contact Steve.Lime at DNR.State.MN.US

Status Pending

Id \$Id: ms-rfc-32.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

Presently MapServer supports GD (www.libgd.org) as its primary raster rendering backend. While GD is sufficient in many instances it is not capable of high quality output especially with regards to anti-aliased line work. MapServer does support pseudo anti-aliased wide lines using variable opacity “fuzzy” buffers, but the results are not as good as they could be.

AGG has emerged as one of the premier software-only rendering solutions and it holds the promise of superior output quality with little or no apparent loss in performance. In fact, AGG may well be faster than GD in some instances even with the higher quality output.

That said, we are still heavily vested in GD for many things. Text positioning and raster rendering in particular use GD functions directly. On the other hand, AGG does not have built in functions to read or write popular graphics formats such as GIF, JPEG or PNG. It makes sense then to consider a hybrid solution wherein we can take advantage of aspects of GD that make sense such as buffer management and I/O capabilities, and let AGG worry about rendering features. That’s exactly what is proposed- AGG rendering into a GD managed image buffer. A secondary benefit is that AGG functionality can be added incrementally as time and resources permit. For example, since an AGG imageObj is really just a gdImagePtr we can use current code that renders to a GD image along side any AGG routines.

Note: See *AGG Rendering Specifics* for more information.

Technical Solution

The goal of this initial implementation is to be able to render the symbol and style definitions the same way as GD does. That is, the AGG renderer should produce output similar to GD but of a higher quality. No attempts are made at this stage to introduce rendering capabilities specific to AGG unless otherwise noted. As a result no additions to the MapServer symbolObj or styleObj are necessary at this point.

For the most part, the AGG renderer can ingest processed shapeObj’s, styleObj’s and symbolObj’s just as GD. That said, since AGG uses sub-pixel computations to render features it does not want feature coordinates rounded to integer values, so a special AGG-only map to image coordinate conversion function `msTransformShapeAGG`. Note, that it may be that GD could also make use of the non-rounded features and could just cast the doubles to ints when passing x’s and y’s to GD (in fact the code already does this), but further testing will be necessary.

C API Changes

In reality adding a new renderer has little or no effect on the MapServer core. The following files are to be modified to add AGG specific processing blocks that are basically straight copies of GD support:

- `mapdraw.c`
- `mapdrawgdal.c`
- `maperror.c`
- `maplegend.c`

- mapoutput.c
- mapraster.c
- mapresample.c
- maputil.c

No new functionality is added to these files, rather just else-if blocks.

The bulk of the AGG functionality can be found in a new source file, mapagg.cpp. For better or worse it mimics the rendering API found in mapgd.c. So, for example msDrawLineSymbolGD has a counter part in msDrawLineSymbolAGG. Various helper functions/methods can also be found in that file.

MapScript

No changes.

Mapfiles

An output block like this will trigger AGG rendering:

```
OUTPUTFORMAT
  NAME 'AGG_PNG24'
  DRIVER AGG/PNG
  IMAGEMODE RGB
END
```

Issues and Caveats

- The AGG driver only supports RGB output at this time. A fundamental difference in how GD and AGG interpret alpha channel values (GD is backwards) means that AGG cannot write to a GD alpha channel and have the output interpreted correctly. This really shouldn't be a deal breaker though since vector rendering in MapServer does not write to the alpha channel except when dealing with layer transparency, otherwise alpha blending occurs. In addition, GD is back under active development and there are plans to define a proper RGBA buffer.
- Text and raster layers are drawn using GD. While text placement could certainly benefit from sub-pixel placement there are not enough resources to complete that support at this time. Hopefully it can be addressed soon. Raster layer rendering doesn't appear to benefit from AGG and will remain a GD function at this time.
- The following symbol, style combinations do not work under AGG: TODO
- The AGG license was changed between versions 2.4 (BSD style) and 2.5 (GPL) (<http://www.antigrain.com/license/index.html>). MapServer should use version 2.4 of the library until all implications of this change are clarified.

Bug ID

None assigned

Voting history

None

14.6.32 MS RFC 33: Removing msLayerWhichItems() from maplayer.c

Date 2007/07/09

Author Steve Lime

Contact steve.lime at dnr.state.mn.us

Last Edited 2007/07/09

Status draft

Version MapServer 5.0

Id \$Id: ms-rfc-33.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

The function `msLayerWhichItems()` is a function in `maplayer.c` that determines exactly which feature attributes need to be retrieved from a given data source. All item-based properties (e.g. `CLASSITEM`, bound properties, `EXPRESSIONS`, etc...) are checked and a master list (an array) is compiled. At the same time index references are made (e.g. `classitemindex`, `labelitemindex`, etc...). The item indexes are used instead of names to access attribute values at runtime.

The problem with this method is that a feature (a `shapeObj`) used for drawing or the first pass of a query is not the same as that used for presentation (the second pass of a query). The second pass of a query uses `msLayerGetShape()` which by default requests all attributes of a feature since we don't know ahead of time which attributes might be output.

By removing `msLayerWhichItems()` and always retrieving all attributes a feature is a feature whether drawing, querying or outputting via a template so caching features now becomes practical.

Technical Solution

As mentioned earlier `msLayerWhichItems()` does a couple of things. Some of these functions would need to be retained somehow and are described below.

1. Layer items array: this array normally would be populated by the `msLayerWhichItems()` call. Instead it should be populated as a layer is opened in `msLayerOpen`. (makes sense?)
2. Index references: linking a attribute name to an index value would have to happen during the course of rendering or querying instead of ahead of time. Basically code that uses attributes would have to first check if the parameter is not `NULL`, and then if the index reference is not set it would have to call a function like that shown below. While this affects a good number of places in the code base the change is relatively minor. The use of attribute binding isolates much of this code so this change is smaller for version 5.0 that it otherwise would have been.

```
int msGetItemIndex(char **itemlist, int numitems, char *item) {
    int i;

    if (!itemlist || numitems <= 0 || !item) return -1;

    for (i=0; i<numitems; i++)
        if (strcasecmp(itemlist[i], item) return i;

    return -1; /* failure */
}
```

1. Logical expression handling: expressions maintain their own individual list of items to process so like 2 above this list would have to be created during processing. The code to do this exists as part of `msLayerWhichItems()` and would be retained under some other name.

Note: this RFC does not address single pass queries, but rather sets the stage for them. Subsequent drawing, query and template output processes would remain unaltered.

General C API Changes

`maplayer.c` - `msLayerWhichItems()` goes away. `msLayerOpen()` now sets the `layer->items` array.

`maputil.c` - Binding functions now must assign index references when executed. Same goes for the functions to assign a `classObj` to a feature.

`mapdraw.c` - `Layeritemindex` and `classitemindex` must now be dynamically assigned values.

Input Driver Changes

It is unclear how each driver made use of the output of `msLayerWhichItems()`. It may be as easy as calling `msLayerGetItems()` instead of using `msLayerGetItemInfo()` in `msLayerOpen()` in which case we'd lose the `msLayerGetItemInfo()` function for each driver too (and the main wrapper function). TODO...

`mapshape.c` - `mapsde.c` - `mapogr.cpp` - `mappostgis.c` - `maporaclespatial.c` - `mapmygis.c` -

MapScript

No changes.

Mapfiles

No changes anticipated.

Backwards Compatibility Issues

None. This is a new feature.

Bug ID

None assigned.

Voting History

None

14.6.33 MS RFC 34: MapServer Release Manager and Release Process

Date 2007/07/13

Author Daniel Morissette

Contact dmorissette at mapgears.com

Last Edited 2007/07/19

Status Adopted (2007/07/19)

Id \$Id: ms-rfc-34.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

This RFC documents the MapServer Release Manager role and the phases of MapServer's Release Process.

The MapServer Release Manager Role

For every release of MapServer, the PSC elects a release manager (this is usually done with a motion and vote on the mapserver-dev list).

The overall role of the release manager is to coordinate the efforts of the developers, testers, documentation and other contributors to lead to a release of the best possible quality within the scheduled timeframe.

The PSC delegates to the release manager the responsibility and authority to make certain final decisions for a release, including:

- Approving or not the release of each beta, release candidate and final release
- Approving or rejecting non-trivial bug fixes or changes after the feature freeze
- Maintaining the release schedule (timeline) and making changes as required

When in doubt or for tough decisions (e.g. pushing the release date by several weeks) the release manager is free to ask the PSC to vote in support of some decisions, but this is not a requirement for the areas of responsibility above.

The release manager's role also includes the following tasks:

- Setup and maintain the Release Plan section of the website for this release
- Coordinate with the developers team
- Coordinate with the QA/testers team
- Coordinate with the docs/website team
- Keep track of progress via Trac (make use of Trac milestones and ensure tickets are properly targeted, push some tickets to a later release if required, etc.)
- Organize regular IRC meetings (including agenda and minutes)
- Tag source code in SVN for each beta, RC and release
- Branch source code in SVN after the final release (trunk becomes the next dev version)
- Update map.h and HISTORY.TXT for each beta/RC/release
- Package source code distribution for each beta/RC/release
- Update appropriate website/download page for each beta/RC/release
- Make announcements on mapserver-users and mapserver-announce for each release
- Produce/coordinate bugfix releases as needed during the 6 months period that follows the final release (i.e. until the next release)

Any of the above tasks can be delegated but they still remain the ultimate responsibility of the release manager.

The MapServer Release Process

(Credit: Inspired by the Plone release process at <http://plone.org/documentation/manual/plone-developer-reference/overview/release-process>)

MapServer uses a time-based release cycle, trying to aim for one release every 6 months.

The normal development process of a MapServer release consists of various phases.

- Development phase

The development phase usually lasts around 4 months. New features are proposed via RFCs voted by the MapServer PSC.

- RFC freeze date

For each release there is a certain date by which all new feature proposals (RFCs) must have been submitted for review. After this date no features will be accepted anymore for this particular release.

- Feature freeze date / Beta releases

By this date all features must have been completed and all code has to be integrated. Only non-invasive changes, user interface work and bug fixes are done now. We usually plan for 3-4 betas and a couple of release candidates over a 6 weeks period before the final release.

- Release Candidate

Ideally, the last beta that was bug free. No changes to the code. Should not require any migration steps apart from the ones required in the betas. If any problems are found and fixed, a new release candidate is issued.

- Final release / Expected release date

Normally the last release candidate that was issued without any show-stopper bugs.

- Bug fix releases

No software is perfect. Once a sufficient large or critical number of bugs have been found for a certain release, the release manager releases a new bug fix release a.k.a. third-dot release (for example 4.10.2).

MapServer Version Numbering

MapServer's version numbering scheme is very similar to Linux's. For example, a MapServer version number of 4.2.5 can be decoded as such:

- 4: Major version number.

We release a major version every two to three years. The major version number usually changes when significant new features are added or when major architectural changes or backwards incompatibilities are introduced.

- 2: Minor version number.

Increments in minor version number almost always relate to additions in functionality and correspond to the 6 months release process described in this RFC.

MapServer uses the same even/odd minor version number scheme as Linux. Even minor version numbers (0..2..4..6) relate to release versions, and odd minor versions (1..3..5..7) correspond to developmental versions. For instance development version 4.1 was released as version 4.2.0, there was never any formal release of 4.1.

- 5: Revision number.

Revisions are bug fixes only. No new functionality is provided in revisions.

Voting history

Vote completed on 2007/07/19.

+1 from DanielM, SteveL, SteveW, FrankW, TamasS, AssefaY, JeffM, PericlesN, UmbertoN and HowardB.

14.6.34 MS RFC 35: Standards Compliance Enforcement

Date 2007/10/16

Author Frank Warmerdam, Daniel Morissette

Contact warmerdam at pobox.com, dmorissette at mapgears.com

Last Edited 2007/12/07

Status Withdrawn (2007/12/07)

Id \$Id: ms-rfc-35.txt 8278 2008-12-23 21:34:31Z hobo \$

Overview

This RFC introduces a mechanism to tell mapserver to enforce OGC standards compliance, or alternatively to be permissive. MapServer should continue to implement the standards as closely as possible, but it has been found that in some cases strict compliance reduces interoperability instead of increasing it.

One example of this is the the requirement for the STYLES parameter in WMS GetMap requests which has been enforced in MapServer 5.0.0. Enforcing this requirement effectively prevented a number of WMS client implementations from connecting to MapServer because they did not include the required STYLES parameter in their GetMap request. This RFC also details short term adjustments to handling of the STYLE= required parameter in MapServer 5.

OWS_COMPLIANCE METADATA

This will be achieved by introducing a new MAP level metadata item named “ows_compliance” with the possible values “pedantic” and “permissive”, with the default being “permissive” if not specified. Variations for specific protocols will also be supported in the usual fashion for OWS metadata (ie. wms_compliance, wfs_compliance, wcs_compliance, sos_compliance).

If this keyword value is “permissive”, then when practical and unambiguous MapServer may attempt to be forgiving of missing or otherwise non-compliant requests and input in the interest of increasing MapServer’s usability in real life interoperability scenarios. This may include things such as allowing protocol parameters to be omitted in requests that are required by the OGC specifications, allowing services to operate even if metadata is missing from the mapfile to fill required fields in capabilities.

If this keyword value is “pedantic”, then to the extent practical MapServer will attempt to generate an error if it’s input (WxS protocol requests, mapfile incomplete, etc) is not in full compliance with the requirements of the OGC specifications.

It is foreseen that users wishing to encourage good standards compliance behavior in clients, and to ensure they don’t accidentally put up web services with incomplete metadata will use the “pedantic” mode. It is anticipated that “pedantic” mode will also be used when MapServer is run through OGC CITE and similar test suites.

It is foreseen that “permissive” mode will be used by those wishing to ensure their service is accessible by the broadest set of possible clients, even if they are not strictly operating to the standards.

Developers should keep in mind that adding more exceptions in the permissive mode should not be taken lightly as this encourages misuse of the specs and bloats the code with unnecessary exceptions.

msOWSLookupMetadata()

In actual code, the compliance keyword might be tested using `msOWSLookupMetadata()` in a manner similar to this:

```

if (stylesfound == 0 && sldfound == 0)
{
    if ( strcasecmp(msOWSLookupMetadata(&(map->metadata), "MO", "compliance"),
                  "pedantic") == 0 ) {
        msSetError(MS_WMSERR, "Missing required parameter STYLES",
                  "msWMSLoadGetMapParams()");
        return msWMSException(map, nVersion, "MissingParameterValue");
    } else {
        msDebug( "WMS request missing STYLES parameter, permissively ignoring." );
    }
}

```

MapServer 5.0.1

For MapServer 5.0.1 the only change anticipated to be made in the code is the above example, requiring the `STYLES` parameter only in pedantic mode.

If other standards enforcement issues are discovered to be unnecessarily interfering with use of MapServer in real life interoperability they may also be changed in 5.0.x to be based off this keyword.

MapServer 5.1

In MapServer 5.1, and the future, if other standards enforcement issues are discovered to be unnecessarily interfering with use of MapServer in real life interoperability scenarios they may also be changed to be based off this keyword.

Once again, developers should keep in mind that adding more exceptions in the permissive mode should not be taken lightly as this encourages misuse of the specs and bloats the code with unnecessary exceptions.

As part of implementation of this RFC the authors will review all “WARNING:” entries emitted in capabilities documents in places like `mapwms.c` to see if they ought to be turned into fatal errors when operating in “pedantic” mode.

Documentation

The `ows_compliance` keyword will be added to the MapServer 5 reference documentation.

Implementation

The described changes will be made in MapServer 5.0 and MapServer 5.1 by Frank Warmerdam and/or Daniel Morissette with the MapServer 5.0 changes in time for a 5.0.1 release. It is hoped that all OWS service developers will apply the permissive/pedantic tests in OWS services in the future as required and as issues come up.

MapScript

No changes are anticipated in `mapscript` as it already has metadata setting methods.

Backwards Compatibility Issues

Implementation of this feature should restore backward compatible default behavior in MapServer 5 with MapServer 4.x. No other compatibility issues are anticipated.

Bug ID

<http://trac.osgeo.org/mapserver/ticket/2427>

Voting History

There was a first vote around 2007/11/08 which was veto'd... and after further discussion the RFC has been withdrawn in favor of simply restoring the behavior of MapServer 4.10 with respect to the `STYLES=` parameter (i.e. make it optional again). See ticket #2427 about this.

14.6.35 MS RFC 36: Simplified template support for query output

Date 2007/10/23

Author Steve Lime

Contact steve.lime at dnr.state.mn.us

Last Edited 2007/10/23

Status Development

Id \$Id: ms-rfc-36.txt 9276 2009-08-28 03:42:16Z sdlime \$

Overview

Problem:

1. Currently a driver like GML isn't available to the CGI as a means of presenting query results
2. The templating scheme (`HEADER/TEMPLATE/FOOTER`) for queries isn't user friendly nor is it ammenable to multiple presentation formats. That is, one layer => one template set

Solution:

1. Use output format objects to define formats that can be used to output query results in addition to drawing images. For example:

```
OUTPUTFORMAT
NAME 'gml3'
DRIVER GML3
MIMETYPE 'text/xml; subtype=gml/3.2.1'
END
```

Might need to extend that object to discriminate between map rendering and query formatters but that can happen in `mapdraw.c` and `mapserv.c` too. That is, drivers are explicitly referenced in those places so if someone tries to draw a map with a GML3 driver it would throw an error.

1. Use the webObj `QUERYFORMAT` property to reference formats: `'QUERYFORMAT gml3'`. Right now that property carries a mime-type but it could be used to reference a format too.

2. Also allow applicable modes (i.e. WFS, WMS, SOS), to utilize DRIVER/TEMPLATE type formats (i.e. advertise in GetCapabilities responses, support through API [e.g. request=GetFeature&outputFormat=text/xml; subtype=gml/3.2.1]), mapped from OUTPUTFORMAT/MIMETYPE. Presently the WCS driver requires the developer to explicitly define supported output formats, other services could do the same and could reference templated output.
3. Define a TEMPLATE driver. Basically this would just invoke the normal query templating scheme. For example:

```
OUTPUTFORMAT
  NAME 'kml'
  DRIVER TEMPLATE
  MIMETYPE 'application/vnd.google-earth.kml+xml'
  TEMPLATE 'myTemplate.kml'
END
```

```
OUTPUTFORMAT
  NAME 'geojson'
  DRIVER TEMPLATE
  MIMETYPE 'application/json; subtype=geojson'
  TEMPLATE 'myTemplate.js'
END
```

1. Note that in the above examples we reference a file, so I'm thinking of supporting a single template system for queries in addition to the current mechanism. To do this I'd propose 4 new template tags: [resultset], [feature], [join] (for one-to-many joins), and [include] (to support code sharing between templates). All but the include tag would be blocks. An example might be:

```
[include src="templates/header.html"]
[resultset name=lakes]
  ... old layer HEADER stuff goes here, if a layer has no results this block disappears...
  [feature]
    ...repeat this block for each feature in the result set...
    [join name=join1]
      ...repeat this block for each joined row...
    [/join]
  [/feature]
  ...old layer FOOTER stuff goes here...
[/resultset]
[resultset name=streams]
  ... old layer HEADER stuff goes here, if a layer has no results this block disappears...
  [feature]
    ...repeat this block for each feature in the result set...
  [/feature]
  ...old layer FOOTER stuff goes here...
[/resultset]
[include src="templates/footer.html"]
```

A specific GML3 example might be:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
[resultset layer=mums]
<MapServerUserMeetings xmlns="http://localhost/ms_ogc_workshop" xmlns:xlink="http://www.w3.org/1999/xlink">
  <gml:description>This is a GML document which provides locations of all MapServer User Meeting that
  <gml:name>MapServer User Meetings</gml:name>
  <gml:boundedBy>
    <gml:Envelope>
      <gml:coordinates>-93.093055556,44.944444444 -75.7,45.4166667</gml:coordinates>
    </gml:Envelope>
```

```

</gml:boundedBy>
[feature]
<gml:featureMember>
  <Meeting>
    <gml:description>[desc]</gml:description>
    <gml:name>[name]</gml:name>
    <gml:location>
      <gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
        <gml:pos>[x] [y]</gml:pos>
      </gml:Point>
    </gml:location>
    <year>[year]</year>
    <venue>[venue]</venue>
    <website>[url]</website>
  </Meeting>
</gml:featureMember>
[/feature]
<dateCreated>2007-08-13T17:17:32Z</dateCreated>
</MapServerUserMeetings>
[resultset]

```

A GeoJSON example might be:

```

[resultset layer=foo] {
"type": "FeatureCollection",
"features": [
[feature trimlast=',']
{
  "type": "Feature",
  "id": "[id]",
  "geometry": {
    "type": "PointLineString",
    "coordinates": [
      {
        "type": "Point",
        "coordinates": [[x], [y]]
      }
    ]
  },
  "properties": {
    "description": "[description]",
    "venue": "[venue]",
    "year": "[year]"
  }
},
[/feature]
]
}
[/resultset]

```

This would allow for relatively complex text files of any sort to be built from multiple layers. All the normal template tags would still be supported but those normally available for query results would only be valid inside a [feature]...[/feature]. These tags would work with existing system too but just wouldn't be as useful as with the 1 template idea.

Note: It is often a problem to have trailing record separator characters after the final record. For example, in the JSON template above the trailing comma in the [feature] block causes problems with Internet Explorer. So I propose supporting a “trimlast” attribute that tells the template processor to remove that string from the end of the output for

the last feature processed. (implemented in 5.2)

Note: A resultset could be applied to multiple layers so the name attribute will take a comma delimited list of layers. The order listed is the order they results will be presented. It's possible that groups could be used as well but at this point that seems like a fairly rare use case. (future)

Note: A resultset will also take a "limit" attribute so that the number of features processed can be controlled. (implemented in 5.2)

Documentation

Documentation detailing the new templated output capabilities will be added to the mapfile reference guide (OUTPUTFORMAT and WEB objects) and to the template reference guide (new [resultset], [feature], [join] and [include] tags).

Implementation

mapoutput.c: No changes necessary (I think), no need to define a default format, nor do I think we need to extend the outputFormatObj structure.

maptemplate.c: Add processor functions for the new tags. Update process line to recognize the [resultset] and [join] tags (the [feature] tag would only be valid within a [resultset] block. Write a new single template processing function similar to msReturnQuery() in that same source file, something like msReturnSingleTemplateQuery().

mapserv.c: Add code at the end of the query processing switch statement to look at the value of web->queryformat. If it references an existing output format by name then use the file the format points to with msReturnSingleTemplateQuery(), otherwise process as currently done.

MapScript

No changes are anticipated in MapScript at this time,

Backwards Compatibility Issues

No other compatibility issues are anticipated. The current templating mechanism would continue to function. In the event the QUERYFORMAT does not reference an outputFormatObj the current system would kick in. In fact, the current system can use several of the new proposed tags, specifically [join] and [include] tags.

Bug ID

#2576

Voting History

None

14.6.36 MS RFC 37: MapServer Spatial Reference Improvements and Additions

Author Howard Butler

Contact hobu.inc at gmail.com

Author Frank Warmerdam

Contact warmerdam at pobox.com

Revision \$Revision: 8919 \$

Date \$Date: 2009-04-16 20:16:33 -0700 (Thu, 16 Apr 2009) \$

Status Draft

Id \$Id: ms-rfc-37.txt 8919 2009-04-17 03:16:33Z hobo \$

Purpose

To provide MapServer with the ability to set its PROJECTION information from a number of sources, including directly from the datasource itself, in an attempt to lessen the burden related to dealing with coordinate system information on users. These improvements will be optionally available and not interfere with previous PROJECTION definition methods.

The History of Spatial References in MapServer

MapServer's spatial reference support is quite anemic by many standards. While most of the data sources MapServer interacts with support describing the spatial reference of contained layers, MapServer has historically dropped the information on the floor or completely ignored it.

MapServer's reprojection machinery keys off the fact that a LAYER's PROJECTION is different than the MAP's. When this is the case, MapServer reprojects the LAYER's data to the MAP's spatial reference during a map draw. OGC services also interact here, and when a spatial reference is specified as "available" using the METADATA mechanism, a client can request maps in a different spatial reference than is specified by default, which starts the reprojection machinery.

Definition

MapServer has historically used two different approaches for defining the spatial reference of its data – EPSG/ESRI codes in the form:

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

And proj4-formated definitions in the form:

```
PROJECTION
  "proj=cea"
  "lon_0=0"
  "lat_ts=45"
  "x_0=0"
  "y_0=0"
  "ellps=WGS84"
  "units=m"
  "no_defs"
END
```

A third, and rather unknown option is available exclusively to WMS – the EPSG AUTO definition, where MapServer attempts to determine the spatial reference from the data itself. This method currently only works for OGR and GDAL data sources, and it is only available when GDAL/OGR is linked into MapServer.

Performance Observations

MapServer's current spatial reference story is focussed on two things – simple description and ensuring that unnecessary data reprojection doesn't happen. MapServer currently uses proj4 directly to do its data reprojection, and this is the impetus for defining coordinate systems in proj4 format. For people wanting the best performance but still requiring data reprojection, defining your spatial references in proj4 format is a must.

Alternatively, the EPSG/ESRI code definition of MapServer's spatial references allows MapServer to offload the lookup of proj4 descriptions to proj4 itself, with a simple file-based lookup table. This mechanism is currently a bottleneck, however, as each lookup requires trolling through a file to match the given identifier and returning the proj4 definition.

Note: This penalty was lessened in early 2009 by the addition of a caching mechanism in proj4 that allows subsequent lookups to be fast.

Usability

The usability of these two mechanism can be a nightmare for users. First, most of the spatial reference descriptions that people work with are of the WKT variety – not proj4. While it is straightforward to set the PROJECTION information for data with a known EPSG value, custom projections or those not generally available in the EPSG database require the user to somehow translate their WKT into proj4 format and paste it into their mapfile.

Additionally, <http://spatialreference.org> exists to help ease this pain, but it is ultimately a stopgap, and not a permanent solution to the problem. It is not practical to be downloading the spatial reference for each and every layer in a mapfile on every map draw from a website. spatialreference.org does provide some conversion utilities to allow a user to paste in WKT and have it return MapServer PROJECTION blocks, but this approach still foists pain and misery on the users.

Specification Features

MapServer will continue to behave as before, assuming the inputted projection is either an EPSG code or proj4 definition. The user will also have the ability to optionally add a TYPE enumeration that will hint the processing of the projection object.

```
PROJECTION
  TYPE TYPEENUM
  "A definition"
END
```

The following optional TYPE enumerations would be supported:

- AUTO
- PROJ4
- EPSG
- FILE
- OGCWKT
- ESRIWKT

See below for some examples:

```
# Use the what the layer defines as the projection definition.
# This may not be available for all data sources or layer types
# (shapefile, SDE, OGR, etc.).
PROJECTION
    AUTO
END

# Use a proj4 definition
PROJECTION
    "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
END

# Use an EPSG code
PROJECTION
    "init=epsg:4326"
END

# Read the definition from a file
PROJECTION
    TYPE FILE
    VALUE "../myfile.prj"
END

# Use an OGC WKT definition (escaping may be required)
PROJECTION
    'GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.01745329251994328,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4326"]]'
END

# Use an ESRI WKT definition (escaping may be required)
PROJECTION
    TYPE ESRIWKT
    'GEOGCS["GCS_WGS_1984",
        DATUM["D_WGS_1984",
            SPHEROID["WGS_1984",6378137,298.257223563]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]]'
END
```

Implementation Details

It is important that MapServer's previous spatial reference definition behavior be preserved. First, drastically changing the PROJECTION definitions would mean a lot of unnecessary mapfile churn. Second, continuing to define spatial references in proj4 format as before will be the most performant.

Implementation of this RFC will encompass four items:

1. Additional spatial reference type enumerations will be added.
2. Addition of a method to the LAYER virtual table so layers can support returning the spatial reference in a variety of formats (OGCWKT, ESRIWKT, proj4, EPSG).
3. Additional methods will be added to operate with the MapScript projectionObj to support setting the projectionObj of a TYPE with a definition.

TYPE Enumerations

The following TYPE enumerations will be added to support SRS definition types:

```
enum MS_SRS_TYPE {MS_SRS_AUTO, MS_SRS_PROJ4, MS_SRS_EPSG, MS_SRS_FILE, MS_SRS_OGCWKT, MS_SRS_ESRIWKT,
```

projectionObj

Two methods will be added to set the projectionObj using various definition types and to return definitions in various flavors.

```
int msSetProjectionByType(projectionObj *p, int type, const char *value)
```

```
char* msGetProjectionByType(projectionObj *p, int type)
```

Virtual Table Method

To support AUTO projection definitions, drivers need to have the ability to return spatial reference information. MapServer's layer plugin architecture provides mechanisms for interacting with MapServer layer providers, but there is currently no regularized method for returning the spatial reference information from providers. The following virtual table member is proposed:

```
int (*LayerGetAutoProjection)(layerObj *layer)
```

Driver-specific implementations

The following drivers will have implementations provided to support TYPE AUTO spatial reference definitions:

- Shapefile
- OGR
- GDAL Raster
- ArcSDE
- PostGIS

MapScript

msSetProjectionByType and msGetProjectionByType will be exposed to MapScript via the projectionObj:

```
setByType(char* definition, int type)
getByType(int type)
```

Files Affected

```
mapserver.h
mapfile.c
mapscript/swiginc/projection.i
maplayer.c
mapproject.h
mapproject.c
mapsde.c
mapogr.cpp
mapraster.c
mappostgis.c
.
.
.
```

Backward Compatibility Issues

All work described in this RFC will provide optional capabilities to MapServer and no backward compatibility issues are expected.

Documentation

This RFC will stand as primary documentation for the feature until such time as the methods and practices described in this document are integrated into the regular MapServer documentation framework.

14.6.37 MS RFC 38: Native Microsoft SQL Server 2008 Driver for MapServer

Author Howard Butler

Contact hobu.inc at gmail.com

Revision \$Revision: 8278 \$

Date \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Implemented

Version 5.2

Id \$Id: ms-rfc-38.txt 8278 2008-12-23 21:34:31Z hobu \$

Purpose

To provide a read-only, native MapServer driver that connects to Microsoft SQL Server 2008 (henceforth called SQL Server) on Windows as a PLUGIN datasource driver.

Background

I.S. Consulting has donated a native driver modeled on the PostGIS driver to support SQL Server 2008's newly added spatial capabilities. This new driver will only be available on the Windows platform, and it will only be available as a PLUGIN datasource driver. No additional enumerations or conditional includes will be added to the MapServer codebase to support this driver.

Usage Details

The driver is a PLUGIN layer, and uses the PLUGIN syntax described in RFC 8 [1] to define relevant layer information:

```
LAYER
  NAME "Roads"
  CONNECTIONTYPE PLUGIN
  PLUGIN "C:\ms4w\plugins\msplugin_mssql2008.dll"

  CONNECTION "server=mysqlserver2008.com;uid=dbusername;pwd=dbpassword;database=Roads Database;Integr

  DATA "the_geom from roads"
  TYPE LINE

  STATUS ON

  PROJECTION
    "init=epsg:4326"
  END

  CLASS
    STYLE
      COLOR 0 0 255
      WIDTH 8
    END
  END

END
```

Files Affected

A single file, mapmssql2008.c will be added to subversion. It will only be compiled on windows using the 'nmake /f makefile.vc plugins' command when options describing the ODBC libraries are switched on.

Backward Compatibility Issues

All work described in this RFC will provide optional capabilities to MapServer and no backward compatibility issues are expected.

Documentation

This RFC will stand as primary documentation for the feature until such time as the methods and practices described in this document are integrated into the regular MapServer documentation framework.

Intellectual Property

This work will become a regular part of MapServer and will be released under MapServer's open source license.

[1] <http://mapserver.gis.umn.edu/development/rfc/ms-rfc-8/>

14.6.38 MS RFC 39: Support of WMS/SLD Named Styles

Date 2008/06/25

Author Yewondwossen Assefa

Contact assefa at dmsolutions.ca

Status Adopted

Version MapServer 5.2

Id \$Id: ms-rfc-39.txt 8278 2008-12-23 21:34:31Z hobo \$

Overview

When WMS and SLD support was added in MapServer few years back, one of features that was not integrated was the ability to specify named styles through WMS GetMap request using the STYLES parameter or through the <NamedStyle> parameter in an SLD document.

Using named styles, the WMS client has the ability to render a specified layer using styles predefined by the WMS server.

Example of this would be:

```
http://.../mapserv.cgi?Request=GetMap&...&LAYERS=Rivers,Roads,Houses&STYLES=CenterLine,CenterLine,
```

```
<StyledLayerDescriptorversion="1.0.0">
  <NamedLayer><Name>Roads</Name>
    <NamedStyle>
      <Name>Casing</Name>
    </NamedStyle>
    <NamedStyle>
      <Name>CenterLine</Name>
    </NamedStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

The main reason this functionality is not yet supported is that It is not currently possible to defined in MapServer several mutually exclusive 'styles' on a layer The intention of this RFC is to introduce a simple mechanism that would allow MapServer to define mutually exclusive 'Styles'. This would then allow for MapServer to advertise and support named styles through the WMS interface

Proposed Changes

The MapServer architecture (layer. class, styles) does not fit well the SLD model where it assumes that you can define mutually exclusive styles on a layer and be able to switch between them.

One possible solution which is reasonably non disruptive would be to introduce the concept of group names at the class object level and have at a layer level a parameter that can be used to specify the classes to use. Something like this is what is proposed:

```
LAYER
...
CLASSGROUP "group1"
...
CLASS
  NAME "name1"
  GROUP "group1"
  ...
END
CLASS
  NAME "name2"
  GROUP "group2"
  ...
END
CLASS
  NAME "name3"
  GROUP "group1"
  ...
END
...
```

- This introduces two new keywords, CLASSGROUP at the layer level and GROUP name in the class object.
- These parameters are optional
- If the CLASSGROUP parameter is set, only classes that have the same group name would be considered at rendering time. If it is not set, all classes (current behavior) would be used.
- Note that CLASSGROUP is acting as the default style if there are classes within the same LAYER with different GROUPs defined. The idea is that STYLES parameter through a wms request (or cgi URL variable) would override the value of the CLASSGROUP. In the example above, only classes “name1” and “name3” would be considered for rendering (if STYLES= or STYLES=default), unless the client overrides this value using STYLES=group2.

Affected/Added functionalities in MapServer

1. MapServer vector rendering (function msShapeGetClass) would use the setting of the classgroup if it is available
2. Raster rendering (function msGetClass) : would use the setting of the classgroup if it is available
3. Possibility to use the URL variables to modify the value of the classgroup (something like this would be valid : ...&map.layer[layername]=classgroup+group2. This would allow through the cgi to switch representation of a layer if needed
4. WMS related functionalities:
 - Advertise STYLES through the GetCapabilities: styles would be based on the different group names of the classes defined in the layer. If all classes do not have the GROUP set, the current behavior (returning the STYLE element with the ‘default’ name) will be kept. If one or more classes have the GROUP set, It is proposed to output something like this for each “group”:

```
<Style>
  <Name>group1</Name>
  <Title>group1</Title>
```

```
<LegendURL width="20" height="10">
  <Format>image/png</Format>
  <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple" xlink:href="1" />
</LegendURL>
</Style>
```

- Support Named Styles in SLD. This would have the effect of setting the layer level CLASSGROUP. We throw an exception if an invalid style is passed.
 - Support style names in the STYLES parameter for the WMS GetMap request. We would still support the empty style names as well as the “default” keyword used for STYLES. We throw an exception if an invalid style is passed.
 - GetLegendGraphic would be extended to support names in the STYLES parameter. The if STYLE parameter is present and the value is not empty, we would throw an exception if the style passes is not valid.
 - GetStyles : the current behavior is to return all the classes as UserStyle elements. the sld 1.0 specification does not seem to be clear on how to deal when multiple styles are available : section 13.1 specifies “...All requested styles that can in fact be described by SLD will be returned as UserStyle elements, and styles that cannot be will returned as NamedStyle elements. ...”. The proposed approach is to keep the current behaviour if the CLASSGROUP is not set and if set, to return UserStyle elements on specified classes only.
1. Legend drawing would need to use the setting of the classgroup if it is available

Other Considerations

- Does this apply to all types of layers such as chart layers ?

Files Affected

```
maplexer.l
mapserver.h
mapfile.h
mapfile.c
mapogcsld.c
mapwms.c
maplegend.c
mapcopy.c
php_mapscript.c
```

MapScript

Need to update set function (at least in the php MapScript) to set the two new parameters at the layer and class level.
Need to check if there is anything to be done for swig MapScript.

Backwards Compatibility

All work described in this RFC will provide optional capabilities to MapServer and no backward compatibility issues are expected.

Documentation

SLD, *WMS Server*, and *Mapfile* documents will be upgraded.

Testing

Addition of a test in msauto to validate the support of names STYLES parameter through a GetMap request

Bug ID

<http://trac.osgeo.org/mapserver/ticket/2431>

Voting History

+1: Assefa, TomK, FrankW +0, DanielM, SteveW

Discussions on mailing list

<http://www.nabble.com/Call-for-comments-RFC-39-td13774241.html> <http://www.nabble.com/call-for-vote-on-RFC-39-td14212600.html>

14.6.39 MS RFC 40: Support Label Text Transformations

Date 2007/12/03

Author Thomas Bonfort

Contact thomas.bonfort at gmail.com

Last Edited 2007/12/03

Status Implemented

Version MapServer 5.4

Id \$Id: ms-rfc-40.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

MapServer 5.0 supports “hard” text wrapping, where the wrap character is unconditionally treated as a newline. This RFC proposes the addition of:

- conditional wrapping, i.e. newlines are only added when the current line length has met a user adjustable criteria
- line alignment, i.e. text aligned per user request, namely right, center and left

These two additions are independent from one another, i.e. can be activated together or not.

Line Wrapping

A new keyword, MAXLENGTH is added to the label object, and interacts with the label WRAP parameter as follows:

- the wrap character specifies what character can be replaced by a newline (no change w.r.t. the current meaning). Probably the most useful wrap character will become ' ' (i.e. the space character)
- the MAXLENGTH value specifies the maximum number of characters that can compose a line before a newline is inserted. A value of 0 specifies the current “hard” wrapping. A positive value triggers wrapping at the WRAP character for lines longer than MAXLENGTH. A negative value triggers wrapping that is allowed to break words, i.e. not necessarily at the WRAP character.

The WRAP / MAXLENGTH combinations are summarized here:

	maxlen =0	maxlen > 0	maxlen < 0
wrap = 'char'	current way	conditional wrap if > maxlen	hard wrap
no wrap	no processing	skip label if > maxlen	hard wrap

Line Centering

A new keyword ALIGN is added to the label object. It supports 3 values: left, center and right, and controls how text lines should be aligned w.r.t. the label bounding box. Precise placement of text can only be done at the renderer level, by exactly defining the starting pixel of the current line. While this approach could be taken, it would considerably burden the renderer code.

I propose to use a more generic though less precise way of doing, by padding the text lines with space characters to approximate indentation. The initial implementation could only rely on the number of characters in each text line:

- loop through text lines to find the line with the most characters l_max
- pad all the other lines with $(l_max - l_cur)/2$ space characters

A more advanced implementation will be to use the exact line lengths as returned by the renderers with the msGetLabelSize function:

- compute the size in pixels of the " " string (two spaces, accounts for kerning): l_2space
- loop through text lines to longest line of length pix_l_max
- pad all the other lines with $(pix_l_max - pix_l_cur)/2 * 1/(2*l_2space)$ space characters

“‘Limitations’”: aligning text to the right will produce ugly results using this method, unless using a monospace font.

Modifications to the source code

- MAXLENGTH and ALIGN will be added to the LABEL object in map.h, in the mapfile parser/writer (mapfile.c) and in MapScript
- msTransformLabelText() in maplabel.c will be updated to support the modifications proposed. No other modifications should be required as the text itself is modified to fit the user’s request (i.e. it is padded with spaces, and/or newlines are added to it)

MapScript Implications

The labelObj will have new maxlength property of type integer and align property (ms_align_*).

Files affected

```
map.h
mapfile.c
maplabel.c
maplexer.l
maplexer.c
```

Backwards compatibility issues

None.

Bug ID

- 2383: <https://trac.osgeo.org/mapserver/ticket/2383>

Voting history

- +1: SteveL, SteveW
- +0: FrankW, TomK

Questions/Comments from the review period

- SteveW: allow preciser computation of line offsets when padding with spaces: done. The precise size of each line is calculated in pixels, and the closest number of space characters required for alignment is computed accordingly
- DanielM: allow multiple characters to be used when wrapping lines: not addressed. Space characters can be *replaced* by a newline, while the hyphen character should generally be kept. Any idea how we can specify that some wrapping characters are to be kept while others are just markers for potential linebreaks?

14.6.40 MS RFC 41: Support of WCS 1.1.x Protocol

Date 2008/04/09

Author Frank Warmerdam

Contact warmerdam@pobox.com

Status Adopted

Version 5.2

Id \$Id: ms-rfc-41.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

It is proposed to extend MapServer to support the WCS 1.1 protocol. MapServer already supports the WCS 1.0 protocol, but WCS 1.1 is significantly different.

Implementation Methodology

WCS 1.1 is closed based on OWS Common metadata (unlike WCS 1.0), and implementation of the WCS service will take advantage of the OWS metadata services in mapowscommon.c. However, mapowscommon.c is based on libxml for xml serialization (unlike the printf() based WCS 1.0 service). For this reason, WCS 1.1 GetCapabilities and DescribeCoverage implementations will be largely separate from WCS 1.0 implementations and will be implemented using libxml.

It should be noted that the format and organization of the WCS 1.1 capabilities and coverage description methods are so different that separate implementations would have been pretty much necessary anyways.

The key entry points in mapwcs.c for WCS services will be updated to “call out” to WCS 1.1 versions of the services in mapwcs11.c.

It is anticipated that the bulk of the WCS 1.0 GetCoverage implementation will be shared with WCS 1.1 with special WCS 1.1 implementations to handle specific issues in the request (RangeSet processing, and multi-part mime return results for instance).

WCS 1.1 Protocol Limitations

- Only one <Field> may be associated with the Range of Coverage when served through MapServer.
- Only two types of Axis will be supported, a “Bands” axis, and a “Time” axis.

Metadata Mapping

The current WCS metadata items are tightly related to the WCS 1.0 protocol, while the WCS 1.1 protocol used a substantially different form and conventions for service, and coverage descriptions as well as for the getcoverage request. The following table indicates which WCS metadata items are mapped to what coverage XML elements in WCS 1.0 and WCS 1.1:

MapServer	WCS 1.1	WCS 1.0
-----	-----	-----
<x>_formats	SupportedFormat	formats
<x>_keywordlist	ows:Keywords	keywords
<x>_label	(unused)	label
<x>_description	ows:Title	description
<x>_abstract (new)	ows:Abstract	(unused)
<x>_metadatalink_href	(unused)	metadataLink
<x>_nativeformat	(unused)	nativeFormat
<x>_rangeset_name	Field.Identifier	RangeSet.name
<x>_rangeset_label	Field.Title	Rangeset.label
<x>_bands_name	Axis.identifier	AxisDescription.name

URNs / Coordinate Systems and Axis Orientation

WCS 1.1 uses URNs like “urn:ogc:def:crs:EPSG::4326” or “urn:ogc:def:crs:OGC::CRS84“. In addition the WCS protocol is required to honour EPSG axis conventions when using coordinate systems within the EPSG authority space. This means, for instance, that any coordinates in the urn:ogc:def:crs:EPSG::4326 coordinate system must be provided in lat,long ordering instead of the conventional long,lat.

In order to implement these requirements, several changes are planned:

- msLoadProjectionString() will be updated to expand URNs in the EPSG and OGC name spaces.

- `msLoadProjectionString()` will add the “+epsgaxis=ne” parameter for URNs for GCS codes in the EPSG name space.
- New `msAxisNormalizePoints()` and `msAxisDenormalizePoints()` will be added for converting between normalized (easting,northing) axis orientation and EPSG preferred (denormalized) axis orientation (sometimes northing,easting). These functions will scan the `p->args[]` list for the `+epsgaxis=ne` to decide.
- `msOWSCommonBoundingBox()` will be modified to use these axis denormalization function to denormalize axis ordering for EPSG GCS URNs.
- the WCS 1.1 `GetCoverage` call will use `msAxisNormalizePoints()` to fix up orientation of request axes when needed.

MapScript

No changes to MapScript are anticipated.

Backwards Compatibility

No alterations to WCS 1.0 support are expected, and it is not expected that the mechanisms for specifying services metadata will be changed though it is possible a few metadata items used only in WCS 1.1 will be added.

Documentation

The *WCS Server* will be extended to discuss WCS 1.1 related issues.

Implementation Resources

Implementation will be done by Frank Warmerdam with financial support from Noetix Research Inc. and the Geoconnections program of the Canadian Government. Preliminary implementation is already operational in svn trunk, and work completion is anticipated by March 1st.

Testing

Tests will be added to `msautotest/wxs` for the WCS 1.1 protocol. Additional assistance with WCS 1.1 validation from other contributors would be welcome.

Bug ID

None Yet.

Voting History

FrankW(+1), AssefaY(+1), TomK(+1), DanielM(+1), PerryN(+1)

14.6.41 MS RFC 42: Support of Cookies Forwarding

Date 2008/03/26

Author Julien-Samuel Lacroix

Contact jlacroix at mapgears.com

Last Edited 2008/04/01

Status Adopted 2008/04/01 - Implementation completed

version MapServer 5.2

Id \$Id: ms-rfc-42.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

This RFC propose to extend MapServer to forward HTTP Cookies when doing OWS requests.

One method of authentication in distributed server environment is the use of HTTP Cookies. HTTP cookies are used by Web servers to differentiate users and to maintain data related to the user during navigation, possibly across multiple visits and for state information. Technically, cookies are arbitrary pieces of data chosen by the Web server and sent to the browser. The browser returns them unchanged to the server. The enhancement to MapServer would allow the application to forward any HTTP cookie (either from a http or https request for get or post) received from a client to any other server or service that MapServer would request information from.

This would require MapServer to receive and forward any cookies to new services during requests for operations (cascades) such as (but not limited to) WMS requests for: getmaps, getfeatureinfo and WFS requests for GetFeature.

Implementation Methodology

A new metadata (http_cookie_data) will be added at the mapfile level to temporarily store HTTP cookie data. Another metadata (ows_http_cookie) will also be added at the layer and mapfile level to control the use of cookies. The later will be set to either "forward", to forward the cookie stored in the http_cookie_data metadata, or to a hardcoded cookie value, to forward this hardcoded value only.

Example of ows_http_cookie in the metadata:

```
LAYER
...
METADATA
  "ows_http_cookie" "forward" # This will forward the cookie sent to MapServer
END
...
```

The user can also pass direct cookie values instead of the cookies being passed to MapServer. To pass multiple cookies, use the following syntax:

```
LAYER
...
METADATA
  "ows_http_cookie" "cookie1=my_value; cookie2=myvalue; " # This will forward the metadata value
END
...
```

By using the metadata configuration, a MapScript application will be able to forward a HTTP Cookie by setting the correct metadata.

Currently only WMS and WFS code will use this since they are the only places where MapServer request an outside webserver.

Implementation Issues

It was pointed out during the RFC review period that passing and storing the cookie data using an “http_cookie_data” metadata is a poor use of MapServer’s metadata mechanism.

Since MapServer currently lacks a mechanism to associate application state information to a mapObj, there is currently no better mechanism in place to store the cookie data received from the client and pass it to the rendering code that calls the remote WMS. Due to lack of a better solution, for the time being we will use the “http_cookie_data” metadata as proposed in this RFC, with the knowledge that this is a poor use of metadata and that we as soon as a better mechanism is in place to store and pass application state in a mapObj then this metadata will be deprecated and replaced by this new mechanism. Developers of MapScript applications setting this “http_cookie_data” metadata should be aware of this and be prepared to change their code in future revisions of MapServer.

Another issue is that no encoding currently is planned to be made with the http_cookie_data metadata. Poorly formatted metadata could break the HTTP header of a request. If it is found that Curl doesn’t encode the cookie value, only character with a value between 32 and 126 (printable ascii characters) will be allowed.

Modifications to the Source Code

The HTTP Cookie data will need to be stored in cgiRequestObj in a new member variable to be able to pass it to the mapfile. If HTTP Cookies are present the cookies will always be stored there. The mapserv.c main function will then be responsible of the HTTP Cookies in the mapfile just after the loadMap() function and before msOWSDispatch() if the ows_http_cookie metadata is set. The WMS and WFS code will set the newly created variable in the httpRequestObj when it prepares the URL for a server request.

The WMS/WFS msPrepareWMSLayerRequest function will check for the ows_http_cookie metadata in the layer and the map objects. If the value is set to forward, the content of the http_cookie_data metadata will be forwarded as HTTP Cookie to the WMS/WFS server.

Curl as an option in the curl_easy_setopt() function when doing a request called CURLOPT_COOKIE to send cookies with a request.

MapScript

By storing the HTTP Cookie data in a mapfile metadata MapScript will be able to use this new functionality. Here’s an example of a PHP/MapScript use of HTTP Cookie:

```
foreach($_COOKIE as $szKey => $szValue)
{
    $szHTTPCookies .= "$szKey=$szValue; ";
}
$oMap->setMetadata("http_cookie_data", $szHTTPCookies);
$oMap->setMetadata("ows_http_cookie", "forward");
$oMap->draw();
```

File Affected

cgiutil.h	(add http_cookies in cgiRequestObj)
cgiutil.c	(read and store the http cookies data in cgiRequestObj)
maphttp.c	(send cookies with the request via curl option)
mapows.h	(add http_cookies to httpRequestObj)
mapserv.c	(store cookies in mapfile metadata)
mapwmslayer.c	(set cookies in httpRequestObj)
mapwfslayer.c	(set cookies in httpRequestObj)

Backwards Compatibility

None.

Bug ID

<http://trac.osgeo.org/mapserver/ticket/2566>

Voting History

Adopted on 2008/04/01 with +1 from FrankW, DanielM, TomK and AssefaY, and +0 from JeffM and PericlesN.

14.6.42 MS RFC 43: Direct tile generation for Google Maps and Virtual Earth API

Date 2008/04/08

Author Paul Ramsey

Contact pramsey at cleverelephant.ca

Last Edited 2008/05/02

Status Adopted on 2008/04/29 - Completed

Version MapServer 5.2

Id \$Id: ms-rfc-43.txt 8278 2008-12-23 21:34:31Z hobu \$

Overview

Providing a simple access API in the mapserv CGI can make using Mapserver with the Google Maps and Virtual Earth user interfaces dramatically easier for new users.

Technical Solution

The GMaps API defines a GTileLayer which can be used as an overlay or base map. The GTileLayer supports a GTileLayerOption, tileUrlTemplate, which allows the TileLayer to be accessed using a simple URL pattern that substitutes Google's x/y/z coordinates into the request:

```
http://host/tile?x={X}&y={Y}&z={Z}.png
```

See <http://code.google.com/apis/maps/documentation/reference.html#GTileLayer>

For MapServer, the simple URL pattern will be:

```
http://host/cgi-bin/mapserv?map=/the.map&layers=foo,bar&mode=tile&tilemode=gmap&tile={X}+{Y}+{Z}
```

The change will add in new handling in the loadForm function for the mode, interface, version, x, y and z parameters. Like the WMS interface, GMaps API will require PROJ to be specified, and the existence of PROJECTION defines for all layers being accessed. Google X/Y/Z coordinates will be converted to “spherical mercator” coordinates and fed into the extent.

The result will make a Google-with-Mapserver map as easy as:

```
var url = "http://host/cgi-bin/mapserv?";
url += "map=/the.map&";
url += "layers=parcels&";
url += "mode=tile&";
url += "tilemode=gmap&";
url += "tile={X}+{Y}+{Z}";
var myLayer = new GTileLayer(null, null, null,
    {
        tileUrlTemplate:url,
        isPng:true,
        opacity:0.5
    }
);
var map = new GMap2(document.getElementById("map"));
map.addOverlay(new GTileLayerOverlay(myLayer));
```

A Virtual Earth with Mapserver map would look like:

```
var url = "http://host/cgi-bin/mapserv?";
url += "map=/the.map&";
url += "layers=parcels&";
url += "mode=tile&";
url += "tilemode=ve&";
url += "tile=%4";
map = new VEMap("map");
map.LoadMap();
var tileSourceSpec = new VETileSourceSpecification("myLayer", url);
tileSourceSpec.Opacity = 0.3;
map.AddTileLayer(tileSourceSpec, true);
```

A request with tilemode of “gmap” implies the following:

- The “tile” parameter will be of the form: x y zoom
- The output CRS will be set to “spherical mercator” (EPSG:900913)
- The service bounds be global in extent. The top “zoom” level (0) will have 1 tile.
- The “zoom” levels will run from 0 upwards
- The “y” axis of the tile coordinates will run from top to bottom
- The “x” axis of the tile coordinates will run from left to right
- The output tiles will be 256x256 in size
- Each zoom level will be related to parent and children by powers of two

A request with tilemode of “ve” implies the following:

- The “tile” parameter will use the VE tile numbering scheme, for example “0312”. See <http://msdn.microsoft.com/en-us/library/bb545006.aspx> for more details.
- The output CRS will be set to “spherical mercator” (EPSG:900913)
- The service bounds be global in extent. The top level will have 4 tiles (0, 1, 2, 3).
- The output tiles will be 256x256 in size
- Each zoom level will be related to parent and children by powers of two

The MapServer tile mode API will *not* explicitly attempt to address issues of meta-tiling or cross-tile labeling. However, the following steps will be taken to try to minimize these issues:

- Future phases will render into a target slightly larger than the tile and then clip off the extra border pixels.

MapScript Implications

None. This affects only the CGI interface and *mapserv* CGI.

Files Affected

```
mapserv.c  
maptile.c <new>  
maptile.h <new>
```

- Documentation will be updated to reflect the new feature
 - Mapserver CGI Reference
 - Mapserver Tile HOWTO <new>
- Test suite will be updated to exercise the new features
 - Frank Warmerdam has volunteered to do this

Backwards Compatibility Issues

None. This functionality is net new and requires no changes to existing behavior.

Bug ID

- <http://trac.osgeo.org/mapserver/ticket/2581>

Voting History

Adopted on 2008/04/29 with +1 from SteveL, DanielM, JeffM, SteveW, TomK, PericlesN and AssefaY.

References

- <http://msdn.microsoft.com/en-us/library/bb545006.aspx>
- <http://code.google.com/apis/maps/documentation/reference.html#GTileLayer>
- <http://www.worldwindcentral.com/wiki/TileService>

14.6.43 MS RFC 44: Restore URL modification of mapfiles to pre-5.0 levels

Date 2008/08/19

Author Steve Lime

Contact Steve.Lime at dnr.state.mn.us

Last Edited 2008/08/19

Status Draft

Version

Id \$Id: ms-rfc-44.txt 8278 2008-12-23 21:34:31Z hobu \$

MS RFC 31: Loading MapServer Objects from Strings introduced a new syntax for modifying *mapfiles* via URL. Object parameters could be specified together in mapfile snippets making it easier to make changes with far fewer characters. At the same time access to a number of parameters, particularly those that mapfile parsing did no value checks on (mostly strings) were removed for security purposes. In hindsight I underestimated the degree to which that functionality was used by developers. This RFC aims to restore that functionality albeit with security in mind.

Proposed Changes

Presently a few widely modified (and risky) parameters (e.g. layer TEMPLATE and DATA) can be changed via URL if a regular expression (e.g. TEMPLATEPATTERN and DATAPATTERN) is set to validate the incoming value. I propose using the same approach for all un-checked mapfile input. Parameters that represent numbers, colors or have a value domain (e.g. ON/OFF/DEFAULT) are subject to the same checks as when a mapfile is read from disk and as a result should be ok. Those that don't would require specific validation values be set before input would be allowed. For example, the LAYER VALIDATION block below defines patterns that would be used to validate DATA or FILTER parameter changes. If the appropriate validation key doesn't exist the value cannot change.

Grouping all validation in a new VALIDATION block will ease use by simplifying key names to match MapServer token names. The block would be valid for *MAP*, *WEB*, *LAYER* and *CLASS* objects and its core type would be a hashTableObj. The MAP level VALIDATION block would be useful for applying a pattern for all LAYERS or CLASSes (since there is only 1 WEB object there is no need to rely on the MAP object). This would save lots of duplication in cases where a mapfile contains similar layers and the same data validation pattern applies to all. The logic would simply be: look for validation pattern in layer, if not found then look for validation pattern at map level, if not found then no modifications are allowed.

```
VALIDATION
data 'my pattern'
filter 'another pattern'
...
END
```

The validation would only be invoked if the token source is a URL. Mapfile file or string-based processing would be unaffected. An example of how this would work can be seen in mapfile.c near line 2683 with the DATA/DATAPATTERN parameters.

Files Affected

- maplexer.l: all parameters (a few will never be modifiable, like VALIDATION) will be changed to be recognized in the URL_VARIABLE lexer state; VALIDATION token needs to be added
- mapfile.h: add VALIDATION token

- `mapfile.c`: all non-value checked parameters will require regex validation before changes will be allowed via URL; recognize validation token; write validation hash with `mapfile`
- `mapserv.c`: update code for runtime substitution and `qstring` validation to check the validation hash as well

A complete list of parameters affected will be attached to this document in the post-implementation notes below.

Mapfile Changes

New `VALIDATION` token will be recognized.

MapScript Changes

None. MapScript already has a general class for `hashTableObj` management.

Backwards Compatibility Issues

The parameters `DATAPATTERN` and `TEMPLATE` pattern will become deprecated though. The objects in question (`LAYER` and `CLASS`) already contain validation blocks that can be used for this.

URL runtime substitution and `qstring` validation are currently supported through metadata, This would become deprecated as well. The runtime variables and the word “`qstring`” can be used as keys in the validation block instead.

Post-Implementation Notes

A `HowTo` will be developed that covers this topic and run-time substitutions.

Bug ID

<http://trac.osgeo.org/mapserver/ticket/2699>

Voting History

+1 Lime, Woodbridge, Morissette, Assefa

14.6.44 MS RFC 45: Symbology, Labeling, and Cartography Improvements

Date 2008/08/26

Authors Stephen Lime, Håvard Tveite, Thomas Bonfort

Contact Steve.Lime at DNR.State.MN.US

Status Draft

Version MapServer 5.4

Id \$Id: ms-rfc-45.txt 8278 2008-12-23 21:34:31Z hobo \$

(partially derived in part from a document prepared by Håvard and discussions with other developers working on cartographic output)

Scale Dependant Rendering

- All symbols can be scaled. When SYMBOLSCALEDENOM is given for a layer, the layer and all of its symbols (height, width, line widths, line patterns, line dashes, ...) are scaled.
- CLASS->STYLE->SIZE gives the vertical dimension of the symbol's bounding box at the symbol scale (given by SYMBOLSCALEDENOM). For all other scales, the symbol is scaled proportionally to the mapscale.
- Symbol scaling can be limited by using CLASS->STYLE->MINSIZE and CLASS->STYLE->MAXSIZE. By using these MINSIZE/MAXSIZE, one can ensure that a symbol will never be rendered smaller/larger than the given sizes.

Precise Symbol Placement

Harmonize the placement of symbols relative to the feature (point or line) being drawn.

Fractional values for sizes

All symbols sizes and symbol geometries should be specified and represented as double - AGG supports fractional line widths so users should be able to set them explicitly. GD could either round or truncate/cast the double to an int.

- STYLE->SIZE/MINSIZE/MAXSIZE - styleObj.size/minsize/maxsize
- STYLE->WIDTH/MINWIDTH/MAXWIDTH - styleObj.width/minwidth/maxwidth
- STYLE->GAP - style.gap (replaces symbol->gap)
- STYLE->PATTERN (new feature) - styleObj.pattern
- STYLE-> CENTER/ORIGIN (new feature) - styleObj.centerx/centery and originx/originy
- STYLE->OFFSET - styleObj.offsetx and styleObj.offsety
- LABEL->SIZE/MINSIZE/MAXSIZE - labelObj.size/minsize/maxsize
- LABEL->OFFSET - labelObj.offsetx and labelObj.offsety

CENTER/ORIGIN [x y]

Defines the centre of the symbol using decimal pixels (x axis increasing to the right, y axis increasing downwards).

- Is used for placing a point symbol on the map or on a "decorated" line
- Defines the center of rotation if an ANGLE is specified for point symbols.
- Default value is the centre of the symbol's bounding box.
- Applies to TYPE ellipse, vector, pixmap, truetype.

ANGLE [double|attribute] [DEFAULT|CLOCKWISE|COMPASS]

- STYLE-> ANGLE [double|attribute] [DEFAULT|CLOCKWISE|COMPASS]
- LABEL-> ANGLE [double|autofollow|attribute] [DEFAULT|CLOCKWISE|COMPASS]

To specify rotation, you need a point of rotation, a reference direction, a direction of rotation and a unit of rotation.

Currently, angles for labels have a reference direction to the right (west), and a counter-clockwise direction of rotation specified in degrees (0..360).

When specifying direction/rotation for symbols, compass direction is often used (reference direction north and clockwise rotation). A mechanism to specify mode of rotation/direction is needed.

The default would be reference direction east and counter-clockwise rotation. If possible, something like:

Modes of operation

- DEFAULT reference direction to the right (east), counter-clockwise direction of rotation.
- CLOCKWISE reference direction to the right (east), clockwise direction of rotation.
- COMPASS reference direction to the north, clockwise direction of rotation (compass directions).

Behavior

- For symbols of Type HATCH, this is the angle of the hatched lines.
- For points, this specifies the rotation of a symbol around its defined CENTER.
- For point symbols on lines, this specifies the rotation of the symbol relative to the direction of the line (0 - the symbol's x-axis is oriented along the line), or a fixed compass direction for the symbols orientation (COMPASS).
- For polygon fills this specifies the rotation of the pattern given by its defining symbol. For its use with hatched lines, see Example#8 in the SYMBOL examples.

Stable rendering for tile generation

how and with what keywords?

polygon fill symbols

give hatches and vector fills a stable geographical coordinate origin (0,0) to better support tiling and continuous rendering of neighbouring polygons

labeling

TODO

Keywords moved from SYMBOL to STYLE

FIXME GAP [double]+ END

Gap between point symbols when rendered on a linear geometry (used eg. for shields)

- Given in decimal pixels (at the symbol scale). Defines distances (centre to centre) between point symbols for line decoration at the layers symbol scale

- Default is 0.0. The first double gives the distance from the start of the line to the first point symbol, the rest of the doubles constitute a pattern of distances between the point symbols for the rest of the line.
- GAP distances are scaled. Applies to TYPEs vector, ellipse, pixmap, truetype
- FIXME : When drawing the symbol along a line segment, a negative GAP will add 180 degrees to the angle.
- A GAP of 0 (the default value) will cause MapServer to use the symbol as a brush to draw the line. (AGG only)
- The defined centre (CENTER) of the symbol is used when placing the symbol on the line. When placing the symbol on the line, the symbol is oriented in such a way that the symbol's x-axis is directed along the line.

DASHPATTERN ([double on] [double off])+ END

- renamed from STYLE
- the distances are scaled

more to come?

add MINSCALEDENOM/MAXCALEDENOM parameters to styleObj

this would allow doing things like removing outlines or the hollow portions of “tube” line work based on scale.

add LABELMETHOD to layerObj

(or perhaps just method to labelObj?)

- this would allow users to choose label placement algorithms that meet their needs. For example, in D2 below improving label placement will come with a reduction in performance. In some situations simply using a shapes bounding box center is perfectly adequate.
- The default would be the improved placement in D2.

add LABEL to layersObj

so that a default can be defined for all classes (sort of like templates). A class label would take precedent.

add OUTLINEWIDTH to styleObj

eg for drawing road lines with a cached outline

- It is currently impossible to have scale dependant road networks with a fixed width outline, this addition will remedy this.
- This would also be a shorthand to be able to specify an outline for linear networks, without having to specify two style blocks.

add TYPE to styleObj for line and polygon types

modifies how the current shape is interpreted. Applies to line and polygon layers (for now: any ideas for applying this to point layers?)

- TYPE BBOX : use the style to render the bounding box of the current shape
- TYPE CENTROID : render a single pointObj at the centroid of the feature
- TYPE VERTEXES : render a pointObj at each vertex of the shape. - how to specify angle (follows line direction, fixed, ...) - need to specify a way to ignore the start and/or end points of the feature
- TYPE START/END : render a pointObj at the start/end of the feature. would be used for example to add arrowheads to lines - default angle is direction of first/last segment

Files Affected

- maplexer.l: C4, C5
- mapgd.c: B1, C2, C3, C4
- mapagg.cpp: B1, C2, C3, C4
- mapfile.c: B1, C1, C2, C3, C4, C5, C6, D1
- mapdraw.c: C1, C5, C6, D1
- mapprimitive.c: C5, D2

Bug IDs

Individual modifications will be tracked with their own tickets which will be listed here.

- B1: <http://trac.osgeo.org/mapserver/ticket/2766>

14.6.45 MS RFC 46: Migrate Website to OSGeo

Date 2008/11/5

Author Howard Butler

Contact hobu.inc at gmail.com

Last Edited \$Date: 2008-12-23 13:34:31 -0800 (Tue, 23 Dec 2008) \$

Status Proposed

Id \$Id: ms-rfc-46.txt 8278 2008-12-23 21:34:31Z hobu \$

Purpose

Developers and users have expressed dissatisfaction with the current MapServer site, the site rather poorly serves the needs of the project in many instances, and its maintenance and upkeep is limited to a single system administrator (Howard Butler). This RFC will aspire to replace the current MapServer website with a hybrid setup that is similar to the infrastructure that the [OpenLayers](#) currently maintains.

Failures of the Current Website

The current MapServer website could be considered the 2.0 version of the MapServer project's web presence. The 1.0 version of the website was a completely static. The current version of the website looked to allow through-the-web editing to lessen the burden for documenters. Over three years into the effort, it is pretty clear that the website has not had the desired effect with respect to documentation, and it is getting in the way of the project doing other business.

Administrative Failures

Our uptime has been ok, but one effect of the current setup is that no one except Howard Butler takes responsibility for our web infrastructure. Part of the reason for this is there are no other Plone admins that have volunteered time in the MapServer community in the three plus years the site has existed, and part of the reason is that Howard bootstrapped the 2.0 version of the site and it was easy to leave it in his hands. Howard doesn't have the time to be able to keep things up at anything over a subsistence level, and administration of MapServer's web presence must be distributed if we are to achieve any progress.

Survey

A [survey](#) that hoped to determine if the community had any feelings about how the site is currently serving the community was rather inconclusive. While generally positive about the site, the self-selected sample size of eighteen dwarfs the nearly three thousand mailing list subscribers, and I am uncertain what was expressed captures the general sentiment.

Goals

Here are some goals that the 3.0 version of the MapServer website should achieve:

- Make it easy for folks to find the docs
- Stay out of developers' way
- Allow documenters to get their job done
- Allow limited user-contributed information in the form of wiki pages
- Have a gallery that works better
- Move off of UMN computing and integrate within OSGeo's infrastructure

Make it easy for folks to find the docs

Some people have complained that it is difficult to find documents on the website unless you know the exact place in the hierarchy. Because our mind-reading webpage finding software isn't quite up to snuff, the new website should make it easy enough for documenters to organize and reorganize information in logical and interlinked ways. It seems the strictly-enforced hierarchy causes more problems than it solves in this regard.

Stay the out of developers' way

The current website is quite slow. Slow to edit, slow to view, and slow to change. There's lots of pointing and clicking involved to do the simplest tasks. So much so, that folks will only update the website when they absolutely have to. Developers have subversion access by definition of being developers. They should be able to edit the website through text files in subversion and have the website be updated automatically.

Allow documenters to get their job done

The website fails documenters in a number of ways, but the most important failure is the inability to tie documents to specific MapServer versions. A new iteration of the MapServer website must allow this to happen. Luckily, we already have tools to version documents (our source code repository), so we should just leverage that to accomplish this goal.

Allow limited user-contributed information in the form of wiki pages

From time to time, users do contribute significant documentation describing how to accomplish a particular task with MapServer. Our new infrastructure must still allow this to happen without too much friction. MapServer's Trac instance already provides this capability (along with single-signon), and we can take advantage of it to accomplish this goal.

Have a gallery that works better

The [OpenLayers Gallery](#) works better than the current MapServer gallery, and it works much easier from a management/administrative standpoint. A benefit of using OpenLayers' gallery software is both projects can enjoy the benefits of improving it, which is not possible with the current MapServer gallery.

Move off of UMN computing and integrate within OSGeo's infrastructure

Just recently (Sept 15th – Sept 16th, 2008), the server that houses the site was having power supply unit issues (they have been resolved), but it is a fact that the site is running on a very old Solaris machine that could be decommissioned at any point without much of a head's up. MapServer no longer brings grant monies to the UMN, and while they have been gracious to continue hosting us, we need to move somewhere where we have more control over our destiny. Reasons like this are exactly why OSGeo exists, there are resources there for us to use, and we should move the website there at the same time.

Implementation

We are going to unabashedly rip off OpenLayers' web infrastructure. This includes the gallery, static website, and Trac integration. OpenLayers' web infrastructure meets a lot of the goals above, it stays out of the way of the developers and does a good job of serving the users' documentation needs. The mechanics of how this transition will take place are described below:

1. Migrate everything in <http://mapserver.gis.umn.edu/development> to Trac and add redirects of /development to a landing page on the Trac wiki.
2. Migrate everything in <http://mapserver.gis.umn.edu/community> to Trac and add redirects of /community to a landing page on the Trac wiki.
3. Migrate everything in <http://mapserver.gis.umn.edu/download> to Trac and add redirects of /download to a landing page on the Trac wiki.
4. Stand up an Apache instance for MapServer on OSGeo infrastructure. Howard will coordinate with OSGeo's System Administration Committee to get this done. Our new URL will be <http://mapserver.osgeo.org>
5. Stand up an instance of the [OpenLayers Gallery](#) at <http://mapserver.osgeo.org/gallery> and port over our existing gallery entries. Any culling of these entries must be done by some volunteer effort.
6. Migrate existing documents (notice of when to be given) in /docs to Subversion <http://svn.osgeo.org/mapserver/trunk/docs/> All subsequent editing on major MapServer documents from that point forward should happen in svn, and the documents on the Plone website will be considered frozen.

7. Stand up a cron process that takes the docs in Subversion and generates a static HTML website from them. This website will be what exists at <http://mapserver.osgeo.org>

14.6.46 MS RFC 47: Move IGNORE_MISSING_DATA to run-time configuration

Date 2008/10/09

Author Paul Ramsey

Contact pramsey at clevelephant.ca

Last Edited 2008/10/15

Status Adopted on 2008-10-15 - Completed

Version MapServer 5.4

Id \$Id: ms-rfc-47.txt 8278 2008-12-23 21:34:31Z hobo \$

Overview

Making the options for ignoring missing data in tile-indexed layers and WMS client layers more flexible, and configurable at run-time via the map file.

Technical Solution

Currently Mapserver has only compile-time (`./configure --ignore-missing-data`) control of behavior when files referenced in tile indexes are missing, via the `IGNORE_MISSING_DATA` define, and that behavior is applied globally.

Under this RFC, the missing data behavior will be defined:

- at run time
- globally for a map file

As well, the option to “fail on a missing layer” will be added to the WMS client code, which currently defaults to “ignore on a missing layer”.

All changes will preserve the current default behavior in cases where new behavior is not requested by the user: missing data in tileindexes will continue to cause failure and missing WMS layers will continue to be ignored, unless Mapserver is compiled with `--ignore-missing-data`.

A map-level configuration option will be added, `CONFIG “ON_MISSING_DATA” “[ACTION]”`, with the following valid actions: “IGNORE”, “LOG”, “FAIL”. The default map-level behavior will be “FAIL”, unless MapServer is compiled with `--ignore-missing-data`, in which case it will be “LOG”. This will preserve the current behavior for all legacy mapfiles and compile set-ups.

In the case of “LOG” behavior, the logging will only occur if the mapfile is set up for logging: `DEBUG` is set and `MS_ERRORFILE` is set. Documentation must note that `DEBUG` and `MS_ERRORFILE` need to be set.

At the same time, this ticket (#2722) can probably be tracked down and resolved.

Mapscript Implications

None.

Files Affected

mapshape.c
mapogr.c
mapraster.c
maprasterquery.c

- Documentation will be updated to reflect the new feature
 - Mapserver Mapfile Reference

Backwards Compatibility Issues

None. Legacy mapfiles and compile setups should retain existing behavior in the presence of this new code.

Bug ID

- <http://trac.osgeo.org/mapserver/ticket/2785>

Voting History

N/A

References

N/A

14.6.47 MS RFC 48: GEOTRANSFORM Geometry operations

Date 2008/11/01

Authors Thomas Bonfort, Stephen Lime,

Contact thomas.bonfort at camptocamp.com, Steve.Lime at DNR.State.MN.US

Status Implemented (2009/02)

Last Edited 2009/02

Version MapServer 5.4

Id \$Id: ms-rfc-48.txt 8806 2009-03-18 11:14:16Z tbonfort \$

Summary

The purpose of this RFC is to add the ability to Mapserver to interpret the geometries based on a user-defined GEOMETRYTRANSFORM keyword, and to render the interpreted geometry with the selected styling

An example usage would be:

```
GEOMETRYTRANSFORM "start ([geom]) "
```

to treat the underlying geometry as a point rendered on the first vertex current feature.

The directive to activate this behavior is the GEOMETRYTRANSFORM keyword, that initially belongs to the STYLE object. Further developments could be to enable the keyword at the LAYER level.

Usages of this parameter include drawing bounding boxes of underlying geometries, or adding arrowheads/tails to lines.

Further developments would allow complex expressions to be passed to the keyword, allowing nested transformations to the underlying geometry, eg:

```
GEOMETRYTRANSFORM "difference([geom],buffer([geom],10))"
```

and binding to attributes, eg:

```
GEOMETRYTRANSFORM "buffer([geom],[distance])"
```

Detailed functionality

The GEOMETRYTRANSFORM keyword initially accepts a shortcut version of these parameters :

- *bbox* : render a shapeObj representing the bounding box of the underlying geometry.
- *start/end* : render a point symbol at the first/last vertex of the underlying geometry. By default, the symbol is oriented to match the orientation of the corresponding geometry segment. This would be used to add arrowheads/tails to linestrings.
- *vertices* : render a point symbol at each vertex of the geometry. The default orientation matches the half angle of the corresponding segments. The start and end vertices are not included in the transformation.

These transforms are not initially implemented, but could be supported in the future:

- *buffer([geom],distance)* : render a shapeObj representing the buffered geometry. “distance” specifies the buffer distance, in layer units.
- *centroid([geom])* : render a pointObj at the location of the centroid of the geometry
- *labelpoint([geom])* : render a pointObj at the location of the at the point where the geometry will/would be labelled.
- *convexhull([geom])*
- *simplify([geom])*

Implementation Details

Apart from the parsing and internal storage of which type should be used, the modifications of this RFC only affect the high-level rendering functions in mapdraw.c .

There are three main modifications:

- AUTO is added to the ANGLE parser for the STYLE block. styleObj has an autoangle member added accordingly.
- inside msDrawVectorLayer, a copy of the shape before being clipped by the current extent must be kept, as the whole shape must be used for all but the VERTICES type. Then, when looping through the different styles that can be applied, we switch off to the type-specific function if a GEOMETRYTRANSFORM keyword is set.

- inside `mapgeomtransform.c`, the type-specific drawing function is added. Depending on the current style type, it either computes a new shape from the original shape (types `bbox`, `convexhull` or `buffer`) and passes it to the `msDrawShadeSymbol`, or calls `msDrawMarkerSymbol` for points of the original shape (types `start`, `end`, `vertices`, `centroid`) after having adjusted the orientation accordingly.

Affected Files

- `mapdraw.c`
- `maplexer.c/l`
- `mapfile.c`
- `mapserver.h`
- `mapgeometrytransform.c` is added, for the parsing of the transform expression, the transformation itself, and the calling of `msDrawMarkerSymbol` / `msDrawShadeSymbol` functions

Limitations

For line layers, the transformations do not fit in with the current cache mechanism that draws the first style of each class in a first pass for all shapes, and the remaining styles in a second pass.

MapScript implications

Getters and setters will have to be added to the different mapscripts.

A possible enhancement could also be to expose the transformation function to MapScript.

Documentation

completed

Backwards Incompatibility Issues

none expected.

Bug ID

<http://trac.osgeo.org/mapserver/ticket/2825>

Voting History

Vote completed on 2008/12/08:

- +1: Lime, Szekeres, Woodbridge, Nacionales, Morissette
- +0: Nicoletti

14.6.48 MS RFC 49: Symbology, Labeling, and Cartography Improvements

Date 2009/01/26

Authors Thomas Bonfort, Stephen Lime

Contact Thomas.Bonfort at camptocamp.com , Steve.Lime at DNR.State.MN.US

Status Passed and Implemented

Version MapServer 5.4

Id \$Id: ms-rfc-49.txt 8455 2009-01-25 17:29:53Z tbonfort \$

Purpose

This RFC regroups a few minor additions or changes to the MapServer rendering or mapfile syntax.

These changes are/were all in RFC45, but were extracted to keep things moving on as some of the stuff in RFC45 lacks funding in the near future.

Fractional values for SIZE and WIDTH

AGG can deal with fractional widths and sizes but styleObj's define those properties as int's. Shouldn't be a big deal since both GD and AGG don't use those values directly, rather they use scaled values with are cast or rounded to an int if necessary.

<http://trac.osgeo.org/mapserver/ticket/2766>

MINSCALEDENOM / MAXCALEDENOM for STYLEs and LABELS

These keywords can reduce mapfile size and maintainance, by avoiding creating multiple layers or classes.

<http://trac.osgeo.org/mapserver/ticket/2865>

OUTLINEWIDTH on line layers

It is currently impossible to draw a outlined line with a scale-dependant width. We propose to add the OUTLINEWIDTH keyword to the STYLE block, that indicates the width in pixels that should be rendered around the main line color.

The implementation of this lies in mapdraw.c. When a line's style block has an outlinewidth, the underlying shape is actually drawn twice: once with it's width adapted to account for the outlinewidth, and once normally. The current caching mechanism for lines is preserved.

example usage:

```
LAYER
  TYPE LINE
  SIZEUNITS meters
  CLASS
    STYLE
      COLOR 255 0 0
      OUTLINECOLOR 0 0 0
      OUTLINEWIDTH 1
      WIDTH 25 #this is in meters (ground units)
      MINWIDTH 1 #minimal width (in pixels) for far zoomed maps
      MAXWIDTH 20 #maximal width (in pixels) for close zoomed maps
```

```
    END
  END
END
```

whatever the map extent or scale, this line layer would always be rendered by a red line with a one pixel black outline.

<http://trac.osgeo.org/mapserver/ticket/2865>

add LABEL to layersObj

Note: This feature has not yet been implemented

so that a default can be defined for all classes (sort of like templates). A class label would take precedent.

<http://trac.osgeo.org/mapserver/ticket/XXX>

Affected Files

- mapfile.c
- mapdraw.c (outlinewidth on lines, scale on styleobj)
- mapserver.h
- maplabel.c (scale on labels)

Documentation

The documentation for the keywords will be added to the main mapfile syntax docs.

Mapscript

Getters and Setters for the keywords will be added for mapscript availability

Backwards Incompatibility

None Expected

Comments from Review period

- the LABELMETHOD proposal is withdrawn
- fractional values also for pattern,gap, offset: a good candidate for 6.0 (when pattern and gap would be moved to the styleObj?)
- there was consensus for adding parametered scaling, usefull for thematic mapping (eg size proportional to a feature attribute)
 - by allowing simple expressions, eg `SIZE (18 + [poptotal] * ((43-18)/(5000-30000)))`
 - or by adding pseudoitems at the layer level:

```

PSEUDOITEM
  'mySize' (mrange([itemname], mininput, maxinput, minoutput,
maxoutput))
  'myText' (commify(round([itemname], 2)) + ' ac')
END

LABELITEM 'myText'
CLASS
  STYLE
    COLOR 255 0 0
    SIZE [mySize]
  END
  LABEL
    ...
  END
END

```

this enhancement will be addressed in a specific rfc.

Voting History

+1 : SteveW, UmbertoN, TamasS, SteveL, DanielM (,AlanB)

14.6.49 MS RFC 50: OpenGL Rendering Support

Date 2008/12/23

Authors Toby Rahilly, Jonathan Bourke

Contact toby.rahilly at gmail.com. bourke.jf at gmail.com

Last Edited 2008/12/23

Version MapServer 5.4

Id

Overview

This RFC proposes the addition of an Opengl Rendering module to MapServer for faster rendering of images.

Presently MapServer supports AGG rendering for high quality anti-aliased images. Although AGG is capable of high quality images, and is significantly faster than GD rendering, it is still not fast enough to suit desirable use cases such as live on-demand rendering.

Opengl is seen as a solution that can provide high performance rendering by making use of hardware specially designed for real-time rendering. Current Opengl prototypes are capable of rendering maps an order of magnitude faster than AGG with similar image quality.

Technical Solution

The Opengl rendering module interface and path will be very similar to the current AGG rendering module. Like AGG, the Opengl module will make use of GD for I/O of the results. However due to the nature of Opengl there are a few key implementation differences that are worth noting.

OpenGL can make use of pre-rendered textures. All symbols can be pre-rendered and stored as an OpenGL texture, when they are used in rendering a map, the rendered texture is simply copied onto the image buffer with transformations. This can hugely increase the performance of complex and heavily repeated symbols, such as train tracks, because not only do they need to be only rendered once, OpenGL also handles the repetition and transformations which it has been heavily optimised for.

Labels also benefit from this method greatly, as an entire alphabet can be pre-rendered and stored as textures. When a curved label is to be drawn, the OpenGL module simply places the rotated textures of the characters.

Pre-rendered textures of symbols can also be shared between different map rendering instances. For this reason it is suggested that the rendering path be modified slightly, such that when a mapfile with OpenGL rendering is first loaded, all the symbols are pre-rendered and stored on the map object. This drastically increases the performance of batch rendering.

Another notable difference between AGG and OpenGL is that OpenGL is incapable of rendering into main memory, and instead renders into an inaccessible buffer in video memory. This means the image must be rendered into video memory in OpenGL, then copied to the GD image buffer before saving the result.

C API Changes

As the OpenGL module follows very closely to the AGG implementation, very little change has to be made to the core of MapServer. Minor changes are made to the following files to add a rendering path for OpenGL:

- mapdraw.c
- mapdrawgdal.c
- maperror.c
- maplegend.c
- mapoutput.c
- mapraster.c
- mapresample.c
- maputil.c

No new functionality is added to these files, rather just else-if blocks.

The bulk of OpenGL functionality will be found in mapogl.cpp.

Mapfiles

An output block like this will trigger OpenGL rendering:

```
OUTPUTFORMAT
  NAME 'OGL_PNG24'
  DRIVER OGL/PNG
  IMAGEMODE RGB
END
```

Issues

- The guts of OpenGL rendering occurs inside of OpenGL drivers. OpenGL drivers vary from hardware and operating system, and the rendering techniques vary with them. What this means is that maps rendered on different computers can have different results, such as color shades, anti-aliasing quality and performance.

- Although Opengl is cross platform, each operating system has a different method of setting up an Opengl context to render into. Currently we have an implementation for pre-vista windows.

Documentation

TBD

Backwards Incompatibility

No issues expected.

Bug ID

Not assigned.

14.6.50 MS RFC 51: XML Mapfile Format

Date 2009/01/16

Authors Alan Boudreault

Contact aboudreault at mapgears.com

Last Edited 2009/09/04

Version MapServer 5.6

Id \$Id: ms-rfc-51.txt 9292 2009-09-04 11:38:23Z dmorissette \$

Overview

Presently, MapServer supports only normal mapfiles that are parsed by Flex. The current mapfiles are parsed very fast by MapServer but can not be parsed by any other software due to the parser complexity. So, a client interface to build mapfiles is difficult to make.

The need we are trying to address is the ability to build MapFile Editors that would be facilitated by the existence of a XML mapfile format (since the current mapfile format makes it impossible to write a forward-compatible parser). XML is seen as a solution that can provide a strict syntax to mapfiles to have a strong validation with XML Schema.

This RFC proposes the addition of XML Mapfile support.

Technical Solution

It was decided that for the time being we should develop a XML schema and a XSLT to convert from XML to text mapfile. If the new XML format takes off then we may consider implementing support for it directly in MapServer in a future release.

Mapfiles

An example of XML mapfile layer definition (prototype):

```
<Layer>
  <name>popplace</name>
  <type>POINT</type>
  <debug>5</debug>
  <status>ON</status>
  <Metadata>
    <item name="DESCRIPTION">Cities</item>
    <item name="TEST">TESTING</item>
    <item name="RESULT_FIELDS">NAME</item>
  </Metadata>
  <data>popplace</data>
  <labelItem>Name</labelItem>
  <classItem>Capital</classItem>

  <Class>
    <name>Cities</name>
    <expression>1</expression>
    <template>ttt_query.html</template>
    <symbol>2</symbol>
    <size>8</size>
    <Label type="TRUETYPE">
      <colorAttribute>[COULEUR]</colorAttribute>
      <font>sans</font>
      <angle>0</angle>
      <size>8</size>
      <outlineColor red="255" green="255" blue="255"/>
    </Label>
    <color red="0" green="255" blue="0"/>
  </Class>

  <Class>
    <name>Cities</name>
    <expression>/2|3</expression>
    <template>ttt_query.html</template>
    <tolerance>5</tolerance>
    <Label type="TRUETYPE">
      <colorAttribute>[COULEUR]</colorAttribute>
      <font>[FONT]</font>
      <angle>[ANGLE]</angle>
      <size>8</size>
      <outlineColor red="255" green="255" blue="255"/>
      <position>AUTO</position>
      <partials>FALSE</partials>
    </Label>
    <Style>
      <symbol>7</symbol>
      <size>6</size>
      <colorAttribute>[COULEUR]</colorAttribute>
    </Style>
  </Class>
</Layer>
```

Future Enhancement

In the future, some enhancement could be good to added:

- Special tags for all WMS options (including styles)

- Reusable xml block with ID

Documentation

TBD

Backwards Incompatibility

No issues expected.

Bug ID

<http://trac.osgeo.org/mapserver/ticket/2872>

14.6.51 MS RFC 52: One-pass query processing

Date 2009/03/08

Authors Steve Lime

Contact sdlime at comcast.net

Last Edited 2009/06/03

Status Draft

Version MapServer 6.0

Id

Overview

This RFC proposes change(s) to the current of query (by point, by box, by shape, etc...) processing in MapServer.

Presently MapServer supports a very flexible query mechanism that utilizes two passes through the data. This works by caching a list of feature IDs (pass one) and then a second pass through the features for presentation: templated output, drawing, or retrieval via MapScript. The obvious problem is the performance hit incurred from the second pass. The real pain is that the `msLayerGetShape()` function, as implemented, provides random access to the data which can be very expensive for certain drivers.

Technical Solution

There are a number of potential solutions:

1. One could cache the returned shapes in memory. While this wouldn't result in a true single-pass, you wouldn't have to go back to the original driver twice. However, it could lead to large memory consumption with even moderately-sized datasets. Multiple clients accessing services at the same time would only compound the problems. Some testing has confirmed this method to be no faster and probably even a bit slower than option 3 below.
2. Another solution would be fold much of the query pre- and post-processing code into the `msLayerWhichShapes()` and `msLayerNextShapes()` functions so that the access paradigm used in drawing layers could be used. Subsequent research has let us to conclude that a true single pass is not possible in some cases. For example, GML requires a result set envelope be written out before writing individual features. There's no way to get that initial envelope without a pass through the features.

3. A final solution would be to change how the `msLayerGetShape()` function behaves. We propose changing the behavior of that function to provide random access to a result set (as defined by `msLayerWhichShapes()`) rather than the entire data set. This removes most of the overhead currently incurred by referencing the results already returned by the data driver in the initial query. For implementation purposes we would retain the current `msLayerGetShape()` implementations to support easy random access and create a new function called `msLayerResultsGetShape()`.

`msLayerGetShape()` would *only* be used via MapScript to preserve true random access functionality. Note that drawing uses `msLayerNextShape()` and does not rely on index values.

Under this last solution data drivers would need to do two things:

- update the population of the `shapeObj` index property (long int) with a value that will allow `msLayerResultsGetShape()` to randomly access a result
- creation of the driver-specific version of `msLayerResultsGetShape()` to retrieve a shape from the results created in `msLayerWhichShapes()`
- the default implementation of `msLayerResultsGetShape()` would simply wrap `msLayerGetShape()` since most drivers would not have to implement the newer function (e.g. OGR or shapefiles)

The query functions would need to:

- not close the layer when finished with a query (we assume that users will want to do something with the results)
- allow `msLayerWhichItems()` to retrieve ALL items so that the retrieved shapes are presentation-ready (draw, template, or ...)

The presentation functions:

- refrain from calling `msLayerOpen()`, `msLayerWhichItems()`, `msLayerWhichShapes()` since that has already been done in the query functions

MapScript `layerObj` wrappers:

- add `resultsGetShape()` method

This solution has been piloted in the single-pass sandbox with very promising results. In some cases queries run orders of magnitude faster. One positive side effect is that primary keys need not be used to retrieve features from the result set. It is the drivers responsibility to provide data to uniquely identify a row in the result set.

Backwards Compatibility Issues

This solution will most likely require changes to MapScript applications that process query results depending on types of data being processed. Those hitting shapefiles for example will still function “as is” since there that driver will still be using `msLayerGetShape()`. Any script hitting a RDBMS data source will have to switch to using `resultsGetShape()` instead of `getFeature()/getShape()`.

One casualty would be the query save/read functions. Since the processing of a set of results would be specific to dataset result handle it won’t be possible to get back to a result once a layer is ultimately closed. A proposed solution to this problem is presented next.

Query File Support

Query files have provided a means of saving the results of a query operation for use in subsequent map production. The series of indexes gathered during a query are written to disk and read later to be used to access the data a feature at a time. With the proposed changes this simply won’t work with RDBMS data sources. It becomes necessary to instead recreate the result set but re-executing the query. Problem is, there’s no easy way to serialize query parameters.

I propose creating a new object, a `queryObj`, to store the various parameters associated with MapServer queries. It might look something like:

```

typedef struct { int type; /* By rect, point, shape, attribute, etc... Types match the query functions. / int qlayer; /
    used by all functions */
    rectObj rect; / used by msQueryByRect() */
    char qitem; / used by msQueryByAttribute() / char *qstring; / used by msQueryByAttribute() */
    ...and so on...
} queryObj;

```

A single queryObj would hang off a mapObj and the mapObj would be the sole parameter passed to the various query methods. MapServer C code, primarily the CGI and OGC interfaces would simply populate the appropriate queryObj members and call the correct query function.

MapScript would be unchanged. The wrappers for the various query functions need only use the user supplied parameters to populate the queryObj and then call the query function. The queryObj would be immutable.

By storing all the information in a single store it should be easily serialized to disk. When read, the reconstituted queryObj could then be used to re-execute the appropriate query. The msSaveQuery() and msLoadQuery() function signatures would remain “as is” although the internals would change.

Files Impacted

- driver files: implement msXXXLayerResultsGetShape() function to shape fetching code if necessary
- maptemplate.c: don't open/close a layer when presenting results
- mapgml.c: don't open/close a layer when presenting results
- mapdraw.c: don't open/close a layer ect... IF if drawing a query map
- maplayer.c: refactor msLayerWhichItems(), add msLayerResultsGetShape() to the layer plugin API
- mapquery.c: re-work msSaveQuery() and msLoadQuery(), change query functions to take a lone mapObj as input, add msInitQuery() and msFreeQuery() functions
- mapserv.c: populate map->query before calling query functions
- mapwxs.c (various): populate map->query before calling query functions
- mapfile.c: leverage msInitQuery() and msFreeQuery() functions
- mapserv.h: define queryObj, add to mapObj
- mapscript (various): update map/layer query methods to populate a queryObj
- others? (mapcopy.c for one)

Although a number of files are impacted the changes in general are relatively simple.

Unknowns

To date only shapefiles, PostGIS and Oracle drivers have been tested with this new scheme, all with positive results. Even shapefiles showed a performance improvement simply due to incurring the overhead of opening files just once. It's not clear how OGR, SDE and raster queries will be impacted. I hope the owners of those drivers can comment further.

MapServer has supported a rather obscure query method called queryByIndex() as basically a wrapper to msLayerGetShape(). This change may render that method obsolete but more checking need be done.

Voting History

None

14.6.52 MS RFC 53: Guidelines for MapScript method return values

Date 2009/03/08

Authors Daniel Morissette

Contact dmorissette at mapgears.com

Last Edited 2009/03/08

Status Adopted on 2009/03/12

Version MapServer 5.4

Id \$Id: ms-rfc-53.txt 8786 2009-03-12 18:17:25Z dmorissette \$

Overview

Ticket #244 pointed out that the return values of MapScript methods were not consistent: some methods return 0/-1 for success/failure and others return MS_SUCCESS/MS_FAILURE.

This mini-RFC defines guidelines for the return values to use with MapScript methods in the future.

Technical Solution

Existing methods will remain untouched to avoid breaking existing MapScript applications.

For new methods added to the MapScript API in the future, the following guidelines should apply:

1. Return values:

- If the method returns only a success/failure status then the MS_SUCCESS/MS_FAILURE values should be used.
- If the method returns a reference to an object, then it should return a valid object on success, or a NULL value on failure
- If the method returns an positive integer, then it should return a positive integer on success and -1 on failure. This is a flexible rule which may not apply in some contexts.

1. Error reporting:

In case of errors/failures, MapScript developers should ensure that the MapScript application code can expect to find an errorObj in the error stack. In most cases the core function that MapScript maps to should take care of calling msSetError(), but in some cases the error may be caught by the wrapper code and then the wrapper code may be responsible for calling msSetError() on failure to ensure that the caller has valid error information.

Backwards Compatability Issues

None. These guidelines apply only to new MapScript methods. The existing MapScript API remains untouched.

Ticket Id

<http://trac.osgeo.org/mapserver/ticket/244>

Voting History

Adopted on 2009/03/12 with +1 from DanielM, UmbertoN, TomK, SteveW, TamasS, HowardB, AssefaY, PericlesN, JeffM and SteveL.

14.6.53 MS RFC 54: Rendering Interface API

Date 2009/03/01

Authors Thomas Bonfort

Contact Thomas.Bonfort at camptocamp.com

Status Planning

Version MapServer 6.0

Id \$Id: ms-rfc-54.txt 8900 2009-04-11 08:56:50Z tbonfort \$

Purpose

This RFC proposes the refactoring of MapServer's rendering backends, following an approach similar to the vtable architecture used by the input drivers.

The motivation for this refactoring is that in the current code, each renderer is responsible both for low-level drawing functionality (eg. strokes, fills, symbology) and for higher level interpretation of the combination of mapfile keywords (eg. scale-dependant size adjustments, line-following orientation for symbols ...). This leads to a lot of code duplication and difficult maintainance.

In a first phase, a cairo (<http://cairographics.org>) renderer will be added following this architecture, and will support traditional raster formats (png, jpeg) as well as vector formats (pdf, svg). Support for OpenGL and KML rendering backends will also be added similarly. During this phase, the new renderers will live alongside the older ones, and the functionality of the existing renderers should not be affected.

In a second phase, the existing renderers will be ported to the new architecture. Ideally all renderers should be ported to the new architecture, although this task might take more time and/or financing. The current PDF and SVG renderers could be phased out and replaced by the ones natively provided by cairo.

Note: While the current GD renderer could be ported to the API architecture, it probably isn't something we want to do as we would be losing some performance induced by the switching from GD's internal pixel representation to the generic one used by this API.

Low Level Rendering API

A new *renderObj* structure is defined and is attached to each outputformat depending on which driver is configured. A *renderObj* contains function pointers to the low-level functionality that each renderer should implement, and configuration options which can tell the higher level calling code what is supported or implemented by the renderer.

Adding a new output driver to mapserver therefore consists in two steps:

- implementing the low-level rendering functions (depending on the renderer's capabilities and configuration options, not all functions need be implemented)

- creating and registering a new outputformat driver, by making the renderObj function pointers point to the low-level functions

```
struct renderer{
    // configuration parameters

    // image creation functions

    // raster handling functions (input and output)

    // image saving functions (only for the renderers that cannot export a
    // raster format)

    // low level rendering functions lines and polygons symbology vector
    // ellipse pixmap truetype text handling label size calculation label
    // rendering

    // support for using an image cache for symbology creation of a tile
    // representing a cached symbol vector ellipse ... placement of a
    // cached tile at a point using a tile as a fill pattern on lines and
    // polygons

    // freeing functions (main imageObj and caches)
}
```

Configuration parameters

Each renderer can inform the higher level calling code of what functionality it supports or implements. This will primarily allow us to support vector and raster renderers. More details on each configuration parameter will be given when appropriate in the rest of this document.

Image Creation Functions

Each renderer must provide a function to create a new image. MapServer's imageObj structure will have an additional void* member that will be used by the renderer to store a structure containing any useful information it may need.

```
imageObj* (*createImage)(int width, int height, outputFormatObj *format,
                        colorObj* bg);
```

Note: Renderer specific creation options would be extracted by each renderer from the outputFormatObj passed to it. (This would be used eg by the PDF renderer for page layout options).

Raster Handling Functions

Raster handling will follow different code paths depending on whether the underlying renderer is already natively a raster renderer (eg. PNG, JPEG) , or if it is a vector one (eg. PDF, SVG).

Raster Buffer Structure A new structure defining a raster object in memory is defined, and is used to replace the current code that uses a gdImage as a common format.


```
typedef struct {
    unsigned int width,height;

    // pointers to the start of each band in the pixel buffer
    unsigned char *a,*r,*g,*b;

    // step between one component and its successor (inside the same band)
    unsigned int pixel_step

    //step between the beginning of each row in the image
    unsigned int row_step;

    // pointer to the actual pixels (should not be used by the high-level
    // code)
    // TODO: as this memeber is actually private, it should probably be
    // a void* so that any internal representation can be used
    unsigned char *pixelbuffer;
} rasterBufferObj;
```

A renderer must provide a function to initialize a rasterBuffer where the pixel alignment and the order of the bands best fits its internal representation of a raster array.

```
void (*initializeRasterBuffer)(rasterBufferObj *buffer, int width, int height);
```

Handling Raster Layers Depending on the renderer's capabilities, there are two possibilities here, that are determined by the *supports_pixel_buffer* parameter:

- if the renderer supports a pixel buffer, then the raster layer code is given a handle to the memory buffer used by the renderer, and directly sets individual pixels where needed. Such a renderer will thus implement a function to export a rasterBufferObj pointing to its internal pixel buffer:

```
void (*getRasterBuffer)(imageObj *img,rasterBufferObj *rb);
```

- if the renderer does not use an internal representation that can be directly filled by the raster layer code, it must provide two functions that will allow the higher level code to merge the created raster:

```
void (*mergeRasterBuffer)(imageObj *dest, rasterBufferObj *overlay,
    double opacity, int dstX, int dstY)
```

Image Saving Functions

Similarly to raster layers, there are two code paths for saving a created image to a stream or to a buffer, depending also on the *supports_pixel_buffer* configuration paramter:

- if the renderer supports a pixel buffer, the exported rasterBufferObj will be treated by some new image writing code (relying on libpng/libjpeg/libgif) This will allow each renderer to not have to implement image saving and will allow us to treat all the formatoptions in a single place.
- if the renderer does not support exproting to a rasterBuffer, it should provide a function for writing itself to a given stream:

```
int (*saveImage)(imageObj *img, FILE *fp, outputFormatObj *format);
```

TODO: exporting to a buffer for mapscript getBytes()

Low Level Functions to be Implemented by a renderer

Polygons

- solid filled: The simplest case, needs only be passed a color
- pattern filled: is passed a tile. The size of the tile (i.e. the amount of transparent space around the actual marker inside the tile allows to tweak the spacing between markers inside the polygon)
- hatch fill: TBD - probably by drawing individual lines at a higher level

```
void (*renderPolygon)(imageObj *img, shapeObj *p, colorObj *color);  
void (*renderPolygonTiled)(imageObj *img, shapeObj *p, void *tile);
```

Lines

- simple stroke: is passed a pattern (dashes) and line style (caps, joins)
- pattern fill: is passed a tile

```
void (*renderLine)(imageObj *img, shapeObj *p, strokeStyleObj *style);  
void (*renderLineTiled)(imageObj *img, shapeObj *p, void *tile);
```

Markers

```
void (*renderVectorSymbol)(imageObj *img, double x, double y,  
    symbolObj *symbol, symbolStyleObj *style);
```

```
void (*renderPixmapSymbol)(imageObj *img, double x, double y,  
    symbolObj *symbol, symbolStyleObj *style);
```

```
void (*renderEllipseSymbol)(imageObj *img, double x, double y,  
    symbolObj *symbol, symbolStyleObj *style);
```

```
void (*renderTrueTypeSymbol)(imageObj *img, double x, double y,  
    symbolObj *symbol, symbolStyleObj *style);
```

```
void (*renderTile)(imageObj *img, void *tile, double x, double y, double angle);
```

Labels and Text

- label size calculation: If passed an “advances pointer”, also calculates individual advances for each glyphs (for anlage folow text)

```
int (*getTrueTypeTextBBox)(imageObj *img, char *font, double size, char *string,  
    rectObj *rect, double **advances);
```

- glyph rendering: no real change

```
void (*renderGlyphs)(imageObj *img, double x, double y, labelStyleObj
    *style, char *text);

void (*renderGlyphsLine)(imageObj *img, labelPathObj *labelpath,
    labelStyleObj *style, char *text);
```

Tiles and Caches For some symbols and renderer combinations, it might be beneficial to use a cache so as not to have to render a complicated symbol over and over again. This behavior would probably have to be deactivated when using attribute binding (on size, angle, color...) as there would be too many cache misses and no real gain.

TBD: use a renderer specific format for a tile (passed back as a void*), or would a simple imageObj created by normally by the renderer suffice? Second solution is much simpler as we'd use the same functions for rendering a symbol than for rendering a tile. First solution is more flexible, as the renderer can cache anything.

Miscellaneous functions

```
void (*transformShape)(shapeObj *shape, rectObj extend, double cellsize);
```

These functions might be needed by some renderers. TBD

```
void (*startNewLayer)(imageObj *img, double opacity);
void (*closeNewLayer)(imageObj *img, double opacity);
```

Cleanup functions

```
void (*freeImage)(imageObj *image);
void (*freeSymbol)(symbolObj *symbol);
void (*freeShape)(shapeObj *shape);
```

Note: The freeSymbol and freeShape functions are added here as we will be adding a void* pointer to the symbolObj and shapeObj objects where a renderer can store a private cache containing an implementation specific representation of the object.

High Level Usage of the rendering API

Image Creation

nothing special. just pass on to the renderer.

Image Saving

Depending on whether the renderer can export a raster buffer or not:

- supported: get a raster buffer, eventually apply formatoptions (quantization) and pass the buffer to the MapServer's saving functions
- unsupported: call the renderer's saving function

Raster Layers

MapServer's Raster layer handling will be modified so that it can write to a rasterBufferObj and not only to a gdImage.

Note: TODO (FrankW) add some detail here about the changes induced here

```
if(renderer->supports_pixel_buffer) {
    rasterBufferObj buffer;
    renderer->getRasterBuffer(img, &buffer);
    msDrawRasterLayer(map, layer, &buffer);
}
else {
    rasterBufferObj buffer;
    renderer->initializeRasterBuffer(&buffer, width, height);
    msDrawRasterLayer(map, layer, &buffer);
    renderer->mergeRasterBuffer(img, &buffer, 0, 0);
    renderer->freeRasterBuffer(&buffer);
}
```

Symbol Cache and Marker Rendering

A symbol cache similar to the actual mapgd.c imagecache will be implemented in mapimagecache.c .

Note: As pointed out earlier, it is yet to be decided if the cache will store plain imageObjbs or if we allow renderers to store a specific structure. This could also be a configuration option, with each renderer specifying if it can treat it's own created imageObj as a cache, or if it needs a void* cache. This would keep the code simple for most renderers, while allowing more flexibility for others (the OpenGL renderer will probably be needing this)

The use of an image cache would be configurable by each renderer via the supports_imagecache configuration option, as its use isn't necessarily useful depending on the format.

```
if(renderer->supports_imagecache) {
    // this function looks in the global image cache if it finds a tile
    // corresponding to the current symbol and styling and returns it. If
    // it wasn't found, it calls the renderer functions to create a new
    // cache tile and adds it to the global cache.
    imageObj *tile = getTile(img, symbol, symbolstyle);
    renderer->renderTile(img, tile, x, y);
}
else {
    switch(symbol->type) {
        case VECTOR:
            renderer->renderVectorSymbol(img, params...);
            break;
        case ELLIPSE:
            /* ... */
    }
}
```

Line Rendering

- simple stroke: call the renderer directly
- marker symbols:

- compute positions and orientations along the geometry
- render a marker symbol with the renderer’s marker functions for each of these positions. It probably isn’t a good idea to use a tile cache if the symbols have to follow line orientation, as rotating a cached tile will produce poor results.
- pattern:
 - create the pattern’s tile (with the renderer’s marker functions)
 - pass the tile to the renderer

Polygon Rendering

- simple fill
- pattern fill:
 - create the pattern tile
 - eventually cache the tile for future use on another geometry
 - pass the tile to the renderer
- hatch fill: - compute the lines corresponding to the hatch - use the renderer’s simple stroking function

Text Rendering

Nothing special to add here, except that the basic preliminary tests (if the font is defined, or if the string to render is NULL) and the lookup of the system path of the truetype font file will be done once by the high level code before calling the renderer drawing function.

Note: Raster fonts will not be supported (at least in a first phase) by this rendering API. Only truetype fonts are implemented.

Image I/O

Currently all image I/O is done through the GD library. To raise this dependency, the RFC proposes to have mapserver implement image loading and saving by directly interfacing the libpng / libjpeg / giflib libraries. The corresponding code will be added to a new file “mapimageio.c”, and will produce or read rasterBufferObj’s.

These functions will implement the formatoptions that are currently supported. Adding new formatoptions would be done by adding the code to these i/o functions, and would thus add the support for all the renderers that use this rendering API architecture:

- jpeg:
 - quality: 0 - 100
- png:
 - interlacing: will be set to OFF by default, as the creation is costlier, produces heavier images, and is incompatible with TileCache’s metatiling functionality.
 - compression: could be added to select the quality of zlib compression
 - quantization: produce palette-based pngs from rgb or rgba imagemodes.

- palette: produce palette-based pngs given a precomputed palette. Further enhancements could include automatically producing a palette by looping through the input mapfile to extract the colors and ensure they end up in the final image (this would only be supported for RGB, not RGBA)

- gif: TBD

Note: The initial implementation of this RFC will not include support for writing GIF images.

Note: The initial implementation of this RFC will not include image loading functions by direct usage of the libjpeg/libpng/giflib libraries. It will instead fallback to GD loading and convert the created gdImage to a rasterBuffer. This can be added in a second phase as it is lower priority, and does not induce a major architectural change.

Miscellaneous

Drawing some layers as rasters in vector renderers

The current pdf (and SVG?) renderers have the option of switching over to GD for some layers, that are then incorporated as a raster image in the final map rather than adding each feature in the usual way.

This functionality could probably be kept with this API with the mergeRasterBuffer function.

Note: A renderer using this functionality will have to take care as the rasterBuffer that it will be passed will be in the native format of the delegated renderer, and may not be in the exact same format it would have created with a call to initializeRasterBuffer.

Legend

The legend rendering code will have to be refactored, as it currently produces a gdImage. Legend rendering will be handled like a normal imageObj.

Note: TODO: “embed” mode is problematic.

If postlabelcache is on, then the created imageObj can easily be added to the main image using the renderTile function (for the renderers that support it. the others will probably have no embedded legend support)

If postlabelcache is off we have a problem, as the created legend is passed as a pixmap symbol to the main map rendering. This is incompatible with the vector renderers, as the imageObj they have created isn't an array of pixels.

Scalebar

TBD

Reference Map

TBD. Should we even still be supporting this ?

Affected Files

- maprendering.c , mapimageio.c, mapcairo.c (added)
- mapfile.c (defaults, and keywords moved from symbolObj to styleObj)
- mapoutput.c (renderObj creation)

- mapdraw.c (msDrawXXX functions moved to maprendering.c)
- mapserver.h
- lots more

Documentation

No end documentation needed aside from the migration guide.

Mapscript

TBD

Backwards Incompatibility

Symbology should be better behaved between renderers, but does imply some backwards incompatible changes in how some symbols can be rendered (although this can probably be seen as fixing bugs rather than backwards incompatibilities)

- reverse orientation of rotation for vector symbols. currently vector symbols are rotated in the opposite direction of the other symbols.
- for line geometries, the given pattern (dashes...) will be scaled by the actual width of the line. A dotted line will therefore be defined as PATTERN 1 1 END whatever the width of the line.
- More to come ?

Comments from Review period

Voting History

14.6.54 MS RFC 55: Improve control of output resolution

Date 2009/03/09

Authors Daniel Morissette

Contact dmorissette at mapgears.com

Last Edited 2009/10/06

Status Adopted (2009/03/16) and Implemented.

Version MapServer 5.6

Id \$Id: ms-rfc-55.txt 9391 2009-10-06 19:40:53Z dmorissette \$

Overview

MapServer is often used to produce a printable map corresponding to the map currently being displayed in the Web interface.

Printed maps will usually require a resolution higher than the default 72 pixels per inch which is commonly used for the screen (e.g. 150 or 300 dpi). In general printing is done by setting the map.resolution and requesting a larger size image for the same map extents. Setting map.resolution to a higher value ensures that the layers minscaledenom/maxscaledenom are properly evaluated for the larger image, but one thing is missing: symbology (line widths,

symbol and font sizes, etc) is not adjusted to reflect the resolution change and the result is a map in which symbology looks smaller than what it looked on screen.

This RFC proposes a mechanism to fix this by automatically scaling the symbology when resolution changes so that the map maintains the same look at each resolution.

Technical Solution

A new optional DEFRESOLUTION keyword will be added in the mapObj with a default value of 72 if not set. DEFRESOLUTION is the reference or default resolution for which all symbology sizes are defined in the mapfile.

At rendering time, if RESOLUTION is set to something different from DEFRESOLUTION, then the symbology in the layer definitions will be adjusted by a factor corresponding to RESOLUTION/DEFRESOLUTION. More specifically, this is done by multiplying the layer->scalefactor value by the resolution/defresolution ratio in msPrepareImage().

The following rendering parameters will be affected:

- size (symbols and labels)
- width (lines)

The following rendering parameters which were not impacted by scaling in the past will also be fixed to follow the scaling from now on:

- offsets (line, point, polygon)
- pattern (line)
- gap (line)
- outlinewidth (labels)
- shadowsize, background-shadow-size (labels)
- buffer around labels (collision)
- minfeaturesize, mindistance (labels)

The case of outlinewidth for line styles is a bit different: it ignores the scalefactor so that we get a fixed outline width no matter what we use for size units, but we want it to follow the resolution/defresolution factor. For this case the line drawing code will need to have a reference to the mapObj passed to msDrawLineSymbol() so that it can access the resolution and defresolution parameters and do the right thing.

In the case of pixmap symbols, they will be scaled only if a size is set. If no size is set then no scaling happens.

Usage example

Example 1:

Let's say we've got a mapfile that we use to produce a 400x400 pixels image for use on the Web, with both defresolution and resolution set to 72.

```
MAP
...
RESOLUTION 72
DEFRESOLUTION 72
SIZE 400 400
...
END
```


In order to produce a 300dpi image for printing, the application code should request an image of 1667 x 1667 pixels ($400 \times 300 / 72 = 1667$) and set `resolution=300`. MapServer will automatically scale all symbology sizes by a factor of $300 / 72 = 4.1667$.

The new values of size and resolution for printing will likely be passed as URL parameters or set via MapScript in most applications, but for the purpose of this example, expressed in mapfile syntax then we'd use the following mapObj settings for printing:

```
MAP
  ...
  RESOLUTION 300
  DEFRESOLUTION 72
  SIZE 1667 1667
  ...
END
```

Example 2:

An application is using a mapfile developed for `map.resolution=96` with MapServer 5.x. In order to take advantage of the new feature, one should set `DEFRESOLUTION=96` in the mapfile, and at rendering time, setting `RESOLUTION=300` will result in all symbology being scaled by a factor of $300 / 96 = 3.125$.

Backwards Compatibility Issues

Existing apps or mapfiles that set `RESOLUTION` to a value other than 72 **without** setting `DEFRESOLUTION` will end up with their symbology scaled. This change in behavior is more likely to be considered as a feature than as a bug in most cases, but in case this is a problem for an existing app then the fix will be to simply add a `DEFRESOLUTION` to the mapfile which matches the resolution for which all mapfile classes and styles are defined.

The rendering parameters listed above that did not follow scaling in the past will be fixed to follow the scaling.

Documentation notes

In addition to documenting the new feature, a note should be added somewhere in the mapserver docs that as of this change, the size values will reflect valid sizes (according to the `SIZEUNITS` settings, `MS_PIXELS` by default) only when the `RESOLUTION` is set to the default 72 or equal to the `DEFRESOLUTION` setting.

The implementation of this feature in v5.6 has a limitation: the `STYLE`'s `MINSIZE`, `MAXSIZE`, `MINWIDTH` and `MAXWIDTH` parameters are not automatically adjusted to reflect the resolution change. This is because we cannot easily access the defresolution/resolution ratio at the level where those parameters are handled, and with the new rendering API in 6.0 this will no longer be an issue.

Files Impacted

- `mapserver.h`: new defresolution member in `mapObj`
- `mapfile.c`, `maplexer.l`, etc: new `DEFRESOLUTION` mapfile keyword
- `mapdraw.c`: adjust scalefactor as needed in `msPrepareImage()`
- `mapscript/php3/php_mapscript.c`: expose new defresolution member

Ticket Id

- <http://trac.osgeo.org/mapserver/ticket/2948>
- <http://trac.osgeo.org/mapserver/ticket/3153>

Voting History

Adopted on 2009/03/16 with +1 from DanielM, HowardB, SteveW, TamasS, PericlesN and AssefaY.

14.6.55 MS RFC 56: Tighten control of access to mapfiles and templates

Date 2009/03/24

Authors Daniel Morissette

Contact dmorissette at mapgears.com

Authors Steve Lime

Contact steve.lime at dnr.state.mn.us

Last Edited 2009/03/26

Status Adopted and implemented (2009/03/26)

Version MapServer 5.4.0, 5.2.2, and 4.10.4.

Id \$Id: ms-rfc-56.txt 8873 2009-04-03 12:23:51Z dmorissette \$

Overview

MapServer versions 5.2.1 and older could potentially be used to access arbitrary files via the creation of mapfiles or templates in untrusted directories.

This RFC proposes a mechanism to tighten access control on mapfiles and templates and limit the risk of leaking arbitrary file contents.

The new access control mechanisms will be implemented and released in MapServer 5.4.0, 5.2.2 and 4.10.4.

Technical Solution

The following mechanisms will be put in place:

- Enforce the requirement for the MAP keyword at the beginning of mapfiles and for the SYMBOLSET keyword at the beginning of SYMBOLSETs.
- Require a Magic String at the beginning of all MapServer templates
- Use of environment variables to control and restrict access to mapfiles by the mapserv CGI:
 - MS_MAP_PATTERN
 - MS_MAP_NO_PATH

Each of the points above are described in more details in the following sections.

Enforce the requirement for the MAP and SYMBOLSET keywords

The MAP and SYMBOLSET keywords used to be optional at the beginning of mapfiles and symbolsets respectively.

With this change, the MAP keyword will be required on the first line of mapfiles and the SYMBOLSET keyword required on the first line of symbolset files.

If the keyword is missing then the parser will reject the file.

Require a Magic String at the beginning of all MapServer templates

With this change, the first line of a template must contain the “MapServer Template” magic string which can be surrounded by comment delimiters in the format of the template to facilitate template editing (see examples below). The first line of the template file will automatically be stripped from the template and will not be included in the MapServer output.

If the magic string is not found then the template will be rejected by MapServer.

HTML template example:

```
<!-- MapServer Template -->
<html>
  <head>...</head>
  <body>
    ...
  </body>
</html>
```

XML template example:

```
<!-- MapServer Template -->
<?xml version="1.0" encoding="UTF-8" ?>
<rootElement>
  ...
</rootElement>
```

GeoJSON template example:

```
// MapServer Template
[resultset layer=foo] {
"type": "FeatureCollection",
"features": [
 [feature trim=',']
 {
  "type": "Feature",
  "id": "[id]",
  "geometry": {
   "type": "PointLineString",
   "coordinates": [
    {
     "type": "Point",
     "coordinates": [[x], [y]]
    }
   ]
  },
  "properties": {
   "description": "[description]",
   "venue": "[venue]",
```

```
    "year": "[year]"
  }
},
[/feature]
]
}
[/resultset]
```

MS_MAP_PATTERN Environment Variable

The optional `MS_MAP_PATTERN` environment variable, set via `mod_env` or other web server equivalents, can be used to specify a Regular Expression that must be matched by all mapfile paths passed to the `mapserv` CGI.

If `MS_MAP_PATTERN` is not set then any `.map` file can be loaded.

Example, use Apache's `SetEnv` directive to restrict mapfiles to the `/opt/mapserver/` directory and subdirectories:

```
SetEnv MS_MAP_PATTERN "^/opt/mapserver/"
```

MS_MAP_NO_PATH Environment Variable

The optional `MS_MAP_NO_PATH` environment variable can be set to any value via `mod_env` or other web server equivalents to forbid the use of explicit paths in the `map=...` URL parameter. Setting `MS_MAP_NO_PATH` to **any value** forces the use of the `map=<env_variable_name>` mechanism in `mapserv` CGI URLs.

If this variable is not set then nothing changes and the `mapserv` CGI still accepts explicit file paths via the `map=...` URL parameter.

Example, set `MS_MAP_NOPATH` and some mapfile paths in Apache's `httpd.conf`:

```
SetEnv MS_MAP_NO_PATH "foo"
SetEnv MY_MAPFILE "/opt/mapserver/map1/mymapfile.map"
```

... and then calls the `mapserv` CGI must use environment variables for the `map=...` parameter:

```
http://localhost/cgi-bin/mapserv?map=MY_MAPFILE&mode=...
```

Backwards Compatibility Issues

The `MAP` and `SYMBOLSET` keywords must be added to any mapfile and symbolset that did not contain them already.

All MapServer templates must be updated to contain the "MapServer Template" magic string on the first line.

The new environment variables are optional and will have no impact on existing applications that don't use them.

Files Impacted

- `mapserver.h`
- `maptemplate.c`
- `mapserv.c`

Ticket Id

Related security tickets:

- <http://trac.osgeo.org/mapserver/ticket/2939>
- <http://trac.osgeo.org/mapserver/ticket/2941>
- <http://trac.osgeo.org/mapserver/ticket/2942>
- <http://trac.osgeo.org/mapserver/ticket/2943>
- <http://trac.osgeo.org/mapserver/ticket/2944>

Voting History

Adopted on 2009-03-26 with +1 from DanielM, TomK, PericlesN, JeffM, SteveW, AssefaY, SteveL and TamasS.

14.6.56 MS RFC 57: Labeling enhancements: ability to repeat labels along a line/multiline

Date 2009/06/26

Authors Alan Boudreault

Contact aboudreault at mapgears.com

Last Edited 2009/09/04

Status Adopted (2009/07/21) and Implemented.

Version MapServer 5.6

Id \$Id: \$

Overview

Currently, MapServer draws labels on the longest segment of a linear shape. Even if the shape is a long line or a multiline shape, only one label is drawn. In some cases, the map quality could be highly improved with more labels.

This RFC proposes a mechanism to fix this by adding the ability to add more labels along long lines or multiline shapes. There are two major enhancements that are proposed.

See the Images section to visualize the current and new behaviors.

Enhancement 1: Label all the lines in MultiLine shape

At the moment, if you've got a MultiLine shape (i.e. a shapeObj with numlines > 1) then only the longest of the Lines is labelled. This is fine in most cases, but in some cases, we may want each individual Line in the MultiLine to get a label. That's the first proposed enhancement: make it possible to label all lines in a MultiLine shape.

Enhancement 2: Ability to repeat labels along a line

At the moment, the label is placed at the center of a line in the case of ANGLE FOLLOW, and at the center of the longest segment of a line in the case of ANGLE AUTO|constant. In the case of very long lines (roads), and especially when using metatiles to render a tile cache, we may want to repeat the label at some interval along the line. That's the second enhancement: make it possible to repeat the label at a given interval along a line.

Technical Solution

The way to control this in the mapfile is to add a LABEL.REPEATDISTANCE parameter. By default this would be turned off and we would keep the current behavior. If REPEATDISTANCE is set to any value > 0 then the labels would be repeated on every line of a multiline shape (enhancement 1 above), and would be repeated multiple times along a given line at an interval of REPEATDISTANCE pixels (item 2 above). In all cases the MINDISTANCE value would still be handled by the label cache so that multiple labels ending up too close to each other for various reasons would be eliminated by the label cache.

To achieve better visual effect, here is an outline of the algorithm that applies when REPEATDISTANCE is set:

1. Calculate the number of labels candidates (N) that can fit in the length of the line.
2. Ensure that N is an odd number. If it's even then subtract one to get an odd number. We want an odd number of label candidates along a given line so that there is always one candidate that falls at the center of the line and which will remain in case of collisions. With an even number of candidates, when collisions happen along a line, the remaining labels are usually shifted one way or the other along the line instead of being centered which does not look as good.
3. Calculate the offset between labels and the position of candidates along the line.
4. Insert labels candidates into the label cache in an order that will ensure that the label cache will eliminate the right labels in case of collisions (keep in mind that collisions are still possible since MINDISTANCE remains in effect). We want to give higher priority to the label candidate at the center of the line and lower priority to other candidates as they get further away from the center. Since the label cache order is last in first out (LIFO), we start by inserting the labels candidates at the extremities of the line into the cache end end with the one at the center, giving it the highest priority.

Usage example

This example will repeat the labels every 80 pixels on each line of a multiline shape.

```
MAP
...
LABEL
  REPEATDISTANCE 80
END
...
END
```

Backwards Compatibility Issues

There is no backwards compatibility issues. By default the LABEL.REPEATDISTANCE parameter is set to 0, so not applied.

Files Impacted

- mappimitive.c: Modify the labels placement algorithm
- mapfile.c, maplexer.l: new LABEL.REPEATDISTANCE mapfile keyword
- mapdraw.c: Modify the way how msPolylineLabel* calls are handled

Ticket Id

- <http://trac.osgeo.org/mapserver/ticket/3030>

Images

Since a picture is worth a thousand words:

- Repeat distance, visual explanation: http://trac.osgeo.org/mapserver/attachment/ticket/3030/repeat_label.gif
- Label placement algorithm: http://trac.osgeo.org/mapserver/attachment/ticket/3030/label_algorithm.gif
- Labels comparison: <http://trac.osgeo.org/mapserver/attachment/ticket/3030/label-comparison.jpg>
- Shields comparison: <http://trac.osgeo.org/mapserver/attachment/ticket/3030/shield-comparison.jpg>

Voting History

Adopted on 2009/07/21 with +1 from SteveW, DanielM, SteveL, AssefaY, ThomasB and PericlesN.

14.7 Mapfile Editing

14.7.1 VIM Syntax

Author Andreas Hirner

Contact andreas.hirner at dlr.de

Author Schuyler Erle

Contact schuyler at tridity.org

Revision \$Revision: 8369 \$

Date \$Date: 2008-12-31 12:21:15 -0800 (Wed, 31 Dec 2008) \$

Last Updated 2006/5/5

Table of Contents

- VIM Syntax
 - General remarks
 - Installation
 - Folding
 - Closing Remarks

General remarks

vi (vim) resides in: /usr/share/vim/current (vim_root).

Syntax definitions are in vim_root/syntax/*.vim files. Linking file types to extensions is done in vim_root/ftdetect/*.vim. (The the star denotes the extension and would be replaced with map to define highlighting of MapServer config files).

Installation

The installation process requires two steps:

1. Copy the map.vim syntax file to the syntax directory

Copy map.vim to the vim syntax directory (on Suse: /usr/share/vim/current/syntax)

2. Register the syntax file in the custom filetype directory

If absent, create a new directory called ftdetect in vim_root (on Suse Linux: /usr/share/vim/current). Create a new file called map.vim in ftdetect according to the example shown below:

```
" MapServer config file
au BufNewFile,BufRead *.map      setf map
```

Now restart vim and open a map file

Folding

Introduction

Vim code folding can be an extremely handy way to simplify the task of editing complex MapServer mapfiles inside the Vim editor. A Vim command file called map_fold.vim is attached to this document, which defines a folding mode for mapfiles. This file also lives at http://iconocla.st/code/dot/map_fold.vim.

Installation

In Vim 6, you can copy *map_fold.vim* as-is to your *\$VIMROOT/ftplugin/* directory, and it should more or less work.

To use locally, create a *~/.vim/ftplugin* directory, copy *map_fold.vim* to it, and then add the following to your *~/.vimrc*:

```
autocmd BufRead *.map set filetype=map
```

map_fold.vim is totally compatible with the map.vim syntax highlighter, and their combined use is even recommended.

Use

The Vim [folding tutorial](#) (at least the first two or three sections) is highly recommended. The Vim [folding reference](#) may also be helpful.

Conclusion

Hope this helps! Please let me know if you find it useful, or if you find ways to improve it. Thanks!

Closing Remarks

The keywords are based on the TextPad Symbol Map_40.syn file contributed by Christopher Thorne, which can also be found on the MapServer utility page. I hope the file is somehow useful to anybody out there. If there are any questions or suggestions, please feel free to contact me.

14.8 External Links

- [Buildbot](#)
- [Developers](#)
- [Ohloh](#)
- [Release Plans](#)

DOWNLOAD

15.1 Source

15.1.1 Current Release(s)

- **2010-01-08** mapserver-5.6.1.tar.gz Changelog

15.1.2 Development Releases

- none available

15.1.3 Past Releases

- **2009-12-18** mapserver-5.6.0.tar.gz Changelog
- **2009-07-23** mapserver-5.4.2.tar.gz Changelog
- **2009-07-23** mapserver-5.2.3.tar.gz Changelog
- **2009-07-23** mapserver-4.10.5.tar.gz Changelog

15.1.4 Development Source

- Current Trunk: <http://svn.osgeo.org/mapserver/trunk/>
- Latest Release Branch: <http://svn.osgeo.org/mapserver/branches/branch-5-6>

Note: More information about MapServer development can be found at *Development*

15.2 Binaries

15.2.1 Windows

OSGeo4W OSGeo4W is a binary distribution of a broad set of Open Source geospatial software for Win32 environments (Windows XP, Vista, etc). OSGeo4W includes GDAL/OGR, GRASS, MapServer, OpenEV, uDig, as well as many other packages (about 70 as of summer 2008).

MS4W MapServer for Windows from MapTools.org. If you are a beginner looking for a complete MapServer solution on the Windows platform, or an advanced user looking to avoid compiling your own, this is what you're looking for. MS4W is a complete Web Server/MapServer/MapScript package. MS4W includes several flavors of MapScript (PHP, C#, Python, Java), and additional dlls for SDE and Oracle support. Several popular Web applications are also available for download, which are pre-configured for MS4W.

FWTools A collection of Open Source GIS tools provided by Frank Warmerdam. This goes beyond your basic web mapping tools and include many useful utilities for creating, manipulating, and serving geo-spatial data.

15.2.2 Linux

FGS The FGS Linux Installer is a self-extracting file that will install MapServer with PHP/MapScript and all of their dependencies on your Linux system. It provides a stand-alone environment with all the required software (incl. Apache and PHP) to run PHP/MapScript webmapping applications. The bundled version of Apache can be configured to run on any port so it will not interfere with previous installations of Apache or other web servers on your system.

DebianGIS DebianGIS provides a number of packages including MapServer, PostGIS, GDAL, QGIS, and GEOS. It sometimes lags a little bit behind the absolute latest release, but DebianGIS provides a solid integration environment for a giant slug of Open Source GIS software.

UbuntuGIS UbuntuGIS provides up to date packages of MapServer and related OSGeo tools for the last three releases of Ubuntu.

15.2.3 Mac OS X

Kyng Chaos William Kyngesburye provides excellent, up-to-date packages for Mac OS X. Most of the Open Source GIS stack is provided, including GEOS, GDAL, QGIS, and PostGIS.

15.3 Demo Application

Download MapServer 5.4.x Demo Download and unzip this file into your computer. Follow the directions in ReadMe.txt. You will most likely need to move the demo directory tree to an appropriate location in your httpd server. You will also need to edit some of the mapfile and template file parameters to reflect the changes in your paths (directory locations).

View Demo Application The link above provides a live version of the demo application.

GLOSSARY

AGG *Anti-Grain Geometry* A high quality graphics rendering engine that MapServer 5.0+ can use. It supports sub-pixel anti-aliasing, as well as many more features.

CGI Wikipedia provides excellent coverage of *CGI*.

EPSG EPSG codes are numeric codes associated with coordinate system definitions. For instance, EPSG:4326 is geographic WGS84, and EPSG:32611 is “UTM zone 11 North, WGS84”. The WMS protocol uses EPSG codes to describe coordinate systems. EPSG codes are published by the *OGC Surveying and Positioning Committee*. A list of PROJ.4 definitions corresponding to the EPSG codes can be found in the file `/usr/local/share/proj/epsg.PROJECTION` describes how to use these in your *Mapfile*.

See Also:

<http://spatialreference.org> for a listing of spatial references and an interface to search for spatial references.

Filter Encoding *Filter Encoding* is an *OGC* standard which defines an XML encoding for filter expressions to allow for spatial and attribute querying.

See Also:

WFS Filter Encoding

FreeType *FreeType* is a font engine that MapServer uses for accessing and rendering *TrueType* fonts.

GD *GD* is a graphics library for dynamic generation of images. It was the first graphics renderer that was available for MapServer, and it is required by MapServer to operate.

GDAL *GDAL* (*Geospatial Data Abstraction Library*) is a multi-format raster reading and writing library. It is used as the primary mechanism for reading raster data in MapServer. It is hosted at <http://www.gdal.org/>

GEOS *Geometry Engine Open Source* is a C/C++ port of the *Java Topology Suite*. It is used for geometric algebra operations like determining if a polygon is contained in another polygon or determining the resultant intersection of two or more polygons. MapServer optionally uses *GEOS* for geometric algebra operations.

GML *Geography Markup Language* is an *OGC* standard which defines an abstract model for geographic features

See Also:

WFS Server

GPX *GPS eXchange Format* is an XML Schema for describing GPS data. *OGR* can be used to transform and render this data with MapServer.

Map Scale A treatise of mapping scale can be found on about.com.

Mapfile *Mapfile* is the declarative language that MapServer uses to define data connections, map styling, templating, and server directives. Its format is xml-like and hierarchical, with closing `END` tags, but the format is not xml.

MapScript *MapScript* is an alternative to the *CGI* application of *mapserv* that allows you to program the MapServer object API in many languages.

Mercator Wikipedia provides excellent coverage of the [Mercator projection](#).

OGC The [Open Geospatial Consortium](#) is a standards organization body in the GIS domain. MapServer supports numerous OGC standards.

See Also:

[WMS Server](#) and [WMS Time](#) and [WMS Client](#) and [WFS Server](#) and [WFS Client](#) and [WCS Server](#) and [Map Context](#) and [SLD](#) and [WFS Filter Encoding](#) and [SOS Server](#)

OGR OGR is the vector data access portion of the *GDAL* library. It provides access to a multitude of data formats.

See Also:

[OGR](#)

OM [Observations and Measurements](#) is an *OGC* standard which defines an abstract model for sensor based data.

See Also:

[SOS Server](#)

OpenLayers [OpenLayers](#) is a JavaScript library for developing draggable, “slippy map” web applications.

Proj.4 [Proj4](#) is a library for projecting map data. It is used by MapServer and GDAL and a multitude of other Open Source GIS libraries.

Projection A map projection is a mathematical transformation of the surface of a sphere (3D) onto a 2D plane. Due to the laws of the universe, each type of projection must make tradeoffs on how and what features it distorts.

Raster A raster is a rectangular grid of pixels. Essentially an image. Rasters are supported in MapServer with a layer type of RASTER, and a variety of formats are supported including GeoTIFF, JPEG, and PNG.

Shapefile Shapefiles are simple GIS vector files containing points, lines or areas. The format was designed and published by ESRI and is widely supported in the GIS world. It is effectively the native and highest performance format for MapServer.

See Also:

[Wikipedia](#)

SLD *SLD* is an *OGC* standard which allows for custom symbolization for portrayal of data.

See Also:

[SLD](#)

SOS *SOS* is an *OGC* standard which provides an API for managing deployed sensors and retrieving sensor and observation data.

See Also:

[SOS Server](#)

Spherical Mercator [Spherical Mercator](#) is a term used to describe the *PROJECTION* used by many commercial API providers.

SVG [Scalable Vector Graphics](#) is an XML format that MapServer can output. It is frequently used in browser and mobile devices.

See Also:

[SVG](#)

SWF Shockwave Flash format that MapServer can generate for output.

See Also:

Flash Output

SWIG Simplified Wrapper Interface Generator is the library that MapServer uses for generating the language bindings for all languages other than C/C++ and PHP. *MapScript* describes these bindings.

Tileindex A tileindex is a *Shapefile* or other *Vector* data source that contains footprints of *Raster* data coverage. MapServer can use a tileindex to render a directory of raster data. The tileindex allows MapServer to only read the data that intersects the requested map extent, rather than reading all of the data.

See Also:

Tile Indexes

Vector Geographic features described by geometries (point, line, polygon) on a (typically) cartesian plane.

WCS *WCS* is an *OGC* standard that describes how to systematically produce structured *Raster* cartographic data from a service and return them to a client

See Also:

WCS Server and WCS Use Cases

WFS *WFS* is an *OGC* standard that describes how to systematically produce structured *Vector* cartographic data from a service and return them to a client.

See Also:

WFS Server and WFS Client

WMC *Web Map Context* is an *OGC* standard which allows for sharing of map views of *WMS* layers in multiple applications.

See Also:

Map Context

WMS *WMS* is an *OGC* standard that describes how to systematically produce rendered map images from a service and return them to a client.

See Also:

WMS Server and WMS Client

ERRORS

17.1 drawEPP(): EPPL7 support is not available

Error displayed when not using EPPL7 data.

This is a confusing error for users who are not even trying to view EPPL7 layers (EPPL7 is a raster format). The full error displayed may appear as follows:

```
msDrawRaster(): Unrecognized or unsupported image format ...
drawEPP(): EPPL7 support is not available.
```

17.1.1 Explanation

When MapServer tries to draw a layer, it will attempt to use all of the drivers it knows about, and the EPPL7 driver is the very last driver it will try. This means that if a layer fails to draw for any reason, you will see this error message.

There are other possible instances when this error can appear however, here are a few:

- the server is returning either a `ServiceException` (which MapServer does not yet detect and parse into a reasonable error message) or it is returning an image in an unrecognized format ... for instance it is returning a GIF image and MapServer is not built to support GIF images.
- WMS servers often advertise multiple image formats but don't respect them in the getmap request.

17.2 loadMapInternal(): Given map extent is invalid

When loading your mapfile or one of your layers, MapServer complains about an invalid extent.

Beginning in MapServer 4.6, MapServer got more strict about LAYER and MAP extents. If `minx` is greater than `maxx`, or `miny` is greater than `maxy`, this error will be generated. Check your MAP's `EXTENT`, LAYER's `EXTENT`, or `wms_extent` setting to make sure this is not the case. MapServer **always** takes in extents in the form of:

```
EXTENT minx miny maxx maxy
```

17.2.1 How to get a file's EXTENT values?

The easiest way to get a vector file's EXTENT is to use the `ogrinfo` utility, that is part of the GDAL/OGR library (for raster files you would use the `gdalinfo` utility). Windows users can download the `FWTools` package, which includes all of the GDAL and OGR commandline utilities. `MS4W` also includes the utilities (in `ms4w/tools/gdal-ogr-utils/`). Linux users will probably already have the GDAL libraries, if not you can also use the `FWTools` package.

For example, here is the results of the `ogrinfo` command on a shapefile (notice the "Extent" line):

```
$ ogrinfo province.shp province -summary
INFO: Open of 'province.shp'
      using driver 'ESRI Shapefile' successful.

Layer name: province
Geometry: Polygon
Feature Count: 1071
Extent: (-2340603.750000, -719746.062500) - (3009430.500000, 3836605.250000)
Layer SRS WKT:
(unknown)
AREA: Real (16.0)
PERIMETER: Real (16.0)
PROVINCE_: Real (16.0)
PROVINCE_I: Real (16.0)
STATUS: String (64.0)
NAME: String (64.0)
NAME_E: String (64.0)
NAME_F: String (64.0)
REG_CODE: Real (16.0)
POLY_FEATU: Real (16.0)
ISLAND: String (64.0)
ISLAND_E: String (64.0)
ISLAND_F: String (64.0)
YYY: Real (16.0)
SIZE: Real (16.0)
ANGLE: Real (16.0)
```

Ogrinfo gives the file's extent in the form of (minx, miny),(maxx, maxy), therefore the EXTENT in a mapfile would be:

```
EXTENT -2340603.750000 -719746.062500 3009430.500000 3836605.250000
```

Note: The EXTENT in a mapfile must be in the same units as the *MAP* -level *PROJECTION*.

17.3 msSaveImageGD(): Unable to access file

Error displayed when attempting to display map image.

This error is displayed if MapServer cannot display the map image. There are several things to check:

- IMAGEPATH and IMAGEURL parameters in mapfile are valid
- In CGI mode, any IMAGEPATH and IMAGEURL variables set in the init pages are valid
- Linux/Unix users should verify that the web user has permissions to write to the IMAGEPATH

17.4 msLoadMap(): Failed to open map file

Error displayed when trying to display map image.

The message tells you that MapServer cannot find mapfile. Verify that you have specified the correct path to the mapfile. Linux/Unix users should make sure that the web user has access permissions to the mapfile path as well.

The error message states where MapServer thinks the mapfile is:

```
[MapServer Error]: msLoadMap(): (D:/ms4w/apps/blah/blah.map)
Failed to open map file D:/ms4w/apps/blah/blah.map
```

17.5 msQueryByPoint: search returned no results

Error displayed when a feature is known to exist at that location.

The query feature requires a `TEMPLATE` object in the `CLASS` object of your `LAYER` definition. The value points to a html fragment using MapServer template syntax.

Example MapFile fragment:

```
LAYER
  NAME "Parcel9"
  TYPE POLYGON
  STATUS OFF
  DATA "Parcels/area09_parcels"
  CLASS
    STYLE
      OUTLINECOLOR 128 128 128
      COLOR 153 205 255
    END
    TEMPLATE "templates/Parcels/area09_parcels.html"
  END

  HEADER "templates/Parcels/area09_parcels_header.html"
  FOOTER "templates/Parcels/area09_parcels_footer.html"

END
```

Example Template:

```
<tr>
  <td>[lrn]</td>
  <td>[PIN]</td>
</tr>
```

The `[lrn]` is a special keyword that indicates the resulting line number which starts at 1. `[PIN]` is the name of a feature attribute.

17.6 msLoadFontset(): Error opening fontset

Error when attempting to display a label.

This message tells you that MapServer cannot find the `FONTSET` specified in the *Mapfile*.

The FONTSET path is relative to the mapfile location.

See Also:

FONTSET

17.7 msGetLabelSize(): Requested font not found

Error displayed when attempting to display a specific font.

This message tells you that MapServer cannot find specified font.

Make sure that the font is properly referenced in the FONTSET lookup file.

See Also:

FONTSET

17.8 loadLayer(): Unknown identifier. Maximum number of classes reached

Error displayed when attempting to draw a layer with a large number of classes.

This error states that MapServer has reached its limit for the maximum number of classes for the layer. This maximum can be modified in the MapServer source, and can then be re-compiled. *map.h* contains the default values, and below are the defaults for MapServer 4.10 and 4.8:

```
#define MS_MAXCLASSES 250
#define MS_MAXSTYLES 5
#define MS_MAXLAYERS 200
```

Note: This limitation was corrected in MapServer 5.0 and should no longer be a problem.

17.9 msReturnPage(): Web application error. Malformed template name

This error may occur when you are attempting to use a URL template for a query. The issue is that URL templates are only allowed for query modes that return only one result (e.g. query or itemquery)

You can only use a URL template for a query in mode=query or mode=itemquery. If you try it with mode=nquery or mode=itemnquery, you will get the error:

```
Content-type: text/html msReturnPage(): Web application error. Malformed template name
```

See Also:

MapServer CGI Controls

17.10 Unable to load dll (*MapScript*)

One of the dll-s could not be loaded that mapsript.dll depends on.

You can get this problem on Windows and in most cases it can be dedicated to a missing or an unloadable shared library. The error message talks about mapsript.dll but surely one or more of the dll-s are missing that libmap.dll depends on. So firstly you might want to check for the dependencies of your libmap.dll in your application directory. You can use the Visual Studio Dependency Walker to accomplish this task. You can also use a file monitoring tool (like SysInternal's filemon) to detect the dll-s that could not be loaded. I propose to store all of the dll-s required by your application in the application folder. If you can run the mapsript sample applications properly your compilation might be correct and all of the dlls are available.

17.10.1 C#-specific information

You may find that the mapsript C# interface behaves differently for the desktop and the ASP.NET applications. Although you can run the drawmap sample correctly you may encounter the dll loading problem with the ASP.NET applications. When creating an ASP.NET project your application folder will be 'Inetpubwwwroot[YourApp]bin' by default. The host process of the application will aspnet_wp.exe or w3wp.exe depending on your system. The application will run under a different security context than the interactive user (under the context of the ASPNET user by default). When placing the dll-s outside of your application directory you should consider that the PATH environment variable may differ between the interactive and the ASPNET user and/or you may not have enough permission to access a dll outside of your application folder.

17.11 msProcessProjection(): Projection library error.major axis or radius = 0 not given

Error displayed when attempting to specify projection parameters.

Since MapServer 4.0, you are required to specify the ellipsoid for the projection. Omitting this ellipsoid parameter in later MapServer versions will cause this error.

17.11.1 Valid Examples

4.0 and newer:

```
PROJECTION
  "proj=latlong"
  "ellps=WGS84"
END
```

before MapServer 4.0:

```
PROJECTION
  "proj=latlong"
END
```

See Also:

PROJECTION and <http://spatialreference.org>

17.12 msProcessProjection(): no options found in ‘init’ file

Error displayed when attempting to use a specific projection.

The message tells you that the projection you’re trying to use isn’t defined in the epsg file. Open your epsg file in a text editor and search for your projection to make sure that it exists.

On Windows, the default location of the epsg file is `c:\projnad`. MS4W users will find the epsg file in `\ms4w\projnad`.

See Also:

PROJECTION and <http://spatialreference.org>

17.13 msProcessProjection(): No such file or directory

Error displayed when trying to refer to an epsg file.

The message tells you that MapServer cannot find the epsg file.

On Windows, the default location of the epsg file is `c:\projnad`. MS4W users will find the epsg file in `\ms4w\projnad`.

Linux/Unix users should be careful to **specify the correct case** when referring to the epsg file, since filenames are case sensitive on Linux/Unix. “init=epsg:4326” refers to the epsg filename, and therefore “init=EPSG:4326” will not work because it will be looking for an *EPSG* file in uppercase.

17.13.1 Setting the location of the epsg file

There are a few options available if you need to set the epsg location:

1. Use a system variable (“environment variable” on windows) called “PROJ_LIB” and point it to your epsg directory.
2. Use the mapfile parameter CONFIG to force the location of the epsg file. This parameter is specified at the MAP level

See Also:

Mapfile

```
MAP
...
CONFIG "PROJ_LIB" "C:/somedir/proj/nad/"
...
END
```

3. Set an environment variable through your web server. Apache has a SetEnv directive that can set environment variables. Add something like the following to your Apache *httpd.conf* file:

```
SetEnv PROJ_LIB C:/somedir/proj/nad/
```

17.14 msQueryByPoint: search returned no results

Why do I get the message “msQueryByPoint(): Search returned no results. No matching record(s) found” when I query a feature known to exist?

The query feature requires a TEMPLATE object in the CLASS object of your LAYER definition. The value points to a html fragment using MapServer template syntax.

Example MapFile fragment:

```
LAYER
  NAME "Parcel9"
  TYPE POLYGON
  STATUS OFF
  DATA "Parcels/area09_parcels"
  CLASS
    STYLE
      OUTLINECOLOR 128 128 128
      COLOR 153 205 255
    END
    TEMPLATE "templates/Parcels/area09_parcels.html"
  END

  HEADER "templates/Parcels/area09_parcels_header.html"
  FOOTER "templates/Parcels/area09_parcels_footer.html"

END
```

Example Template:

```
<tr>
  <td>[lrn]</td>
  <td>[PIN]</td>
</tr>
```

The [lrn] is a special keyword that indicates the resulting line number which starts at 1. [PIN] is the name of a feature attribute.

18.1 Where is the MapServer log file?

See *MS RFC 28: Redesign of LOG/DEBUG output mechanisms*

18.2 What books are available about MapServer?

“Mapping Hacks” by Schuyler Erle, Rich Gibson, and Jo Walsh is available from O’Reilly.

“Web Mapping Illustrated” by Tyler Mitchell is available from O’Reilly. Introduces MapServer and many other related technologies including, GDAL/OGR, MapScript, PostGIS, map projections, etc.

“MapServer: Open Source GIS Development” by Bill Kropla.

18.3 How do I compile MapServer for Windows?

See *Compiling on Win32*. Also, you can use the development libraries in *OSGeo4W* as a starting point instead of building all of the dependent libraries yourself.

18.4 What do MapServer version numbers mean?

MapServer’s version numbering scheme is very similar to Linux’s. For example, a MapServer version number of 4.2.5 can be decoded as such:

- 4: Major version number. MapServer releases a major version every two to three years.
- 2: Minor version number. Increments in minor version number almost always relate to additions in functionality.
- 5: Revision number. Revisions are bug fixes only. No new functionality is provided in revisions.

From a developer’s standpoint, MapServer version numbering scheme is also like Linux. Even minor version numbers (0..2..4..6) relate to *release* versions, and odd minor versions (1..3..5..7) correspond to developmental versions.

18.5 Is MapServer Thread-safe?

Q: Is MapServer thread-safe?

A: Generally, no (but see the next question). Many components of MapServer use static or global data that could potentially be modified by another thread. Under heavy load these unlikely events become inevitable, and could result in sporadic errors.

Q: Is it possible to safely use any of MapServer in a multi-threaded application?

A: Some of it, yes, with care. Or with Python :) Programmers must either avoid using the unsafe components of MapServer or carefully place locks around them. Python's global interpreter lock immunizes against MapServer threading problems; since no mapscript code ever releases the GIL all mapscript functions or methods are effectively atomic. Users of mapscript and Java, .NET, mod_perl, or mod_php do not have this extra layer of protection.

A: Which components are to be avoided?

Q: Below are lists of unsafe and unprotected components and unsafe but locked components.

Unsafe:

- OGR layers: use unsafe CPL services
- Cartoline rendering: static data
- Imagemap output: static data
- SWF output: static data and use of unsafe `msGetBasename()`
- SVG output: static data
- WMS/WFS server: static data used for state of dispatcher
- Forcing a temporary file base (an obscure feature): static data
- MyGIS: some static data

Unsafe, but locked:

- Map config file loading: global parser
- Setting class and layer filter expressions (global parser)
- Class expression evaluation (global parser)
- Setting map and layer projections (PROJ)
- Raster layer rendering and querying (GDAL)
- Database Connections (mappool.c)
- PostGIS support
- Oracle Spatial (use a single environment handle for connection)
- SDE support (global layer cache)
- Error handling (static repository of the error objects)
- WMS/WFS client connections: potential race condition in Curl initialization
- Plugin layers (static repository of the loaded dll-s)

Rather coarse locks are in place for the above. Only a single thread can use the global parser at a time, and only one thread can access GDAL raster data at a time. Performance is exchanged for safety.

18.6 What does STATUS mean in a LAYER?

STATUS ON and STATUS OFF set the default status of the layer. If a map is requested, those layers will be ON/OFF unless otherwise specified via the layers parameter. This is particularly the case when using MapScript

and MapServer's built-in template mechanism, but is also useful as a hint when writing your own apps and setting up the initial map view.

STATUS DEFAULT means that the layer is always on, even if not specified in the layers parameter. A layer's status can be changed from DEFAULT to OFF in MapScript, but other than that, it's always on.

CGI turns everything off that is not "STATUS DEFAULT" off so all layers start from the same state (e.g. off) and must be explicitly requested to be drawn or query. That common state made (at least in my mind) implementations easier. I mean, if a layer "lakes" started ON the doing layer=lakes would turn it OFF. So I wanted to remove the ambiguity of a starting state.

18.7 How can I make my maps run faster?

There are a lot of different approaches to improving the performance of your maps, aside from the obvious and expensive step of buying faster hardware. Here are links to some individual howtos for various optimizations.

- [Tuning your mapfile for performance](#)
- [Optimizing the performance of vector data sources](#)
- [Optimizing the performance of raster data sources](#)
- [Tileindexes for mosaicing and performance](#)

Some general tips for all cases:

- First and foremost is hardware. An extra GB of RAM will give your map performance increases beyond anything you're likely to achieve by tweaking your data. With the price of RAM these days, it's cheap and easy to speed up every map with one inexpensive upgrade.
- Use the scientific method. Change one thing at a time, and see what effect it had. Try disabling all layers and enabling them one at a time until you discover which layer is being problematic.
- Use *shp2img* program to time your results. This runs from the command line and draws an image of your entire map. Since it's run from the command line, it is immune to net lag and will give more consistent measurements than your web browser.

18.8 What does Polyline mean in MapServer?

There's confusion over what POLYLINE means in MapServer and via ESRI. In MapServer POLYLINE simply means a linear representation of POLYGON data. With ESRI polyline means multi-line. Old versions of the Shapefile technical description don't even refer to polyline shapefiles, just line. So, ESRI polyline shapefiles are just linework and can only be drawn and labeled as LINE layers. Those shapefiles don't have feature closure enforced as polygon shapefiles do which is why the distinction is so important. I suppose there is a better choice than POLYLINE but I don't know what it would be.

Note: The only difference between POLYLINE and LINE layers is how they are labeled.

18.9 What is MapScript?

MapScript is the scripting interface to MapServer, usually generated by *SWIG* (except in the case of *PHP MapScript*). MapScript allows you to program with MapServer's objects directly instead of interacting with MapServer through its *CGI* and *Mapfile*.

18.10 Does MapServer support reverse geocoding?

No.

Reverse geocoding is an activity where you take a list of street features that you already have and generate postal addresses from them. This kind of spatial functionality is provided by proprietary packages such as the ESRI suite of tools, as well as services such as those provided by GDT. MapServer is for *map rendering*, and it does not provide for advanced spatial operations such as this.

18.11 Does MapServer support geocoding?

No.

Geocoding is an activity where you take a list of addresses and generate lat/lon points for them. This kind of spatial functionality is provided by proprietary packages such as the ESRI suite of tools, as well as services such as those provided by GDT. MapServer is for *map rendering*, and it does not provide for advanced spatial operations such as this.

If you are using MapScript, there is a free geocoder available through XMLRPC and SOAP at <http://geocoder.us> . You could hook you application up to use this service to provide lat/lon pairs for addresses, and then use MapServer to display those points.

18.12 How do I set line width in my maps?

You must set the symbol for the LAYER to be 'circle' and then you can set the symbol SIZE to be the width you want. A 'circle' symbol can be defined as

```
SYMBOL
  NAME 'circle'
  TYPE ELLIPSE
  FILLED TRUE
  POINTS 1 1 END
END
```

18.13 Why do my JPEG input images look crappy via MapServer?

The default output format for MapServer is 8bit pseudo-colored PNG or GIF. Inherently there will be some color degradation in converting a 24bit image (16 million colors) image into 8bit (256 colors).

But in order to ensure fast rendering MapServer uses quite a simple method to do the transformation, converting pixels to the nearest color in a 175 color colorcube. This will usually result in blotchy color in a fairly smoothly varying image.

Solutions include:

- Select 24bit output. This might be as easy as "IMAGETYPE JPEG" in your MAP section.
- Enable dithering (PROCESSING "DITHER=YES") to produce a better color appearance.
- Preprocess your image to 8bit before using it in MapServer with an external application like the GDAL `rgb2pct.py` script.

For more information review the *Raster Data*.

18.14 Which image format should I use?

Although MapScript can generate the map in any desired image format it sufficient to only consider the three most prevalent ones: JPEG, PNG, and GIF.

JPEG is an image format that uses a lossy compression algorithm to reduce an image's file size and is mostly used when loss of detail through compression is either not noticeable or negligible, as in most photos. Maps on the other hand mainly consist of fine lines and areas solidly filled in one colour, which is something JPEG is not known for displaying very well. In addition, maps, unless they include some aerial or satellite imagery, generally only use very few different colours. JPEG with its 24bit colour depth capable of displaying around 16.7 million colours is simple not suitable for this purpose. GIF and PNG however use an indexed colour palette which can be optimized for any number (up to 256) of colours which makes them the perfect solution for icons, logos, charts or maps. The following comparison (generated file sizes only; not file generation duration) will therefore only include these two file formats:

Table 18.1: GIF vs. PNG vs. PNG24 Generated Map File Sizes

	GIF	PNG	PNG24
Vector Data only	59kb	26kb	69kb
Vector Data & Satellite Image coloured	156kb	182kb	573kb
Vector Data & Satellite Image monochrome	142kb	134kb	492kb

(results based on an average 630x396 map with various colours, symbols, labels/annotations etc.)

Although GIF shows a quantitative as well as qualitative advantage over PNG when generating maps that contain full coloured remote sensing imagery, PNG is the clear quantitative winner in terms of generated file sizes for maps with or without additional monochrome imagery and should therefore been the preferred image format. If the mapping application however can also display fullcolour aerial or satellite imagery, the output file format can be changed dynamically to either GIF or even PNG24 to achieve the highest possible image quality.

18.15 Why doesn't PIL (Python Imaging Library) open my PNGs?

PIL does not support interlaced PNGs at this time (no timetable on when it actually will either). To be able to read PNGs in PIL, they must not be interlaced. Modify your OUTPUTFORMAT with a FORMATOPTION like so:

```
OUTPUTFORMAT
NAME png
DRIVER "GD/PNG"
MIMETYPE "image/png"
IMAGEMODE RGB
EXTENSION ".png"
FORMATOPTION "INTERLACE=OFF"
END
```

18.16 Why do my symbols look poor in JPEG output?

When I render my symbols to an 8bit output (PNG, GIF) they look fine, but in 24bit jpeg output they look very blocky and gross.

In order to render some classes of symbols properly in 24bit output, such as symbols from true type fonts, it is necessary to force rendering to occur in RGBA. This can be accomplished by including the "TRANSPARENCY ALPHA" line in the layer definition. Don't use this unnecessarily as there is a performance penalty.

This problem also affects PNG24 output or any RGB output format. 8bit (PC256) or RGBA output types are already ok.

18.17 How do I add a copyright notice on the corner of my map?

You can use an inline feature, with the *FEATURE* object, to make a point on your map. Use the *TEXT* parameter of the *FEATURE* object for the actual text of the notice, and a *LABEL* object to style the notice.

18.17.1 Example Layer

```
LAYER
  NAME "copyright"
  STATUS ON
  TYPE annotation
  TRANSFORM ll #set the image origin to be lower left
  FEATURE
    POINTS
      60 -10 #set the offset from lower left position in pixels
    END
    TEXT "@ xyz company 2006" #this is your displaying text
  END
  CLASS
    LABEL #defines the font, colors etc. of the text
      FONT "sans"
      TYPE TRUETYPE
      SIZE 8
      BUFFER 1
      COLOR 0 0 0
      BACKGROUNDCOLOR 255 255 255
      FORCE TRUE
    END
  END
  UNITS PIXELS #sets the units for the feature object
END
```

18.17.2 Result



18.18 How do I have a polygon that has both a fill and an outline with a width?

How do I have a polygon that has both a fill and an outline with a width? Whenever I put both a color (fill) and an outlinecolor with a width on a polygon within a single STYLE, the outline width isn't respected.

For historical reasons, width has two meanings within the context of filling polygons and stroke widths for the outline. If a polygon is filled, then the width defines the width of the symbol *inside* the filled polygon. In this event, the outline width is disregarded, and it is always set to 1. To achieve the effect of *both* a fill and an outline width, you need to use two styles in your class.

```
STYLE # solid fill
  COLOR 255 0 0
END
STYLE # thick outline (could use a circle symbol with size too)
  OUTLINECOLOR 0 0 0
  WIDTH 3
  ANTIALIAS TRUE
END
```

18.19 How can I create simple antialiased line features?

The easiest way to produce antialiased lines is to:

- use a 24-bit output image type (IMAGEMODE RGB (or RGBA))
- set TRANSPARENCY ALPHA in the layer using antialiased lines
- set ANTIALIAS TRUE in the STYLE element of the CLASS with antialiased lines

The following mapfile snippets enable antialiased county borders:

```
...
IMAGETYPE "png24"
...
OUTPUTFORMAT
  NAME "png24"
  DRIVER "GD/PNG"
  MIMETYPE "image/png"
  IMAGEMODE RGB
  EXTENSION "png"
END
...
LAYER
  NAME "counties"
  TYPE line
  STATUS default
  DATA "bdry_counln2"
  TRANSPARENCY alpha
  SYMBOLSCALE 5000000
  CLASS
    STYLE
      WIDTH 3
      COLOR 1 1 1
      ANTIALIAS true
    END
  END
END
...
```

Note: The `bdry_counln2` shapefile referenced in the counties layer is a line shapefile. A polygon shapefile could be substituted with roughly the same results, though owing to the nature of shapefiles each border would be rendered twice and the resulting output line would likely appear to be slightly thicker. Alternatively, one could use a polygon shapefile, set TYPE to POLYGON, and use OUTLINECOLOR in place of COLOR in the STYLE element.

Note: You can tweak the combination of STYLE->WIDTH and SYMBOLSCALE to modify line widths in your output images.

See Also:

Cartoline symbols can be used to achieve fancier effects.

18.20 Which OGC Specifications does MapServer support?

- Web Map Service (OGC:WMS) 1.0.0, 1.0.7, 1.1.0 and 1.1.1
- Web Feature Service (OGC:WFS) 1.0.0, 1.1.0
- Web Coverage Service (OGC:WCS) 1.0.0, 1.1.0

- Geography Markup Language (OGC:GML) 2.1.2, 3.1.0 Level 0 Profile
- Web Map Context Documents (OGC:WMC) 1.0.0, 1.1.0
- Styled Layer Descriptor (OGC:SLD) 1.0.0
- Filter Encoding Specification (OGC:FES) 1.0.0
- Sensor Observation Service (OGC:SOS) 1.0.0
- Observations and Measurements (OGC:OM) 1.0.0
- SWE Common (OGC:SWE) 1.0.1
- OWS Common (OGC:OWS) 1.0.0, 1.1.0

18.21 Why does my requested WMS layer not align correctly?

Requesting a layer from some ArcIMS WMS connectors results in a map with misaligned data (the aspect ratio of the pixels looks wrong).

Some ArcIMS sites are not set up to stretch the returned image to fit the requested envelope by default. This results in a map with data layers that overlay well in the center of the map, but not towards the edges. This can be solved by adding “*reaspect=false*” to the request (by tacking it on to the connection string).

For example, if your mapfile is in a projection other than EPSG:4326, the following layer will not render correctly:

```
LAYER
  NAME "hillshade"
  TYPE RASTER
  STATUS OFF
  TRANSPARENCY 70
  CONNECTIONTYPE WMS
  CONNECTION "http://gisdata.usgs.net:80/servlet19/com.esri.wms.Esrimap/USGS_WMS_NED?"
  PROJECTION
    "init=epsg:4326"
  END
  METADATA
    "wms_srs" "EPSG:4326"
    "wms_title" "US_NED_Shaded_Relief"
    "wms_name" "US_NED_Shaded_Relief"
    "wms_server_version" "1.1.1"
    "wms_format" "image/png"
  END
END
```

Adding “*reaspect=false*” to the connection string solves the problem:

```
LAYER
  NAME "hillshade"
  TYPE RASTER
  STATUS OFF
  TRANSPARENCY 70
  CONNECTIONTYPE WMS
  CONNECTION "http://gisdata.usgs.net:80/servlet19/com.esri.wms.Esrimap/USGS_WMS_NED?reaspect=false"
  PROJECTION
    "init=epsg:4326"
  END
  METADATA
```

```
"wms_srs" "EPSG:4326"  
"wms_title" "US_NED_Shaded_Relief"  
"wms_name" "US_NED_Shaded_Relief"  
"wms_server_version" "1.1.1"  
"wms_format" "image/png"
```

END
END

18.22 When I do a GetCapabilities, why does my browser want to download mapserv.exe/mapserv?

A beginner question here... I'm new to MS and to Apache. I've got MS4W up and running with the Itasca demo. Now I want to enable it as a WMS server. `mapserv -v` at the command line tells me it supports WMS_SERVER. When I point my browser to it, my browser just wants to download `mapserv.exe`!

What am I missing?

Here is the URL I'm using to issue a GetCapabilities WMS request: <http://localhost/cgi-bin/mapserv.exe?map=./htdocs/itasca/demo.map&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetCapabilities>

The OGC:WMS 1.1.0 and 1.1.1 specifications (which are both supported by MapServer) state that, for GetCapabilities responses, the OGC:WMS server returns a specific MIME type (i.e. `application/vnd.ogc.xml` (see subclause 6.5.3 of [OGC:WMS 1.1.1](#)).

A MIME type is passed from the web server to the client (in your case, a web browser), from which a client can decide how to process it.

Example 1: if using a web browser, if a web server returns an HTTP Header of “Content-type:image/png”, then the web browser will know that this is a PNG image and display it accordingly.

Example 2: if using a web browser, if a web server returns an HTTP Header of “Content-type:text/html”, then the web browser will know that this is an HTML page and display it accordingly (i.e. tables, divs, etc.)

Basically, what is happening is that the OGC:WMS is returning, in the headers of the HTTP response, a MIME type which your web browser does not understand, which usually prompts a dialog box on whether to open or download the content (i.e. `Content-type:application/vnd.ogc.wms_xml`).

You could configure your web browser to handle the `application/vnd.ogc.wms_xml` MIME type a certain way (i.e. open in Notepad, etc.).

18.23 Why do my WMS GetMap requests return exception using MapServer 5.0?

Before upgrading to MapServer 5.0, I was able to do quick GetMap tests in the form of: <http://wms.example.com/wms?service=WMS&version=1.1.1&request=GetMap&layers=foo>

Now when I try the same test, MapServer WMS returns an XML document saying something about missing required parameters. What's going on here?

This was a major change for WMS Server support in MapServer 5.0. MapServer WMS Server GetMap requests now require the following additional parameters:

- srs
- bbox

- width
- height
- format
- styles

Note: These parameters were always required in all versions of the WMS specification, but MapServer previously had not required them in a client request (even though most OGC WMS clients would issue them anyway to be consistent with the WMS spec).

The request below now constitutes a valid GetMap request:

```
http://wms.example.com/wms?service=WMS&version=1.1.1&request=GetMap&layers=foo&srs=EPSG:4326&bbox=-1
```

Which is consistent with the WMS specification.

More information on these parameters can be found in the *WMS Server* and the *OGC WMS 1.1.1 specification*

For more detailed information, see [ticket 1088](#)

Warning: STYLES, though a required WMS parameter, is now optional again in MapServer. For more detailed information, see [ticket 2427](#)

18.24 Where do I find my *EPSG* code?

There is a text file “epsg” in your PROJ4 installation (e.g. “/usr/local/share/proj/epsg”) which contain the EPSG information used by PROJ4. In Windows, this is often located in C:\proj\nad or is found with an environment variable called PROJ_LIB.

<http://spatialreference.org> allows you to search for EPSG codes.

You can also have a look at: <http://ocean.csl.co.uk>

More information to EPSG: <http://www.epsg.org>

More information to PROJ4: <http://trac.osgeo.org/proj>

18.25 How can I reproject my data using ogr2ogr?

With ogr2ogr of course! ogr2ogr is a powerful utility that will transform the projections of your shapefiles when passed the appropriate parameters. In my case, I was using MapServer to serve data in RI State Plane Feet. In order to do so, the data had to first be converted to meters. Here is the command I used:

```
ogr2ogr -t_srs EPSG:32130 output.shp input.shp
```

Since my data already had a projection defined, I did not need to explicitly state a source projection. This command uses the EPSG definition for NAD83 Rhode Island (32130) and performs the feet to meters conversion.

Now say my data wasn’t already projected? Here’s how we deal with that:

```
ogr2ogr -s_srs "+proj=tmerc +lat_0=41.08333333333334 +lon_0=-71.5 +k=0.999994 +x_0=100000 +y_0=0 +e1
```

Let's examine what is going on here:

The `-s_srs` parameter explicitly defines a projection for the data. The parameters used here were taken out of the EPSG definition (in this case, 32130) in the `epsg` file (see the projection FAQ for more details on locating EPSG definitions). The entry for RI in the `epsg` file is as follows:

```
# NAD83 / Rhode Island
<32130> +proj=tmerc +lat_0=41.08333333333334 +lon_0=-71.5 +k=0.999994 +x_0=100000 +y_0=0 +ellps=GRS80
```

You can see how the definition in the initial command is formulated. Notice that the `"+units=m"` parameter has been changed to `"+to_meter=0.3408"`. This is important for the conversion. Where did the value of 0.3408 come from you ask? From the EPSG file! It has many goodies buried in it so by simply running `'grep "to_meter" epsg'` you can refresh your memory if you need to.

The next parameter in the command is `"-t_srs EPSG:32130"`. This command tells `ogr2ogr` to transform the data to the EPSG code of 32130. After this is declared, all you need to do is declare a file name for your new shape file and to set which file is being used as the input (note: make sure you don't confuse the order of these two).

Hit enter, bombs away, and enjoy your new data in meters!

LICENSE

Copyright (c) 1996-2008 Regents of the University of Minnesota.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies of this Software or works derived from this Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CREDITS

Major funding for development of MapServer has been provided by NASA through cooperative agreements with the University of Minnesota, Department of Forest Resources. Additional enhancements have been made by the State of Minnesota, Department of Natural Resources and the Land Management Information Center.

- MapServer and MapScript have been developed by Stephen Lime.
- Raster access module developed by Pete Olson and Stephen Lime.
- PHP/MapScript module was developed by [DM Solutions](#) and is maintained by [MapGears](#).
- Portions copyright (c) 1998 State of Minnesota, Land Management Information Center.

BIBLIOGRAPHY

[FW1] Frank Warmerdam on the mapserver-dev:

Message-Id: <smtpd.490f.4303f2ee.d8246.1@mtaout-c.tc.umn.edu>

Date: Wed, 17 Aug 2005 22:31:09 -0400

Subject: Re: Mapserver Plug-in Infastructure: RFC and PATCH

INDEX

A

AGG, 671
ANTIALIAS
 LABEL, 80
 SYMBOL, 100

C

CGI, 55, 671
CLASS
 DEBUG, 63
 EXPRESSION, 63
 GROUP, 64
 KEYIMAGE, 64
 MAXSCALEDENOM, 64
 MINSCALEDENOM, 64
CLASSITEM, 82
 EXPRESSION, 66
cloning
 LAYER, 221
 mapfile, 221
CONFIG
 MAP, 89

D

DEBUG
 CLASS, 63
 LAYER, 83
download
 php, 45

E

EPSG, 671
EXPRESSION
 CLASS, 63
 CLASSITEM, 66

F

Filter Encoding, 671
FreeType, 671

G

GD, 671

GDAL, 671
GEOS, 671
GML, 671
GPX, 671
GROUP
 CLASS, 64

I

IIS, 55
INCLUDE
 MAP, 72
install
 Oracle, 58
installation
 php, 44
introduction
 php, 173

J

Join, 75

K

KEYIMAGE
 CLASS, 64

L

LABEL, 79
 ANTIALIAS, 80
LABELMAXSCALEDENOM
 LAYER, 84
LABELMINSCALEDENOM
 LAYER, 85
LAYER
 cloning, 221
 DEBUG, 83
 LABELMAXSCALEDENOM, 84
 LABELMINSCALEDENOM, 85
 MAXSCALEDENOM, 85
 MINSCALEDENOM, 85
 SYMBOLSCALEDENOM, 87
LEGEND, 89

M

MAP

- CONFIG, 89
- INCLUDE, 72
- SCALEDENOM, 92

Map Scale, 671

Mapfile, 671

mapfile

- cloning, 221
- MapScript, 220
- saving, 221

MapScript, 671

- mapfile, 220
- php, 184
- python, 215

MAXSCALEDENOM

- CLASS, 64
- LAYER, 85
- WEB, 135

Mercator, 672

MINSCALEDENOM

- CLASS, 64
- LAYER, 85
- WEB, 135

O

OGC, 672

OGR, 672

OM, 672

OpenLayers, 672

Oracle, 274

- install, 58

P

php, 55

- constants, 185
- download, 45
- installation, 44
- introduction, 173
- MapScript, 184

PROCESSING, 86

Proj.4, 672

Projection, 672

python

- MapScript, 215

Q

QUERYMAP, 96

R

Raster, 672

Raster options, 86

Regular expressions, 67

S

saving

- mapfile, 221

SCALEDENOM

- MAP, 92

Shapefile, 672

SLD, 672

SOS, 672

Special options, 86

Spherical Mercator, 672

SVG, 672

SWF, 672

SWIG, 673

SYMBOL

- ANTIALIAS, 100

SYMBOLSCALEDENOM

- LAYER, 87

T

Tileindex, 673

V

Vector, 673

W

WCS, 673

WEB

- MAXSCALEDENOM, 135

- MINSCALEDENOM, 135

WFS, 673

WMC, 673

WMS, 673