



MapServer
open source web mapping

MapServer Documentation

Release 6.4.1

The MapServer Team

November 03, 2015

1	Introduction	3
1.1	An Introduction to MapServer	3
1.1.1	MapServer Overview	3
1.1.2	Anatomy of a MapServer Application	4
1.1.3	Installation and Requirements	6
	Windows Installation	6
	Hardware Requirements	11
	Software Requirements	11
	Skills	12
1.1.4	Introduction to the Mapfile	12
	MAP Object	13
	LAYER Object	14
	CLASS and STYLE Objects	14
	SYMBOLS	15
	LABEL	17
	CLASS Expressions	18
	INCLUDE	19
	Get MapServer Running	20
	Get Demo Running	20
1.1.5	Making the Site Your Own	20
	Adding Data to Your Site	20
	Vector Data	20
	Raster Data	21
	Projections	21
1.1.6	Enhancing your site	21
	Adding Query Capability	21
	Attribute queries	21
	Spatial queries	21
	Interfaces	22
	Data Optimization	22
1.1.7	How do I get Help?	22
	Documentation	22
	Users Mailing List	22
	IRC	23
	Reporting bugs	23
	Tutorial	23
	Test Suite	23
	Books	23

2	Tutorial	25
2.1	MapServer Tutorial	25
2.1.1	Tutorial background	25
	Tutorial Timeframe	25
	Tutorial Data	25
	Before Using the Tutorial	26
	Windows, UNIX/Linux Issues	26
	Other Resources	27
2.1.2	Section 1: Static Maps and the MapFile	27
2.1.3	Section 2: CGI variables and the User Interface	28
	HTML Templates	28
	Examples	28
2.1.4	Section 3: Query and more about HTML Templates	28
2.1.5	Section 4: Advanced User Interfaces	29
3	Installation	31
3.1	Installation	31
3.1.1	Compiling on Unix	31
	Introduction	31
	Obtaining the necessary software	32
	libgd	33
	Anti-Grain Geometry Support	35
	OGC Support	35
	Spatial Warehousing	36
	Compiling	37
	Installation	39
3.1.2	Compiling on Win32	40
	Introduction	41
	Compiling	41
	Set up a Project Directory	41
	Download MapServer Source Code and Supporting Libraries	42
	The MapServer source code	42
	Set Compilation Options	43
	Compile the Libraries	44
	Compile MapServer	45
	Compiling MapServer with PostGIS support	45
	Common Compiling Errors	45
	Installation	46
	Other Helpful Information	47
	Acknowledgements	47
3.1.3	PHP MapScript Installation	47
	Introduction	47
	Obtaining, Compiling, and Installing PHP and the PHP/MapScript Module	48
	FAQ / Common Problems	51
3.1.4	.NET MapScript Compilation	53
	Compilation	53
	Installation	56
	Known issues	56
	Most frequent errors	58
	Bug reports	58
3.1.5	IIS Setup for MapServer	58
	Base configuration	59
	Php.ini file	59
	Internet Services Manager	59

	Under the tree for your new website - add virtual directories for	60
	Test PHP	60
	Mapfiles for IIS	61
	Configuration files:	61
3.1.6	Oracle Installation	61
	Preface	61
	System Assumptions	62
	Compile MapServer	62
	Set Environment Variables	62
4	Mapfile	65
4.1	Mapfile	65
4.1.1	Cartographical Symbol Construction with MapServer	65
	Abstract	66
	Introduction	67
	Using Cartographical Symbols in MapServer	69
	Construction of Point Symbols	70
	Construction of Line Symbols	77
	Area Symbols	88
	Examples (MapServer 4)	104
	Tricks	109
	Mapfile changes related to symbols	113
	Current Problems / Open Issues	113
	The End	114
4.1.2	Geometry Transformations	114
	Transformations for simple styling (<i>CLASS STYLE</i> only)	115
	Labels (<i>LABEL STYLE</i> only)	119
	Expressions and advanced transformations (<i>LAYER</i> and <i>CLASS STYLE</i>)	120
4.1.3	CLASS	128
4.1.4	CLUSTER	132
	Description	132
	Supported Layer Types	132
	Mapfile Parameters	132
	Supported Processing Options	133
	Mapfile Snippet	133
	Feature attributes	133
	PHP MapScript Usage	133
	Example: Clustering Railway Stations	134
4.1.5	Display of International Characters in MapServer	136
	Credit	137
	Related Links	137
	Requirements	137
	How to Enable in Your Mapfile	137
	Example Using PHP MapScript	139
	Notes	139
4.1.6	Expressions	140
	Introduction	140
	Expression Types	142
	“MapServer expressions”	144
4.1.7	FEATURE	150
4.1.8	FONTSET	151
	Format of the fontset file	151
4.1.9	GRID	152
	Description	152

Mapfile Parameters:	152
Example1: Grid Displaying Degrees	152
Example2: Grid Displaying Degrees with Symbol	153
Example2: Grid Displayed in Other Projection (Google Mercator)	154
4.1.10 INCLUDE	156
Notes	156
Example	156
4.1.11 JOIN	157
Description	157
Supported Formats	157
Mapfile Parameters:	157
Example 1: Join from Shape dataset to DBF file	158
Example 2: Join from Shape dataset to PostgreSQL table	159
Example 3: Join from Shape dataset to CSV file	160
Example 4: Join from Shape dataset to MySQL	162
Example 5: One-to-many join	162
4.1.12 LABEL	163
4.1.13 LAYER	168
4.1.14 LEADER	177
Description	178
Supported Layer Types	178
Mapfile Parameters	178
Mapfile Snippet	178
Example: World Countries Labels	178
4.1.15 LEGEND	181
4.1.16 MAP	182
4.1.17 OUTPUTFORMAT	186
4.1.18 PROJECTION	189
Background	189
Projections with MapServer	189
“Web Mercator” or “Google Mercator”	190
PROJECTION AUTO	191
Important Notes	191
For More Information	191
4.1.19 QUERYMAP	191
4.1.20 REFERENCE	192
4.1.21 SCALEBAR	192
4.1.22 STYLE	193
4.1.23 SYMBOL	201
4.1.24 Symbology Examples	202
Example 1. Dashed Line	203
Example 2. TrueType font marker symbol	203
Example 3. Vector triangle marker symbol	203
Example 4. Non-contiguous vector marker symbol (Cross)	204
Example 5. Circle vector symbol	204
Example 6. Downward diagonal fill	204
Example 7. Using the Symbol Type HATCH (new in 4.6)	205
Example 8. Styled lines using GAP	205
4.1.25 Templating	206
Introduction	206
Format	207
Example Template	215
4.1.26 Union Layer	216
Description	216

	Requirements	216
	Mapfile Configuration	216
	Feature attributes	217
	Classes and Styles	218
	Projections	218
	Supported Processing Options	218
	Examples	218
4.1.27	WEB	221
4.1.28	XML Mapfile support	223
	Enabling the support	223
	Usage:	223
4.1.29	Notes	224
5	MapScript	225
5.1	MapScript	225
5.1.1	Introduction	225
	Appendices	225
	Documentation Elements	225
	fooObj	225
	Additional Documentation	226
5.1.2	SWIG MapScript API Reference	226
	Introduction	228
	MapScript Constants	228
	MapScript Functions	234
	MapScript Classes	235
5.1.3	PHP MapScript	269
	Introduction	269
	PHP MapScript API	270
	PHP MapScript Migration Guide	303
	By Example	306
5.1.4	Python MapScript Appendix	315
	Introduction	316
	Classes	316
	Exception Handling	317
5.1.5	Python MapScript Image Generation	317
	Introduction	318
	Imagery Overview	318
	The imageObj Class	318
	Image Output	319
	Images and Symbols	319
5.1.6	Mapfile Manipulation	320
	Introduction	320
	Mapfile Overview	320
	The mapObj Class	320
	Children of mapObj	321
	Metadata	322
5.1.7	Querying	323
	Introduction	323
	Querying Overview	323
	Attribute Queries	324
	Spatial Queries	324
6	MapCache	327
6.1	MapCache	327

6.1.1	Compilation & Installation	327
	Getting the Source	328
	Linux Instructions	328
	Windows Instructions	335
6.1.2	Configuration File	337
	Source	338
	Cache	339
	Format	340
	Grid	341
	Tileset	343
	Services	346
	Miscellaneous	347
6.1.3	Supported Tile Services	348
	TMS service	348
	KML Service	349
	OGC WMTS Service	350
	OGC WMS Service	350
	GoogleMaps XYZ Service	351
	Virtual Earth Tile service	352
6.1.4	Seeder	353
	Usage	353
6.1.5	Cache Types	355
	Disk Caches	355
	BerkeleyDB Caches	356
	Sqlite Caches	357
	Memcache Caches	359
	(Geo)TIFF Caches	359
6.1.6	Image Formats	361
	JPEG Format	362
	PNG Format	362
	Mixed Format	362
6.1.7	Tileset Dimensions	362
6.1.8	FeatureInfo Requests	363
6.1.9	Proxying Unsupported Requests	363
	Parameter Filtering	363
	Proxy Destination	364
6.1.10	Data Sources	364
	HTTP Service Definition	365
	WMS Sources	365
	MapFile Sources	365
6.1.11	Tile Assembling	365
6.1.12	Features	365
7	Input	367
7.1	Data Input	367
7.1.1	Vector Data	367
	Data Format Types	368
	ArcInfo	369
	ArcSDE	370
	Contour	373
	DGN	375
	ESRI File Geodatabase	376
	ESRI Personal Geodatabase (MDB)	378
	ESRI Shapefiles (SHP)	380

GML	382
GPS Exchange Format (GPX)	383
Inline	385
KML - Keyhole Markup Language	387
MapInfo	390
MSSQL	391
MySQL	395
NTF	399
OGR	400
Oracle Spatial	412
PostGIS/PostgreSQL	417
SDTS	425
S57	427
Spatialite	429
USGS TIGER	432
Vector field rendering - UVraster	434
Virtual Spatial Data	436
WFS	440
7.1.2 Raster Data	441
Introduction	442
How are rasters added to a Map file?	442
Supported Formats	444
Rasters and Tile Indexing	445
Raster Warping	446
24bit RGB Rendering	446
Special Processing Directives	447
Raster Query	449
Raster Display Performance Tips	450
Preprocessing Rasters	451
Georeference with World Files	452
8 Output	455
8.1 Output Generation	455
8.1.1 AGG Rendering Specifics	455
Introduction	455
Setting the OutputFormat	455
New Features	456
Modified Behavior	457
8.1.2 AntiAliasing with MapServer	457
What needs to be done	458
8.1.3 Dynamic Charting	461
Setup	461
Adding a Chart Layer to a Mapfile	462
Pie Charts	464
Bar Graphs	465
8.1.4 Flash Output	465
Introduction	466
Installing MapServer with Flash Support	466
How to Output SWF Files from MapServer	467
What is Currently Supported and Not Supported	470
8.1.5 HTML Legends with MapServer	471
Introduction	471
Sample Site Using the HTML Legend	479
8.1.6 HTML Imagemaps	480

Introduction	480
Mapfile Layer Definition	481
Templates	481
Request URL	482
Additional Notes	482
More Information	482
8.1.7 OGR Output	482
Introduction	483
OUTPUTFORMAT Declarations	483
LAYER Metadata	484
MAP / WEB Metadata	485
Geometry Types Supported	485
Attribute Field Definitions	485
Return Packaging	486
Test Suite Example	486
8.1.8 PDF Output	486
Introduction	487
What is currently supported and not supported	487
Implementing PDF Output	488
PHP/MapScript and PDF Output	489
8.1.9 SVG	491
Introduction	492
Feature Types and SVG Support Status	492
Testing your SVG Output	494
goSVG	495
8.1.10 Tile Mode	498
Introduction	498
Configuration	498
Utilization	499
8.1.11 Template-Driven Output	502
Introduction	502
OUTPUTFORMAT Declarations	503
Template Substitution Tags	503
Examples	504
8.1.12 Kml Output	507
Introduction	508
General Fonctionnality	508
Output format	508
Build	508
Limiting the number of features	508
Map	509
Layers	509
Styling	512
Attributes	513
Coordinate system	513
Warning and Error Messages	513
9 OGC	515
9.1 OGC Support and Configuration	515
9.1.1 MapServer OGC Specification support	515
9.1.2 WMS Server	515
Introduction	516
Setting Up a WMS Server Using MapServer	517
Changing the Online Resource URL	522

WMS 1.3.0 Support	525
Reference Section	526
FAQ / Common Problems	539
9.1.3 INSPIRE View Service	540
Introduction	540
Activation of INSPIRE support	541
Multi-language support for certain capabilities fields	541
Provision of INSPIRE specific metadata	542
Named group layers	543
Style section for root layer and possibly existing group layers	544
9.1.4 WMS Client	546
Introduction	546
Compilation / Installation	546
MapFile Configuration	547
Limitations/TODO	552
9.1.5 WMS Time	553
Introduction	553
Enabling Time Support in MapServer	553
Future Additions	557
Limitations and Known Bugs	557
9.1.6 WMS Dimension	557
Introduction	558
Enabling Dimension Support in MapServer	558
GetCapabilities Output	559
Supported Dimension Requests	559
Processing Dimension Requests	559
9.1.7 Map Context	560
Introduction	560
Implementing a Web Map Context	561
9.1.8 WFS Server	567
Introduction	567
Configuring your MapFile to Serve WFS layers	568
Reference Section	572
To-do Items and Known Limitations	576
9.1.9 WFS Client	576
Introduction	576
Setting up a WFS-client Mapfile	577
TODO / Known Limitations	579
9.1.10 WFS-T Server	579
WFS-T	579
9.1.11 WFS Filter Encoding	579
Introduction	580
Currently Supported Features	580
Get and Post Requests	581
Use of Filter Encoding in MapServer	582
Limitations	584
Tests	584
9.1.12 SLD	587
Introduction	588
Server Side Support	588
Client Side Support	596
Named Styles support	597
Other Items Implemented	597
Issues Found During Implementation	598


9.1.13	WCS Server	598
	Introduction	598
	Configuring Your Mapfile to Serve WCS Layers	599
	Test Your WCS 1.0 Server	601
	WCS 1.1.0+ Issues	603
	WCS 2.0	604
	HTTP-POST support	609
	Reference Section	610
	Rules for handling SRS in a MapServer WCS	614
	Spatio/Temporal Indexes	614
	WCS 2.0 Application Profile - Earth Observation (EO-WCS)	615
	To-do Items and Known Limitations	615
9.1.14	WCS Use Cases	615
	Landsat	616
	SPOT	617
	DEM	617
	NetCDF	618
9.1.15	SOS Server	620
	Introduction	621
	Setting Up an SOS Server Using MapServer	621
	Limitations / TODO	626
	Reference Section	626
	Use of sos_procedure and sos_procedure_item	631
9.1.16	How to set up MapServer as a client to access a service over https	632
	Introduction	632
	Requirements	632
	Default Installation (with apt-get install, rpm, manual, etc)	632
	Non-Standard Installation (common with ms4w and fgs)	633
	Remote Server with a Self-Signed SSL Certificate	633
9.1.17	MapScript Wrappers for WxS Services	634
	Introduction	634
	Python Examples	635
	Perl Example	636
	Java Example	638
	PHP Example	639
	Use in Non-CGI Environments (mod_php, etc)	641
	Post Processing Capabilities	641
10	TinyOWS	643
10.1	TinyOWS	643
10.1.1	TinyOWS Installation	643
	Requires	643
10.1.2	Configuring TinyOWS with an XML File	644
	Configuration file simple Example	644
	Testing your config.xml file	645
	Structure of the config.xml file	645
10.1.3	Configuring TinyOWS with a standard Mapfile	650
	Mapfile Config File support for TinyOWS	650
	Mapfile path of each TinyOWS config element	651
10.1.4	Sample: WFS-T with TinyOWS and OpenLayers	652
10.1.5	Server Tuning: How to speed up your TinyOWS server	657
	Tips and Tricks for PostgreSQL / PostGIS databases	657
	Tips and Tricks for Apache	657
	Using Fast-CGI	657

HTTP GZip compression	658
10.1.6 Working Around the LibXML2 XSD Schema GML Bug	658
Issue	658
Workaround and options	658
11 Optimization	661
11.1 Optimization	661
11.1.1 Debugging MapServer	661
Introduction	661
Steps to Enable MapServer Debugging	662
Debugging MapServer using Compiler Debugging Tools	669
Debugging Older Versions of MapServer (before 5.0)	671
11.1.2 FastCGI	672
Introduction	672
Obtaining the necessary software	672
mod_fcgid Configuration	673
Deprecated mod_fcgi Configuration	673
Common mod_fcgid/mod_fcgi Configuration	673
Common Problems	674
FastCGI on Win32	674
11.1.3 Mapfile	675
Introduction	676
11.1.4 Raster	678
Overviews	678
Tileindexes and Internal Tiling	678
Image formats	678
Remote WMS	679
11.1.5 Tile Indexes	679
Introduction	679
What is a tileindex and how do I make one?	679
Using the tileindex in your mapfile	680
Tileindexes may make your map faster	680
Tileindexes with tiles in different projections	680
11.1.6 Vector	681
Splitting your data	681
Shapefiles	681
PostGIS	682
Databases in General (PostGIS, Oracle, MySQL)	682
12 Utilities	683
12.1 Utilities	683
12.1.1 legend	683
Purpose	683
Syntax	683
12.1.2 msencrypt	683
Purpose	683
Syntax	683
Use in Mapfile	684
12.1.3 scalebar	685
Purpose	685
Syntax	685
12.1.4 shp2img	685
Purpose	685
Syntax	685

12.1.5	shptree	686
	Purpose	686
	Description	687
	Syntax	687
	Mapfile Notes	687
12.1.6	shptreetst	688
	Purpose	688
	Syntax	688
12.1.7	shptreevis	689
	Purpose	689
	Syntax	689
12.1.8	sortshp	690
12.1.9	sym2img	691
	Purpose	691
	Syntax	691
12.1.10	tile4ms	691
	Purpose	691
	Description	692
	Syntax	692
	Short Example	692
	Long Example	692
12.1.11	Batch Scripting	695
	Windows	695
	Linux	695
12.1.12	File Management	695
	File Placement	695
	Temporary Files	695
13	CGI	697
13.1	CGI	697
13.1.1	MapServer CGI Introduction	697
	Notes	697
	Changes	697
13.1.2	mapserv	698
13.1.3	Map Context Files	698
	Support for Local Map Context Files	698
	Support for Context Files Accessed Through a URL	698
	Default Map File	699
13.1.4	MapServer CGI Controls	699
	Variables	699
	Changing map file parameters via a form or a URL	702
	Specifying the location of mapfiles using an Apache variable	704
	ROSA-Applet Controls	704
13.1.5	Run-time Substitution	704
	Introduction	704
	Basic Example	705
	Parameters Supported	705
	Default values if not provided in the URL	706
	VALIDATION	706
	Magic values	707
13.1.6	A Simple CGI Wrapper Script	707
	Introduction	707
	Script Information	707
13.1.7	MapServer OpenLayers Viewer	708

Using the OpenLayers viewer	709
14 Environment Variables	711
14.1 Environment Variables	711
15 Glossary	715
15.1 Glossary	715
16 Errors	719
16.1 Errors	719
16.1.1 drawEPP(): EPPL7 support is not available	719
Explanation	719
16.1.2 loadLayer(): Unknown identifier. Maximum number of classes reached	719
16.1.3 loadMapInternal(): Given map extent is invalid	720
How to get a file's EXTENT values?	720
16.1.4 msGetLabelSize(): Requested font not found	720
16.1.5 msLoadFontset(): Error opening fontset	721
16.1.6 msLoadMap(): Failed to open map file	721
16.1.7 msProcessProjection(): no options found in 'init' file	721
16.1.8 msProcessProjection(): No such file or directory	721
Setting the location of the epsg file	722
16.1.9 msProcessProjection(): Projection library error.major axis or radius = 0 not given	722
Valid Examples	722
16.1.10 msQueryByPoint: search returned no results	722
16.1.11 msReturnPage(): Web application error. Malformed template name	723
16.1.12 msSaveImageGD(): Unable to access file	723
16.1.13 msWMSLoadGetMapParams(): WMS server error. Image Size out of range, WIDTH and HEIGHT must be between 1 and 2048 pixels	724
16.1.14 Unable to load dll (MapScript)	724
C#-specific information	724
17 FAQ	725
17.1 FAQ	725
17.1.1 Where is the MapServer log file?	725
17.1.2 What books are available about MapServer?	725
17.1.3 How do I compile MapServer for Windows?	725
17.1.4 What do MapServer version numbers mean?	725
17.1.5 Is MapServer Thread-safe?	726
17.1.6 What does STATUS mean in a LAYER?	727
17.1.7 How can I make my maps run faster?	727
17.1.8 What does Polyline mean in MapServer?	727
17.1.9 What is MapScript?	727
17.1.10 Does MapServer support reverse geocoding?	728
17.1.11 Does MapServer support geocoding?	728
17.1.12 How do I set line width in my maps?	728
17.1.13 Why do my JPEG input images look crappy via MapServer?	728
17.1.14 Which image format should I use?	729
17.1.15 Why doesn't PIL (Python Imaging Library) open my PNGs?	729
17.1.16 Why do my symbols look poor in JPEG output?	730
17.1.17 How do I add a copyright notice on the corner of my map?	730
Example Layer	730
Result	731
17.1.18 How do I have a polygon that has both a fill and an outline with a width?	731
17.1.19 How can I create simple antialiased line features?	731
17.1.20 Which OGC Specifications does MapServer support?	732

17.1.21	Why does my requested WMS layer not align correctly?	732
17.1.22	When I do a GetCapabilities, why does my browser want to download mapserv.exe/mapserv?	733
17.1.23	Why do my WMS GetMap requests return exception using MapServer 5.0?	734
17.1.24	Using MapServer 6.0, why don't my layers show up in GetCapabilities responses or are not found anymore?	735
17.1.25	Where do I find my EPSG code?	735
17.1.26	How can I reproject my data using ogr2ogr?	735
17.1.27	How can I help improve the documentation on this site?	736
17.1.28	What's with MapServer's logo?	736
18	Copyright	737
18.1	License	737
18.2	Credits	737

Note: The entire documentation is also available as a single PDF document  and ePub document [previous versions](#).
If you are upgrading from an earlier version of MapServer, be sure to review the MapServer Migration Guide.

Introduction

1.1 An Introduction to MapServer

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

Last Updated 2014-03-27

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?*

1.1.1 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.

1.1.2 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 Shape format. 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 it's 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:

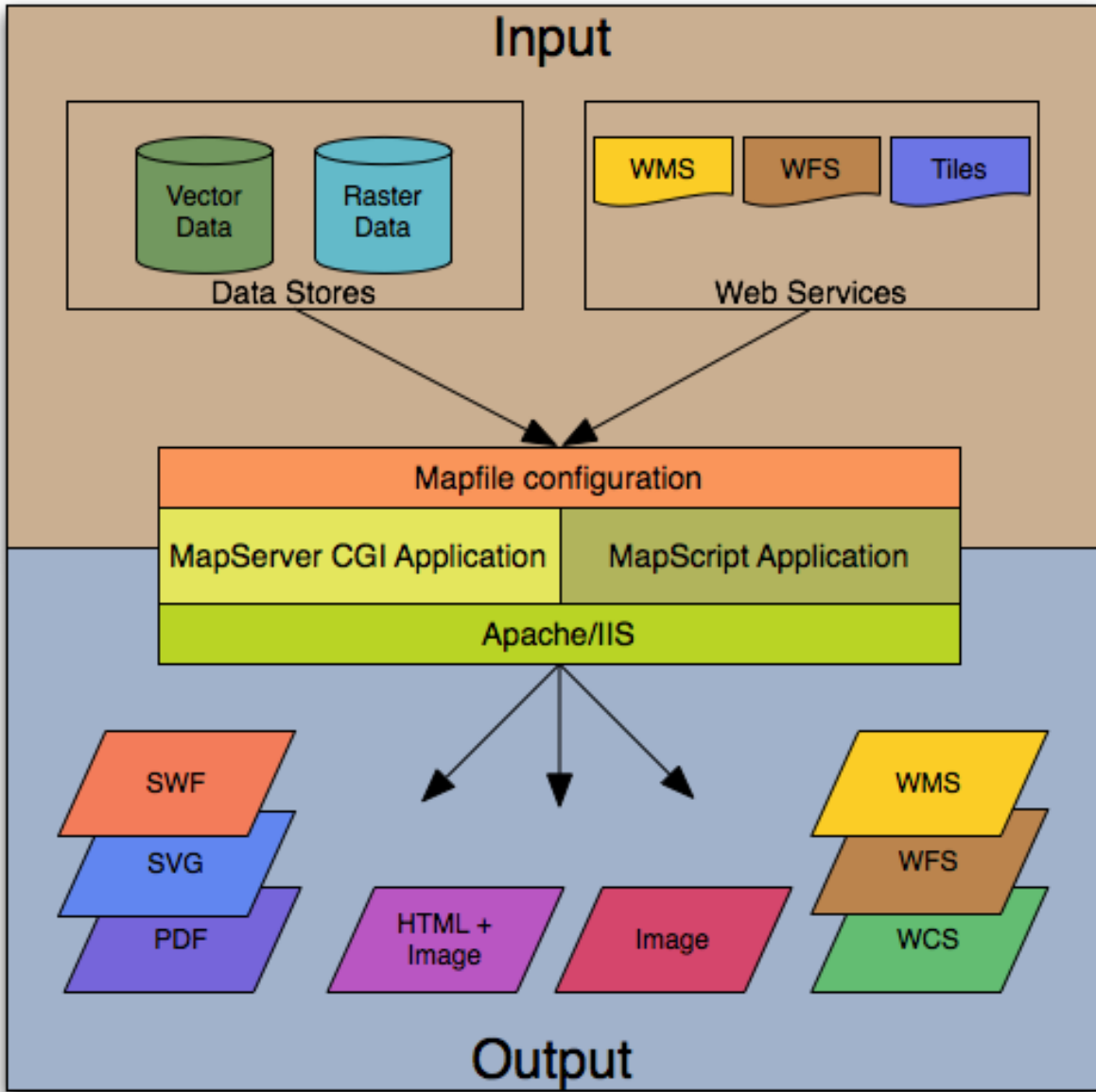


Fig. 1.1: The basic architecture of MapServer applications.

- **Initialization File** - uses a form with hidden variables to send an initial query to the web server and MapServer. This form could be placed on another page or be replaced by passing the initialization information as variables in a URL.
- **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 web 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*
- **Web/HTTP Server** - serves up the HTML pages when hit by the user's browser. You need a working Web (HTTP) server, such as *Apache* or Microsoft Internet Information Server, on the machine on which you are installing MapServer.

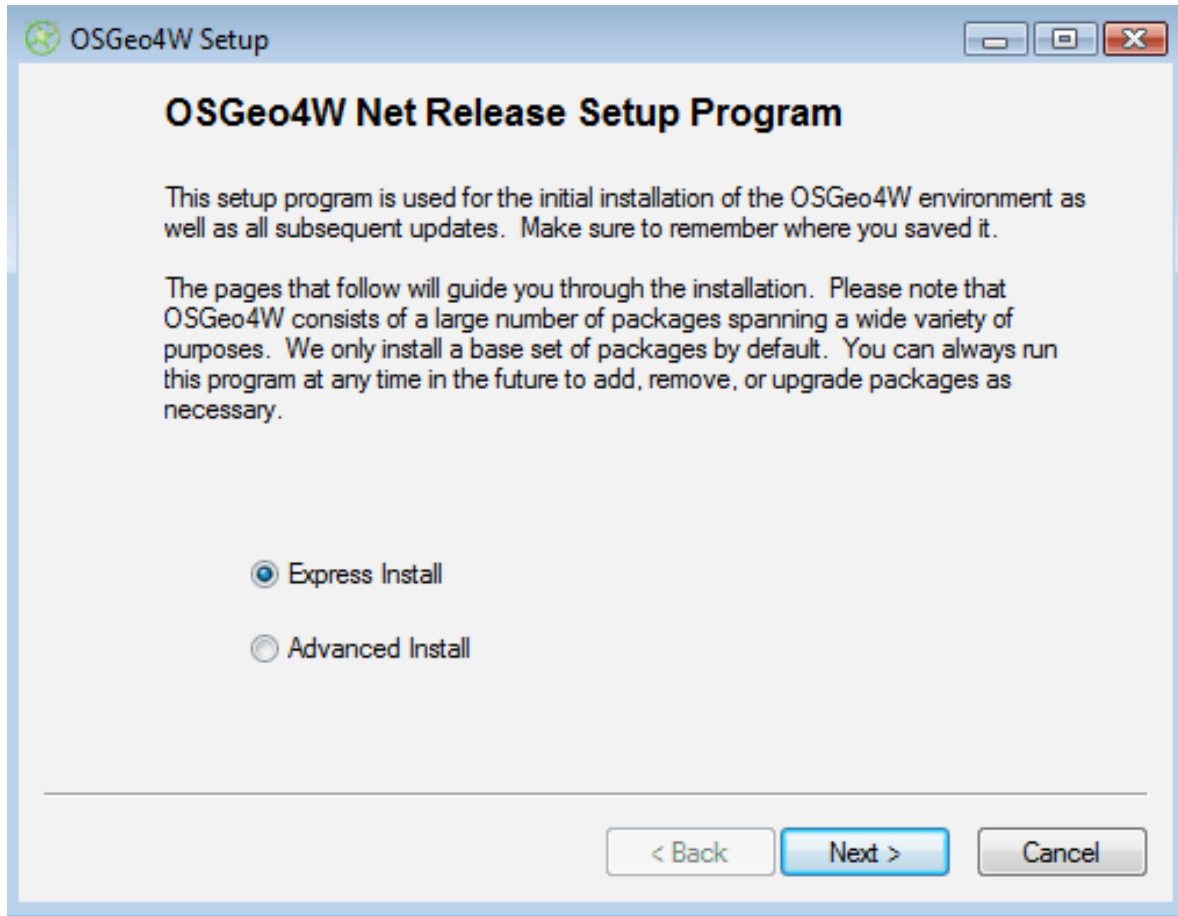
1.1.3 Installation and Requirements

Windows Installation

Note: Pre-compiled binaries for MapServer are available from a variety of sources, refer to the windows section of the Downloads page.

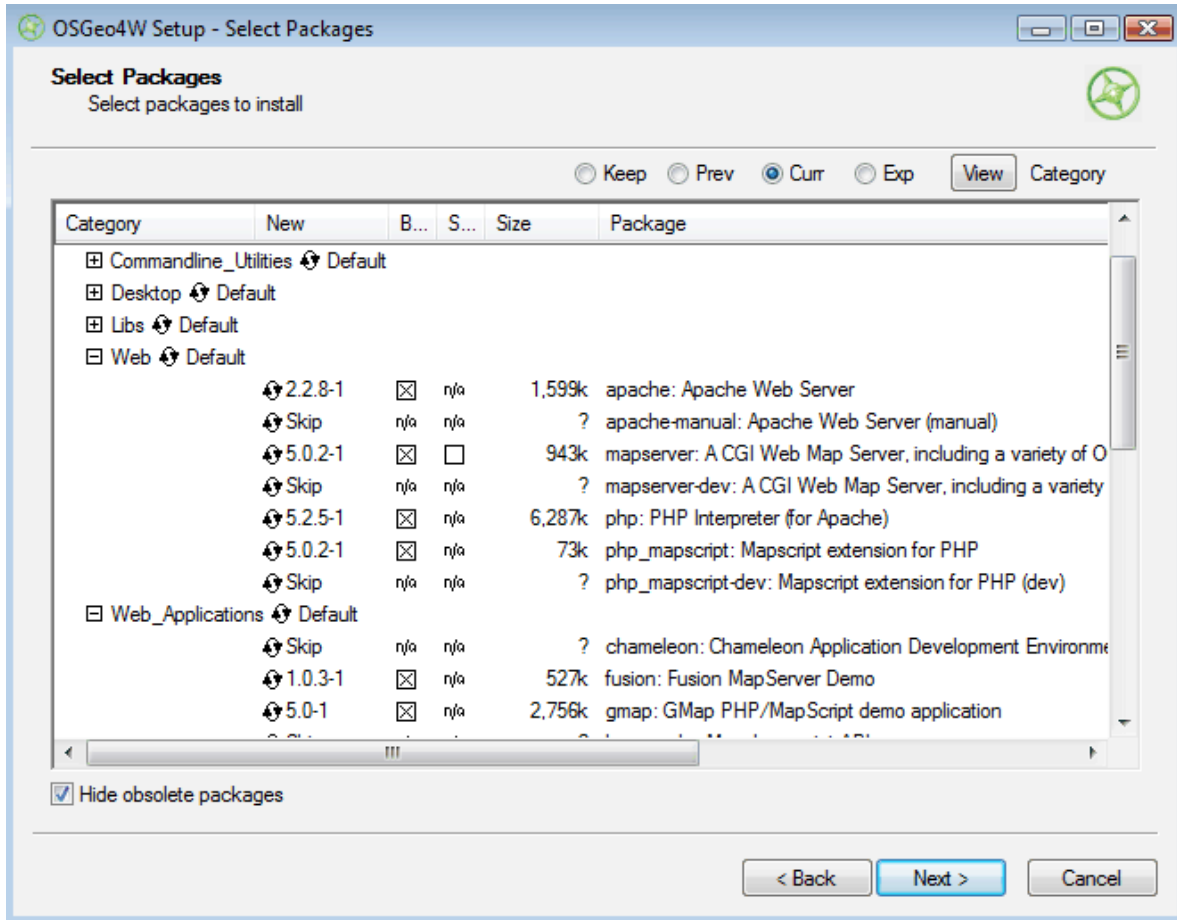
OSGeo4W is a new Windows installer that downloads and/or updates MapServer, add-on applications, and also other Open Source geospatial software. The following steps illustrate how to use OSGeo4W:

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



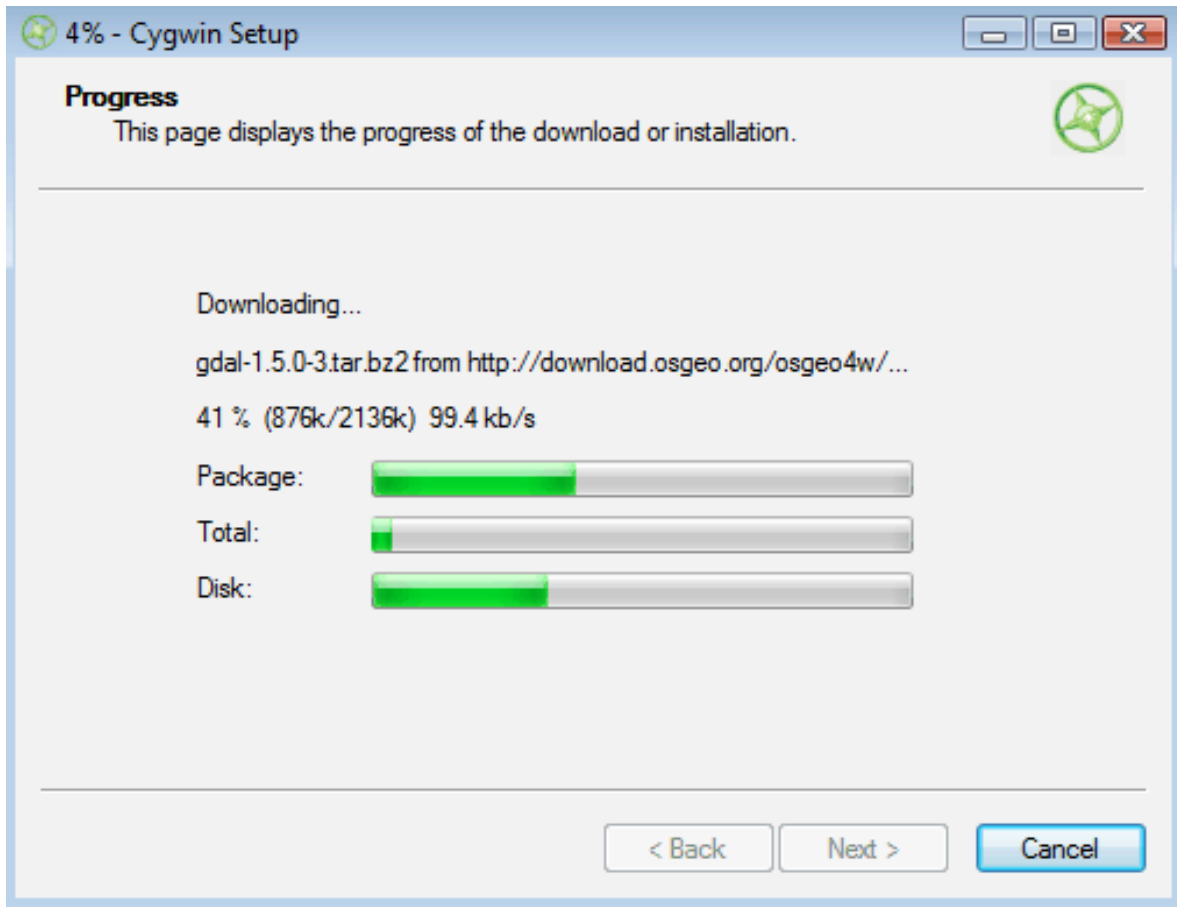
Note: Express contains options for higher-level packages such as MapServer, GRASS, and uDig. Advanced gives you full access to choosing commandline tools and applications for MapServer that are not included in the Express install

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

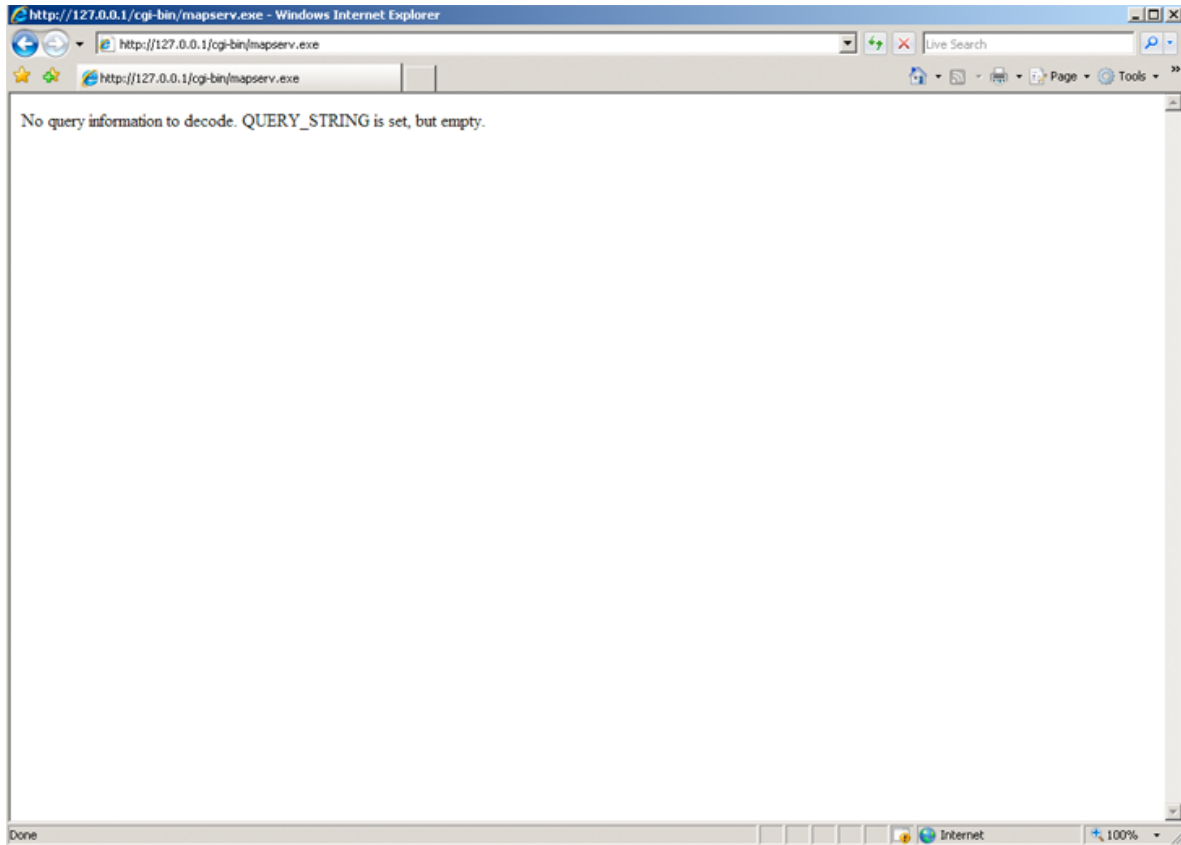
Note: An `apache-uninstall.bat` script is also available to remove the Apache service installation.

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

```
apache-restart.bat
```



8. Verify that MapServer is working



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.

Software Requirements

You need a working and properly configured Web (HTTP) 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 web server installed, you may want to check out [MS4W](#), which will install a pre-configured web 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*.

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.

1.1.4 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:

```
MAP
  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 # WEB

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

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)
- The above example is built on the following directory structure:

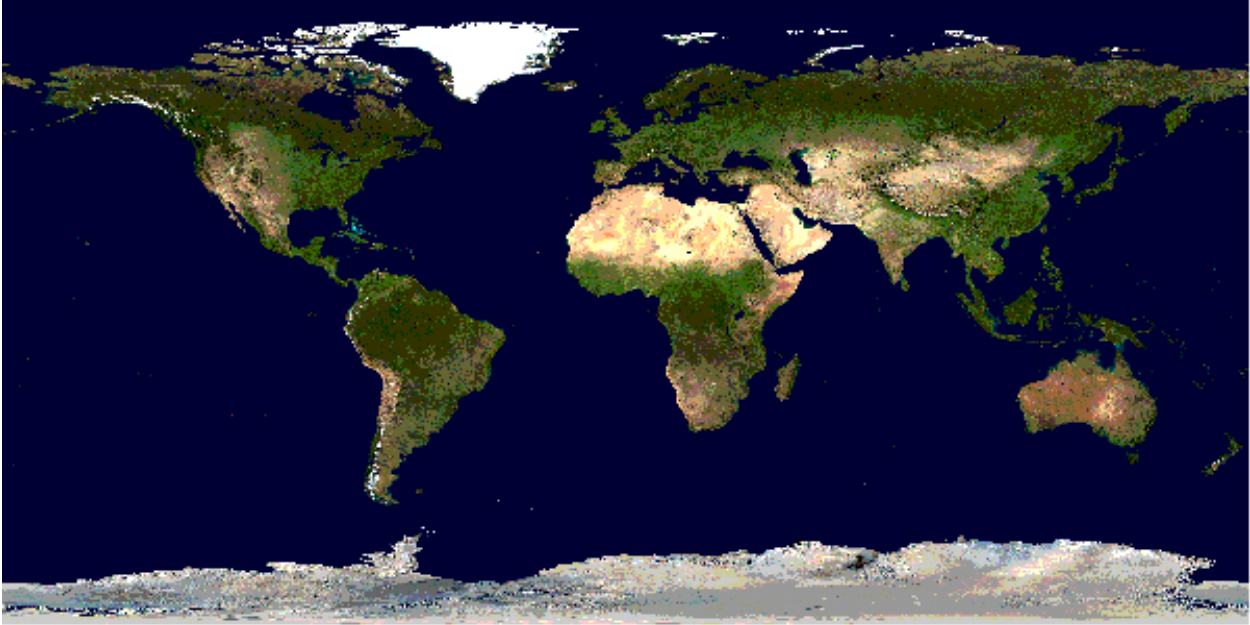


Fig. 1.2: Rendered Bluemarble Image

- The mapfile could be placed anywhere, as long as it is accessible to the web server. Normally, one would try to avoid placing it at a location that makes it accessible on the web. Let us say it is placed in `/home/msuser/mapfiles/`
- The location of the font file is given relative to the map file, in this case: `/home/msuser/etc/fonts.txt`
- The location of the datasets (*blumarble.gif*) is given relative to the map file, in this case: `/home/msuser/data/`
- The location of the symbol file is given relative to the map file, in this case: `/home/msuser/etc/symbols.txt`
- The files generated by MapServer will be placed in the directory `/ms4w/tmp/ms_tmp/`. The web server must be able to place files in this directory. The web server must make this directory available as `/ms_tmp` (if the web server is on `www.ms.org`, the web address to the directory must be: `http://www.ms.org/ms_tmp/`).

MAP Object

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

- 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

Note: MapServer currently uses a pixel-center based extent model which is a bit different from what GDAL or WMS use.

LAYER Object

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

Raster Layers

```
LAYER
  NAME "bathymetry"
  TYPE RASTER
  STATUS DEFAULT
  DATA "bath_mapserver.tif"
END # LAYER
```

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 # STYLE
  END # CLASS
END # LAYER
```

See also:

Vector Data

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"
```

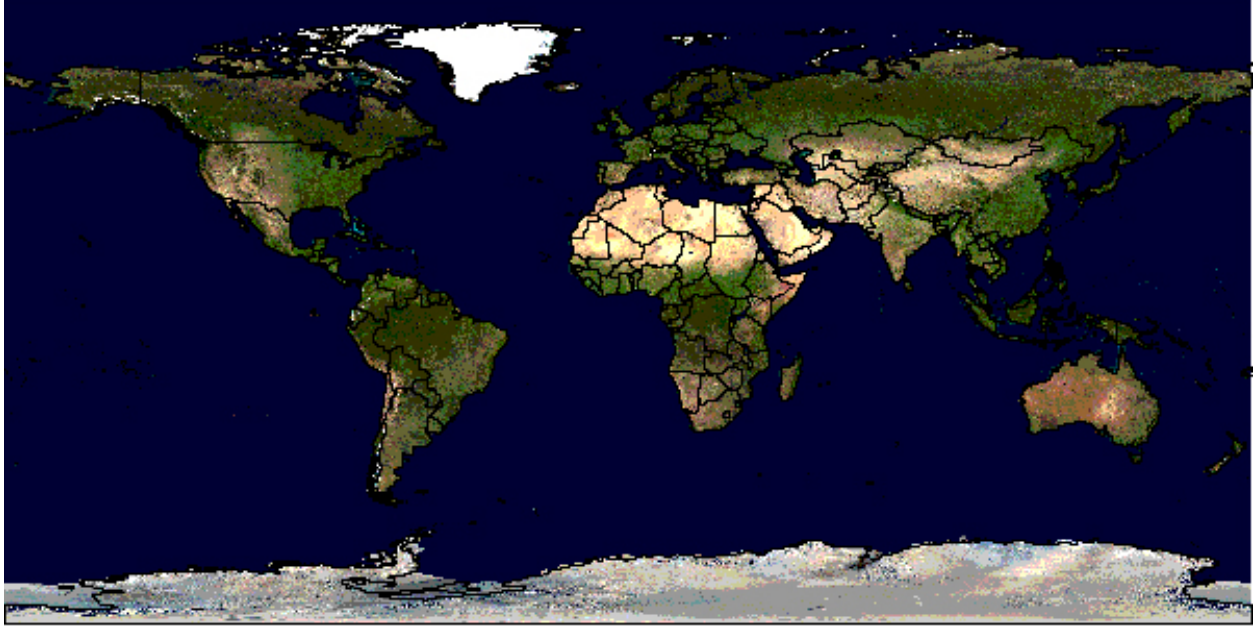


Fig. 1.3: Rendered Bluemarble image with vector boundaries

```

COLOR 178 114 1
SIZE 15
END # STYLE
STYLE
  SYMBOL "circle"
  COLOR 254 161 0
  SIZE 7
END # STYLE
END # CLASS

```

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 # MAP

```

where symbols.txt might contain:

```

SYMBOL
  NAME "ski"
  TYPE PIXMAP
  IMAGE "ski.png"
END # SYMBOL

```

and the mapfile would contain:

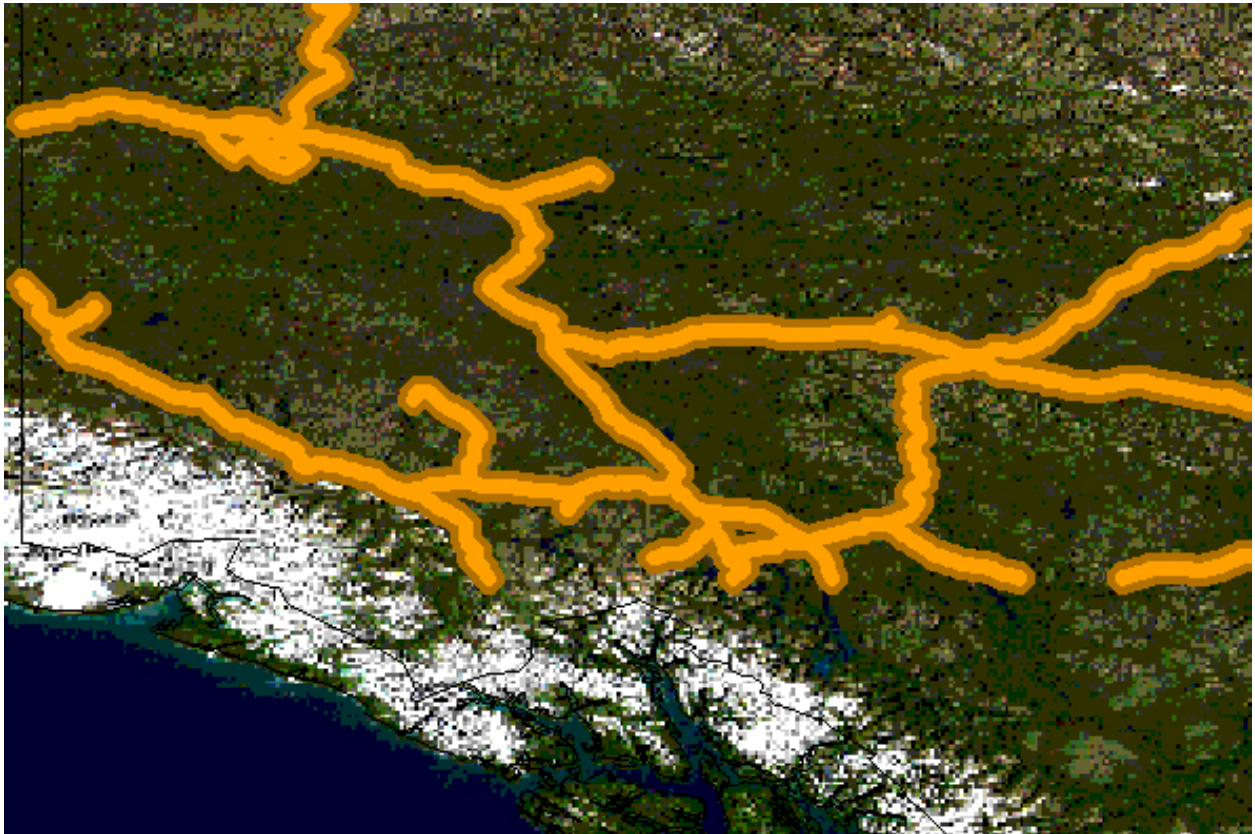


Fig. 1.4: Rendered Bluemarble image with styled roads


```

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

```

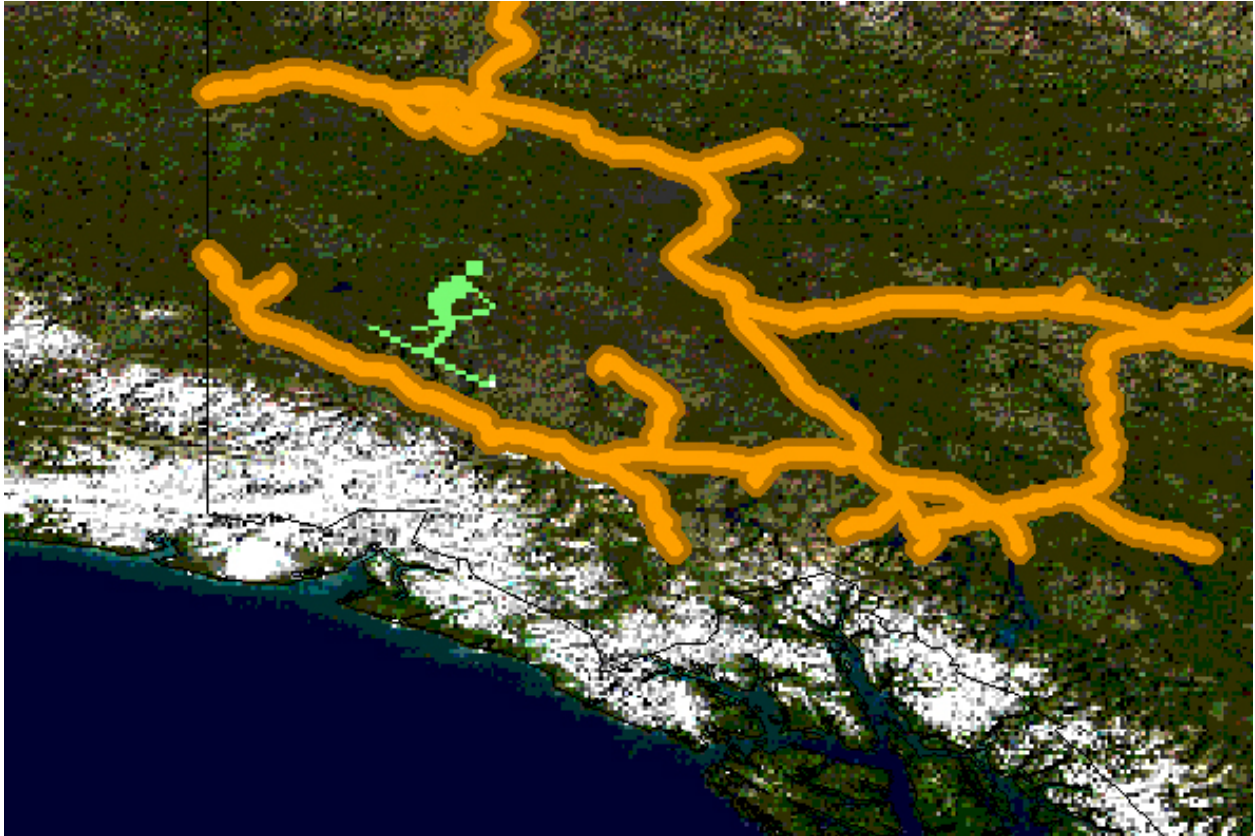


Fig. 1.5: Rendered Bluemarble image with skier symbol

See also:

Cartographical Symbol Construction with MapServer, Symbology Examples, and SYMBOL

LABEL

- defined within a *CLASS* object
- the *LABELITEM* parameters in the *LAYER* object can be used to specify an attribute in the data to be used for labeling. The label is displayed by the *FONT*, declared in the *FONTSET* file (set in the *MAP* object). The *FONTSET* file contains references to the available font names. *ENCODING* describes which encoding is used in the file (see *Display of International Characters in MapServer*).

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

```
LABEL
  FONT "sans-bold"
  TYPE truetype
  ENCODING "UTF-8"
  SIZE 10
  POSITION LC
  PARTIALS FALSE
  COLOR 100 100 100
  OUTLINECOLOR 242 236 230
END # LABEL
```

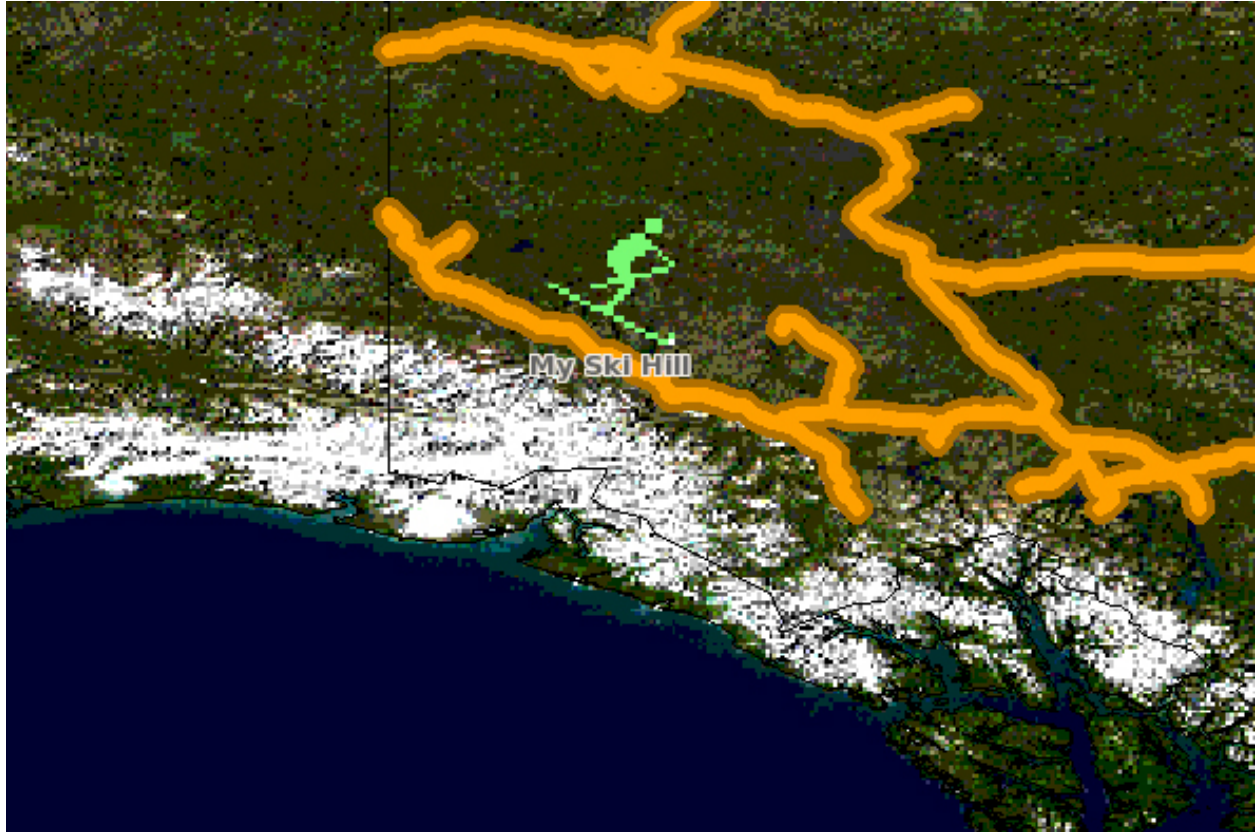


Fig. 1.6: Rendered Bluemarble image with skier symbol and a label

See also:

LABEL, *FONTSET*

CLASS Expressions

MapServer supports three types of *CLASS Expressions* in a *LAYER* (*CLASSITEM* in *LAYER* determines the attribute to be used for the two first types of expressions):

1. String comparisons

```
EXPRESSION "africa"
```

2. Regular expressions

```
EXPRESSION /^9|^10/
```

3. Logical expressions

```
EXPRESSION ([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

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 # LAYER
```

therefore the main mapfile would contain:

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

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:

```
MAP
  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
```

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

See also:

INCLUDE

Get MapServer Running

You can test if MapServer is working by running the MapServer executable (`mapserv`) with the `-v` parameter on the command line (`./mapserv -v`). Depending on your configuration, the output could be something like this:

```
MapServer version 6.0.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG
SUPPORTS=PROJ SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=ICONV
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.

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 web 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.

1.1.5 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 with different geographical reference systems, you will need to modify your Map File to add a *PROJECTION* block to the *MAP* (defines the output projection / geographical reference system) and each of the *LAYERS* (defines the geographical reference system of the layer dataset).

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*.

Vector Data

Vector data includes features made up of points, lines, and polygons. MapServer support the ESRI Shape format by default, but it can be compiled to support spatially enabled databases such as [PostgreSQL-PostGIS](#), and file formats such as [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.

Raster Data

Raster data is image or grid data. Through [GDAL](#), Mapserver supports most raster formats - see the [GDAL format list](#). More specific information can be found in the [Raster Data reference](#).

Note: Since version 6.2 Mapserver relies on GDAL for all raster access.

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](#).

1.1.6 Enhancing your site

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](#).

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 Shape dataset 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.

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](#) documentation.

Interfaces

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

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 Shape datasets 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 when searching through a large Shape dataset 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. Shape datasets can be broken into smaller tiles and then a tileindex Shape dataset can be created using the *tile4ms* utility. A *tileindex* Shape dataset 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

1.1.7 How do I get Help?

Documentation

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

Users Mailing List

Register and post questions to the [MapServer Users](#) mailing list. 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.

IRC

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

Reporting bugs

Bugs (software and documentation) are reported on the [MapServer issue tracker](#).

Tutorial

[Here](#) is a quick tutorial for new users.

Test Suite

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

Books

[Web Mapping Illustrated](#), a 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. 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.

2.1 MapServer Tutorial

Author Pericles S. Nacionales

Contact pnaciona at gmail.com

Author Jeff McKenna

Contact jmkenna at gatewaygeomatics.com

Updated 2010-04-07

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.

2.1.1 Tutorial background

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.

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 (and all html files) for this tutorial at <http://download.osgeo.org/mapserver/docs/mapserver-tutorial.zip>.

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 their 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 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, precompiled binaries exist for a number of platform (see the download page).

Windows, UNIX/Linux Issues

Paths

This tutorial was created on 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/articles/slashforward/> for a few insights on URLs.

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".

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](#)

2.1.2 Section 1: Static Maps and the MapFile

- Take a Shapefile dataset. Any Shapefile dataset. We can use MapServer to display that Shapefile dataset in a web browser. Look...
 - Example 1.1 - A map with a single layer
 - We can display the same Shapefile dataset repeatedly. We can display the polygon attributes in one LAYER and the line attributes in another...
 - Example 1.2 - A map with two layers
 - And we can select which parts of the Shapefile dataset 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 (for example 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 mode and browse mode
-

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

2.1.3 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.

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 in 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 MapServer CGI variables can be found [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.

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](#) | [Proceed to Section 3](#)

2.1.4 Section 3: Query and more about HTML Templates

To learn more about query and HTML templates with MapServer, see examples 3.1 to 3.4 in the [Tutorial Viewer](#).

[Back to Section 2 index](#) | [Proceed to Section 4](#)

2.1.5 Section 4: Advanced User Interfaces

To learn more about advanced navigation such as pan and rubber-band zoom with Javascript and MapServer CGI, see examples 4.1 to 4.4 in the [Tutorial Viewer](#).

[Back to Section 3 index](#) | [Tutorial home](#)

[Begin tutorial](#)

Installation

3.1 Installation

3.1.1 Compiling on Unix

Author J.F. Doyon

Contact jdoyon at nrcan.gc.ca

Author Howard Butler

Contact hobu.inc at gmail.com

Author Thomas Bonfort

Contact thomas.bonfort at gmail.com

Date 2014/03

Table of Contents

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

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.

Detailed configuration options are maintained in the [INSTALL.CMAKE](#) file packaged at the root of the source directory:

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

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 git.

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.
- **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.

Warning: Direct JPEG support is deprecated in MapServer 5.8+, and you should now depend on GDAL for raster read support in MapServer. JPEG support is however still required for producing (i.e. writing) images.

- **zlib**: Zlib should be on your system by default. 1.2.1 is the current release with security patches. Though not used directly by mapserver, it's a mandatory dependency of libpng.

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. Installing libcurl:

```
apt-get install -y libcurl4-gnutls-dev
```

- **OGR**: OGR provides access to at least 18 different vector formats.
- **GDAL**: GDAL provides access to at least 42 different raster formats.

Optional External Libraries

- **libtiff**: libtiff provides TIFF (Tagged Image File Format) reading support to MapServer.

Warning: Direct libtiff support is deprecated in MapServer 5.8+, and you should now depend on GDAL for raster read support in MapServer.

- **libgeotiff**: libgeotiff provides support to read GeoTIFF files (TIFF files with geographic referencing).

Warning: Direct GeoTIFF support is deprecated in MapServer 5.8+, and you should now depend on GDAL for raster read support in MapServer.

- **GEOS**: GEOS allows MapServer to do spatial predicate and algebra operations (within, touches, etc & union, difference, intersection).

New in version 4.10.

- **libxml:** libxml is required to use *OGC SOS* support in MapServer
New in version 4.10.
- **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.
- **‘libgif-dev’_:** giflib is used for reading GIF files used as PIXMAP symbols.
- **FastCGI:** FastCGI is a popular protocol for interfacing MapServer with various web servers. You will need to install the development package. More details on how to use this feature in MapServer is here *FastCGI*. On Ubuntu, that would be:

```
$ apt-get -y install libfcgi-dev
```

- **cairo (svg, pdf) support:** This library is required to produce PDF and SVG outputs. If you’re on an ubuntu system, it can be installed with “apt-get install -y libcairo2-dev”
- **kml support:** This renderer is has no external dependency.
- **‘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).

Optional Features

- **cairo svg parser support:** The WITH_SVGCAIRO option is part of a proposal to improve SVG support. Using this feature requires installing the libsvg-cairo library available here: <http://cairographics.org/snapshots/> . You will need to compile and install cairo, libsvg, and libsvg-cairo.
- SVG support can be enabled either through the unmaintained libsvg / libsvg-cairo combo, or through libsvg at the cost of more dependencies. Use libsvg if your distro provides a package for it, or fall back to libsvgcairo if the cost of compiling the libsvg dependencies is too important.

libgd

Warning: It is not recommended to enable the GD renderer unless you know what you are doing. It suffers from major rendering limitations, and will be completely removed in version 7.0

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:

```
$ gdlb-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 [GD on 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.

Anti-Grain Geometry Support

Since version 5.0 MapServer supports the AGG rendering backend. MapServer 5.6+ embeds it directly in the source tree and you do not have to do anything special to have support for it.

OGC Support

MapServer provides support for many OGC specifications. For an overview, see [MapServer OGC Specification support](#).

WMS support

WMS Server Support for *WMS server* is automatically enabled.

You can check it by looking for the following in your configure output:

```
-- * WMS SERVER: ENABLED
```

If, for some reason you don't want WMS support, you can force it off using "-DWITH_WMS=OFF".

More information on using this feature is available in [WMS Server](#).

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_CLIENT_WMS option to the configure script. MapServer will automatically look for libcurl, which is also required.

To verify that this feature is enabled, check the configure output for:

```
-- * WMS CLIENT: ENABLED
```

Note: This feature is disabled by default, you have to specifically request it.

More information on using this feature is available in [WMS Client](#).

WFS support

WFS Server Support for *WFS server* is enabled by default. OGR and PROJ.4 support is required.

To verify that this feature is enabled, check the configure output for:

```
-- * WFS SERVER: ENABLED
```

If, for some reason you don't want WFS support, you can force it off using "--DWITH_WFS=OFF".

More information on using this feature is available in *WFS Server*.

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 have OGR (built with Xerces support) and PROJ.4 support, and pass the WITH_CLIENT_WFS option to your configure script. MapServer will automatically look for libcurl, which is also required.

To verify that this feature is enabled, check the configure output for:

```
-- * WFS CLIENT: ENABLED
```

Note: This feature is disabled by default, you have to specifically request it.

More information on using this feature is available in *WFS Client*.

WCS Server Support for *WCS server* is enabled by default. WCS must be compiled against certain libraries. More information on this service is available in *WCS Server*.

To verify that this feature is enabled, check the configure output for:

```
-- * WCS SERVER: ENABLED
```

If, for some reason you don't want WCS support, you can force it off using "--DWITH_WCS=OFF".

SOS Server Support for SOS is enabled by using the WITH_SOS option. More information on this service is available in *SOS Server*.

To verify that this feature is enabled, check the configure output for:

```
-- * SOS SERVER: ENABLED
```

Note: This feature is disabled by default, you have to specifically request it.

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. PostGIS support is enabled by default.

To verify that this feature is enabled, check the configure output for:

```
-- * POSTGIS: /usr/local/pgsql/lib/libpq.so
```

If, for some reason you don’t want PostGIS support, you can force it off using “-DWITH_POSTGIS=OFF”. To help cmake find your PostGIS installation, you can use the CMAKE_PREFIX_PATH option (for instance “-DCMAKE_PREFIX_PATH=/usr/local/pgsql”).

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 use the following options: WITH_SDE (“-dWITH_SDE=ON”) and SDE_VERSION (for example “-DSDE_VERSION=91”).

To verify that this feature is enabled, check the configure output for:

```
-- * SDE: /opt/arcsde/lib/libsde90.so
```

Oracle Spatial

Oracle’s Spatial is also supported by MapServer. In order to connect to it, you will need to compile MapServer against the Oracle libraries by using the WITH_ORACLESPATIAL option. You will very likely need an ORACLE_HOME environment variable set to have it configure things correctly.

To verify that this feature is enabled, check the configure output for:

```
-- * Oracle Spatial: <path to oracle spatial shared library>
```

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. This library is only used along with other features, so “--with-curl-config” won’t do anything unless something like “--with-wmsclient” or “--with-wfsclient” is also selected.

Note: If you want to configure MapServer to use SSL when accessing a WMS/WFS server libcurl must be configured / compiled with the “--with-ssl” option. Details about how to set this up is available in *How to set up MapServer as a client to access a service over https*.

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:

```
$ tar -zxvf mapserver-X.Y.Z.tar.gz
```

7. Create the build directory and configure your environment.

Create the build directory:

```
$ cd mapserver-X.Y.Z
$ mkdir build
$ cd build
```

Configure your environment using “cmake” (this is an example):

```
$ cmake -DCMAKE_INSTALL_PREFIX=/opt \
-DMAKE_PREFIX_PATH=/usr/local/pgsql/91:/usr/local:/opt \
-DWITH_CLIENT_WFS=ON \
-DWITH_CLIENT_WMS=ON \
-DWITH_CURL=ON \
-DWITH_SOS=ON \
-DWITH_PHP=ON \
-DWITH_PYTHON=ON \
-DWITH_SVGCAIRO=ON \
-DWITH_ORACLESPATIAL=ON \
-DWITH_MSSQL2008=ON \
-DWITH_SDE=ON \
-DSDE_VERSION=91 .. >../configure.out.txt
```

The following options are enabled by default (version 6.4): WITH_CAIRO, WITH_FCGI, WITH_FRIBIDI, WITH_GDAL, WITH_GEOS, WITH_GIF, WITH_ICONV, WITH_LIBXML2, WITH_OGR, WITH_POSTGIS, WITH_PROJ, WITH_WCS, WITH_WFS, WITH_WMS.

If you want to also build a static version of the library, the BUILD_STATIC and LINK_STATIC_LIBMAPSERVER options can be used,

There are a number of other options available. For an up-to-date list of available cmake options, refer to the CMakeLists.txt.

It can be a good idea to place the configuration commands in a file and change its mode to executable (+x) to save typing and have a record of how MapServer was configured.

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.

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

```
$ make
```

9. Install the Mapserver libraries:

```
# make install
```

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

```
$ ls -al mapserv
-rwxr-xr-x 1 user user 13745 mars 11 17:45 mapserv
```

A simple test is to try and run it:

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

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

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/apache2`. 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.

If you don’t have apache installed, and you want apache, php, fastcgi, etc, that might look something like this:

```
$ apt-get install -y apache2 apache2-mpm-worker libapache2-mod-fastcgi
$ a2enmod actions fastcgi alias
$ apt-get install libapache2-mod-php5 php5-common php5-cli php5-fpm php5
```

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

1. Locate your `cgi-bin` directory. Under a default configuration, the CGI directory is “`/usr/local/apache2/cgi-bin`” (RedHat: “`/home/httpd/cgi-bin`”, Debian: “`/usr/lib/cgi-bin`”). If you’re using apache, there should be a `ScriptAlias` directive in your `http.conf`, or default site, something like:

```
$ cat /etc/apache2/sites-available/default | grep 'cgi-bin'
ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
```

2. Locate the installation path of your freshly compiled `mapserv` executable. This is shown when you run “make install”, something like this:

```
-- Installing: /usr/local/bin/mapserv
-- Set runtime path of "/usr/local/bin/mapserv" to
   "/usr/local/lib:/usr/local/pgsql/91/lib"
```

3. You’ll want to setup a symlink to that executable from your `cgi-bin` directory:

```
# ln -s /usr/local/bin/mapserv /usr/lib/cgi-bin/mapserv
```

Warning: Make sure you are linking against the installed `mapserv` file (after running ‘make install’) and NOT against where it was compiled in your source tree.

Testing your new Install 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:

```
$ 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. The apache docs also have pointers on [setting up cgi-bin](#).

```
:: Check your server logs $ tail /var/log/apache2/error.log
```

Where to go once you’ve got it compiled

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

3.1.2 Compiling on Win32

Author Pericles Nacionales

Contact pnaciona at gmail.com

Revision \$Revision\$

Date \$Date\$

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*

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.

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.

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.

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.

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 [git](#); however, the [GitHub](#) 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 [git](#).

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.gzip.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.

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.

2. 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
```

3. 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 `makemsvimport.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 Libary
#
# 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.

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
```

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.

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.

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-0.12regex.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).

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.

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.

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.

3.1.3 PHP MapScript Installation

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision\$

Date \$Date\$

Table of Contents

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

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 API* 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.

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.

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.

3.1.4 .NET MapScript Compilation

Author Tamas Szekeres

Contact `szekerest at gmail.com`

Revision \$Revision\$

Date \$Date\$

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 `mapscript_csharp.dll.config` is needed along with the `mapscript_csharp.dll` file. This file is created during the make process

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.

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"
  xmlns:asmv1="urn:schemas-microsoft-com:asm.v1"
  xmlns:asmv2="urn:schemas-microsoft-com:asm.v2"
  xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
  xmlns="urn:schemas-microsoft-com:asm.v1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<assemblyIdentity name="drawmap.exe" version="1.0.0.0" type="win32" />
<dependency>
<dependentAssembly asmv2:dependencyType="install"
  asmv2:codebase="Microsoft.VC80.CRT.manifest" asmv2:size="522">
<assemblyIdentity name="Microsoft.VC80.CRT" version="8.0.50608.0"
  publicKeyToken="1fc8b3b9a1e18e3b" processorArchitecture="x86"
  type="win32" />
<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>UM0lhUBGeKRrrg9DaaPNgyhRjyM=</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'
  publicKeyToken='1fc8b3b9a1e18e3b' />
</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 mapsript.dll.manifest /outputresource:mapsript.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 (mapsript.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 (mapsript_csharp.dll) and the C code (mapsript.dll) takes place using platform invoke like:

```
[DllImport("mapsript", 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 (mapsript.dll). On the Unix systems the library file name normally starts with the lib prefix and appended with the .so extension (libmapsript.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 (mapsript_csharp.dll.config):

```
<configuration>
  <dllmap dll="mapsript" target="libmapsript.so" />
</configuration>
```

and with the OSX builds:

```
<configuration>
  <dllmap dll="mapsript" target="libmapsript.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
```

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 `mapscript.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.

Bug reports

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

3.1.5 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.

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.

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)

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 and 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

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”.

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\phpdlls and I copied it to the cgi-bin directory.

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"
```

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
```

3.1.6 Oracle Installation

Author Till Adams

Last Updated 2007/02/16

Table of Contents

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

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

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.

Compile MapServer

Compile as normal compilation and set this flag:

```
--with-oraclespatial=/path/to/oracle/home/</p>
```

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.

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.

4.1 Mapfile

Author Steve Lime

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

Author Jeff McKenna

Contact jmkenna 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.

The Mapfile consists of a *MAP* object, which has to start with the word *MAP*.

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”

4.1.1 Cartographical Symbol Construction with MapServer

Author Peter Freimuth

Contact pf at mapmedia.de

Author Arnulf Christl

Contact arnulf.christl at wheregroup.com

Author Håvard Tveite

Contact havard.tveite at nmbu.no

Table of Contents

- *Abstract*
- *Introduction*
 - *Multiple Rendering and Overlay*
 - *Symbol Scaling*
 - *MapServer and symbol specification*
- *Using Cartographical Symbols in MapServer*
 - *Output formats*
 - *Symbol units*
 - *Scaling of Symbols*
- *Construction of Point Symbols*
 - *Symbols of TYPE vector and ellipse*
 - *Symbols of TYPE truetype*
 - *Symbols of TYPE pixmap*
 - *Symbol definitions for the figure that demonstrates point symbols*
 - *Combining symbols*
- *Construction of Line Symbols*
 - *Overlaying lines*
 - *Use of the PATTERN and GAP parameters*
 - * *LINECAP*
 - * *LINEJOIN*
 - * *LINEJOINMAXSIZE (only relevant for LINEJOIN miter)*
 - *Use of the OFFSET parameter*
 - *Asymmetrical line styling with point symbols*
- *Area Symbols*
 - *Hatch fill*
 - *Polygon fills with symbols of TYPE pixmap*
 - *Polygon fills with symbols of TYPE vector*
 - * *Excerpts from the map file for the polygon fill vector examples above*
 - *Polygon outlines*
- *Examples (MapServer 4)*
 - *Basic Symbols*
 - *Complex Symbols*
- *Tricks*
 - *Changing the center of a point symbol*
- *Mapfile changes related to symbols*
 - *Version 6.2*
 - *Version 6.0*
- *Current Problems / Open Issues*
 - *GAP - PATTERN incompatibility*
- *The End*

Abstract

This Document refers to the syntax of *map* and *symbol* files for MapServer 6. The first version of the document was based on 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.

The document has been heavily revised for MapServer 6.

Introduction

A map is an abstract representation that makes use of point, line and area symbols. Bertin (1974) created a clear and logical symbol scheme in which symbols can be varied referring to graphical variables. The following graphical variables can be used within MapServer: FORM, SIZE, PATTERN, COLOR and LIGHTNESS. Point and area symbols as well as text fonts (ttf) can additionally be displayed with a frame which we call OUTLINE.

The following figure shows the theoretical structure of cartographical symbols, which is also used in MapServer:

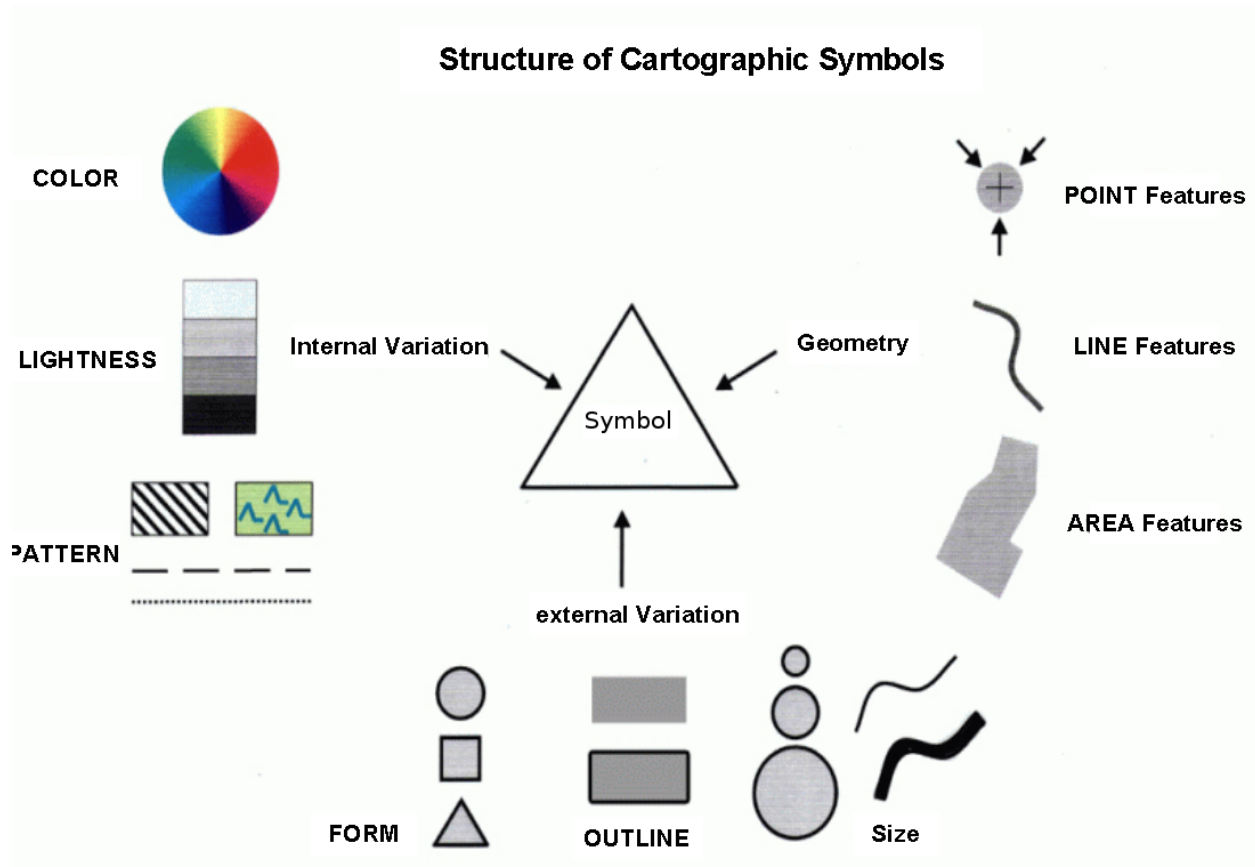


Fig. 4.1: Structure of Cartographical Symbols⁴

Multiple Rendering and Overlay

Say you want to display a highway with a black border line, two yellow lanes and a red center lane. This calls for a combination of signatures.

Complex cartographical effects can be achieved by rendering the same vector data with different symbols, sizes and colours on top of each other. This can be done using separate *LAYERS*. This could, however, have performance effects for the application, as every rendering process of the same geometry will take up additional processor time. The preferred solution is to use multiple *STYLES* to create complex symbols by overlay.

To create the highway symbol mentioned above with a total width of 9 units, the lowest *STYLE* (in drawing order) will be a broad black line with a width of 9 units. The second level *STYLE* will be a yellow line with a width of 7 units,

and the topmost *STYLE* will be a red line with a width of 1 unit. That way each yellow coloured lane will have a width of $(7-1)/2 = 3$ units.

Combining symbols can be a solution for many kinds of cartographical questions. A combination of different geometry types is also possible. A polygon data set can be rendered as lines to frame the polygons with a line signature. It can also be rendered as polygons with a symbol filling the polygon. When the polygon fill is rendered on top of the lines, the inner part of the underlying outline is covered by the fill symbol of the polygon. What is observed here is a clipping effect that will result in an asymmetric symbol for the boundary line. To present the outline without clipping, just reorder the *LAYERS* or *STYLES* and put the outline presentation on top of the fill.

Yet another way to construct advanced line signatures for framed polygons 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. PostGIS can help achieve a lot of effects.

The *OPACITY* parameter of *LAYER* and *STYLE* can be used to achieve transparency (making lower symbols “shine” through upper symbols).

Symbol Scaling

There are two basically different ways of handling the display size of symbols and cartographical elements in a map at different scales. The size of cartographical elements is either set in screen pixels or in real world units.

- If the size is set in real world units (for example meters), the symbol will shrink and grow according to the scale at which the map is displayed.
- If the size is set in screen pixels, symbols look the same at all scales.

The default behaviour of MapServer is to implement the “screen pixels” size type for displaying cartographical elements.

“Real world units”, as described above, can be achieved using either the *SIZEUNITS* or the *SYMBOLSCALEDENOM* parameter of the *LAYER*.

- When *SIZEUNITS* is set (and is not *pixels*), symbol sizes are specified in real world units (for instance meters). For available units, see the *SIZEUNIT* documentation.
- When *SYMBOLSCALEDENOM* is set, the given symbols size is used for the map scale 1:*SYMBOLSCALEDENOM*, for other scales, the symbols are scaled proportionally.

STYLE MAXSIZE and *MINSIZE* limits the scaling of symbols.

MapServer and symbol specification

In a MapServer application, *SYMBOL* parameters are organised in the *map* file as follows:

- Each *LAYER* has a *TYPE* parameter that defines the type of geometry (point, line or polygon). The symbols are rendered at points, along lines or over areas accordingly.
- Basic symbols are defined in *SYMBOL* elements, using the parameters *TYPE*, *POINTS*, *IMAGE*, *FILLED*, *ANCHORPOINT* and more (*SYMBOL* elements can be collected in separate *symbol files* for reuse).
- Colour, lightness, size and outline are defined inside the *STYLE* sections of a *CLASS* section using the parameters *COLOR*, *SIZE*, *WIDTH* and *OUTLINECOLOR*.
- Patterns for styling lines and polygons are defined in *STYLE* sections using *GAP* and *PATTERN*.
- Several basic elements can be combined to achieve a complex signature using several *STYLES* inside one *CLASS*.

The following example shows the interaction of some of these elements and explains the configuration in the *LAYER* and the *SYMBOL* sections necessary for rendering a cartographical point symbol (a red square with a 1 pixel wide black outline and a smaller blue circle inside):



Fig. 4.2: The generated overlay symbol

Table 4.1: Commented LAYER and SYMBOL sections.

<i>LAYER</i> section of the map file	<i>SYMBOL</i> (from a separate symbol file or in-line in the map file)
<pre> # Start of layer definition LAYER # Name of the layer NAME "mytest" TYPE POINT # Point geometries STATUS DEFAULT # Always draw # Use the dataset test.shp DATA test # Start of a Class definition CLASS # Start of the first Style STYLE # Symbol to be used (reference) SYMBOL "square" # Size of the symbol in pixels SIZE 16 # Colour (RGB) - red COLOR 255 0 0 # Outline colour (RGB) - black OUTLINECOLOR 0 0 0 END # end of STYLE # Start of the second Style STYLE # Symbol to be used (reference) SYMBOL "circle" # Size of the symbol in pixels SIZE 10 # Colour (RGB) - blue COLOR 0 0 255 END # end of STYLE END # end of CLASS END # end of LAYER </pre>	<pre> # Start of symbol definition SYMBOL # Symbol name (referenced in STYLES) NAME "square" TYPE vector # Type of symbol # Start of the symbol geometry POINTS 0 0 0 1 1 1 1 0 0 0 END # end of POINTS # The symbol should be filled FILLED true # Place the according to its center ANCHORPOINT 0.5 0.5 END # end of SYMBOL # Start of symbol definition SYMBOL # Symbol name (referenced in STYLES) NAME "circle" TYPE ellipse # Type of symbol # Start of the symbol geometry POINTS 1 1 END # end of POINTS # The symbol should be filled FILLED true # Place the according to its center ANCHORPOINT 0.5 0.5 END # end of SYMBOL </pre>

Using Cartographical Symbols in MapServer

Vectors, truetype fonts and raster images are basic graphical elements that are defined by the *TYPE* parameter in the *STYLE* element. This and the following sections explain how these elements can be combined to create complex cartographical symbols, and they describes some other important aspects of map rendering in MapServer .

Output formats

MapServer support raster output formats (e.g. PNG, JPEG and GIF) and vector output formats (e.g. PDF, SVG). The raster formats (except for GIF) use anti-aliasing. See *OUTPUTFORMAT* (and *MAP IMAGETYPE*) for more.

Symbol units

The units used for specifying dimensions is defined in the *SIZEUNITS* parameter of the *LAYER*. The available units are listed there. The default unit is *pixels*.

The *MAP* element's *RESOLUTION* and *DEFRESOLUTION* parameters will determine the resolution of the resulting map and hence the size in pixels of the symbols on the map. *DEFRESOLUTION* is by default 72 dpi (dots per inch). If *RESOLUTION* is set to 144 (and *DEFRESOLUTION* is 72), all dimensions specified in the map file will be multiplied by $144/72 = 2$. This can be used to produce higher resolution images.

Dimensions can be specified using decimals.

Scaling of Symbols

The *SYMBOLSCALEDENOM* parameter in the *LAYER* section specifies the scale at which the symbol or text label is displayed in exactly the dimensions defined in the *STYLE*s (for instance using *SIZE* and *WIDTH*). Observe that all the parameters concerned with the symbol dimensions (*SIZE*, *WIDTH*, ...) are tightly connected to the *SYMBOLSCALEDENOM* parameter. The *MAXSIZE* and *MINSIZE* parameters inside the *STYLE* element limit the scaling of symbols to the maximum and minimum size specified here (but does not affect the size calculations).

When symbols are scaled as the scale changes, the elements (defined in *STYLE*s) of a composite cartographical symbol may change their positions relative to each other. This is due to rounding effects when creating the image. The effect is most noticeable at small scales (large scale denominators), when the symbols get small. Due to the same effects, symbols can also slightly change their shape when they get small.

It is not possible to define the display intervals with *MINSCALEDENOM* and *MAXCALEDENOM* in the *STYLE*-section, so this kind of tuning has to be solved at the *LAYER* level. To do this, create several *LAYER*s with the same geometries for different scale levels.

Always observe that cartographical symbols depend a lot on the scale! So be careful with the interaction of content, symbols and scale. All three parameters heavily interact and have to be coordinated to produce a good map.

Construction of Point Symbols

In the figure below, point symbols of *TYPE* *truetype*, *pixmap*, *ellipse* and *vector* are demonstrated. The precise position of the point for which the symbol is rendered is shown with a small red dot. A small blue dot is used to show an offset position.

All point symbols can be rotated using the *ANGLE* parameter.

Since version 6.2, the anchor point / reference point of all point symbols can be set using the *SYMBOL ANCHORPOINT* parameter. The default anchorpoint is at the center of the boundingbox of the symbol (*ANCHORPOINT* 0.5 0.5).

Symbols of *TYPE* *vector* and *ellipse*

For symbols of *TYPE* *vector* and *ellipse* the shape of the symbol by setting X and Y values in a local two dimensional coordinate system with X values increasing to the right and Y values increasing downwards. The coordinates defining

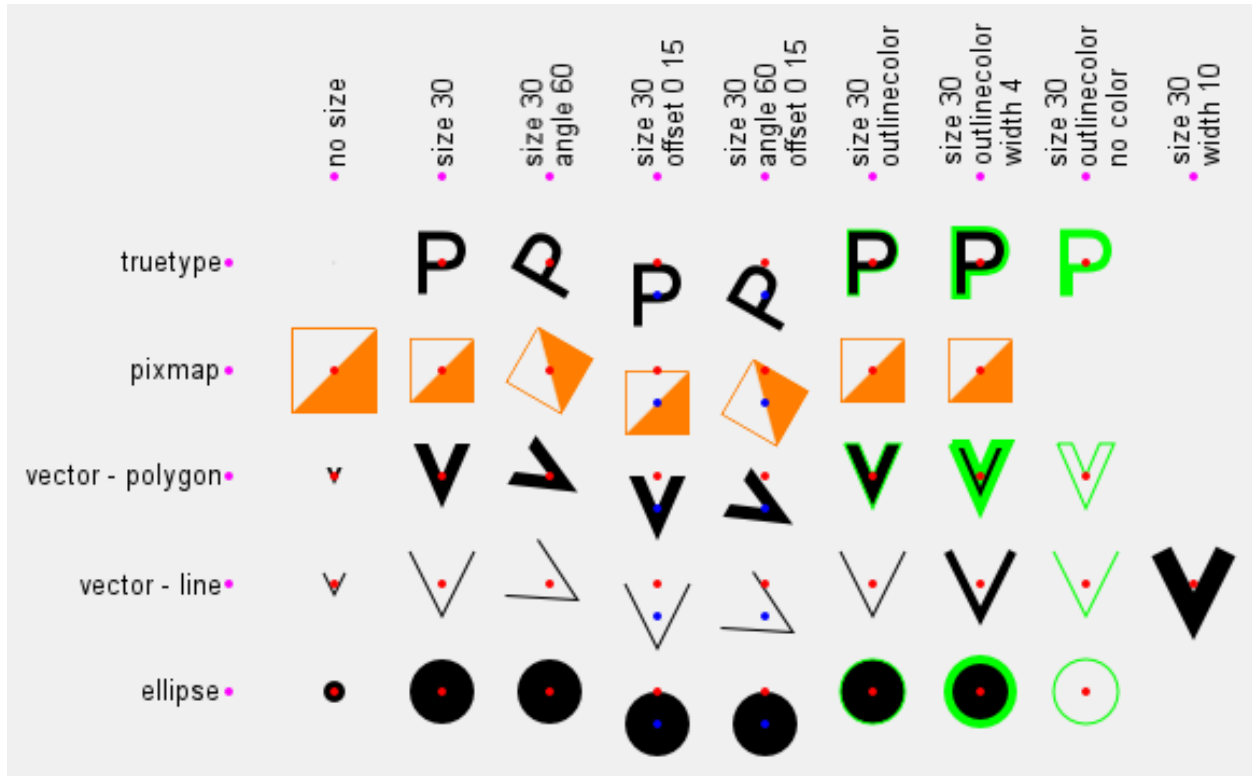


Fig. 4.3: Basic point symbol *TYPES*, showing effects of size, offset, angle and outlinecolor

the symbol is listed in the *POINTS* parameter, which is explicitly ended using *END*. Negative values should not be used.

- *TYPE ellipse* is used to create ellipses (and circles). The shape of the ellipse is defined in the *POINTS* parameter (*X* - size in the horizontal direction, *Y* - size in the vertical direction). To create a circle, let *X* and *Y* have the same value.
- *TYPE vector* is used to define advanced vector symbols. The shape of the symbol is defined in the *POINTS* parameter. A *vector* symbol can consist of several elements. The coordinates -99 -99 are used to separate the elements.

To create a polygon vector symbol, the *SYMBOL FILLED* parameter must be set to *true*. If the end point is not equal to the start point of a polygon geometry, it will be closed automatically.

The maximum number of points is 100, but this can be increased by changing the parameter *MS_MAXVECTORPOINTS* in the file *mapsymbols.h* before compilation.

When creating symbols of *TYPE vector* you should observe some style guidelines.

- Avoid downtilted lines in area symbols, as they will lead to heavy aliasing effects.
- Do not go below a useful minimum size. This is relevant for all types of symbols.
- Keep in mind that for pixel images, every symbol of *TYPE vector* has to be rendered using pixels.

Note: The bounding box of a vector symbol has (0,0) in the symbol coordinate system as its upper left corner. This can be used to precisely control symbol placement. Since version 6.2 *SYMBOL ANCHORPOINT* should be used instead.

Symbols of *TYPE truetype*

You can use symbols from truetype fonts. The symbol settings are defined in the *SYMBOL* element. Specify the character or the ASCII number of the character to be used in the *CHARACTER* parameter. The *FONT* parameter is used to specify the font to be used (the alias name from the font file - often “fonts.list”). The *FONTSET* parameter of the *MAP* element must be set for fonts to work.

For gif output (GD renderer), you can define that you want to apply antialiasing to the characters by using the parameter *ANTI_ALIAS*. 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.

Colours for *truetype* symbols can be specified in *LAYER CLASS STYLE* (as with symbols of the *TYPE vector* and *ellipse*). You can specify both fill colour and outline colour.

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 $61440 + 84 = \&\#61524$. (ain’t that a pain!!)

You can also place truetype characters and strings on the map using *LABEL*. Then you can control the placing of the text by using the *POSITION* parameter [ulluclurllclclrlllllcllr], that specifies the position relative to the geometric origin of the geometry.

Symbols of *TYPE pixmap*

Symbols of the *TYPE pixmap* are simply small raster images. The file name of the raster image is specified in the *IMAGE* parameter of the *SYMBOL* element. MapServer supports the raster formats GIF and PNG for *pixmap*s.

Observe the colour depth of the images and avoid using 24 bit PNG symbols displayed in 8 bit mode as this may cause unexpected colour leaps.

When using raster images, the colour cannot be modified in the *SYMBOL* element subsequently.

You can specify a colour with the *TRANSPARENT* parameter which will not be displayed - i.e. it will be transparent. As a result, underlying objects and colours are visible.

The *SIZE* parameter defines the height of *pixmap* symbols when rendered. The pixel structure will show when the *SIZE* grows too large. If you are using symbol scaling (*LAYER SYMBOLSCALEDENOM* is set or *LAYER SIZEUNITS* is not *pixels*) and want to prevent this from happening, you should set the *STYLE MAXSIZE* parameter.

Symbol definitions for the figure that demonstrates point symbols

This code was used to produce the symbols in the point symbol figure.

First, the symbol definitions:

```
SYMBOL
  NAME "o-flag-trans"
  TYPE pixmap
  IMAGE "o-flag-trans.png"
END # SYMBOL

SYMBOL
```



```

NAME "circlef"
TYPE ellipse
FILLED true
POINTS
  10 10
END # POINTS
END # SYMBOL

SYMBOL
NAME "P"
TYPE truetype
FONT "arial"
CHARACTER "P"
END # SYMBOL

SYMBOL
NAME "v-line"
TYPE vector
FILLED false
POINTS
  0 0
  5 10
  10 0
END # POINTS
END # SYMBOL

SYMBOL
NAME "v-poly"
TYPE vector
FILLED true
POINTS
  0 0
  3.5 8
  7 0
  5.2 0
  3.5 4
  1.8 0
  0 0
END # POINTS
END # SYMBOL

```

Then, the *LAYERs* and *STYLEs* used for producing the polygon V symbols in the point symbol figure:

```

LAYER # Vector v - polygon
STATUS DEFAULT
TYPE POINT
FEATURE
  POINTS
    10 30
  END # Points
END # Feature
CLASS
  STYLE
    SYMBOL "v-poly"
    COLOR 0 0 0
  END # STYLE
  STYLE
    SYMBOL "circlef"
    COLOR 255 0 0

```

```
        SIZE 4
    END # STYLE
END # CLASS
END # LAYER

LAYER # Vector v - polygon, size
STATUS DEFAULT
TYPE POINT
FEATURE
POINTS
    20 30
END # Points
END # Feature
CLASS
STYLE
    SYMBOL "v-poly"
    COLOR 0 0 0
    SIZE 30
END # STYLE
STYLE
    SYMBOL "circlef"
    COLOR 255 0 0
    SIZE 4
END # STYLE
END # CLASS
END # LAYER

LAYER # Vector v - polygon, size, angle
STATUS DEFAULT
TYPE POINT
FEATURE
POINTS
    30 30
END # Points
END # Feature
CLASS
STYLE
    SYMBOL "v-poly"
    COLOR 0 0 0
    SIZE 30
    ANGLE 60
END # STYLE
STYLE
    SYMBOL "circlef"
    COLOR 255 0 0
    SIZE 4
END # STYLE
END # CLASS
END # LAYER

LAYER # Vector v - polygon, size, offset
STATUS DEFAULT
TYPE POINT
FEATURE
POINTS
    40 30
END # Points
END # Feature
```

```

CLASS
  STYLE
    SYMBOL "v-poly"
    COLOR 0 0 0
    SIZE 30
    OFFSET 0 15
  END # STYLE
  STYLE
    SYMBOL "circlef"
    COLOR 255 0 0
    SIZE 4
  END # STYLE
END # CLASS
END # LAYER

LAYER # Vector v - polygon, size, angle, offset
STATUS DEFAULT
TYPE POINT
FEATURE
  POINTS
    50 30
  END # Points
END # Feature
CLASS
  STYLE
    SYMBOL "v-poly"
    COLOR 0 0 0
    SIZE 30
    ANGLE 60
    OFFSET 0 15
  END # STYLE
  STYLE
    SYMBOL "circlef"
    COLOR 255 0 0
    SIZE 4
  END # STYLE
END # CLASS
END # LAYER

LAYER # Vector v - polygon, size outline
STATUS DEFAULT
TYPE POINT
FEATURE
  POINTS
    60 30
  END # Points
END # Feature
CLASS
  STYLE
    SYMBOL "v-poly"
    COLOR 0 0 0
    SIZE 30
    OUTLINECOLOR 0 255 0
  END # STYLE
  STYLE
    SYMBOL "circlef"
    COLOR 255 0 0
    SIZE 4
  END # STYLE
END # CLASS
END # LAYER

```

```
    END # STYLE
  END # CLASS
END # LAYER

LAYER # Vector v - polygon, size, outline, width
  STATUS DEFAULT
  TYPE POINT
  FEATURE
    POINTS
      70 30
    END # Points
  END # Feature
  CLASS
    STYLE
      SYMBOL "v-poly"
      COLOR 0 0 0
      SIZE 30
      OUTLINECOLOR 0 255 0
      WIDTH 4
    END # STYLE
    STYLE
      SYMBOL "circlef"
      COLOR 255 0 0
      SIZE 4
    END # STYLE
  END # CLASS
END # LAYER

LAYER # Vector v - polygon, size, outline, no color
  STATUS DEFAULT
  TYPE POINT
  FEATURE
    POINTS
      80 30
    END # Points
  END # Feature
  CLASS
    STYLE
      SYMBOL "v-poly"
      SIZE 30
      OUTLINECOLOR 0 255 0
    END # STYLE
    STYLE
      SYMBOL "circlef"
      COLOR 255 0 0
      SIZE 4
    END # STYLE
  END # CLASS
END # LAYER
```

Combining symbols

The following figure shows how to combine several basic symbols to create a complex point symbol. The combination is achieved by adding several *STYLE*s within one *LAYER*. Each *STYLE* element references one *SYMBOL* element. All the basic symbols are centered and overlaid when rendered.

Notice that the *SIZE* parameter in the *STYLE* element refers to the height of the symbol (extent in the Y direction).

A standing rectangle will thus display with a smaller area than a lying rectangle, although both have the same *SIZE* parameter and the same maximum Y value in the *SYMBOL* element. When combining several basic point symbols on top of each other, they will not always be centered correctly due to the integer mathematics required when rendering raster images. It is recommended not to combine elements with even and odd numbered *SIZE* parameters, as this tends to produce larger irregularities.

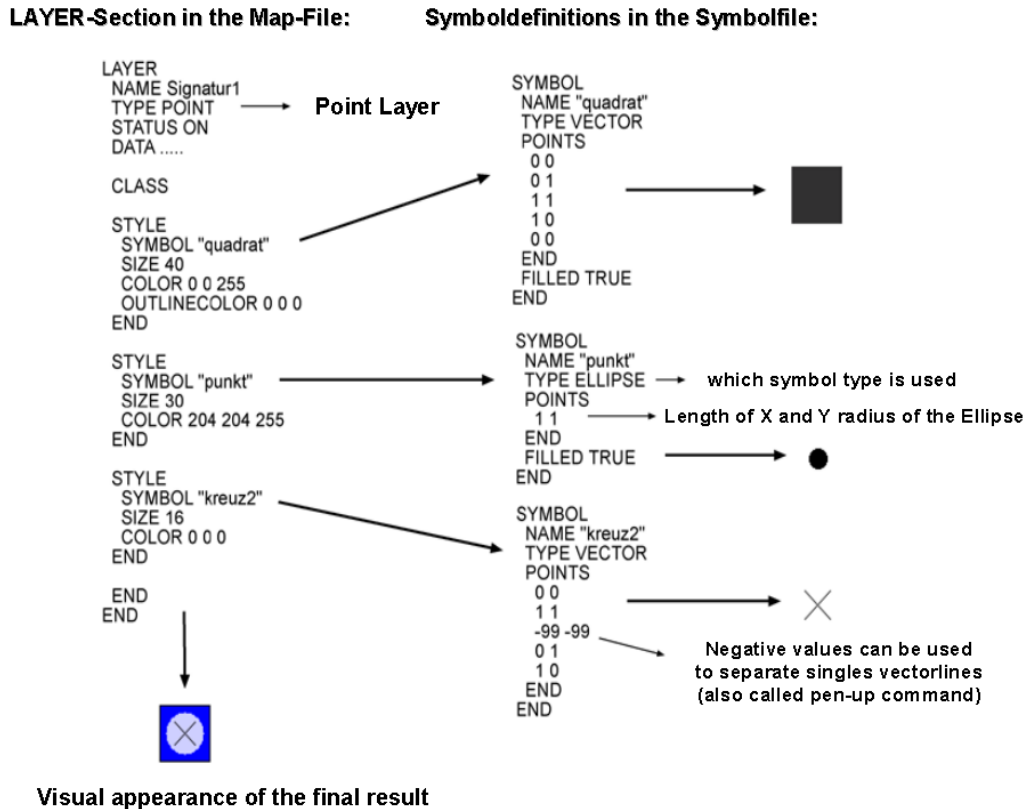


Fig. 4.4: Construction of Point Symbols

Construction of Line Symbols

For displaying line geometries, you specify the width of the lines using the *WIDTH* parameter and the colour using the *COLOR* parameter. If no colour is specified, the line will not be rendered. If no width is specified, a thin line (one unit (pixel) wide) will be rendered. The *LINECAP*, *LINEJOIN* and *LINEJOINMAXSIZE* parameters are used to specify how line ends and corners are to be rendered.

Overlaying lines

When combining several styles / symbols on a line, they will be positioned on the baseline which is defined by the geometry of the object. In most cases MapServer correctly centers symbols. The combination of a line displayed in 16 units width and overlaid with a 10 unit width line, results in a line symbol with a 3 unit border. If the cartographical symbol is to contain a centered line with a width of 1 pixel, then the widths should be reconfigured, for example to 11 and 17 units. As a rule of thumb don't combine even numbered and odd numbered widths.

Use of the *PATTERN* and *GAP* parameters

The *PATTERN* and *GAP* parameters can be used to produce styled lines in MapServer.

To create line patterns, use the *PATTERN* parameter of the *STYLE*. Here you define dashes by specifying the length of the first dash, followed by the length of the first gap, then the length of the second dash, followed by the second gap, and so on. This pattern will be repeated as many times as that pattern will fit into the line. *LINECAP* can be used to control how the ends of the dashes are rendered. *LINEJOIN* can be used to control how sharp bends are rendered. In the left column of the figure, you will find three examples where *PATTERN* has been used. Number 2 from below uses *LINECAP butt*, number 3 from below uses *LINECAP round* (and *LINEJOIN miter*) and number 4 from below uses *LINECAP butt* (and is overlaid over a wider, dark grey line). To produce dots, use 0 for dash length with *LINECAP 'round'*.

Styled lines can be specified using *GAP* and a symbol for styling. In the figure, you will find examples where *GAP* has been used (in the right column). At the bottom a *SYMBOL* of *TYPE ellipse* has been used, then a *SYMBOL* of *TYPE vector*, then a *SYMBOL* of *TYPE font* and then a *SYMBOL* of *TYPE pixmap*. To control the placement of the symbols relative to the line (to get asymmetrical styling), use *SYMBOL ANCHORPOINT* (as explained later).

Note: Since version 6.2 it is possible to specify an offset (start gap) when creating asymmetrical patterns using the *STYLE INITIALGAP* parameter. *INITIALGAP* can be used with *GAP* and with *PATTERN*.

The following figure shows how to use styles to define different kinds of line symbols.

- *PATTERN* usage is demonstrated in the 2nd, 3rd, 4th and 5th symbol from the bottom in the left column.
- *GAP* usage is demonstrated in the 2nd symbol from the bottom in the left column and all the symbols in the right column.
- negative *GAP* value usage is demonstrated in the all the symbols in the right column, except for the one at the bottom.
- *INITIALGAP* usage is demonstrated in the 2nd and 5th symbol from the bottom in the left column.
- *STYLE OFFSET* usage is demonstrated in the 5th symbol from the bottom in the right column

Below you will find the *SYMBOLs* and *STYLEs* that were used to produce the line symbols in “Construction of Line Symbols”. The *LAYERS* are ordered from bottom to top of the figure.

Styles and symbols for lines

```

SYMBOL
  NAME "circlef"
  TYPE ellipse
  FILLED true
  POINTS
    1 1
  END # POINTS
END # SYMBOL

SYMBOL
  NAME "p"
  TYPE truetype
  FONT "arial"
  CHARACTER "p"
END # SYMBOL

SYMBOL
  NAME "vertline"
  TYPE vector
  FILLED true

```

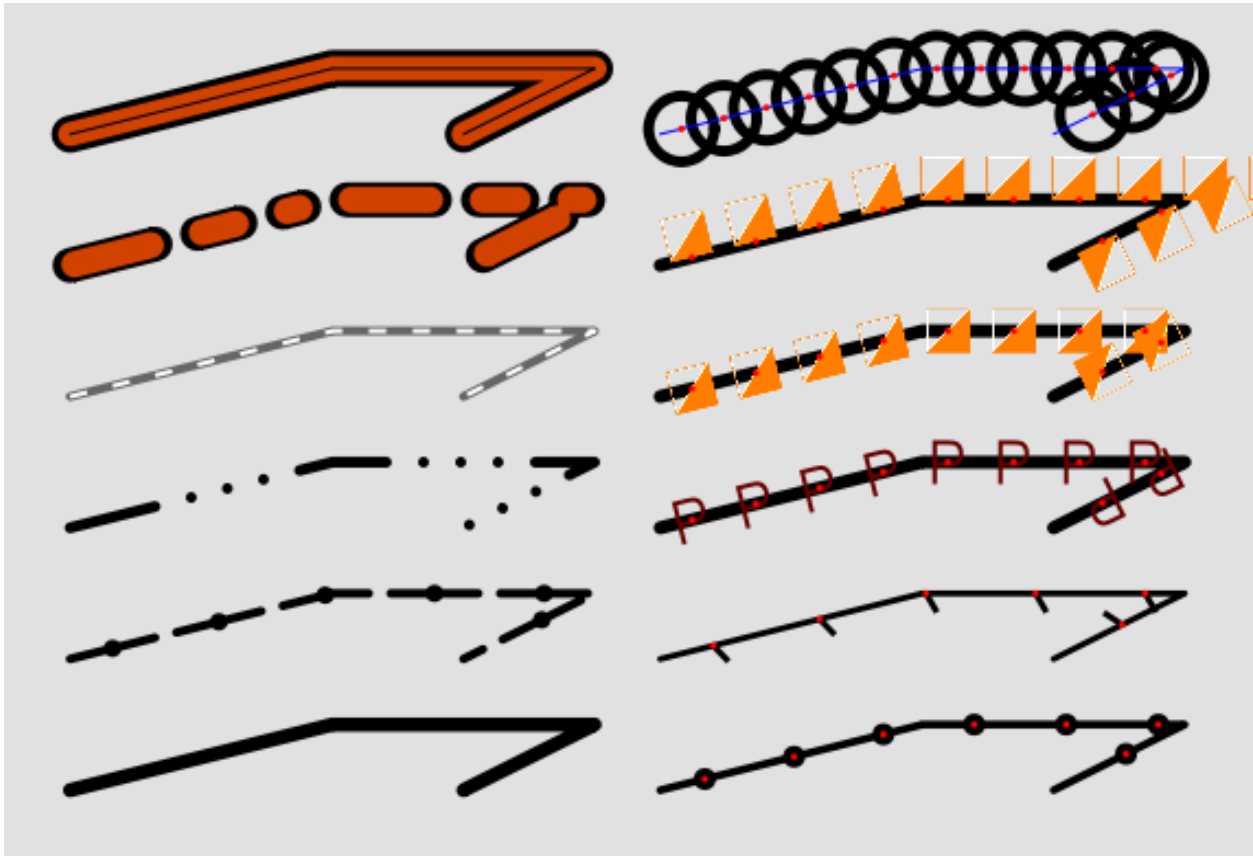


Fig. 4.5: Construction of Line Symbols

```
POINTS
  0 0
  0 10
  2.8 10
  2.8 0
  0 0
END # POINTS
ANCHORPOINT 0.5 0
END # SYMBOL

SYMBOL
NAME "o-flag-trans"
TYPE pixmap
IMAGE "o-flag-trans.png"
END # SYMBOL

##### Left column #####

LAYER # Simple line
STATUS DEFAULT
TYPE LINE
FEATURE
  POINTS
    5 5
    25 10
    45 10
    35 5
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 6.5
  END # STYLE
END # CLASS
END # LAYER

LAYER # Dashed line with symbol overlay
STATUS DEFAULT
TYPE LINE
FEATURE
  POINTS
    5 15
    25 20
    45 20
    35 15
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 5.0
    PATTERN 40 10 END
  END # STYLE
  STYLE
    SYMBOL "circlef"
    COLOR 0 0 0
    SIZE 8
```



```

    INITIALGAP 20
    GAP 50
    END
    END # CLASS
END # LAYER

LAYER # Dashed line, varying
    STATUS DEFAULT
    TYPE LINE
    FEATURE
    POINTS
        5 25
        25 30
        45 30
        35 25
    END # Points
    END # Feature
    CLASS
    STYLE
        COLOR 0 0 0
        WIDTH 5.0
        LINECAP round #[butt/round/square/triangle]
        LINEJOIN miter #[round/miter/bevel]
        LINEJOINMAXSIZE 3
        PATTERN 40 17 0 17 0 17 0 17 END
    END # STYLE
    END # CLASS
END # LAYER

LAYER # Line dash overlay
    STATUS DEFAULT
    TYPE LINE
    FEATURE
    POINTS
        5 35
        25 40
        45 40
        35 35
    END # Points
    END # Feature
    CLASS
    STYLE
        COLOR 102 102 102
        WIDTH 4.0
    END # STYLE
    STYLE
        COLOR 255 255 255
        WIDTH 2.0
        LINECAP BUTT
        PATTERN 8 12 END
    END # STYLE
    END # CLASS
END # LAYER

LAYER # Line dashed with dashed overlay
    STATUS DEFAULT
    TYPE LINE
    FEATURE

```

```

POINTS
  5 45
  25 50
  45 50
  35 45
END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 16.0
    PATTERN 40 20 20 20 10 20 END
  END # STYLE
  STYLE
    COLOR 209 66 0
    WIDTH 12.0
    INITIALGAP 2
    PATTERN 36 24 16 24 6 24 END
  END # STYLE
END # CLASS
END # LAYER

LAYER # Line overlay - 3
STATUS DEFAULT
TYPE LINE
FEATURE
  POINTS
    5 55
    25 60
    45 60
    35 55
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 17.0
  END # STYLE
  STYLE
    COLOR 209 66 0
    WIDTH 11.0
  END # STYLE
  STYLE
    COLOR 0 0 0
    WIDTH 1.0
  END # STYLE
END # CLASS
END # LAYER

##### right column #####

LAYER # Line - ellipse overlay
STATUS DEFAULT
TYPE LINE
FEATURE
  POINTS
    50 5
    70 10

```

```

    90 10
    80 5
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 3.6
  END # STYLE
  STYLE
    COLOR 0 0 0
    SYMBOL "circlef"
    SIZE 10
    GAP 42
  END # STYLE
  STYLE
    COLOR 255 0 0
    SYMBOL "circlef"
    SIZE 3
    GAP 42
  END # STYLE
END # CLASS
END # LAYER

LAYER # Line - symbol overlay
STATUS DEFAULT
TYPE LINE
FEATURE
  POINTS
    50 15
    70 20
    90 20
    80 15
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 2.8
  END # STYLE
  STYLE
    COLOR 0 0 0
    SYMBOL "vertline"
    SIZE 10.0
    ANGLE 30
    GAP -50
  END # STYLE
  STYLE
    COLOR 255 0 0
    SYMBOL "circlef"
    SIZE 3
    GAP 50
  END # STYLE
END # CLASS
END # LAYER

LAYER # Line - font overlay
STATUS DEFAULT

```

```
TYPE LINE
FEATURE
  POINTS
    50 25
    70 30
    90 30
    80 25
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 6
  END # STYLE
  STYLE
    COLOR 102 0 0
    SYMBOL "P"
    SIZE 20
    GAP -30
  END # STYLE
  STYLE
    COLOR 255 0 0
    SYMBOL "circlef"
    SIZE 3
    GAP 30
  END # STYLE
END # CLASS
END # LAYER

LAYER # Line - pixmap overlay
STATUS DEFAULT
TYPE LINE
FEATURE
  POINTS
    50 35
    70 40
    90 40
    80 35
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 6
  END # STYLE
  STYLE
    COLOR 102 0 0
    SYMBOL "o-flag-trans"
    SIZE 20
    GAP -30
  END # STYLE
  STYLE
    COLOR 255 0 0
    SYMBOL "circlef"
    SIZE 3
    GAP 30
  END # STYLE
END # CLASS
```

```

END # LAYER

LAYER # Line - pixmap overlay
STATUS DEFAULT
TYPE LINE
FEATURE
POINTS
  50 45
  70 50
  90 50
  80 45
END # Points
END # Feature
CLASS
STYLE
  COLOR 0 0 0
  WIDTH 6
END # STYLE
STYLE
  COLOR 102 0 0
  SYMBOL "o-flag-trans"
  SIZE 20
  GAP -30
  OFFSET -10 -99
END # STYLE
STYLE
  COLOR 255 0 0
  SYMBOL "circlef"
  SIZE 3
  GAP 30
END # STYLE
END # CLASS
END # LAYER

```

LINECAP By default, all lines (and patterns) will be drawn with rounded ends (extending the lines slightly beyond their ends). This effect gets more obvious the larger the width of the line is. It is possible to alter this behaviour using the *LINECAP* parameter of the *STYLE*. *LINECAP butt* will give butt ends (stops the line exactly at the end), with no extension of the line. *LINECAP square* will give square ends, with an extension of the line. *LINECAP round* is the default.

LINEJOIN The different values for the parameter *LINEJOIN* have the following visual effects. Default is *round*. *miter* will follow line borders until they intersect and fill the resulting area. *none* will render each segment using linecap *butt*. The figure below illustrates the different linejoins.

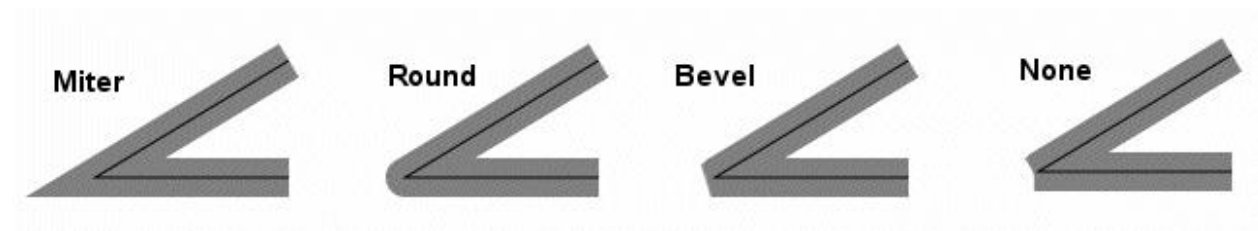


Fig. 4.6: Different kinds of linejoins

LINEJOINMAXSIZE (only relevant for *LINEJOIN miter*) Specify the maximum length of *miter* linejoin factor *m* (see the figure below). The value is a multiplication factor (default 3).

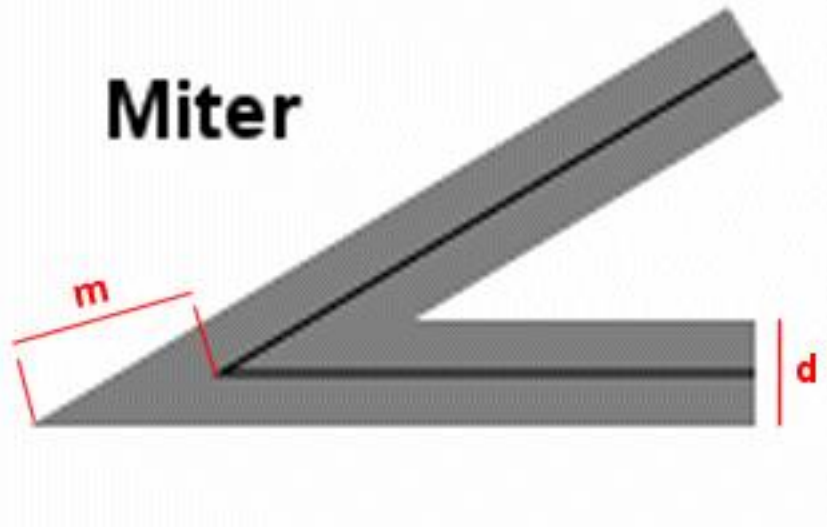


Fig. 4.7: Miter linejoin

The max length of the *miter* join is calculated as follows (*d* is the line width, specified with the *WIDTH* parameter of the *STYLE*):

$$m_{max} = d * LINEJOINMAXSIZE$$

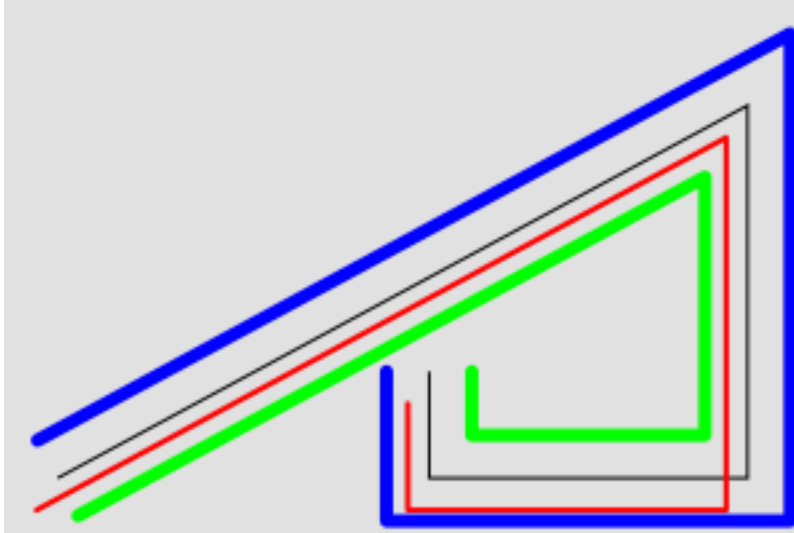
If $m > m_{max}$, then the connection length will be set to m_{max} .

Use of the *OFFSET* parameter

In *STYLE*, an *OFFSET* parameter can be set to shift symbols in the X and Y direction. The displacement is not influenced by the direction of the line geometry. Therefore the point symbols used for styling are all shifted in the same direction, independent of the direction of the line (as defined in style number 2 in the map file example below - red line in the map image). A positive X value shifts to the right. A positive Y value shifts downwards.

To generate lines that are shifted relative to the original lines, -99 has to be used for the Y value of the *OFFSET*. Then the X value defines the distance to the line from the original geometry (perpendicular to the line). A positive X value will shift to the right (when viewed in the direction of the line), a negative X value will shift to the left.

The example below shows how *OFFSET* works with the use of -99 (blue and green lines) and without the use of -99 (red line). The thin black line shows the location of the line geometry.

Fig. 4.8: Use of the *OFFSET* parameter with lines - map imageUse of the *OFFSET* parameter with lines - Map file excerpt

```

LAYER #
  STATUS DEFAULT
  TYPE LINE
  FEATURE
    POINTS
      20 20
      280 160
      280 20
      160 20
      160 60
    END # Points
  END # Feature
  CLASS
    STYLE # no offset
      COLOR 0 0 0 # black
      WIDTH 1
    END # STYLE
    STYLE # simple offset left and down
      COLOR 255 0 0 # red
      WIDTH 2
      OFFSET -8 12
    END # STYLE
    STYLE # left offset rel. to line direction
      COLOR 0 0 255 # blue
      WIDTH 5
      OFFSET -16 -99
    END # STYLE
    STYLE # right offset rel. to line direction
      COLOR 0 255 0 # green
      WIDTH 5
      OFFSET 16 -99
    END # STYLE
  END # CLASS
END # LAYER

```

Asymmetrical line styling with point symbols

Line number 2 and 5 from the bottom in the right column of the “Construction of Line Symbols” figure are examples of asymmetrical line styling using a point symbol. This can be achieved either by using an `OFFSET` (with a `Y` value of `-99`), or by using `ANCHORPOINT`, as described in the tricks section below. Line number 2 from the bottom can be produced using `ANCHORPOINT` - this is the best method for placing symbols on lines. Line number 5 from the bottom is produced using `STYLE OFFSET`. As can be seen, the symbols are here rendered on the offset line, meaning that at sharp bends, some symbols will be placed far away from the line.

Area Symbols

Areas (polygons) can be filled with full colour. Areas can also be filled with symbols to create for instance hatches and graticules.

The symbols are by default used as tiles, and rendered (without spacing) one after the other in the `x` and `y` direction, filling the whole polygon.

If the `SIZE` parameter is used in the `STYLE`, the symbols will be scaled to the specified height.

The `GAP` parameter of the `STYLE` can be used to increase the spacing of the symbols.

The AGG renderer uses anti-aliasing by default, so edge effects around the symbols can occur.

Hatch fill

Simple line hatches (e.g. horizontal, vertical and diagonal) can be created by filling the polygon with a hatch symbol.

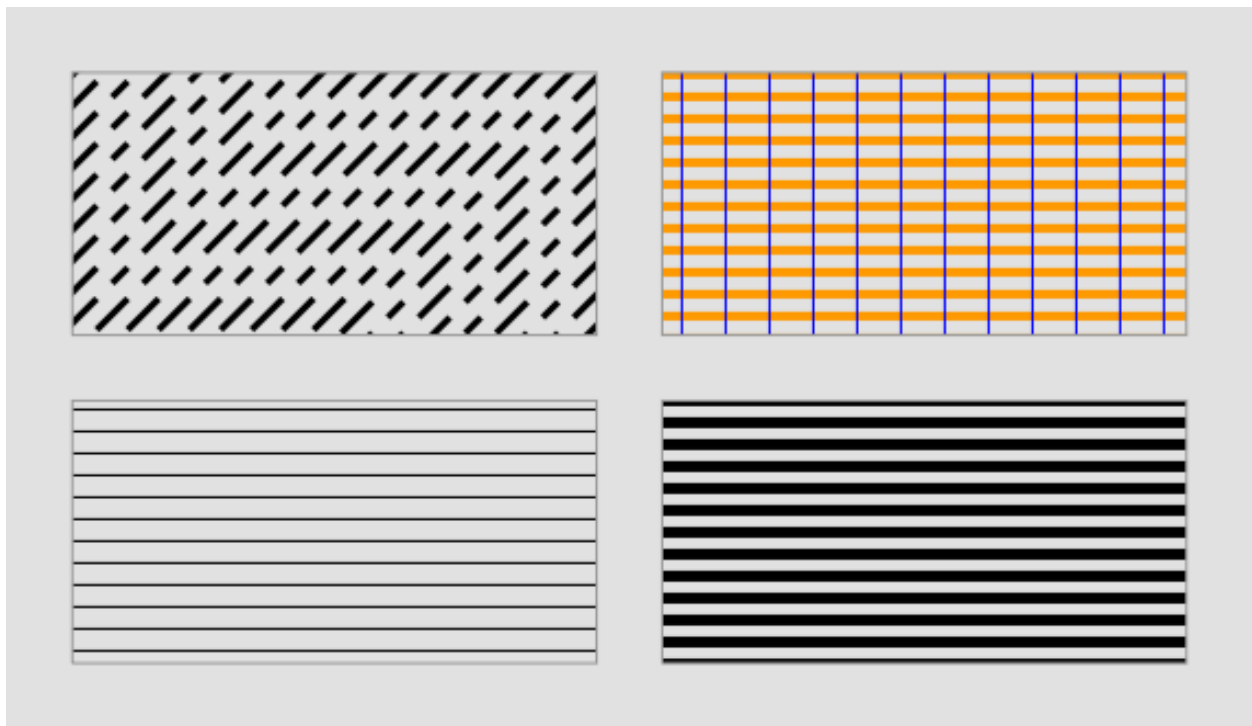


Fig. 4.9: Hatch examples

The *SIZE* parameter in the *STYLE* that uses a *SYMBOL* of type *hatch* specifies the distance from center to center between the lines (the default is 1). The *WIDTH* parameter specifies the width of the lines in the hatch pattern (default is 1). The *ANGLE* parameter specifies the direction of the lines (default is 0 - horizontal lines). Since version 6.2, the *PATTERN* parameter can be used to create hatches with dashed lines.

The figure demonstrates the use of *SIZE* (bottom left); *WIDTH* (bottom right); *ANGLE*, *PATTERN* and *SIZE* (top left); and overlay (top right) of hatches.

The code below shows excerpts of the map file that was used to produce the figure.

First, the *SYMBOL* definition:

```
SYMBOL
  NAME "hatchsymbol"
  TYPE hatch
END
```

Then the *CLASS* definitions:

Table 4.2: Hatches

<i>CLASS</i> definitions
<pre> LAYER # hatch ... CLASS STYLE SYMBOL "hatchsymbol" COLOR 0 0 0 SIZE 10 END # <i>STYLE</i> END # <i>CLASS</i> END # <i>LAYER</i> LAYER # hatch with angle and pattern ... CLASS STYLE SYMBOL "hatchsymbol" COLOR 0 0 0 SIZE 10 WIDTH 3 ANGLE 45 PATTERN 20 10 10 10 END END # <i>STYLE</i> END # <i>CLASS</i> END # <i>LAYER</i> LAYER # hatch with wide lines ... CLASS STYLE SYMBOL "hatchsymbol" COLOR 0 0 0 SIZE 10 WIDTH 5 END # <i>STYLE</i> END # <i>CLASS</i> END # <i>LAYER</i> LAYER # cross hatch ... CLASS STYLE SYMBOL "hatchsymbol" COLOR 255 153 0 SIZE 10 WIDTH 4 END # <i>STYLE</i> STYLE SYMBOL "hatchsymbol" COLOR 0 0 255 SIZE 20 ANGLE 90 END # <i>STYLE</i> END # <i>CLASS</i> END # <i>LAYER</i> </pre>

Polygon fills with symbols of *TYPE pixmap*

Polygons can be filled with pixmaps.

Note: If the *STYLE SIZE* parameter is different from the image height of the pixmap, there can be rendering artefacts around the pixmaps (visible as a grid with the “background” colour).

Pixmap symbols can be rotated using the *ANGLE* parameter, but for polygon fills, this produces strange effects, and is not recommended.

To create complex area symbols, e.g. with defined distances between single characters or hatches with broad lines, pixmap fill is probably the best option. Depending on the desired pattern you have to generate the raster image with high precision using a graphical editor. The figure below is an example of how to obtain a regular allocation of symbols with defined spacing.

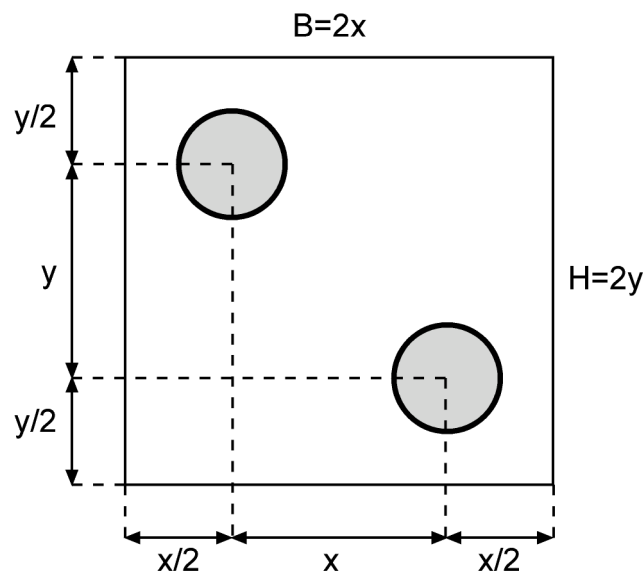


Fig. 4.10: Raster image for a regular symbol fill

You can use other shapes than circles. 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 the same x and y distances to the imageborder is enough.

The following figure shows an example of how you can design a pixmap to produce a hatch with wide lines.

To create a 45 degree hatch use:

$B = H$ and $x = y$

Note: 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.

The example below shows 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 to the colour palette.

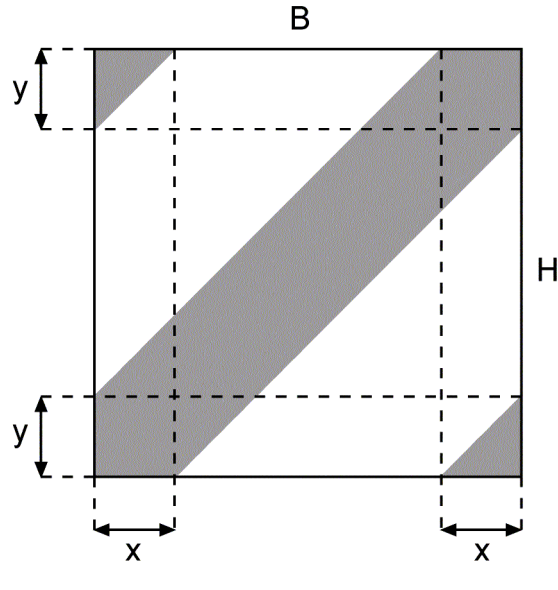


Fig. 4.11: Raster image for a hatched fill

Table 4.3: Construction of a horizontally arranged area symbol



<i>CLASS</i> section	<i>SYMBOL</i> definition
<pre> CLASS STYLE COLOR 255 255 0 END STYLE SYMBOL "in_the_star" END STYLE OUTLINECOLOR 0 0 0 WIDTH 1 END END </pre>	<pre> SYMBOL NAME "in_the_star" TYPE PIXMAP IMAGE "stern.png" TRANSPARENT 8 END </pre> 



Fig. 4.12: Polygon fill - regular grid pattern

Table 4.4: Construction of a diagonally arranged area symbol

<i>CLASS</i> section	<i>SYMBOL</i> definition
<pre> CLASS STYLE SYMBOL "in_point1" END STYLE OUTLINECOLOR 0 0 0 WIDTH 1 END END </pre>	<pre> SYMBOL NAME "in_point1" TYPE PIXMAP IMAGE "flaechel_1.png" TRANSPARENT 13 END </pre> 

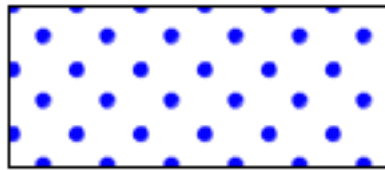


Fig. 4.13: Polygon fill - diagonal pattern

Table 4.5: Construction of a hatch


<i>CLASS</i> section	<i>SYMBOL</i> definition
<pre> CLASS STYLE COLOR 255 255 0 END STYLE SYMBOL "in_hatch" END STYLE OUTLINECOLOR 0 0 0 WIDTH 1 END END </pre>	<pre> SYMBOL NAME "in_hatch" TYPE PIXMAP IMAGE "schraffur.png" TRANSPARENT 2 END </pre> 



Fig. 4.14: Polygon fill - hatch

Polygon fills with symbols of *TYPE vector*

Polygons can be filled with symbols of *TYPE vector*. As for the other symbol fills, the pattern will be generated by using the specified symbol for the tiles. The bounding box of the symbol is used when tiling.

Creating vector symbols for polygon fills is done in much the same way as for pixmap symbols. Precision is necessary to get nice symmetrical symbols.

The upper left corner of the bounding box of a symbol of *TYPE vector* is always (0, 0) in the symbol's coordinate system. The lower right corner of the bounding box is determined by the maximum x and y values of the symbol definition (*POINTS* parameter). The fact that the upper left corner always is at (0,0) makes it convenient to construct symbols such as the dash signature found as number two from the bottom in the centre column of the example below.

Both polygon (*FILLED true*) and line (*FILLED false*) vector symbols can be used. For line symbols, the *WIDTH* parameter of the *STYLE* will give the line width and the *SIZE* parameter will specify the height of the symbol.

Note: For vector line symbols (*FILL off*), if a width greater than 1 is specified, the lines will grow to extend outside the original bounding box of the symbol. The parts that are outside of the bounding box will be cut away.

STYLE *ANGLE* can be used for polygon fills, but will only rotate each individual symbol, not the pattern as a whole. It is therefore quite demanding to generate rotated patterns.

Below you will find some examples of vector symbols used for polygon fills. The polygon fill is accompanied by the vector symbol used for the fill. The center of the vector symbol is indicated with a red dot.

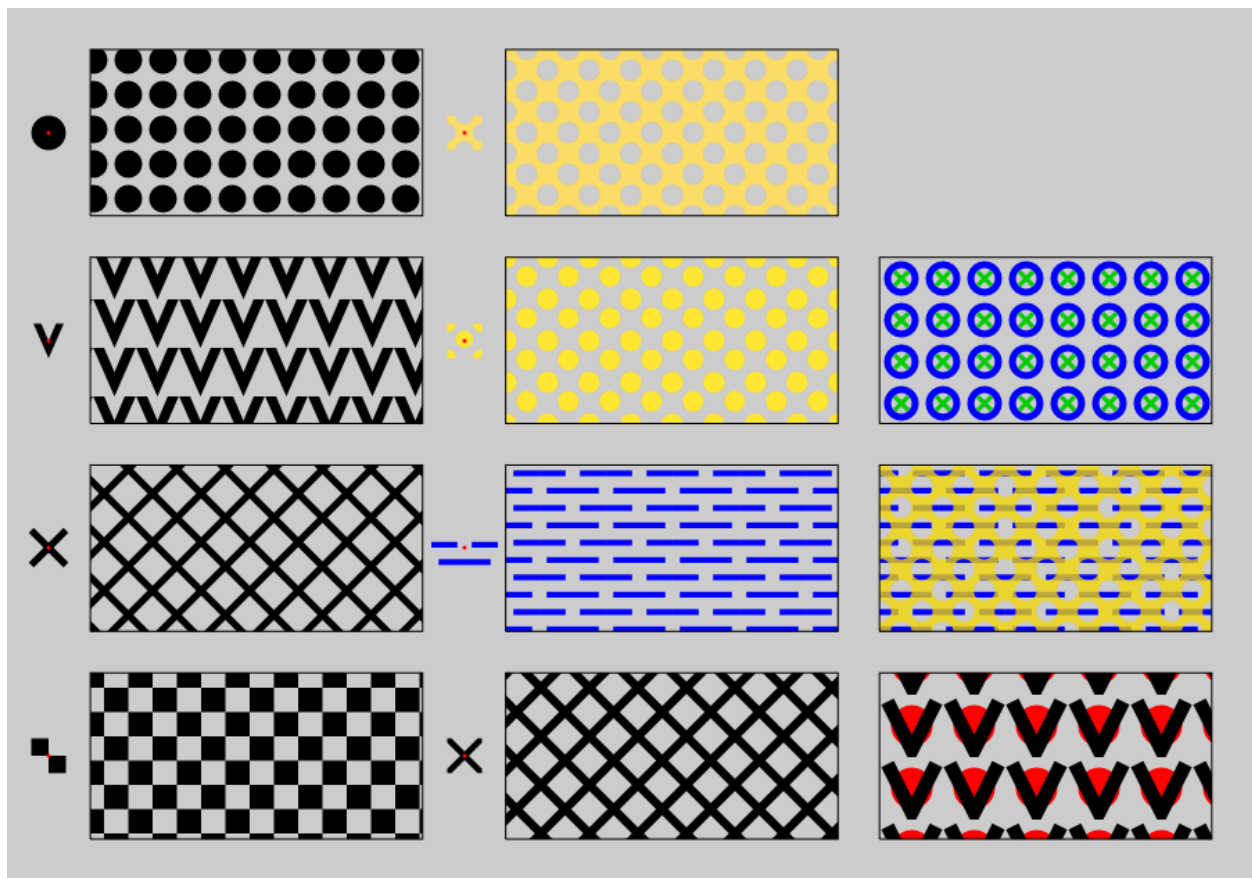


Fig. 4.15: Polygon fills - vector

Excerpts from the map file for the polygon fill vector examples above First, the *LAYERS*

```

LAYER # chess board
STATUS DEFAULT
TYPE POLYGON
FEATURE
  POINTS
    5 5
    5 25
    45 25
    45 5
    5 5
  END # Points
END # Feature
CLASS
  STYLE
    SYMBOL "chess"
    COLOR 0 0 0
    SIZE 35
  END # STYLE
END # CLASS
END # LAYER

LAYER # x - line
STATUS DEFAULT
TYPE POLYGON
FEATURE
  POINTS
    5 30
    5 50
    45 50
    45 30
    5 30
  END # Points
END # Feature
CLASS
  STYLE
    SYMBOL "x-line"
    COLOR 0 0 0
    WIDTH 5
    SIZE 35
  END # STYLE
END # CLASS
END # LAYER

LAYER # v polygon
STATUS DEFAULT
TYPE POLYGON
FEATURE
  POINTS
    5 55
    5 75
    45 75
    45 55
    5 55
  END # Points
END # Feature
CLASS
  STYLE
    SYMBOL "v-poly"

```

```
        COLOR 0 0 0
        SIZE 35
    END # STYLE
END # CLASS
END # LAYER

LAYER # Circles
STATUS DEFAULT
TYPE POLYGON
FEATURE
POINTS
    5 80
    5 100
    45 100
    45 80
    5 80
END # Points
END # Feature
CLASS
STYLE
    SYMBOL "circlef"
    COLOR 0 0 0
    SIZE 20
    GAP 25
END # STYLE
END # CLASS
END # LAYER

LAYER # x polygon
STATUS DEFAULT
TYPE POLYGON
FEATURE
POINTS
    55 5
    55 25
    95 25
    95 5
    55 5
END # Points
END # Feature
CLASS
STYLE
    COLOR 0 0 0
    SYMBOL "x-poly-fill"
    SIZE 35
END # STYLE
END # CLASS
END # LAYER

LAYER # indistinct marsh
STATUS DEFAULT
TYPE POLYGON
FEATURE
POINTS
    55 30
    55 50
    95 50
    95 30
```



```

    55 30
    END # Points
END # Feature
CLASS
    STYLE
        COLOR 0 0 255
        SYMBOL "ind_marsh_poly"
        SIZE 25
    END # STYLE
END # CLASS
END # LAYER

LAYER # diagonal circles
STATUS DEFAULT
TYPE POLYGON
FEATURE
    POINTS
        55 55
        55 75
        95 75
        95 55
        55 55
    END # Points
END # Feature
CLASS
    STYLE
        COLOR 255 230 51
        SYMBOL "diag_dots"
        SIZE 30
    END # STYLE
END # CLASS
END # LAYER

LAYER # diagonal holes in yellow
STATUS DEFAULT
TYPE POLYGON
FEATURE
    POINTS
        55 80
        55 100
        95 100
        95 80
        55 80
    END # Points
END # Feature
CLASS
    STYLE
        SYMBOL "diag_holes"
        SIZE 30
        COLOR 250 220 102
    END # STYLE
END # CLASS
END # LAYER

LAYER # v line + circle
STATUS DEFAULT
TYPE POLYGON

```

```

FEATURE
  POINTS
    100 5
    100 25
    140 25
    140 5
    100 5
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 255 0 0
    SYMBOL "circlef"
    SIZE 30
    GAP 45
  END # STYLE
  STYLE
    COLOR 0 0 0
    SYMBOL "v-line"
    LINEJOIN miter
    LINECAP butt
    SIZE 35
    WIDTH 10
    GAP 45
  END # STYLE
END # CLASS
END # LAYER

LAYER # indistinct marsh + diagonal holes in yellow
  STATUS DEFAULT
  TYPE POLYGON
  FEATURE
    POINTS
      100 30
      100 50
      140 50
      140 30
      100 30
    END # Points
  END # Feature
  CLASS
    STYLE
      COLOR 0 0 255
      SYMBOL "ind_marsh_poly"
      SIZE 25
    END # STYLE
    STYLE
      SYMBOL "diag_holes"
      SIZE 30
      COLOR 250 220 0
      OPACITY 75
    END # STYLE
  END # CLASS
END # LAYER

LAYER # x line + circle
  STATUS DEFAULT
  TYPE POLYGON

```

```

FEATURE
  POINTS
    100 55
    100 75
    140 75
    140 55
    100 55
  END # Points
END # Feature
CLASS
  STYLE
    COLOR 0 0 255
    SYMBOL "circle"
    WIDTH 5
    SIZE 20
    GAP 30
  END # STYLE
  STYLE
    COLOR 0 204 0
    SYMBOL "x-line"
    SIZE 10
    WIDTH 3
    GAP 30
  END # STYLE
END # CLASS
END # LAYER

```

Then the *SYMBOLS*:

```

SYMBOL
  NAME "circlef"
  TYPE ellipse
  FILLED true
  POINTS
    10 10
  END # POINTS
END # SYMBOL

SYMBOL
  NAME "circle"
  TYPE ellipse
  FILLED false
  POINTS
    10 10
  END # POINTS
END # SYMBOL

SYMBOL
  NAME "v-line"
  TYPE vector
  POINTS
    0 0
    5 10
    10 0
  END
END

SYMBOL
  NAME "v-poly"

```

```
TYPE vector
FILLED false
FILLED true
POINTS
  0 0
  3.5 8
  7 0
  5.2 0
  3.5 4
  1.8 0
  0 0
END
END

SYMBOL
NAME "x-line"
TYPE vector
POINTS
  0 0
  1 1
  -99 -99
  0 1
  1 0
END
END

SYMBOL
NAME "chess"
TYPE vector
FILLED true
POINTS
  0 0
  10 0
  10 10
  0 10
  0 0
  -99 -99
  10 10
  20 10
  20 20
  10 20
  10 10
END
END

SYMBOL
NAME "x-poly-fill"
TYPE vector
FILLED true
POINTS
  0 1.131
  0 0
  1.131 0
  4.566 3.434
  8 0
  9.131 0
  9.131 1.131
  5.697 4.566
```

```

9.131 8
9.131 9.131
8 9.131
4.566 5.697
1.131 9.131
0 9.131
0 8
3.434 4.566
0 1.131
END # POINTS
END # SYMBOL

SYMBOL
NAME "ind_marsh_poly"
TYPE vector
FILLED true
POINTS
  # Half line
  0 2
  4.5 2
  4.5 3
  0 3
  0 2
  -99 -99
  # Half line
  7 2
  11.5 2
  11.5 3
  7 3
  7 2
  -99 -99
  # Hole line
  1.25 5
  10.25 5
  10.25 6
  1.25 6
  1.25 5
END
END

SYMBOL
NAME "diag_dots"
TYPE vector
FILLED true
POINTS
  # Central circle:
  0.7450 0.4500
  0.7365 0.5147
  0.7115 0.5750
  0.6718 0.6268
  0.6200 0.6665
  0.5597 0.6915
  0.4950 0.7000
  0.4303 0.6915
  0.3700 0.6665
  0.3182 0.6268
  0.2785 0.5750
  0.2535 0.5147

```

```

0.2450 0.4500
0.2535 0.3853
0.2785 0.3250
0.3182 0.2732
0.3700 0.2335
0.4303 0.2085
0.4950 0.2000
0.5597 0.2085
0.6200 0.2335
0.6718 0.2732
0.7115 0.3250
0.7365 0.3853
0.7450 0.4500
-99 -99
0.25 0.0
0.2415 0.0647
0.2165 0.1250
0.1768 0.1768
0.1250 0.2165
0.0647 0.2415
0.0 0.25
0.0 0.0
0.25 0.0
-99 -99
1 0.25
0.9252 0.2415
0.8649 0.2165
0.8132 0.1768
0.7734 0.1250
0.7485 0.0647
0.74 0.0
1 0.0
1 0.25
-99 -99
0.74 1
0.7485 0.9252
0.7734 0.8649
0.8132 0.8132
0.8649 0.7734
0.9252 0.7485
1 0.74
1 1
0.74 1
-99 -99
0.0 0.74
0.0647 0.7485
0.1250 0.7734
0.1768 0.8132
0.2165 0.8649
0.2415 0.9252
0.25 1
0.0 1
0.0 0.74

```

END

END

SYMBOL

NAME "diag_holes"

```

TYPE vector
FILLED true
POINTS
  0.0      0.0
  # Left half circle
  0.0      0.24
  0.0647   0.2485
  0.1250   0.2734
  0.1768   0.3132
  0.2165   0.3649
  0.2415   0.4252
  0.25     0.5
  0.2415   0.5647
  0.2165   0.6250
  0.1768   0.6768
  0.1250   0.7165
  0.0647   0.7415
  0.0      0.75

  0.0      1.0
  # Bottom half circle
  0.24     1
  0.2485   0.9252
  0.2734   0.8649
  0.3132   0.8132
  0.3649   0.7734
  0.4252   0.7485
  0.5      0.74
  0.5647   0.7485
  0.6250   0.7734
  0.6768   0.8132
  0.7165   0.8649
  0.7415   0.9252
  0.75     1

  1.0      1.0
  # Right half circle
  1 0.75
  0.9252   0.7415
  0.8649   0.7165
  0.8132   0.6768
  0.7734   0.6250
  0.7485   0.5647
  0.74     0.5
  0.7485   0.4252
  0.7734   0.3649
  0.8132   0.3132
  0.8649   0.2734
  0.9252   0.2485
  1 0.24

  1.0      0.0
  # Top half circle
  0.75     0.0
  0.7415   0.0647
  0.7165   0.1250
  0.6768   0.1768
  0.6250   0.2165

```

```
0.5647 0.2415
0.5 0.25
0.4252 0.2415
0.3649 0.2165
0.3132 0.1768
0.2734 0.1250
0.2485 0.0647
0.24 0.0

0.0 0.0
END
END
```

Polygon outlines

Polygon outlines can be created by using *OUTLINECOLOR* in the *STYLE*. *WIDTH* specifies the width of the outline.

```
STYLE
  OUTLINECOLOR 0 255 0
  WIDTH 3
END # STYLE
```

Dashed polygon outlines can be achieved by using *OUTLINECOLOR*, *WIDTH* and *PATTERN* (together with *LINECAP*, *LINEJOIN* and *LINEJOINMAXSIZE*). For more information on the use of *PATTERN*, see *Use of the PATTERN and GAP parameters*.

```
STYLE
  OUTLINECOLOR 0 255 0
  WIDTH 3
  PATTERN
    10 5
  END # PATTERN
  LINECAP BUTT
END # STYLE
```

For some symbol types, it is even possible to style polygon outlines using *OUTLINECOLOR*, *SYMBOL* and *GAP*.

```
STYLE
  OUTLINECOLOR 0 255 0
  SYMBOL 'circle'
  SIZE 5
  GAP 15
END # STYLE
```

Examples (MapServer 4)

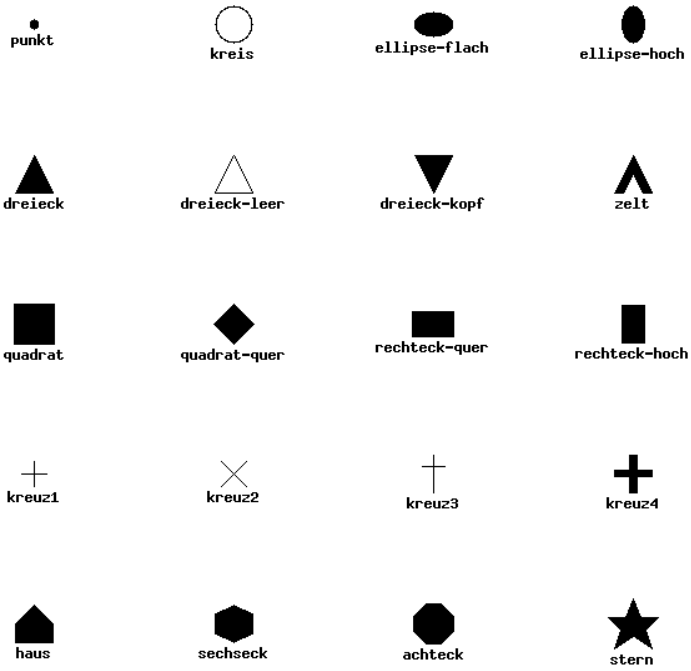
The examples in this section were made for MapServer 4.

Note: Many of these symbols will not work with later versions of MapServer, but they contain a lot of useful symbol definitions and are therefore provided as reference.

The symbols were created with *map* files and *symbol* files ([download_old_symbols](#)). If you want to use these MAP files please note, that your MapServer must at least be able to handle 50 symbols. Otherwise you will get an error while loading the *symbol* files.

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

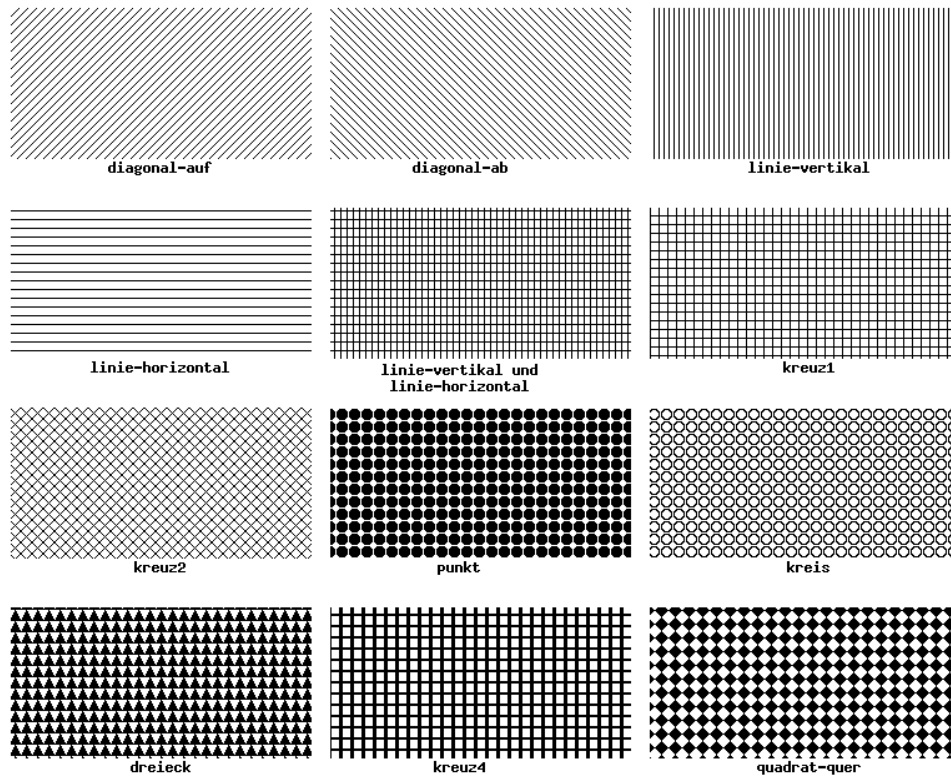
grenze1 (based on circle symbol of type ELLIPSE)

grenze2 (based on rectangle symbol of type VECTOR)

rechteck-quer-st

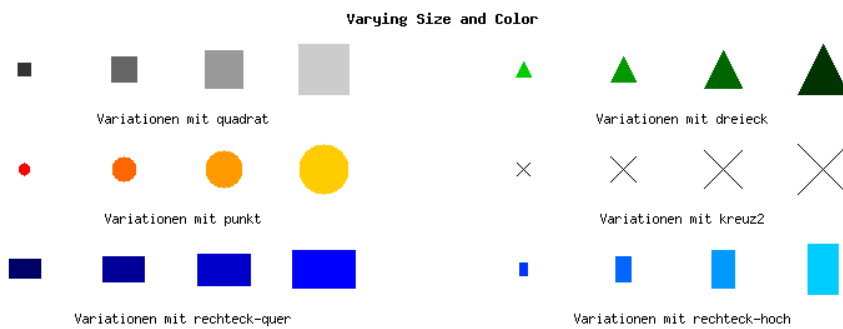
rechteck-bahn

Graphical Primitives for Polygon-Symbolizers located in the defined Symbolfile symbols.sym

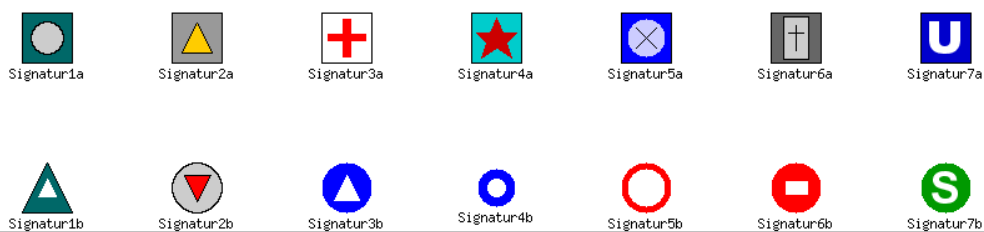


Complex Symbols

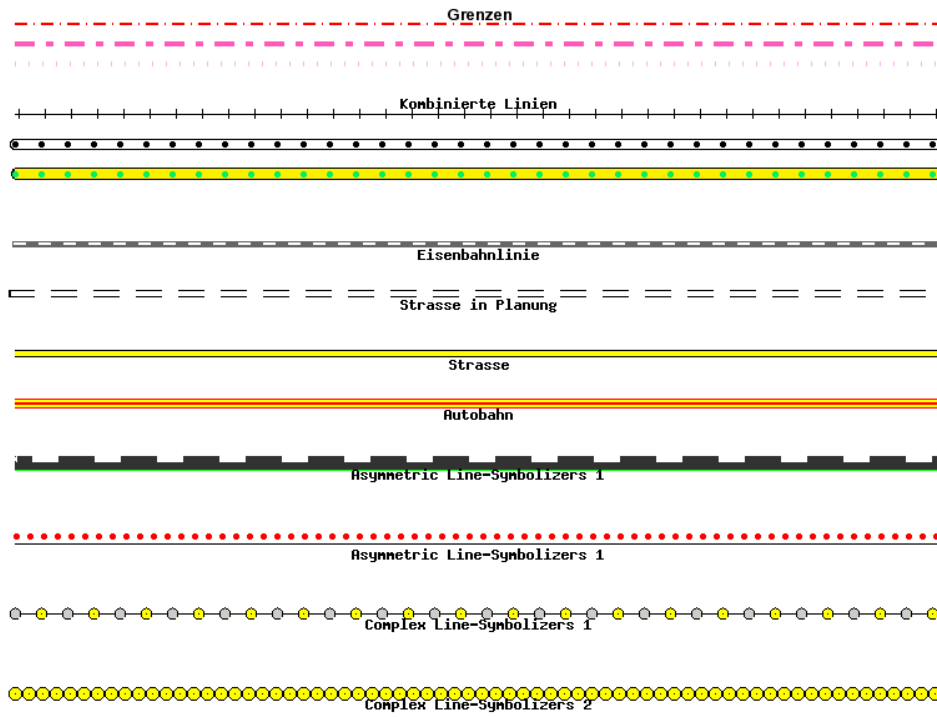
Examples of Point-Symbolizers varying some graphical Attributes



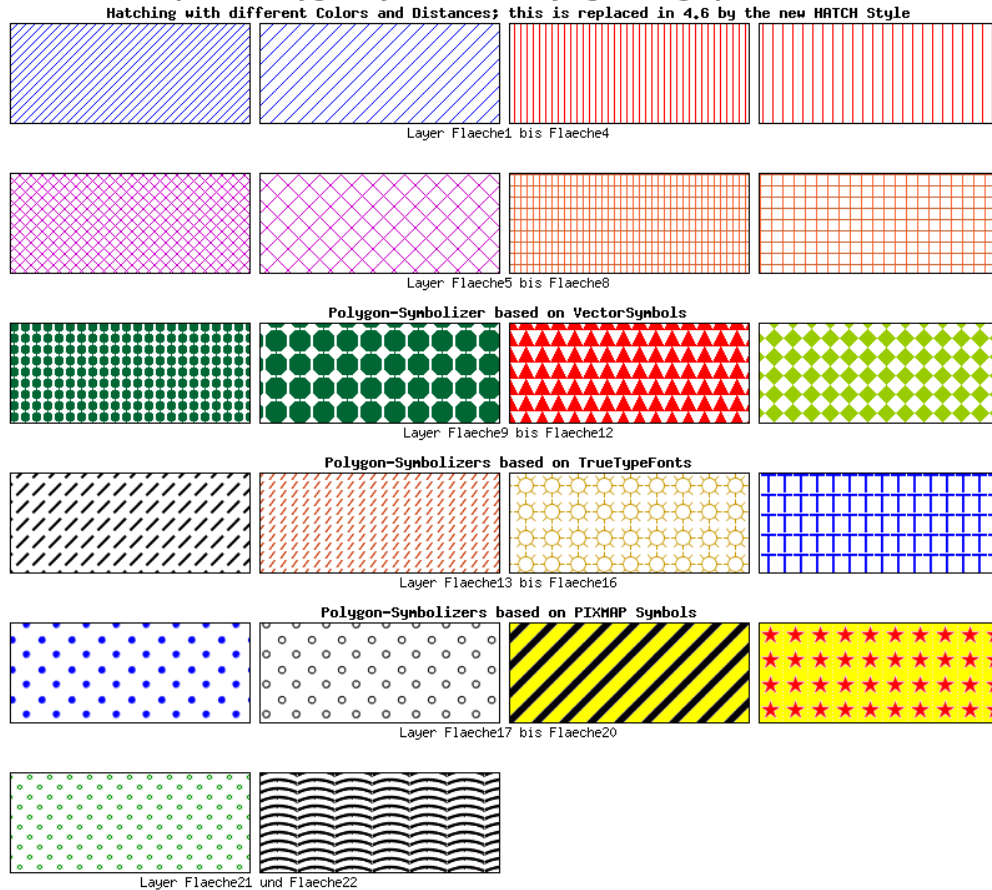
Examples for combinations of several Basetypes



Examples of combined Line-Symbolizers varying some graphical Attributes



Examples of Polygon-Symbolizers varying some graphical Attributes



Tricks

Changing the center of a point symbol

MapServer does all transformations (offset, rotation) with respect to the symbol anchor point. By default, the anchor point is calculated from the symbol's bounding box. In some cases it can be useful to change the anchor point of a symbol. Since version 6.2, this can be done using the *SYMBOL ANCHORPOINT*.

Here are some examples of what can be achieved by using the *ANCHORPOINT* mechanisms for point symbols and decorated lines. There are three examples in the illustration, and each example shows the result with and without the use of *ANCHORPOINT*. At the top, arrows are added to lines using *GEOMTRANSFORM start / end*. In the middle, tags are added to lines using *GAP* and *ANGLE*. At the bottom, a point symbol is shifted and rotated. The red dots represent the center points, and the blue dots the offsets.

Below you will find three tables that contain the *SYMBOLS* and the *STYLE* mechanisms that were used to generate the shifted symbols in the illustration.

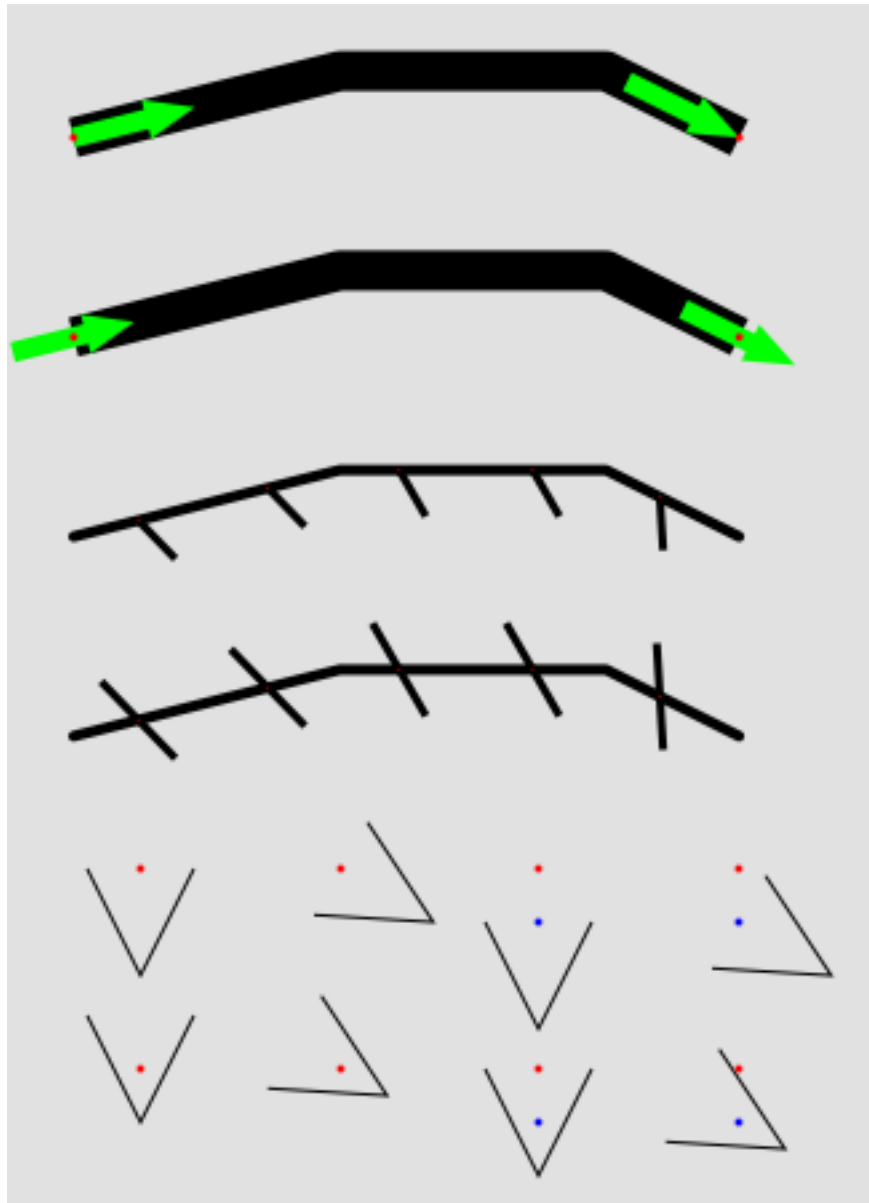


Fig. 4.16: Shifting trick

Table 4.6: Symbol tricks - shift - arrows

<i>SYMBOLs</i>	<i>LAYER STYLEs</i>
<pre> SYMBOL NAME "arrow-start" TYPE vector FILLED true POINTS 0 0.4 3 0.4 3 0 5 0.8 3 1.6 3 1.2 0 1.2 0 0.4 END # POINTS ANCHORPOINT 0 0.5 END # SYMBOL SYMBOL NAME "arrow-end" TYPE vector FILLED true POINTS 0 0.4 3 0.4 3 0 5 0.8 3 1.6 3 1.2 0 1.2 0 0.4 END # POINTS ANCHORPOINT 1 0.5 END # SYMBOL </pre>	<pre> LAYER # Line STATUS DEFAULT TYPE LINE FEATURE POINTS 20 80 40 85 60 85 70 80 END # Points END # Feature CLASS STYLE COLOR 0 0 0 WIDTH 15 LINECAP butt END # STYLE STYLE GEOMTRANSFORM "start" COLOR 0 255 0 SYMBOL "arrow-start" SIZE 15.0 ANGLE AUTO END # STYLE STYLE GEOMTRANSFORM "start" COLOR 255 0 0 SYMBOL "circlef" SIZE 3 END # STYLE STYLE GEOMTRANSFORM "end" COLOR 0 255 0 SYMBOL "arrow-end" SIZE 15.0 ANGLE AUTO END # STYLE STYLE GEOMTRANSFORM "end" COLOR 255 0 0 SYMBOL "circlef" SIZE 3 END # STYLE END # CLASS END # LAYER </pre>

Table 4.7: Symbol tricks - shift - asymmetrical tags

SYMBOLs	LAYER STYLEs
<pre> SYMBOL NAME "vert-line-shift" TYPE vector POINTS 0 0 0 10 END # POINTS ANCHORPOINT 0.5 0 END # SYMBOL SYMBOL NAME "vert-line" TYPE vector POINTS 0 0 0 10 END # POINTS END # SYMBOL </pre>	<pre> LAYER # Line - symbol overlay STATUS DEFAULT TYPE LINE FEATURE POINTS 20 50 40 55 60 55 70 50 END # Points END # Feature CLASS STYLE COLOR 0 0 0 WIDTH 4 END # STYLE STYLE COLOR 0 0 0 SYMBOL "vert-line-shift" SIZE 20.0 WIDTH 3 ANGLE 30 GAP -50 END # STYLE STYLE COLOR 255 0 0 SYMBOL "circlef" SIZE 3 GAP 50 END # STYLE END # CLASS END # LAYER </pre>

Table 4.8: Symbol tricks. Unshifted symbol (top) and shifted symbol

<i>SYMBOLs</i>
<pre> SYMBOL NAME "v-line" TYPE vector POINTS 0 0 5 10 10 0 END # <i>POINTS</i> END # <i>SYMBOL</i> SYMBOL NAME "v-line-shift" TYPE vector POINTS 0 0 5 10 10 0 END # <i>POINTS</i> ANCHORPOINT 0.5 0 END # <i>SYMBOL</i> </pre>

Mapfile changes related to symbols

Version 6.2

The *ANCHORPOINT SYMBOL* parameter was added.

The *INITIALGAP STYLE* parameter was added.

The *GAP STYLE* parameter's behaviour was modified to specify center to center spacing.

PATTERN support for symbols of *TYPE hatch*.

Version 6.0

Parameters related to styling was moved from the *SYMBOL* element to the *STYLE* element of *CLASS* (in *LAYER*):

PATTERN (introduced in 5.0, previously called *STYLE*), *GAP*, *LINECAP*, *LINEJOIN*, *LINEJOINMAX-SIZE*

The *SYMBOL TYPE cartoline* is no longer needed, and therefore not available in version 6.0.

Current Problems / Open Issues

GAP - PATTERN incompatibility

Creating advanced line symbols involving dashed lines is difficult due to the incompatibility of the dashed line mechanisms (*PATTERN*) and the symbol on line placement mechanisms (*GAP*). A solution could be to allow *GAP* to be a list instead of a single number (perhaps renaming to *GAPS* or *DISTANCES*), but it would also be necessary to introduce

a new parameter to specify the distance to the first symbol on the line (INTIALGAP has been implemented in the development version - 6.2).

GAP does not support two dimensions (relevant for polygon fills), so the same gap will have to be used for for the x and the y directions. The introduction of new parameters - *GAPX* and *GAPY* could be a solution to this.

The End

We hope that this document will help you to present your data in a cartographically nice manner with MapServer and explains some basics and possibilities of the concept of MapServer as well as some weaknesses. It would be great to put together a cartographical symbols library for the profit of everyone. This especially concerns truetype fonts, which have been developed for some projects and contain some typical signatures for cartographical needs.

You can also view the [discussion paper for the improvement of the MapServer Graphic-Kernel](#) (German only).

4.1.2 Geometry Transformations

Author Håvard Tveite

Contact havard.tveite@nmbu.no

Table of Contents

- *Transformations for simple styling (CLASS STYLE only)*
 - *bbox*
 - *centroid*
 - *end and start*
 - *vertices*
- *Labels (LABEL STYLE only)*
 - *labelpnt and labelpoly*
- *Expressions and advanced transformations (LAYER and CLASS STYLE)*
 - *Combining / chaining expressions*
 - *buffer*
 - *generalize ([shape], tolerance)*
 - *simplify([shape], tolerance)*
 - *simplifypt([shape], tolerance)*
 - *smoothsia ([shape], smoothing_size, smoothing_iterations, preprocessing)*
 - * *Tuning the behaviour of smoothsia*
 - * *Dataset resolution is too high*
 - * *Dataset resolution is too low*
 - * *Curves*

Geometry transformations return a new geometry. The purpose of a geometry transformation can be to achieve special effects for symbol rendering and labeling.

Geometry transformation is available at the *LAYER* level and the *STYLE* level. At the *LAYER* level (since 6.4), the original vector geometry (“real world” coordinates) is used. At the *STYLE* level, pixel coordinates are used.

It may be useful to apply pixel values also at the *LAYER* level, and that is possible. If *UNITS* is defined in the *LAYER*, the [map_cellsize] variable can be used to convert to pixel values at the *LAYER* level:

```
GEOMTRANSFORM (simplify([shape], [map_cellsize]*10))
```

Transformations for simple styling (*CLASS STYLE* only)

The following simple geometry transformations are available at the *CLASS STYLE* level:

- `bbox`
- `centroid`
- `end`
- `start`
- `vertices`

`bbox`

- *GEOMTRANSFORM* `bbox` returns the bounding box of the geometry.
 - *GEOMTRANSFORM* “`bbox`”

Note: Only available for *STYLE* in the *CLASS* context.

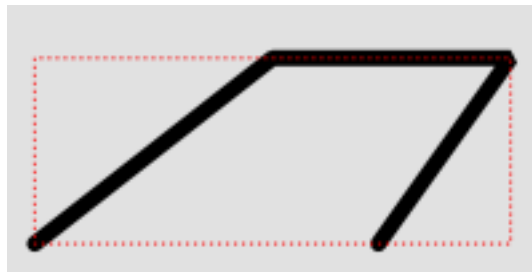


Fig. 4.17: Geomtransform `bbox`

Class definitions for the example:

```

CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 6
  END # STYLE
  STYLE
    GEOMTRANSFORM "bbox"
    OUTLINECOLOR 255 0 0
    WIDTH 1
    PATTERN 1 2 END
  END # STYLE
END # CLASS
  
```

`centroid`

- *GEOMTRANSFORM* `centroid` returns the centroid of the geometry.
 - *GEOMTRANSFORM* “`centroid`”

Note: Only available for *STYLE* in the *CLASS* context.

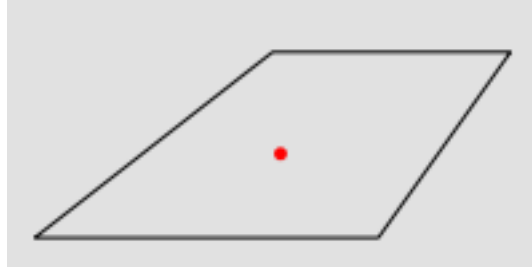


Fig. 4.18: Geomtransform centroid

Style definitions for the example.:

```
STYLE
  GEOMTRANSFORM "centroid"
  COLOR 255 0 0
  SYMBOL circlef
  SIZE 5
END # STYLE
```

Symbol definition for the *circlef* symbol:

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

end and start

- *GEOMTRANSFORM end* returns the end point of a line.
- *GEOMTRANSFORM start* returns the start point of a line.
 - GEOMTRANSFORM “start”
 - GEOMTRANSFORM “end” (since *END* is used to end objects in the map file, *end* must be embedded in quotes)

The direction of the line at the start / end point is available for rendering effects.

Note: Only available for *STYLE* in the *CLASS* context.

Class definitions for the example.

Lower part of the figure:

```
CLASS
  STYLE
    GEOMTRANSFORM "start"
    SYMBOL "circlef"
    COLOR 255 0 0
    SIZE 20
  END # STYLE
```

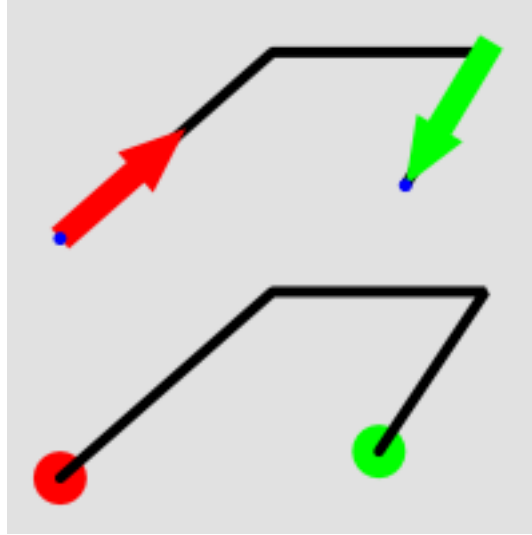


Fig. 4.19: Geomtransform start and end usage

```

STYLE
  COLOR 0 0 0
  WIDTH 4
END # STYLE
STYLE
  GEOMTRANSFORM "end"
  SYMBOL "circlef"
  COLOR 0 255 0
  SIZE 20
END # STYLE
END # CLASS

```

Upper part of the figure:

```

CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 4
  END # STYLE
  STYLE
    GEOMTRANSFORM "start"
    SYMBOL "startarrow"
    COLOR 255 0 0
    SIZE 20
    ANGLE auto
  END # STYLE
  STYLE
    GEOMTRANSFORM "start"
    SYMBOL "circlef"
    COLOR 0 0 255
    SIZE 5
  END # STYLE
  STYLE
    GEOMTRANSFORM "end"
    SYMBOL "endarrow"
    COLOR 0 255 0
  END # STYLE
END # CLASS

```

```
    SIZE 20
    ANGLE auto
  END # STYLE
  STYLE
    GEOMTRANSFORM "end"
    SYMBOL "circlef"
    COLOR 0 0 255
    SIZE 5
  END # STYLE
END # CLASS
```

The startarrow symbol definition (endarrow is the same, except for ANCHORPOINT (value for endarrow: 1 0.5)):

```
SYMBOL
  NAME "startarrow"
  TYPE vector
  FILLED true
  POINTS
    0 0.4
    3 0.4
    3 0
    5 0.8
    3 1.6
    3 1.2
    0 1.2
    0 0.4
  END # POINTS
  ANCHORPOINT 0 0.5
END # SYMBOL
```

vertices

- *GEOMTRANSFORM vertices* produces the set of vertices of a line (with direction information).
 - GEOMTRANSFORM “vertices”

Note: Only available for *STYLE* in the *CLASS* context.

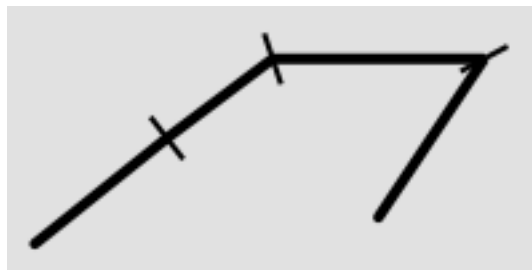


Fig. 4.20: Geomtransform vertices

Class definitions for the example:

```
CLASS
  STYLE
    COLOR 0 0 0
    WIDTH 4
```

```

END # STYLE
STYLE
  GEOMTRANSFORM "vertices"
  SYMBOL "vertline"
  COLOR 0 0 0
  WIDTH 2
  SIZE 20
  ANGLE AUTO
END # STYLE
END # CLASS

```

The vertline symbol definition:

```

SYMBOL
  NAME "vertline"
  TYPE vector
  POINTS
    0 0
    0 1
  END # POINTS
END # SYMBOL

```

Labels (*LABEL STYLE* only)

The following simple geometry transformations are available at the *LABEL STYLE* level:

- *labelpnt*
- *labelpoly*

These are used for label styling (background colour, background shadow, background frame).

Note: The result of using *labelpnt* is affected by the *LAYER LABELCACHE* setting. If *LABELCACHE* is *ON* (the default), the label will be shifted when a non-zero sized symbol is added using *labelpnt*.

labelpnt and *labelpoly*

- *GEOMTRANSFORM labelpnt* produces the geographic position the label is attached to. This corresponds to the center of the label text only if the label is in position *CC*.
 - *GEOMTRANSFORM "labelpnt"*
- *GEOMTRANSFORM labelpoly* produces a polygon that covers the label plus a 1 pixel padding.
 - *GEOMTRANSFORM "labelpoly"*

Note: Only available for *STYLE* in the *LABEL* context.

These transformations can be used to make background rectangles for labels and add symbols to the label points.

Class definitions for the example:

```

CLASS
  STYLE
    OUTLINECOLOR 255 255 204
  END # STYLE
  LABEL

```



Fig. 4.21: Geomtransform labelpnt and labelpoly

```

SIZE giant
POSITION UC
STYLE # shadow
  GEOMTRANSFORM "labelpoly"
  COLOR 153 153 153
  OFFSET 3 3
END # Style
STYLE # background
  GEOMTRANSFORM "labelpoly"
  COLOR 204 255 204
END # Style
STYLE # outline
  GEOMTRANSFORM "labelpoly"
  OUTLINECOLOR 0 0 255
  WIDTH 1
END # Style
STYLE
  GEOMTRANSFORM "labelpnt"
  SYMBOL 'circlef'
  COLOR 255 0 0
  SIZE 15
END # Style
END # Label
END # Class

```

Symbol definition for the *circlef* symbol:

```

SYMBOL
  NAME "circlef"
  TYPE ellipse
  FILLED true
  POINTS
    1 1
  END # POINTS
END # SYMBOL

```

Expressions and advanced transformations (*LAYER* and *CLASS STYLE*)

Combining / chaining expressions

A geometry transformation produces a geometry, and that geometry can be used as input to another geometry transformation. There are (at least) two ways to accomplish this. One is to combine basic geometry transformation expressions into more complex geometry transformation expressions, and another is to combine a geometry transformation expres-

sion at the *LAYER* level with a geometry transformation expressions or a simple geometry transformation at the *CLASS STYLE* level.

Combining geometry transformation expressions A geometry transformation expression contains a *[shape]* part. The *[shape]* part can be replaced by a geometry transformation expression.

For example:

```
GEOMTRANSFORM (simplify(buffer([shape], 20),10))
```

In this transformation, *buffer* is first applied on the geometry (*[shape]*). The resulting geometry is then used as input to *simplify*.

A style that demonstrates this:

```
STYLE
  GEOMTRANSFORM (simplify(buffer([shape], 20),10))
  OUTLINECOLOR 255 0 0
  WIDTH 2
END # STYLE
```

The result of this transformation is shown at the top of the following figure (red line). The original polygon is shown with a full black line and the buffer with a dashed black line.

Combining expressions with simple geometry transformations Simple geometry transformations are only available for *CLASS STYLE*, but can be combined with geometry transformation expressions at the *LAYER* level.

Excerpts from a layer definitions that does this kind of combination:

```
LAYER
  ...
  GEOMTRANSFORM (simplify(buffer([shape], 10),5))
  CLASS
    ...
    STYLE
      GEOMTRANSFORM "vertices"
      COLOR 255 102 102
      SYMBOL vertline
      SIZE 20
      WIDTH 2
      ANGLE auto
    END # STYLE
  END # CLASS
END # LAYER
```

The result of this transformation is shown at the bottom of the following figure (the red lines). The result of the *LAYER* level geomtransform is shown with a full black line. The original polygon is the same as the one used at the top of the figure.

buffer

- *GEOMTRANSFORM buffer* returns the buffer of the original geometry. The result is always a polygon geometry.
 - *GEOMTRANSFORM (buffer ([shape], buffersize))*

Note: Negative values for *buffersize* (setback) is not supported.

Note: Can be used at the *LAYER* level and for *STYLE* in the *CLASS* context.

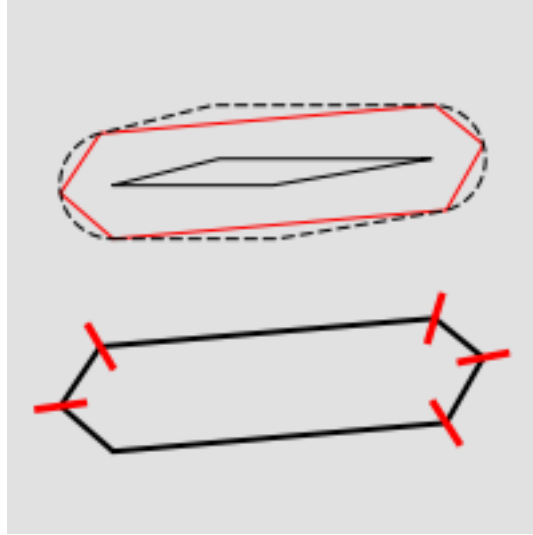


Fig. 4.22: Combining geomtransform expressions

Note: Buffer does not seem to work for point geometries.

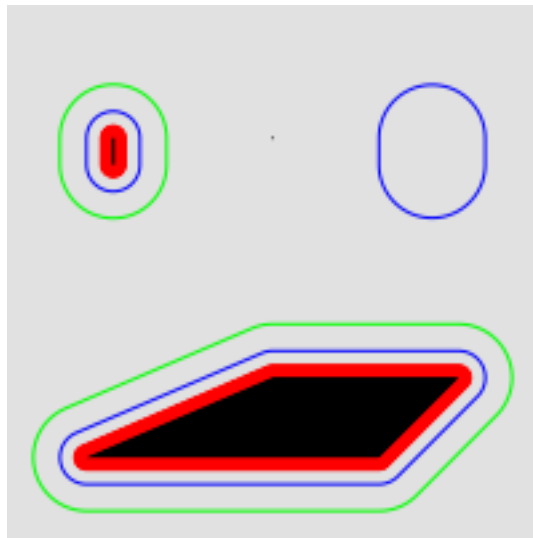


Fig. 4.23: Geomtransform buffer

Some class definitions for the example.

Lower part (polygon with buffers):

```

CLASS
  STYLE
    OUTLINECOLOR 0 255 0
    GEOMTRANSFORM (buffer([shape], 20))
    WIDTH 1
  END # STYLE
  STYLE
    OUTLINECOLOR 0 0 255
  
```

```

    GEOMTRANSFORM (buffer([shape], 10)) #
    WIDTH 1
  END # STYLE
  STYLE
    COLOR 255 0 0
    GEOMTRANSFORM (buffer([shape], 5)) #
  END # STYLE
  STYLE
    COLOR 0 0 0
  END # STYLE
END # CLASS

```

Upper right part (layer level geomtransform):

```

LAYER # line buffer layer
  STATUS DEFAULT
  TYPE LINE
  FEATURE
    POINTS
      80 70
      80 75
    END # Points
  END # Feature
  GEOMTRANSFORM (buffer([shape], 10))
  CLASS
    STYLE
      COLOR 0 0 255
    END # STYLE
  END # CLASS
END # LAYER

```

generalize ([shape], tolerance)

- *GEOMTRANSFORM generalize* simplifies a geometry ([shape]) in a way comparable to FME's ThinNoPoint algorithm. See <http://trac.osgeo.org/gdal/ticket/966> for more information.
 - *GEOMTRANSFORM (generalize([shape], tolerance))*

tolerance is mandatory, and is a specification of the maximum deviation allowed for the generalized line compared to the original line. A higher value for tolerance will give a more generalised / simplified line.

Note: Can be used at the *LAYER* level and for *STYLE* in the *CLASS* context.

Note: Depends on GEOS.

The figure below shows the result of applying *generalize* at the *STYLE* level with increasing values for tolerance (10 - green, 20 - blue and 40 - red).

One of the *STYLE* definitions for the example (tolerance 40):

```

STYLE
  GEOMTRANSFORM (generalize([shape], 40))
  COLOR 255 0 0
  WIDTH 1
  PATTERN 3 3 END
END # STYLE

```

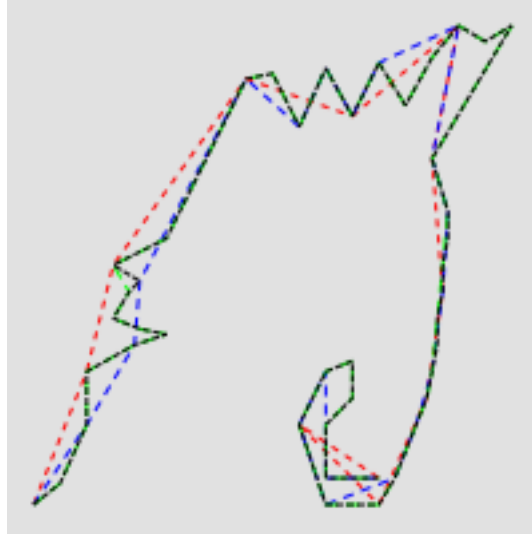


Fig. 4.24: Geomtransform generalize

simplify([shape], tolerance)

- *GEOMTRANSFORM simplify* simplifies a geometry ([shape]) using the standard Douglas-Peucker algorithm.
 - *GEOMTRANSFORM (simplify([shape], tolerance))*

tolerance is mandatory, and is a specification of the maximum deviation allowed for the generalized line compared to the original line. A higher value for tolerance will give a more generalised / simplified line.

Note: Can be used at the *LAYER* level and for *STYLE* in the *CLASS* context.

The figure below shows the result of applying *simplify* at the *STYLE* level with increasing values for tolerance (10 - green, 20 - blue and 40 - red).

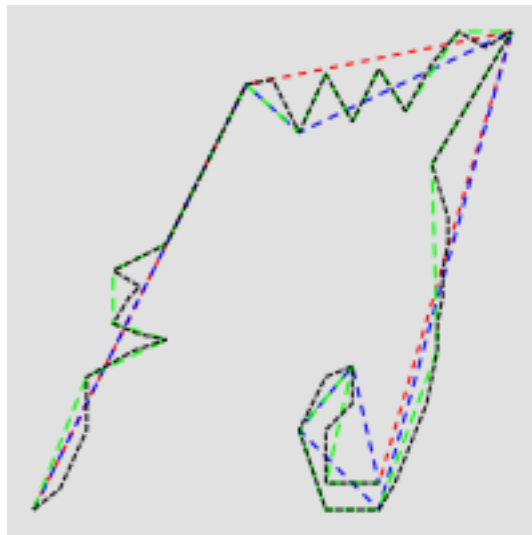


Fig. 4.25: Geomtransform simplify

One of the *STYLE* definitions for the example (tolerance 40):

```
STYLE
  GEOMTRANSFORM (simplify([shape], 40))
  COLOR 255 0 0
  WIDTH 1
  PATTERN 3 3 END
END # STYLE
```

simplifypt([shape], tolerance)

- *GEOMTRANSFORM simplifypt* simplifies a geometry ([shape]), ensuring that the result is a valid geometry having the same dimension and number of components as the input. *tolerance* must be non-negative.
 - *GEOMTRANSFORM (simplifypt([shape], tolerance))*

tolerance is mandatory, and is a specification of the maximum deviation allowed for the generalized line compared to the original line. A higher value for *tolerance* will give a more generalised / simplified line.

Note: Can be used at the *LAYER* level and for *STYLE* in the *CLASS* context.

The figure below shows the result of applying *simplifypt* at the *STYLE* level with increasing values for tolerance (10 - green, 20 - blue and 40 - red).

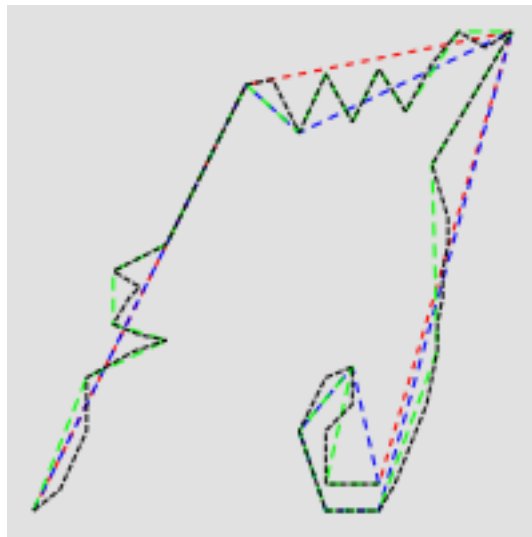


Fig. 4.26: Geomtransform simplifypt

One of the *STYLE* definitions for the example (tolerance 40):

```
STYLE
  GEOMTRANSFORM (simplifypt([shape], 40))
  COLOR 255 0 0
  WIDTH 1
  PATTERN 3 3 END
END # STYLE
```

`smoothsia` (`[shape]`, `smoothing_size`, `smoothing_iterations`, `preprocessing`)

- `GEOMTRANSFORM smoothsia` returns a smoothed version of a line.
 - `GEOMTRANSFORM (smoothsia ([shape], smoothing_size, smoothing_iterations, preprocessing))`

The following parameters are used:

- `shape` (mandatory). Specify the geometry to be used
- `smoothing_size` (optional). The window size (number of points) used by the algorithm. The default is 3.
- `smoothing_iterations` (optional). The number of iterations of the algorithm. The default is 1.
- `preprocessing` (optional). Preprocessing method to add more vertices to the geometry prior to smoothing, described below. There are two possible preprocessing methods:
 - * `all` Adds two intermediate vertices on each side of each original vertex. This is useful to preserve the general shape of the line with low resolution data.
 - * `angle` Add vertices at some specific places based on angle detection.

Note: Can be used at the `LAYER` level and for `STYLE` in the `CLASS` context.

Example of a simple layer definition:

```
LAYER NAME "my_layer"
TYPE LINE
STATUS DEFAULT
DATA roads.shp
GEOMTRANSFORM (smoothsia([shape], 3, 1, 'angle'))
CLASS
  STYLE
    WIDTH 2
    COLOR 255 0 0
  END
END
```

Here are some examples showing results with different parameter values.

Tuning the behaviour of `smoothsia` `smoothsia` has several parameters that can be used to tune its behaviour. The following sections describe some cases / possibilities.

Dataset resolution is too high If you are trying to smooth a line that has a very high resolution (high density of vertices at the current view scale), you may not get the expected result because the vertices are too dense for the smoothing window size. In this case you might want to simplify the geometries before the smoothing. You can combine smoothing and simplification in a single geomtransform for that:

```
GEOMTRANSFORM (smoothsia(simplifypt([shape], 10)))
```

See RFC 89: Layer Geomtransform for more info. Here's a visualization of the issue:

Dataset resolution is too low If you are trying to smooth a long line that has a low density of vertices, you may not get the expected result in some situations. You may lose some important parts of the geometry during the smoothing, for instance around acute angles. You can improve the result by enabling a preprocessing step to add intermediate vertices along the line prior to smoothing.

This behavior is controlled using the `all` value in the preprocessing argument of the `smoothsia` geomtransform:

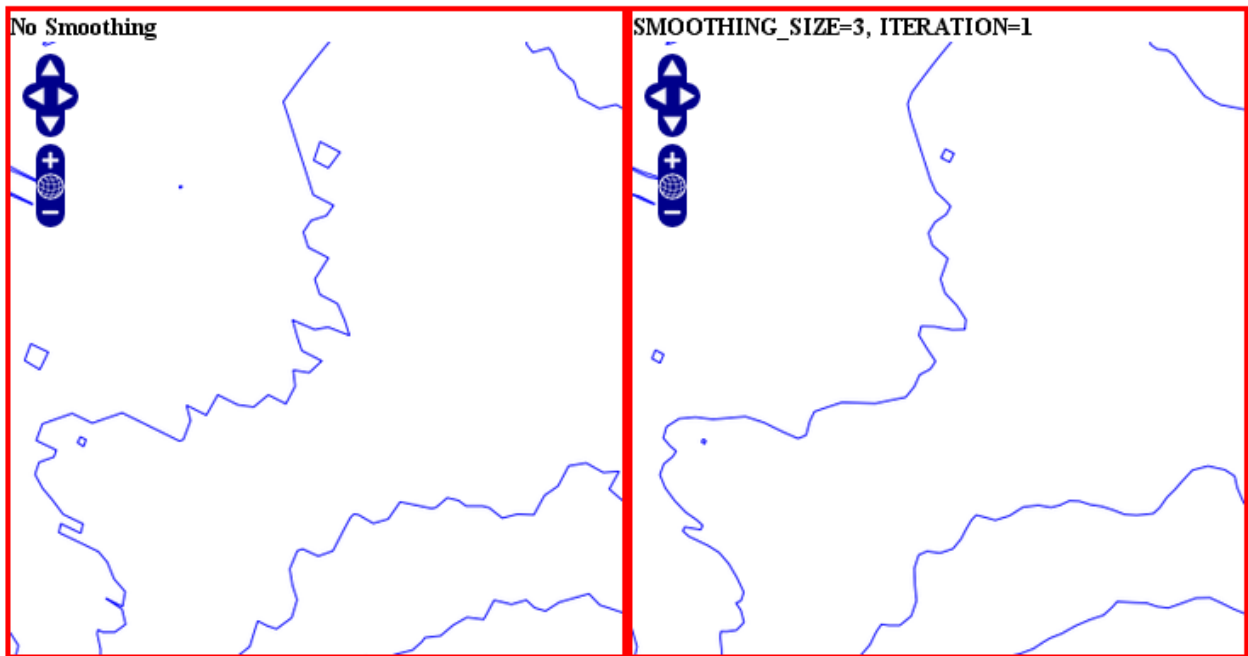


Fig. 4.27: Original geometry (left) and smoothsia with default parameters (right)

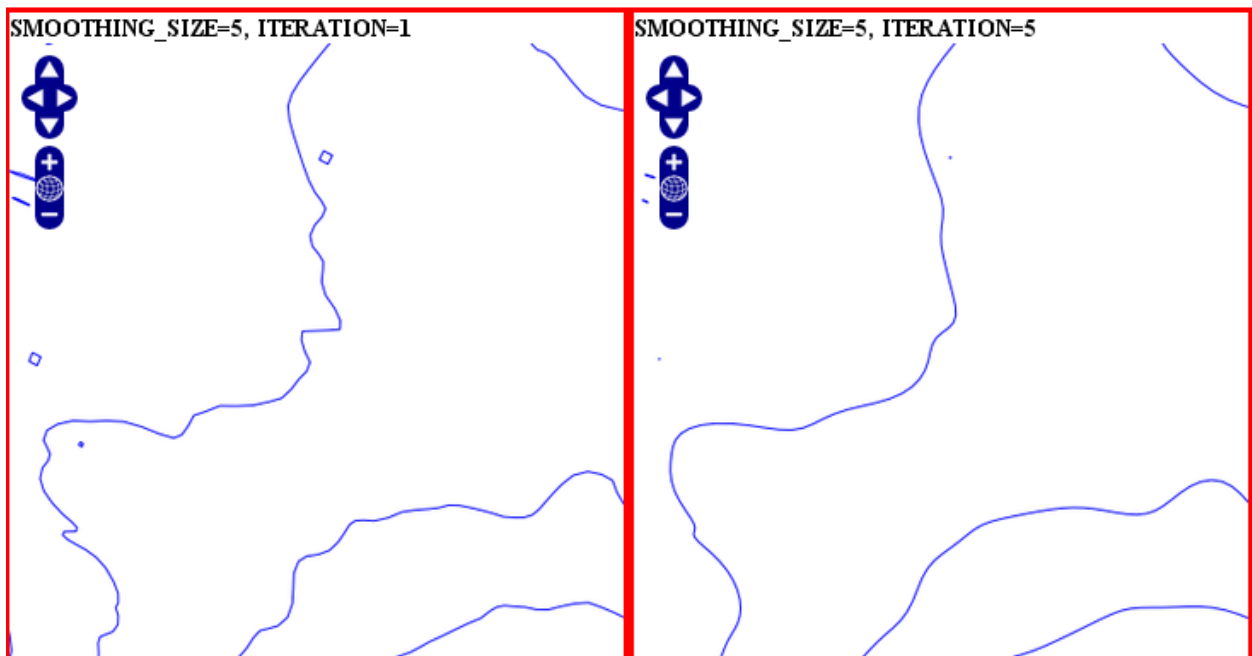


Fig. 4.28: Smoothsia - Larger window size (left) and larger window size with more iterations (right)

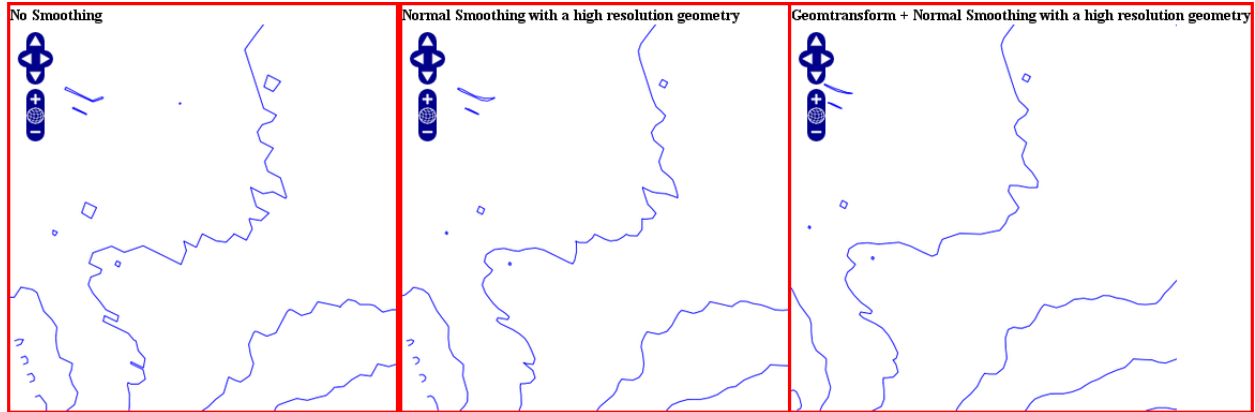


Fig. 4.29: High resolution geometry, smoothing and simplification

```
GEOMTRANSFORM (smoothsia([shape], 3, 1, 'all'))
```

This preprocessing will be performed before the smoothing. It adds 2 intermediate vertices on each side of each original vertex. This is useful if we really need to preserve the general shape of the low resolution line. Note that this might have an impact on the rendering since there will be more vertices in the output.

Here's a visualization of the issue:

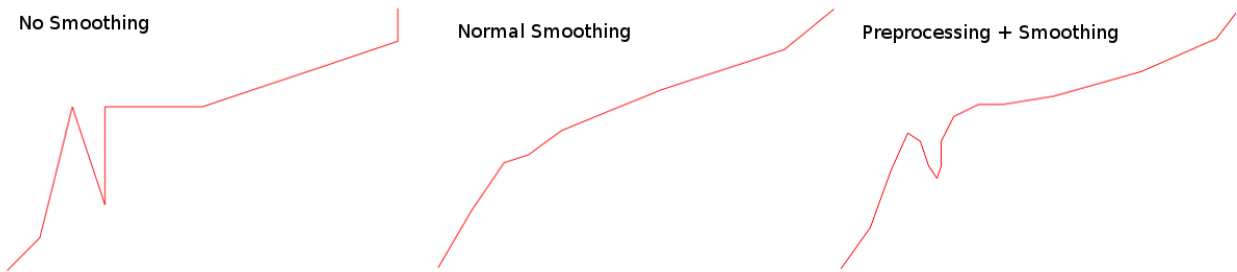


Fig. 4.30: Effects of normal smoothing and preprocessing

Curves The preprocessing step might not be appropriate for all cases since it can impact the smoothing result significantly. However, without it, you might notice bad smoothing for curved lines with large distances between the line vertices. See this example:

You can improve that by enabling another type of preprocessing: *angle*. This one will add points at some specific places based on angle detection to recognize the curves. Here's how you can enable it:

```
GEOMTRANSFORM (smoothsia([shape], 3, 1, 'angle'))
```

4.1.3 CLASS

BACKGROUND_COLOR [r] [g] [b] - *deprecated*

Deprecated since version 6.0: Use *CLASS STYLE*s.

COLOR [r] [g] [b]

Deprecated since version 6.0: Use *CLASS STYLE*s.

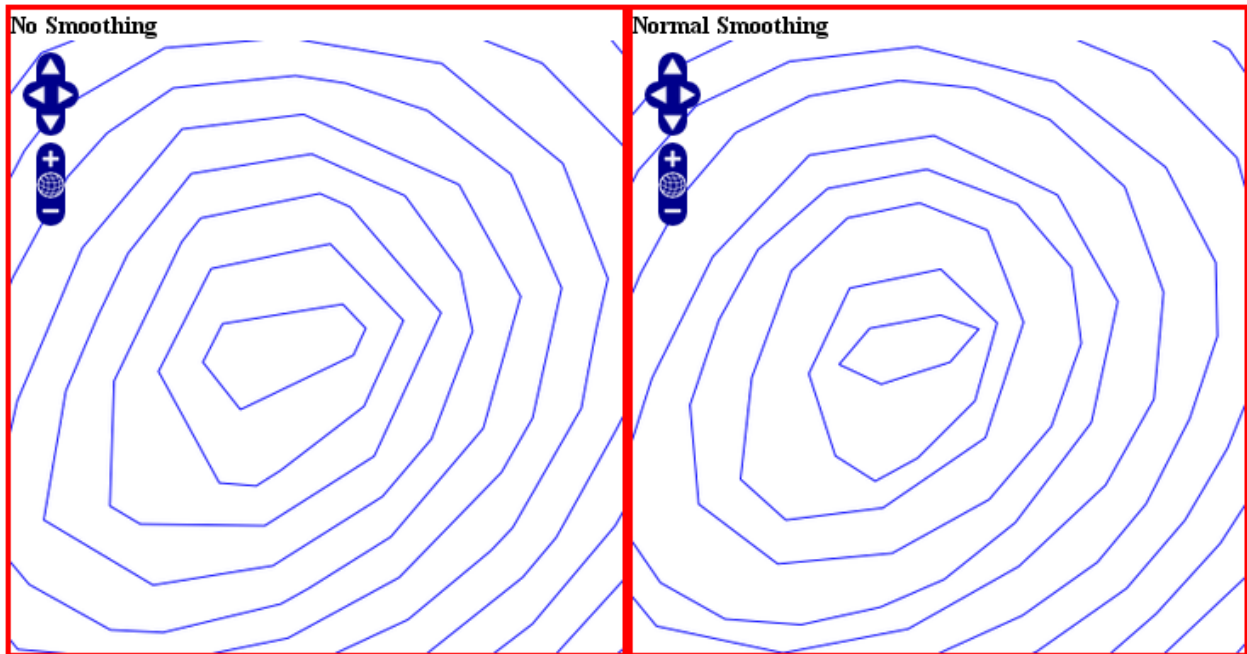


Fig. 4.31: Effects of normal smoothing (without preprocessing)

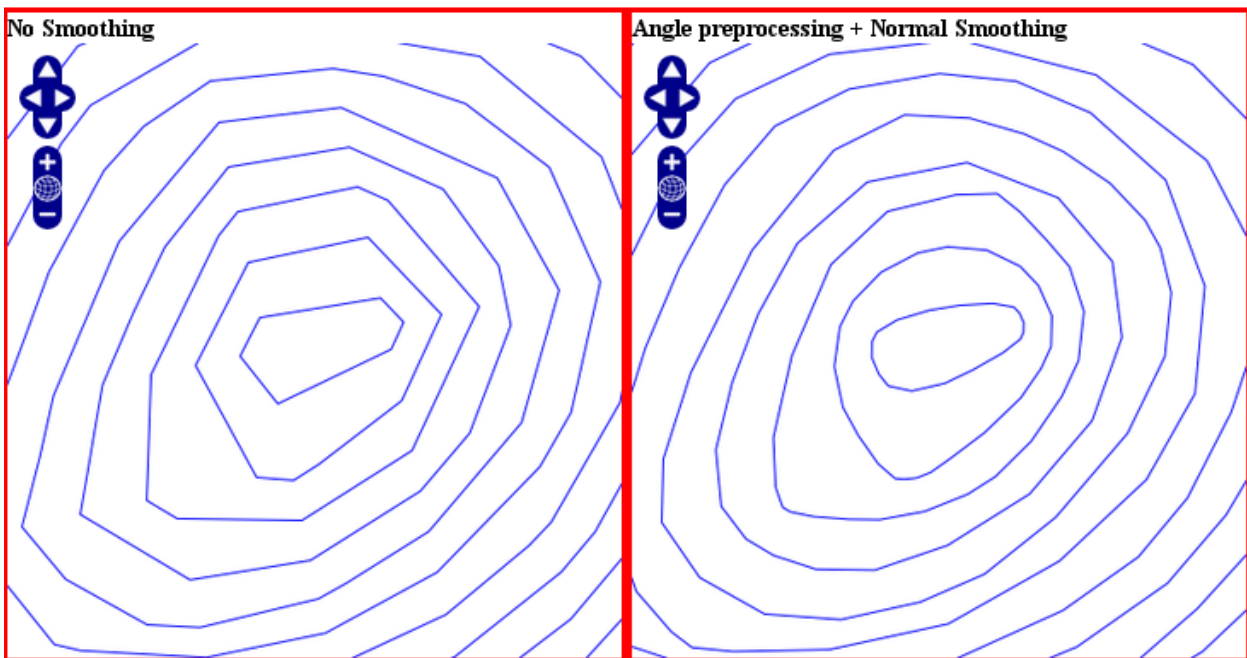


Fig. 4.32: The use of *angle* with *smoothsia*

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:

rfc28

EXPRESSION [string] Four types of expressions are now supported to define which class a feature belongs to: String comparisons, regular expressions, logical expressions, and string functions (see *Expressions*). 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 strings must be quoted if they contain special characters. (As a matter of good habit, it is recommended that you quote all strings). The attribute to use for comparison is defined in the *LAYER CLASSITEM* parameter.
- Regular expression are limited using slashes (/regex/). The attribute to use for comparison is defined in the *LAYER CLASSITEM* parameter.
- Logical expressions allow the building of 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]”. Attribute 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, in, ~, ~*. As one might expect, this level of complexity is slower to process.

- One string function exists: length(). It computes the length of a string:

```
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. A class can contain multiple labels (since MapServer 6.2).

LEADER Signals the start of a *LEADER* object. Use this along with a *LABEL* object to create label leader lines.

New in version 6.2.

MAXSCALEDENOM [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 *MAXSCALE* parameter.

See also:

Map Scale

MAXSIZE [integer]

Deprecated since version 6.0: Use *CLASS STYLE*s.

MINSCALEDENOM [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 *MINSCALE* parameter.

See also:

Map Scale

MINSIZE [integer]

Deprecated since version 6.0: Use *CLASS STYLE*s.

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

OUTLINECOLOR [r] [g] [b]

Deprecated since version 6.0: Use *CLASS STYLE*s.

SIZE [integer]

Deprecated since version 6.0: Use *CLASS STYLE*s.

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. Multiple styles can be used create complex symbols (by overlay/stacking). See *Cartographical Symbol Construction with MapServer* for more information on advanced symbol construction.

SYMBOL [integer|string|filename]

Deprecated since version 6.0: Use *CLASS STYLE*s.

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

TEXT [stringexpression] Text to label features in this class with. This overrides values obtained from the *LAYER LABELITEM*. The string can contain references to feature attributes. This allows you to concatenate multiple attributes into a single label. You can for example concatenate the attributes *FIRSTNAME* and *LASTNAME* like this:

```
TEXT ' [FIRSTNAME] [LASTNAME] '
```

More advanced *Expressions* can be used to specify the labels. Since version 6.0, there are functions available for formatting numbers:

```
TEXT ("Area is: " + tostring([area], "%.2f"))
```

VALIDATION Signals the start of a *VALIDATION* block.

As of MapServer 5.4.0, *VALIDATION* blocks are the preferred mechanism for specifying validation patterns for CGI param runtime substitutions. See *Run-time Substitution*.

4.1.4 CLUSTER

Table of Contents

- *CLUSTER*
 - *Description*
 - *Supported Layer Types*
 - *Mapfile Parameters*
 - *Supported Processing Options*
 - *Mapfile Snippet*
 - *Feature attributes*
 - *PHP MapScript Usage*
 - *Example: Clustering Railway Stations*

Description

Since version 6.0, MapServer has the ability to combine multiple features from a point layer into single (aggregated) features based on their relative positions. Only *POINT* layers are supported. This feature was added through rfc69.

Supported Layer Types

POINT

Mapfile Parameters

MAXDISTANCE [**double**] Specifies the distance of the search region (rectangle or ellipse) in pixel positions.

REGION [**string**] Defines the search region around a feature in which the neighbouring features are negotiated. Can be 'rectangle' or 'ellipse'.

BUFFER [**double**] Defines a buffer region around the map extent in pixels. Default is 0. Using a buffer allows that the neighbouring shapes around the map are also considered during the cluster creation.

GROUP [**string**] This expression evaluates to a string and only the features that have the same group value are negotiated. This parameter can be omitted. The evaluated group value is available in the 'Cluster:Group' feature attribute.

FILTER [**string**] We can define the **FILTER** expression filter some of the features from the final output. This expression evaluates to a boolean value and if this value is false the corresponding shape is filtered out. This expression is evaluated after the feature negotiation is completed, therefore the 'Cluster:FeatureCount' parameter can also be used, which provides the option to filter the shapes having too many or to few neighbors within the search region.

Supported Processing Options

The following processing options can be used with the cluster layers:

CLUSTER_GET_ALL_SHAPES Return all shapes contained by a cluster instead of a single shape. This setting affects both the drawing and the query processing.

Mapfile Snippet

```
LAYER
  NAME "my-cluster"
  TYPE POINT
  ...
  CLUSTER
    MAXDISTANCE 20 # in pixels
    REGION "ellipse" # can be rectangle or ellipse
    GROUP (expression) # an expression to create separate groups for each value
    FILTER (expression) # a logical expression to specify the grouping condition
  END
  LABELITEM "Cluster:FeatureCount"
  CLASS
    ...
    LABEL
    ...
  END
END
...
```

Feature attributes

The clustered layer itself provides the following aggregated attributes:

1. Cluster:FeatureCount - count of the features in the clustered shape
2. Cluster:Group - The group value of the cluster (to which the group expression is evaluated)

These attributes (in addition to the attributes provided by the original data source) can be used to configure the labels of the features and can also be used in expressions. The ITEMS processing option can be used to specify a subset of the attributes from the original layer in the query operations according to the user's preference.

We can use simple aggregate functions (Min, Max, Sum, Count) to specify how the clustered attribute should be calculated from the original attributes. The aggregate function should be specified as a prefix separated by ':' in the attribute definition, like: [Max:itemname]. If we don't specify aggregate functions for the source layer attributes, then the actual value of the cluster attribute will be non-deterministic if the cluster contains multiple shapes with different values. The Count aggregate function in fact provides the same value as Cluster:FeatureCount.

PHP MapScript Usage

The *CLUSTER* object is exposed through PHP MapScript. An example follows:

```
$map = ms_newMapobj("/var/www/vhosts/mysite/httpdocs/test.map");
$layer1=$map->getLayerByName("test1");
$layer1->cluster;
```

Example: Clustering Railway Stations

The following example uses a point datasource, in this case in KML format, to display clusters of railway stations. Two classes are used: one to style and label the cluster, and one to style and label the single railway station.

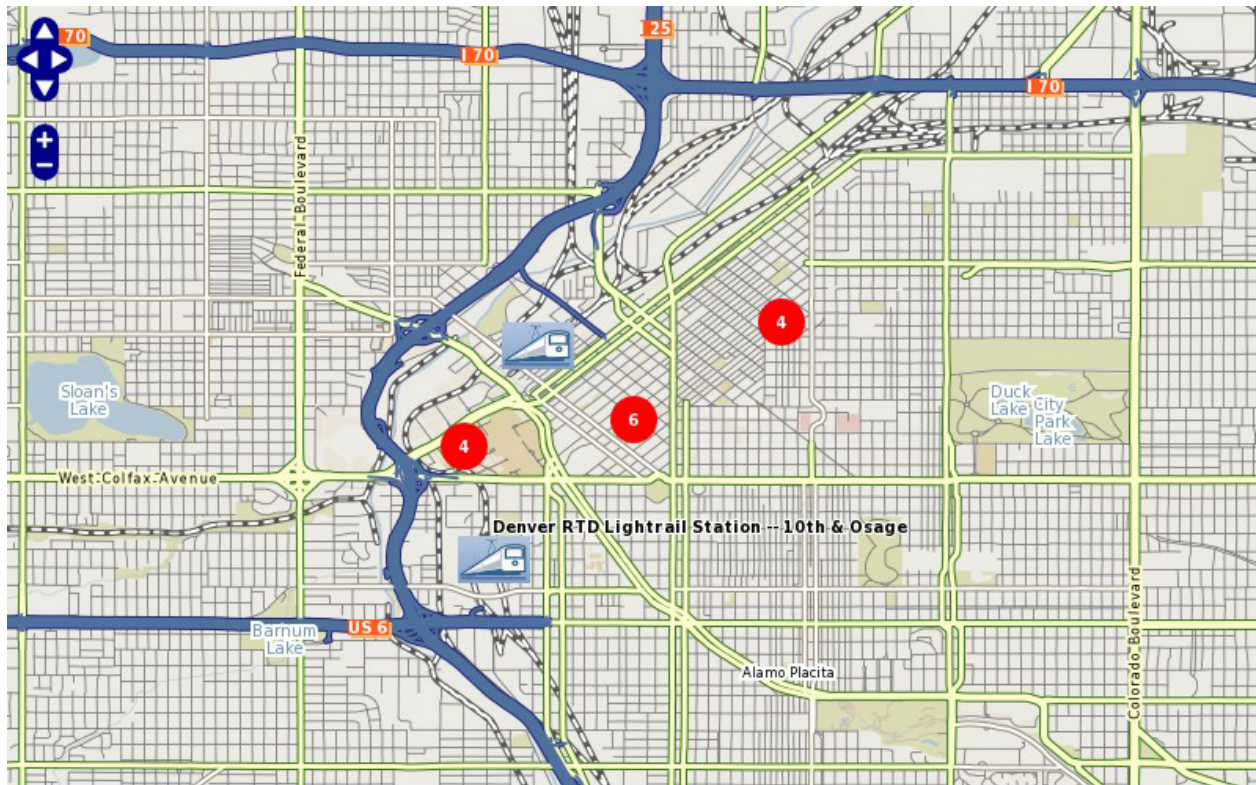
Note: Since we can't declare 2 labelitems, for the single railway class we use the *TEXT* parameter to label the station.

Mapfile Layer

```
#####
# Lightrail Stations
#####
SYMBOL
  NAME "lightrail"
  TYPE PIXMAP
  IMAGE "../etc/lightrail.png"
END
LAYER
  NAME "lightrail"
  GROUP "default"
  STATUS DEFAULT
  TYPE POINT
  CONNECTIONTYPE OGR
  CONNECTION "lightrail-stations.kml"
  DATA "lightrail-stations"
  LABELITEM "Cluster:FeatureCount"
  CLASSITEM "Cluster:FeatureCount"
  #####
  # Define the cluster object
  #####
  CLUSTER
    MAXDISTANCE 50
    REGION "ellipse"
  END
  #####
  # Class1: For the cluster symbol
  #####
  CLASS
    NAME "Clustered Lightrail Stations"
    EXPRESSION ("[Cluster:FeatureCount]" != "1")
    STYLE
      SIZE 30
      SYMBOL "citycircle"
      COLOR 255 0 0
    END
    LABEL
      FONT scb
      TYPE TRUETYPE
      SIZE 8
      COLOR 255 255 255
      ALIGN CENTER
      PRIORITY 10
      BUFFER 1
      PARTIALS TRUE
      POSITION cc
    END
  END
END
```

```
END
#####
# Class2: For the single station
#####
CLASS
  NAME "Lightrail Stations"
  EXPRESSION "1"
  STYLE
    SIZE 30
    SYMBOL "lightrail"
  END
  TEXT "[Name]"
  LABEL
    FONT scb
    TYPE TRUETYPE
    SIZE 8
    COLOR 0 0 0
    OUTLINECOLOR 255 255 255
    ALIGN CENTER
    PRIORITY 9
    BUFFER 1
    PARTIALS FALSE
    POSITION ur
  END
END
# the following is used for a query
TOLERANCE 50
UNITS PIXELS
HEADER "../htdocs/templates/cluster_header.html"
FOOTER "../htdocs/templates/cluster_footer.html"
TEMPLATE "../htdocs/templates/cluster_query.html"
END
```


Map Image



4.1.5 Display of International Characters in MapServer

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision: 12506 \$

Date \$Date: 2011-08-29 14:26:49 +0200 (Mon, 29 Aug 2011) \$

Table of Contents

- *Display of International Characters in MapServer*
 - *Credit*
 - *Related Links*
 - *Requirements*
 - *How to Enable in Your Mapfile*
 - * *Step 1: Verify ICONV Support and MapServer Version*
 - * *Step 2: Verify That Your Files' Encoding is Supported by ICONV*
 - * *Step 3: Add ENCODING Parameter to your LABEL Object*
 - * *Step 4: Test with the shp2img utility*
 - *Example Using PHP MapScript*
 - *Notes*

Credit

The following functionality was added to MapServer 4.4.0 as a part of a project sponsored by the Information-technology Promotion Agency (IPA), in Japan. Project members included: Venkatesh Raghavan, Masumoto Shinji, Nonogaki Susumu, Nemoto Tatsuya, Hirai Naoki (Osaka City University, Japan), Mario Basa, Hagiwara Akira, Niwa Makoto, Mori Toru (Orkney Inc., Japan), and Hattori Norihiro (E-Solution Service, Inc., Japan).

Related Links

- MapServer [ticket:858](#)

Requirements

- MapServer \geq 4.4.0
- MapServer compiled with the libiconv library

How to Enable in Your Mapfile

The mapfile *LABEL* object's parameter named *ENCODING* can be used to convert strings from its original encoding system into one that can be understood by the True Type Fonts. The *ENCODING* parameter accepts the encoding name as its parameter.

MapServer uses GNU's libiconv library (<http://www.gnu.org/software/libiconv/>) to deal with encodings. The libiconv web site has a list of supported encodings. One can also use the “iconv -l” command on a system with libiconv installed to get the complete list of supported encodings on that specific system.

So, theoretically, every string with an encoding system supported by libiconv can be displayed as labels in MapServer as long as it has a matching font-set.

Step 1: Verify ICONV Support and MapServer Version

Execute “`mapserv -v`” at the commandline, and verify that your MapServer version \geq 4.4.0 and it includes “`SUPPORTS=ICONV`”, such as:

```
> mapserv -v

MapServer version 5.6.5 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG
OUTPUT=WBMP OUTPUT=PDF OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ
SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=ICONV SUPPORTS=FRIBIDI
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
SUPPORTS=TILECACHE INPUT=JPEG INPUT=POSTGIS INPUT=OGR INPUT=GDAL
INPUT=SHAPEFILE
```

Step 2: Verify That Your Files' Encoding is Supported by ICONV

Since MapServer uses the libiconv library to handle encodings, you can check the list of supported encodings here: <http://www.gnu.org/software/libiconv/>

Unix users can also use the `iconv -l` command on a system with libiconv installed to get the complete list of supported encodings on that specific system.

Step 3: Add ENCODING Parameter to your LABEL Object

Now you can simply add the ENCODING parameter to your mapfile LAYER object, such as:

```
MAP
...
LAYER
...
CLASS
...
LABEL
...
ENCODING "SHIFT_JIS"
END
END
END
END
```

One of the benefits of having an “ENCODING” parameter within the LABEL object is that different LAYERS with different encoding systems can be combined together and display labels within a single map. For example, labels from a Layer using Shapefile as its source which contains attributes in SHIFT-JIS can be combined with a Layer from a PostGIS database server with EUC-JP attributes. A sample Mapfile can look like this:

```
LAYER
NAME "chimei"
DATA chimei
STATUS DEFAULT
TYPE POINT
LABELITEM "NAMAЕ"
CLASS
NAME "CHIMEI"
STYLE
COLOR 10 100 100
END
LABEL
TYPE TRUETYPE
FONT kochi-gothic
COLOR 220 20 20
SIZE 10
POSITION CL
PARTIALS FALSE
BUFFER 0
ENCODING SHIFT_JIS
END
END
END

LAYER
NAME "chimeipg"
CONNECTION "user=username password=password dbname=gis host=localhost port=5432"
CONNECTIONTYPE postgis
DATA "the_geom from chimei"
STATUS DEFAULT
TYPE POINT
LABELITEM "NAMAЕ"
CLASS
NAME "CHIMEI PG"
STYLE
COLOR 10 100 100
```

```

END
LABEL
  TYPE TRUETYPE
  FONT kochi-mincho
  COLOR 20 220 20
  SIZE 10
  POSITION CL
  PARTIALS FALSE
  BUFFER 0
  ENCODING EUC-JP
END
END
END

```

Step 4: Test with the shp2img utility

- see *shp2img commandline utility*

Example Using PHP MapScript

For PHP Mapscript, the *Encoding* parameter is included in the LabelObj Class, so that the encoding parameter of a layer can be modified such as:

```

// Loading the php_mapscript library
dl("php_mapscript.so");

// Loading the map file
$map = ms_newMapObj("example.map");

// get the desired layer
$layer = $map->getLayerByName("chimei");

// get the layer's class object
$class = $layer->getClass(0);

// get the class object's label object
$clabel= $class->label;

// get encoding parameter
$encode_str = $clabel->encoding;
print "Encoding = ".$encode_str."\n";

// set encoding parameter
$clabel->set("encoding", "UTF-8");

```

Notes

Note: During initial implementation, this functionality was tested using the different Japanese encoding systems: Shift-JIS, EUC-JP, UTF-8, as well as Thai's TIS-620 encoding system.

Examples of encodings for the Latin alphabet supported by libiconv are: ISO-8859-1 (Latin alphabet No. 1 - also known as LATIN-1 - western European languages), ISO-8859-2 (Latin alphabet No. 2 - also known as LATIN-2 - eastern European languages), CP1252 (Microsoft Windows Latin alphabet encoding - English and some other Western languages).

4.1.6 Expressions

Author Dirk Tilger

Contact dirk at MIRIUP.DE

Author Umberto Nicoletti

Contact umberto.nicoletti at gmail.com

Revision \$Revision\$

Date \$Date\$

Last Updated 2011/06/30

Contents

- *Expressions*
 - *Introduction*
 - * *String quotation*
 - * *Quotes escaping in strings*
 - * *Using attributes*
 - * *Character encoding*
 - *Expression Types*
 - * *String comparison (equality)*
 - * *Regular expression comparison*
 - * *List expressions*
 - *“MapServer expressions”*
 - * *Logical expressions*
 - * *String expressions that return a logical value*
 - * *Arithmetic expressions that return a logical value*
 - * *Spatial expressions that return a logical value (GEOS)*
 - * *String operations that return a string*
 - * *Functions that return a string*
 - * *String functions that return a number*
 - * *Arithmetic operations and functions that return a number*
 - * *Spatial functions that return a number (GEOS)*
 - * *Spatial functions that return a shape (GEOS)*
 - * *Temporal expressions*

Introduction

As of version 6.0, expressions are used in four places:

- In *LAYER FILTER* to specify the features of the dataset that are to be included in the layer.
- In *CLASS EXPRESSION* to specify to which features of the dataset the *CLASS* applies to.
- In *CLASS TEXT* to specify text for labeling features.
- In *STYLE GEOMTRANSFORM*.

String quotation

Strings can be quoted using single or double quotes:

```
'This is a string'
"And this is also a string"
```

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:

```
"National \"hero\" statue"
```

To escape a single quote use the following sequence instead:

```
"National \'hero\' statue"
```

Starting with MapServer 6.0 you don't need to escape single quotes within double quoted strings and you don't need to escape double quotes within single quoted strings. In 6.0 you can also write the string as follows:

```
'National "hero" statue'
...
```

To escape a single quote use the following sequence instead:

```
"National 'hero' statue"
```

Using attributes

Attribute values can be referenced in the Map file and used in expressions. Attribute references are case sensitive and can be used in the following types of expressions:

- In *LAYER FILTER*
- In *CLASS EXPRESSION*
- In *CLASS TEXT*

Referencing an attribute is done by enclosing the attribute name in square brackets, like this: [ATTRIBUTENAME]. Then, every occurrence of “[ATTRIBUTENAME]” will be replaced by the actual value of the attribute “ATTRIBUTE-NAME”.

Example: The data set of our layer has the attribute “BUILDING_NAME”. We want the value of this attribute to appear inside a string. This can be accomplished as follows (single or double quotes):

```
'The [BUILDING_NAME] building'
```

For the building which has its BUILDING_NAME attribute set to “Historical Museum”, the resulting string is:

```
'The Historical Museum building'
```

For *Raster Data* layers special attributes have been defined that can be used for classification, for example:

- [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.

Character encoding

With MapServer there is no way to specify the character encoding of the mapfile or the layer data sources, so MapServer can't do the character encoding translation. If the character encoding of the data source is not the same as the character encoding of the map file, they could be converted to a common encoding.

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 (equality)

String comparison means, as the name suggests, that attribute values are checked if they are equal to some value. String comparisons 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* in the *LAYER* object 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 reasons explained later, the values for both *CLASSITEM* and *FILTERITEM* should start with neither a ‘/’ nor 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”) on Unix systems.

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 for string comparison, when using a regular expressions, *FILTERITEM* (*LAYER FILTER*) or *CLASSITEM* (*CLASS EXPRESSION*) has to be defined if the items are not included in the *LAYER FILTER* or *CLASS EXPRESSION*.

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 meanings 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.

MapServer supports two regex operators:

- `~` case sensitive regular expression
- `~*` case insensitive regular expression

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: For *FILTER*, the regular expression is case-sensitive, thus records having “Hotel” in them would not have matched.

Example: Match records that have a value from 2000 to 2010 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]*$/
```

Example: Match all the features where the *name* attribute ends with “by”, “BY”, “By” or “bY” (case insensitive matching):

```
EXPRESSION ('[name]' ~* 'by$')
```

Example: Match all the features where the *rdname* attribute starts with “Main”.

```
LAYER
  ...
  CLASSITEM 'rdname'
  CLASS
    EXPRESSION /^Main.*$/
```

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.

List expressions

New in version 6.4.

List expressions (see *rfc95*) are a performant way to compare a string attribute to a list of multiple possible values. Their behavior duplicates the existing regex or mapserver expressions, however they are significantly more performant. To activate them enclose a comma separated list of values between {}, **without** adding quotes or extra spaces.

```
LAYER
  NAME 'list-example'
  CLASSITEM 'roadtype'
  ...
  CLASS
    EXPRESSION {motorway,trunk}
    #equivalent to regex EXPRESSION /motorway|trunk/
    #equivalent to mapserver EXPRESSION ("roadtype" IN "motorway,trunk")
    ...
  END
  CLASS
    EXPRESSION {primary,secondary}
    ...
  END
END
```

Warning: List expressions do not support quote escaping, or attribute values that contain a comma in them.

“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 that allow certain arithmetic, string and time operations.

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

- ((Expression1) AND (Expression2))
((Expression1) && (Expression2))

returns true when both of the logical expressions (Expression1 and Expression2) are true.

- ((Expression1) OR (Expression2))
((Expression1) || (Expression2))
returns true when at least one of the logical expressions (Expression1 or Expression2) is true.
- NOT (Expression1)
! (Expression1)
returns true when Expression1 is false.

String expressions that return a logical value

Syntactically, a string is something encapsulated in single or double quotes.

- (“String1” eq “String2”)
(“String1” == “String2”) - deprecated since 6.0
(“String1” = “String2”)
returns true when the strings are equal. Case sensitive.
- (“String1” =* “String2”)
returns true when the strings are equal. Case insensitive.
- (“String1” != “String2”)
(“String1” ne “String2”)
returns true when the strings are not equal.
- (“String1” < “String2”)
(“String1” lt “String2”)
returns true when “String1” is lexicographically smaller than “String2”
- (“String1” > “String2”)
(“String1” gt “String2”)
returns true when “String1” is lexicographically larger than “String2”.
- (“String1” <= “String2”)
(“String1” le “String2”)
returns true when “String1” is lexicographically smaller than or equal to “String2”
- (“String1” >= “String2”)
(“String1” ge “String2”)
returns true when “String1” is lexicographically larger than or equal to “String2”.
- (“String1” IN “token1,token2,...,tokenN”)
returns true when “String1” is equal to one of the given tokens.

Note: The separator for the tokens is the comma. That means that there can not be unnecessary white space in the list and that tokens that have commas in them cannot be compared.

- (“String1” ~ “regexp”)
returns true when “String1” matches the regular expression “regexp”. This operation is identical to the regular expression matching described earlier.
- (“String1” ~* “regexp”)
returns true when “String1” matches the regular expression “regexp” (case insensitive). This operation is identical to the regular expression matching described earlier.

Arithmetic expressions that return a logical value

The basic element for arithmetic operations is the number. Arithmetic operations that return numbers will be covered in the next section.

- (n1 eq n2)
(n1 == n2) - deprecated since 6.0
(n1 = n2)
returns true when the numbers are equal.
- (n1 != n2)
(n1 ne n2)
returns true when the numbers are not equal.
- (n1 < n2)
(n1 lt n2)
returns true when n1 is smaller than n2.
- (n1 > n2)
(n1 gt n2)
returns true when n1 is larger than n2.
- (n1 <= n2)
(n1 le n2)
returns true when n1 is smaller than or equal to n2.
- (n1 >= n2)
(n1 ge n2)
returns true when n1 is larger than or equal to n2.
- (n1 IN “number1,number2,...,numberN”)
returns true when n1 is equal to one of the given numbers.

Spatial expressions that return a logical value (GEOS)

- (shape1 eq shape2)
returns true if shape1 and shape2 are equal

- (shape1 intersects shape2)
returns true if shape1 and shape2 intersect
New in version 6.0.
- (shape1 disjoint shape2)
returns true if shape1 and shape2 are disjoint
New in version 6.0.
- (shape1 touches shape2)
returns true if shape1 and shape2 touch
New in version 6.0.
- (shape1 overlaps shape2)
returns true if shape1 and shape2 overlap
New in version 6.0.
- (shape1 crosses shape2)
returns true if shape1 and shape2 cross
New in version 6.0.
- (shape1 within shape2)
returns true if shape1 is within shape2
New in version 6.0.
- (shape1 contains shape2)
returns true if shape1 contains shape2
New in version 6.0.
- (shape1 dwithin shape2)
returns true if the distance between shape1 and shape2 is equal to 0
New in version 6.0.
- (shape1 beyond shape2)
returns true if the distance between shape1 and shape2 is greater than 0
New in version 6.0.

String operations that return a string

- “String1” + “String2”
returns “String1String2”, that is, the two strings concatenated to each other.

Functions that return a string

- tostring (n1, “Format1”)
uses “Format1” to format the number n1 (C style formatting - sprintf).
New in version 6.0.

- `commify ("String1")`
adds thousands separators (commas) to a long number to make it more readable
New in version 6.0.

String functions that return a number

- `length ("String1")`
returns the number of characters of "String1"

Arithmetic operations and functions that return a number

- `round (n1 , n2)`
returns n1 rounded to a multiple of n2: $n2 * \text{round}(n1/n2)$
New in version 6.0.
- `n1 + n2`
returns the sum of n1 and n2
- `n1 - n2`
returns n2 subtracted from n1
- `n1 * n2`
returns n1 multiplied with n2
- `n1 / n2`
returns n1 divided by n2
- `-n1`
returns n1 negated
- `n1 ^ n2`
returns n1 to the power of n2

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 )
```

would return true, but

```
( 5 - 5 )
```

would return false.

Spatial functions that return a number (GEOS)

- `area (shape1)`
returns the area of shape1
New in version 6.0.

Spatial functions that return a shape (GEOS)

- `fromtext ("String1")`
returns the shape corresponding to String1 (WKT - well known text)

```
fromText ('POINT(500000 500000)')
```

New in version 6.0.

- `buffer (shape1 , n1)`
returns the shape that results when shape1 is buffered with bufferdistance n1
New in version 6.0.
- `difference (shape1 , shape2)`
returns the shape that results when the common area of shape1 and shape2 is subtracted from shape1
New in version 6.0.

Temporal expressions

MapServer uses an internal time type to do comparison. To convert a string into this time type it will check the list below from the top and down to check if the specified time matches, and if so, it will do the conversion. The following are integer values: **YYYY** - year, **MM** - month, **DD** - date, **hh** - hours, **mm** - minutes, **ss** - seconds. The following are character elements of the format: **-** (dash) - date separator, **:** (colon) - time separator, **T** - marks the start of the time component (ISO 8601), **space** - marks the end of the date and start of the time component, **Z** - zulu time (0 UTC offset).

- YYYY-MM-DDThh:mm:ssZ
- YYYY-MM-DDThh:mm:ss
- YYYY-MM-DD hh:mm:ss
- YYYY-MM-DDThh:mm
- YYYY-MM-DD hh:mm
- YYYY-MM-DDThh
- YYYY-MM-DD hh
- YYYY-MM-DD
- YYYY-MM
- YYYY
- Thh:mm:ssZ
- Thh:mm:ss

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

- `(t1 eq t2)`
`(t1 == t2)` - deprecated since 6.0
`(t1 = t2)`
returns true when the times are equal.

- (t1 != t2)
(t1 ne t2)
returns true when the times are not equal.
- (t1 < t2)
(t1 lt t2)
returns true when t1 is earlier than t2
- (t1 > t2)
(t1 gt t2)
returns true when t1 is later than t2.
- (t1 <= t2)
(t1 le t2)
returns true when t1 is earlier than or equal to t2
- (t1 >= t2)
(t1 ge t2)
returns true when t1 is later than or equal to t2.

4.1.7 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.

4.1.8 FONTSET

Author Kari Guerts

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision\$

Date \$Date\$

Last Updated 2008/10/08

Contents

- *FONTSET*
 - *Format of the fontset file*

FONTSET is a *MAP* parameter. The syntax is:

```
FONTSET [filename]
```

Where *filename* gives the location of the fontset file of the system. The location of the system fontset file could for instance be `/usr/share/fonts/truetype/font.list` (Debian). The location can be specified using a relative or absolute path.

Format of the fontset file

The format of the fontset file 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

4.1.9 GRID

Description

The GRID object can be used to add labeled graticule lines to your map. Initially developed in 2003 by John Novak, the GRID object is designed to be used inside a *LAYER* object to allow multiple GRID objects for a single map (allowing for example: a lat/long GRID, a State Plane GRID, and a UTM GRID to be displayed on the same map image).

Mapfile Parameters:

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.

Example1: Grid Displaying Degrees

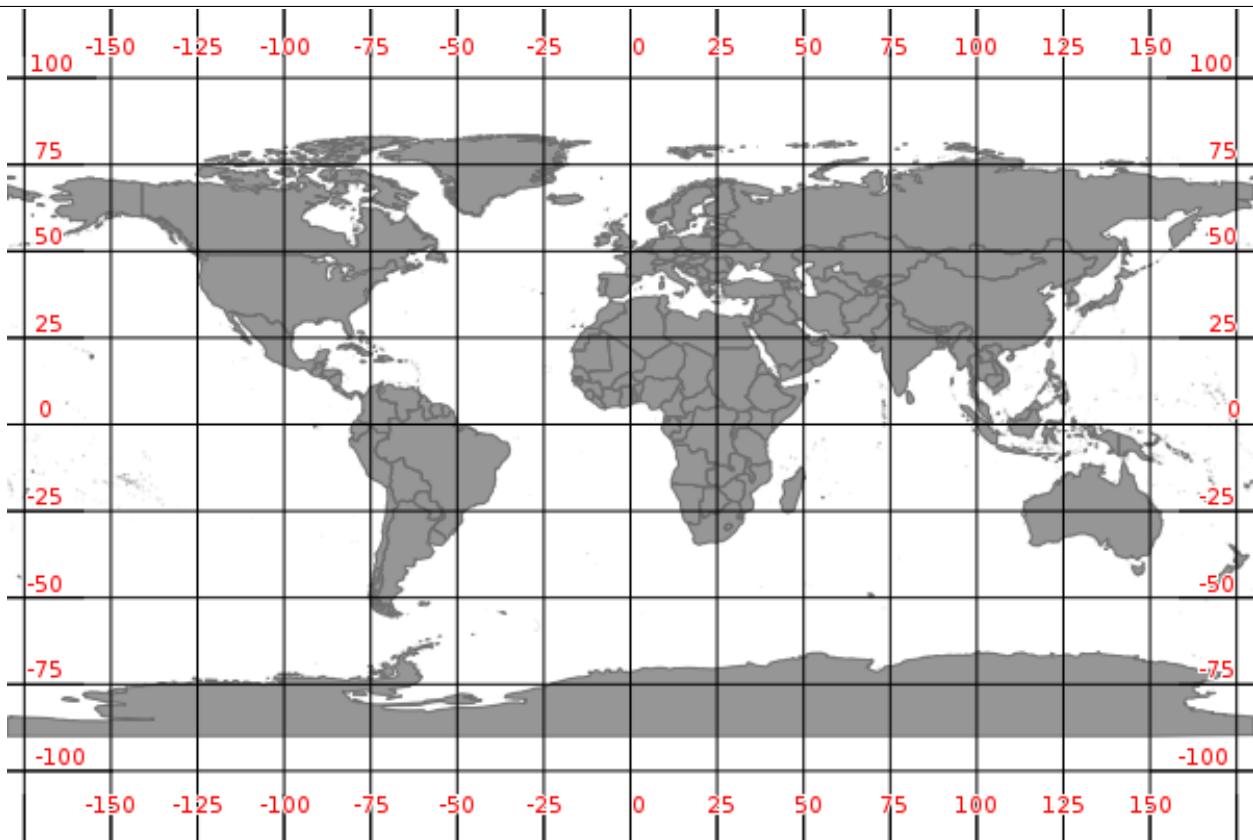
```
LAYER
  NAME "grid"
  METADATA
    "DESCRIPTION" "Grid"
  END
  TYPE LINE
  STATUS ON
  CLASS
    NAME "Graticule"
    COLOR 0 0 0
    LABEL
      COLOR 255 0 0
```



```

FONT "sans"
TYPE truetype
SIZE 8
POSITION AUTO
PARTIALS FALSE
BUFFER 2
OUTLINECOLOR 255 255 255
END
END
PROJECTION
  "init=epsg:4326"
END
GRID
  LABELFORMAT "DD"
END
END # Layer

```



Example2: Grid Displaying Degrees with Symbol

```

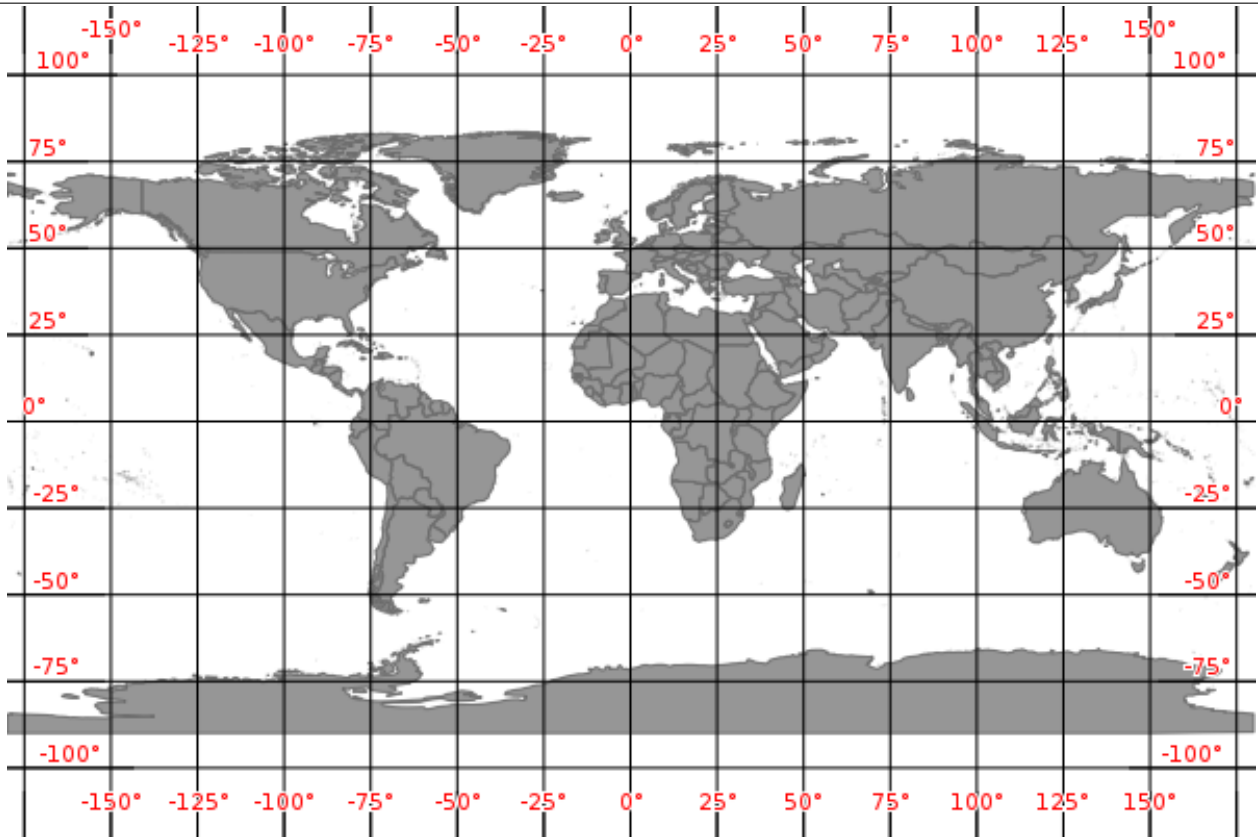
LAYER
  NAME "grid"
  METADATA
    "DESCRIPTION" "Grid"
  END
  TYPE LINE
  STATUS ON
  CLASS
    NAME "Graticule"

```

```

COLOR 0 0 0
LABEL
  COLOR 255 0 0
  FONT "sans"
  TYPE truetype
  SIZE 8
  POSITION AUTO
  PARTIALS FALSE
  BUFFER 2
  OUTLINECOLOR 255 255 255
END
END
PROJECTION
  "init=epsg:4326"
END
GRID
  LABELFORMAT '%g°'
END
END # Layer

```



Example2: Grid Displayed in Other Projection (Google Mercator)

```

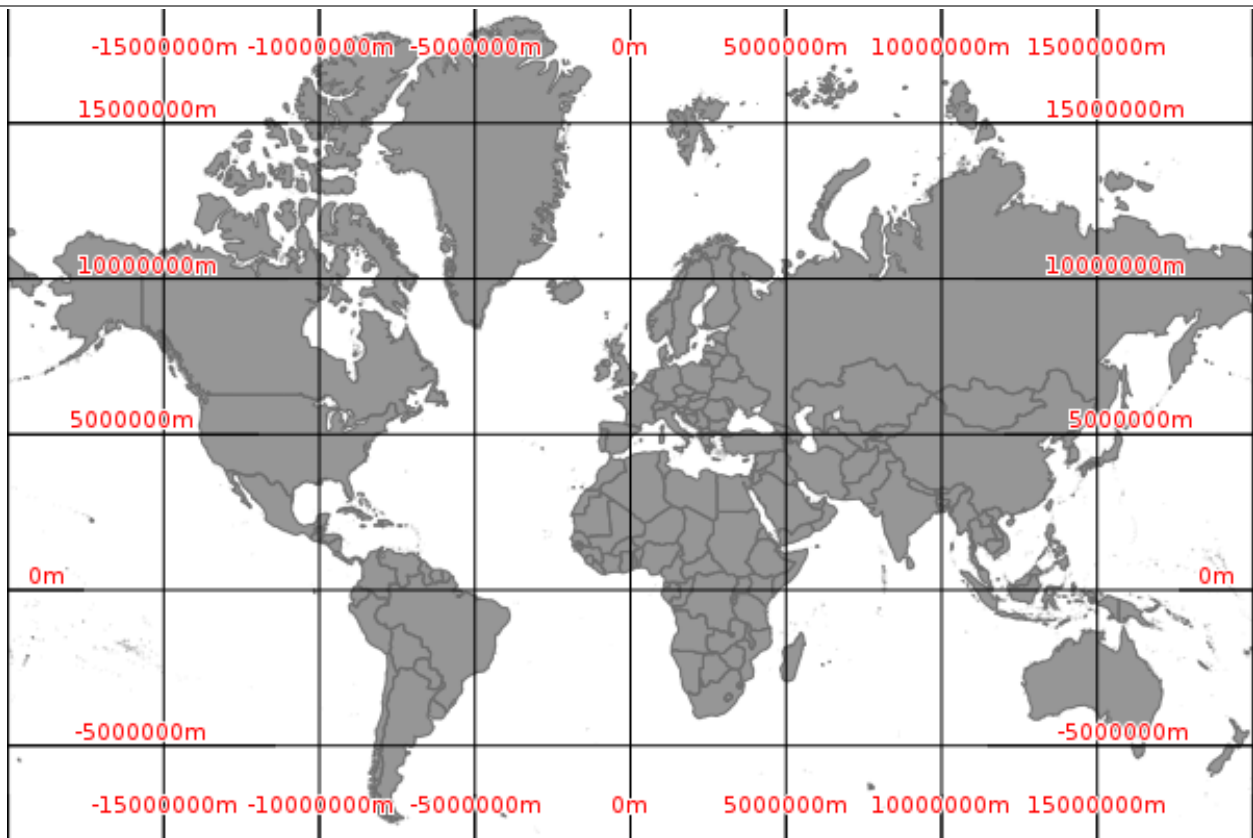
LAYER
  NAME "grid"
  METADATA
    "DESCRIPTION" "Grid"
  END
  TYPE LINE

```

```

STATUS ON
CLASS
  NAME "Graticule"
  COLOR 0 0 0
  LABEL
    COLOR 255 0 0
    FONT "sans"
    TYPE truetype
    SIZE 8
    POSITION AUTO
    PARTIALS FALSE
    BUFFER 2
    OUTLINECOLOR 255 255 255
  END
END
PROJECTION
  "init=epsg:3857"
END
GRID
  LABELFORMAT '%.0fm'
  MININTERVAL 5000000
END
END # Layer

```



Note: Pay attention to the values you use for the INTERVAL parameter; it is possible to confuse/overload MapServer by telling it to draw a graticule line every meter (MININTERVAL 1).

4.1.10 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.

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"
```

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

```

4.1.11 JOIN

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.

Supported Formats

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

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 connection for *PostgreSQL*:

```

CONNECTION "host=127.0.0.1 port=5432 user=pg password=pg dbname=somename"
CONNECTIONTYPE POSTGRESQL

```

CONNECTIONTYPE [**csv|mysql|postgresql**] Type of connection (not required for DBF joins). For PostgreSQL use *postgresql*, for CSV use *csv*, for MySQL use *mysql*.

FOOTER [**filename**] Template to use *after* a layer's set of results have been sent. In other words, this header HTML will be displayed after the contents of the *TEMPLATE* HTML.

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

HEADER [**filename**] Template to use *before* a layer's set of results have been sent. In other words, this header HTML will be displayed before the contents of the *TEMPLATE* HTML.

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 support this is the name of the PostgreSQL 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 columns 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 [column] Join column 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.

Example 1: Join from Shape dataset 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>
```

Example 2: Join from Shape dataset 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=pg password=pg dbname=join"
    CONNECTIONTYPE postgresql
    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=pg password=pg dbname=join"
      lookup -summary
INFO: Open of `PG:host=127.0.0.1 port=5432 user=pg password=pg 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="#EFEFEF">
  <td align="left">[NAME]</td>
  <td align="left">[test_val]</td>
</tr>
```

Example 3: Join from Shape dataset 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"
    CONNECTIONTYPE CSV
    TABLE "../data/lookup.csv"
    FROM "ID"
    #TO "IDENT" # see note below
    TO "1" # see note below
    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
```

Note: The CSV driver currently doesn't read column names from the first row. It just uses indexes (1, 2, ... n) to reference the columns. It's ok to leave column names as the first row since they likely won't match anything but they aren't used. Typically you'd see something like TO "1" in the JOIN block. Then in the template you'd use [name_1], [name_2], etc...

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 (prov.html)

Ideally this the template should look like this:

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

But since attribute names are not supported for CSV files (see note above), the following will have to be used:

```
<!-- MapServer Template -->
<tr bgcolor="#EFEFEF">
  <td align="left">[NAME]</td>
  <td align="left">[test_2]</td>
</tr>
```

Example 4: Join from Shape dataset to MySQL

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 # style
  END # class
  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 "mysql-join"
    CONNECTIONTYPE MYSQL
    CONNECTION 'server:user:password:database'
    TABLE "mysql-tablename"
    FROM "ID"
    TO "mysql-column"
    TYPE ONE-TO-ONE
  END # join
END # layer
```

Example 5: One-to-many join

In a join of type *ONE-TO-MANY*, the *JOIN* object needs to contain a *TEMPLATE*. This *TEMPLATE* is used for each matching record in the join table. Columns in the join table are referenced using `<join_name>_<join_column_name>`. Columns in the layer table are referenced using `<column_name>`.

For a one-to-many join, the *LAYER TEMPLATE* file has to contain a reference to the the *JOIN* object, as follows: `[join_<join_name>]`.

In this example, it is assumed that the join table `many.dbf` contains the columns `MANYFIELD1` and `MANYFIELD2` in addition to the join column (`IDENT`).

Layer object:

```
LAYER
  NAME "joinonetomany"
  TYPE POLYGON
  STATUS DEFAULT
  DATA "prov.shp"
  CLASS
    NAME "Province"
    STYLE
      OUTLINECOLOR 120 120 120
      COLOR 255 255 0
    END # style
  END # class
```

```

TEMPLATE "oneToMany.html"
HEADER "oneToMany_header.html"
FOOTER "oneToMany_footer.html"
JOIN
  NAME "onetomanytest"
  TABLE "many.dbf"
  FROM "ID"
  TO "IDENT"
  TYPE ONE-TO-MANY
  TEMPLATE "oneToMany_join.html"
END # join
END # layer

```

Template oneToMany_header.html:

```

<!-- Mapserver Template -->
<html>
  <head><title>One to Many Join</title></head>
  <body>
    <h1>Mapserver output</h1>
    <table>

```

Template oneToMany.html:

```

<!-- Mapserver Template -->
  <tr>
    <td><strong>[ID]</strong></td>
    <td><table>
[join_onetomanytest]
    </table></td>
  </tr>

```

Template oneToMany_join.html:

```

<!-- Mapserver Template -->
  <tr>
    <td>[NAME]</td>
    <td>[onetomanytest_MANYFIELD1]</td>
    <td>[onetomanytest_MANYFIELD2]</td>
  </tr>

```

Template oneToMany_footer.html:

```

<!-- Mapserver Template -->
  </table>
</body>
<html>

```

4.1.12 LABEL

ALIGN [**left**|**center**|**right**] 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.

New in version 5.4.

ANGLE [**double**|**auto**|**auto2**|**follow**|**attribute**]

- Angle, counterclockwise, given in degrees, to draw the label. Default is 0.

- **AUTO** allows MapServer to compute the angle. Valid for LINE layers only.
- **AUTO2** same as **AUTO**, except no logic is applied to try to keep the text from being rendered in reading orientation (i.e. the text may be rendered upside down). Useful when adding text arrows indicating the line direction.
- **FOLLOW** was introduced in version 4.10 and tells MapServer to compute a curved label for appropriate linear features (see RFC11 for specifics). See also *MAXOVERLAPANGLE*.
- **[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 "sans"
  TYPE truetype
  SIZE 6
  ANGLE [MYANGLE]
  POSITION AUTO
  PARTIALS FALSE
END
```

The associated RFC document for this feature is RFC19.

ANTI_ALIAS [true|false] Should text be antialiased? Note that this requires more available colors, decreases drawing performance, and results in slightly larger output images. Only useful for GD (gif) rendering. Default is false. Has no effect for the other renderers (where anti-aliasing can not be turned off).

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

Note: Removed in 6.0. Use a *LABEL STYLE* object with *GEOMTRANSFORM labelpoly* and *COLOR*.

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

Note: Removed in 6.0. Use a *LABEL STYLE* object with *GEOMTRANSFORM labelpoly*, *COLOR* and *OFF-SET*.

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

Note: Removed in 6.0. Use a *LABEL STYLE* object with *GEOMTRANSFORM labelpoly*, *COLOR* and *OFF-SET*.

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 [MYCOLOR]
  OUTLINECOLOR 255 255 255
  FONT "sans"
```

```

TYPE truetype
SIZE 6
POSITION AUTO
PARTIALS FALSE
END

```

The associated RFC document for this feature is RFC19.

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 in the [Label Encoding document](#).

EXPRESSION [**string**] Expression that determines when the *LABEL* is to be applied. See *EXPRESSION* in *CLASS*.

New in version 6.2.

FONT [**namelattribute**]

- Font alias (as defined in the FONTSET) to use for labeling.
- [*Attribute*] was introduced in version 5.6 to specify the font alias.
- May contain a comma-separated list of up to MS_MAX_LABEL_FONTS (usually 5) font aliases used as fallback fonts in renderers supporting it, if a glyph is not available in a font. If specified directly, be sure to enclose the list with quotes. See RFC80.

FORCE [**truelfalse**] Forces labels for a particular class on, regardless of collisions. Available only for cached labels. Default is false. If *FORCE* is true and *PARTIALS* is false, *FORCE* takes precedence, and partial labels are drawn.

MAXLENGTH [**integer**] This keyword interacts with the *WRAP* keyword so that line breaks only occur after the defined number of characters.

Table 4.9: Interaction with WRAP keyword

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

The associated RFC document for this feature is RFC40.

New in version 5.4.

MAXOVERLAPANGLE [**double**] Angle threshold to use in filtering out ANGLE FOLLOW labels in which characters overlap (floating point value in degrees). This filtering will be enabled by default starting with MapServer 6.0. The default MAXOVERLAPANGLE value will be 22.5 degrees, which also matches the default in GeoServer. Users will be free to tune the value up or down depending on the type of data they are dealing with and their tolerance to bad overlap in labels. As per RFC 60, if MAXOVERLAPANGLE is set to 0, then we fall back on pre-6.0 behavior which was to use $\text{maxoverlapangle} = 0.4 * \text{MS_PI}$ (40% of 180 degrees = 72degree).

The associated RFC document for this feature is RFC60.

MAXSCALEDENOM [**double**] Minimum scale at which this *LABEL* 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.

New in version 5.4.

See also:

Map Scale

MAXSIZE [double] Maximum font size to use when scaling text (pixels). Default is 256. Starting from version 5.4, the value can also be a fractional value (and not only integer). See *LAYER SYMBOLSCALEDENOM*.

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.

MINSCALEDENOM [double] Maximum scale at which this *LABEL* 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.

New in version 5.4.

See also:

Map Scale

MINSIZE [double] Minimum font size to use when scaling text (pixels). Default is 4. Starting from version 5.4, the value can also be a fractional value (and not only integer). See *LAYER SYMBOLSCALEDENOM*.

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.

When used with FOLLOW angle, two additional options are available to render the label parallel to the original feature:

- OFFSET x -99 : will render the label to the left or to the right of the feature, depending on the sign of {x}.
- OFFSET x 99 : will render the label above or below the feature, depending on the sign of {x}.

See *LAYER SYMBOLSCALEDENOM*.

OUTLINECOLOR [r] [g] [b] | [attribute]

- Color to draw a one pixel outline around the characters in 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 [MYOUTCOLOR]
  FONT "sans"
  TYPE truetype
  SIZE 6
  POSITION AUTO
  PARTIALS FALSE
END
```

The associated RFC document for this feature is RFC19.

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. If *FORCE* is true and *PARTIALS* is false, *FORCE* takes precedence, and partial labels are drawn.

POSITION [`ullucurlcllclcrllllcllrlauto`] 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`][`attribute`] The priority parameter 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.

[*Attribute*] was introduced in version 5.6.

New in version 5.0.

REPEATDISTANCE [`integer`] The label will be repeated on every line of a multiline shape and will be repeated multiple times along a given line at an interval of REPEATDISTANCE pixels.

The associated RFC document for this feature is RFC57.

New in version 5.6.

SHADOWCOLOR [`r`] [`g`] [`b`] Color of drop shadow. A label with the same text will be rendered in this color before the main label is drawn, resulting in a shadow effect on the the label characters. The offset of the rendered shadow is set with SHADOWSIZE.

SHADOWSIZE [`x`][`y`][`attribute`][`attribute`][`x`][`attribute`][`attribute`][`y`] Shadow offset in pixels, see SHADOWCOLOR.

[*Attribute*] was introduced in version 6.0, and can be used like:

```
SHADOWSIZE 2 2
SHADOWSIZE [shadowsizeX] 2
SHADOWSIZE 2 [shadowsizeY]
SHADOWSIZE [shadowsize] [shadowsize]
```

SIZE [`double`][`tiny`|`small`|`medium`|`large`|`giant`][`attribute`]

- Text size. Use a number to give the size in pixels of your TrueType font based label, or any of the other 5 listed keywords for bitmap fonts.

When scaling is in effect (*SYMBOLSCALEDENOM* is specified for the *LAYER*), *SIZE* gives the size of the font to be used at the map scale 1:*SYMBOLSCALEDENOM*.

- Starting from version 5.4, the value can also be a fractional value (and not only integer).
- [*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 "sans"
  TYPE truetype
  SIZE [MYSIZE]
  POSITION AUTO
```

```
PARTIALS FALSE
END
```

The associated RFC document for this feature is RFC19.

STYLE The start of a *STYLE* object.

Label specific mechanisms of the *STYLE* object are the GEOMTRANSFORM options:

GEOMTRANSFORM [labelpnt|labelpoly] Creates a geometry that can be used for styling the label.

- **labelpnt** draws a marker on the geographic position the label is attached to. This corresponds to the center of the label text only if the label is in position CC.
- **labelpoly** generates the bounding rectangle for the text, with 1 pixel of padding added in all directions.

The resulting geometries can be styled using the mechanisms available in the *STYLE* object.

Example - draw a red background rectangle for the labels (i.e. billboard) with a “shadow” in gray:

```
STYLE
  GEOMTRANSFORM 'labelpoly'
  COLOR 153 153 153
  OFFSET 3 2
END # STYLE
STYLE
  GEOMTRANSFORM 'labelpoly'
  COLOR 255 0 0
END # STYLE
```

New in version 6.0.

TEXT [stringexpression] Text to label features with (useful when multiple labels are used). Overrides values obtained from the *LAYER LABELITEM* and the *CLASS TEXT*. See *TEXT* in *CLASS*.

New in version 6.2.

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.

Note: Bitmap fonts are only supported with the AGG and GD renderers.

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

4.1.13 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.

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*.

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

CLUSTER Signals the start of a *CLUSTER* object.

The CLUSTER configuration option provides to combine multiple features from the layer into single (aggregated) features based on their relative positions. Supported only for POINT layers.

See also:

rfc69

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:

Vector Data for specific connection information for various data sources.

CONNECTIONTYPE [contour|localogr|oraclespatial|plugin|postgis|sdelunion|uvraster|wfs|wms] Type of connection. Default is local. See additional documentation for any other type.

See also:

Vector Data for specific connection information for various data sources. See *Union Layer* for combining layers, added in MapServer 6.0

Note: *mygis* is another connectiontype, but it is deprecated; please see the *MySQL section* of the *Vector Data* document for connection details.

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:

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 RFC28.

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 [`true|false`] Since 6.0, *DUMP* is not used anymore. *LAYER METADATA* is used instead.

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

Deprecated since version 6.0: *LAYER METADATA* is used instead.

See also:

WMS Server

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.

GEOMTRANSFORM [**<expression>**] Used to indicate that the current feature will be transformed. Introduced in version 6.4.

- *<expression>*: Applies the given expression to the geometry.

Supported expressions:

- (*buffer([shape],dist)*): Buffer the geometry (*[shape]*) using *dist* pixels as buffer distance. For polygons, a negative *dist* will produce a setback.
- (*simplify([shape],tolerance)*): simplifies a geometry (*[shape]*) using the standard Douglas-Peucker algorithm.
- (*simplifypt([shape], tolerance)*): simplifies a geometry (*[shape]*), ensuring that the result is a valid geometry having the same dimension and number of components as the input. *tolerance* must be non-negative.
- (*generalize([shape],tolerance)*): simplifies a geometry (*[shape]*) in way comparable to FME's ThinNoPoint algorithm. See <http://trac.osgeo.org/gdal/ticket/966> for more information.
- (*smoothsia([shape], smoothing_size, smoothing_iteration, preprocessing)*): will smooth a geometry (*[shape]*) using the SIA algorithm

See also:

Geometry Transformations and *shape_smoothing*

There is a difference between STYLE and LAYER GEOMTRANSFORM. LAYER-level will receive ground coordinates (meters, degrees, etc) and STYLE-level will receive pixel coordinates. The argument to methods such as `simplify()` must be in the same units as the coordinates of the shapes at that point of the rendering workflow, i.e. pixels at the STYLE-level and in ground units at the LAYER-level.

```
LAYER NAME "my_layer"
  TYPE LINE
  STATUS DEFAULT
  DATA "lines.shp"
  GEOMTRANSFORM (simplify([shape], 10)) ## 10 ground units
  CLASS
    STYLE
      GEOMTRANSFORM (buffer([shape], 5)) ## 5 pixels
      WIDTH 2
      COLOR 255 0 0
    END
  END
END
```

The `[map_cellsize]` variable is available if you need to pass a pixel value at the LAYER-level.

```
LAYER NAME "my_layer"
  TYPE LINE
  STATUS DEFAULT
  DATA "lines.shp"
  UNITS meters
  # 10 * [map_cellsize] == 10 pixels converted to ground units
  GEOMTRANSFORM (simplify([shape], [map_cellsize]*10))
  ...
```

To get this variable working in the math expression parser, the [map_cellsize] has to be converted into the layer ground unit. If you choose to use [map_cellsize] in your GEOMTRANSFORM expression, you must explicitly set the UNITS option in the layer.

See also:

Geometry Transformations

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.

If a group name is present in the LAYERS parameter of a CGI request, all the layers of the group are returned (the *STATUS* of the *LAYERs* have no effect).

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 [on/off] 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] 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 LABELMAXSCALE parameter.

See also:

Map Scale

LABELMINSCALEDENOM [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 LABELMINSCALE parameter.

See also:

Map Scale

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.

MASK [layername] The data from the current layer will only be rendered where it intersects features from the [layername] layer. [layername] must reference the NAME of another *LAYER* defined in the current mapfile. can be any kind of mapserver layer, i.e. vector or raster. If the current layer has labelling configured, then only

labels whose label-point fall inside the unmasked area will be added to the labelcache (the actual glyphs for the label may be rendered ontop of the masked-out area).

Note: Unless you want the features of [layername] to actually appear on the generated map, [layername] should usually be set to *STATUS OFF*.

See also:

rfc79

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).

MAXGEOWIDTH [double] Maximum width, in the map's geographic units, at which this LAYER is drawn. If MAXSCALEDENOM is also specified then MAXSCALEDENOM will be used instead.

The width of a map in geographic units can be found by calculating the following from the extents:

```
[maxx] - [minx]
```

New in version 5.4.0.

MAXSCALEDENOM [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.

New in version 5.0.0: Replaced MAXSCALE.

See also:

Map Scale

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
```

MINGEOWIDTH [double]

Minimum width, in the map's geographic units, at which this LAYER is drawn. If MINSCALEDENOM is also specified then MINSCALEDENOM will be used instead.

The width of a map in geographic units can be found by calculating the following from the extents:

```
[maxx] - [minx]
```

New in version 5.4.0.

MINSCALEDENOM [double] 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 MINSCALE parameter.

See also:

Map Scale

NAME [string] Short name for this layer. 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. It is recommended that the name not contain

spaces, special characters, or begin with a number (which could cause problems through interfaces such as OGC services).

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.

PLUGIN [**filename**] Additional library to load by MapServer, for this layer. This is commonly used to load specific support for SDE and Microsoft SQL Server layers, such as:

```
CONNECTIONTYPE PLUGIN
CONNECTION "hostname,port:xxx,database,username,password"
PLUGIN "C:/ms4w/Apache/specialplugins/msplugin_sde_92.dll"
DATA "layername,geometrycolumn,SDE.DEFAULT"
```

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"
```

- **Label Directive** - The LABEL_NO_CLIP processing option can be used to skip clipping of shapes when determining associated label anchor points. This avoids changes in label position as extents change between map draws. It also avoids duplicate labels where features appear in multiple adjacent tiles when creating tiled maps.

```
PROCESSING "LABEL_NO_CLIP=True"
```

- **Line Rendering Directive** - The POLYLINE_NO_CLIP processing option can be used to skip clipping of shapes when rendering styled lines (dashed or styled with symbols). This avoids changes in the line styling as extents change between map draws. It also avoids edge effects where features appear in multiple adjacent tiles when creating tiled maps.

```
PROCESSING "POLYLINE_NO_CLIP=True"
```

- **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 [feet|inches|kilometers|meters|miles|nauticalmiles|pixels] Sets the unit of *CLASS* object SIZE values (default is pixels). Useful for simulating buffering. *nauticalmiles* was added in MapServer 5.6.

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.

Note: The STATUS of the individual layers of a GROUP has no effect when the group name is present in the LAYERS parameter of a CGI request - all the layers of the group will be returned.

STYLEITEM [<attribute>|auto] Item to use for feature specific styling. The style information may be represented by a separate attribute (style string) attached to the feature. MapServer supports the following style string representations:

- **MapServer STYLE definition** - The style string can be represented as a MapServer *STYLE* block according to the following example:

```
STYLE BACKGROUNDCOLOR 128 0 0 COLOR 0 0 208 END
```

- **MapServer CLASS definition** - By specifying the entire *CLASS* instead of a single style allows to use further options (like setting expressions, label attributes, multiple styles) on a per feature basis.
- **OGR Style String** - MapServer support rendering the OGR style string format according to the *OGR - Feature Style Specification* documentation. Currently only a few data sources support storing the styles along with the features (like MapInfo, AutoCAD DXF, Microstation DGN), however those styles can easily be transferred to many other data sources as a separate attribute by using the *ogr2ogr* command line tool as follows:

```
ogr2ogr -sql "select *, OGR_STYLE from srclayer" "dstlayer" "srclayer"
```

The value: *AUTO* can be used for automatic styling.

- Automatic styling can be provided by the driver. Currently, only the OGR driver supports automatic styling.
- When used for a *Union Layer*, the styles from the source layers will be used.

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

TEMPLATE [**fileurl**] Used as a global alternative to *CLASS TEMPLATE*. See *Templating* for more info.

TILEINDEX [**filenamelayername**] 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.

Note: Starting with MapServer 6.4, raster layers can use a tileindex with tiles of different projections. For that, the *TILESRS* parameter must be specified.

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

TILESRS [**attribute**] Name of the attribute that contains the SRS of an individual tile. That SRS can be expressed in WKT format, as an EPSG:XXXX code or as a PROJ.4 string. If the tileindex contains rasters in different projections, this option must be specified. If the tileindex has been generated with gdaltindex (GDAL >= 2.0), the value of TILESRS is the value of the -src_srs_name option of gdaltindex. See *Tileindexes with tiles in different projections*

Note: This option is currently available only on raster layers.

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. This setting does not apply to WFS GetFeature operations.

TOLERANCEUNITS [**pixels|feet|inches|kilometers|meters|miles|nauticalmiles|dd**] Units of the TOLERANCE value. Default is pixels. *Nauticalmiles* was added in MapServer 5.6.

TRANSPARENCY [**integer|alpha**] - deprecated

Deprecated since version 5.0: Use *OPACITY* instead.

TRANSFORM [**true|false** **ulluclurllccllrllllllcllr**] 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 copyright statement to another portion of the image in a way that is independent of image size.

TYPE [**chart|circle|line|point|polygon|raster|query**] 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.

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 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
```

TYPE query means the layer can be queried but not drawn.

Note: *TYPE* annotation has been deprecated since version 6.2. Identical functionality can be obtained by adding *LABEL* level *STYLE* blocks, and do not require loading the datasets twice in two different layers as was the case with layers of *TYPE* annotation.

See also:

The *Dynamic Charting* HowTo for *TYPE* chart.

UNITS [**dd|feet|inches|kilometers|meters|miles|nauticalmiles|percentages|pixels**] Units of the layer. *percentages* (in this case a value between 0 and 1) was added in MapServer 4.10 and is mostly geared for inline features. *nauticalmiles* was added in MapServer 5.6.

VALIDATION Signals the start of a *VALIDATION* block.

As of MapServer 5.4.0, *VALIDATION* blocks are the preferred mechanism for specifying validation patterns for CGI param runtime substitutions. See *Run-time Substitution*.

4.1.14 LEADER

Table of Contents

- *LEADER*
 - *Description*
 - *Supported Layer Types*
 - *Mapfile Parameters*
 - *Mapfile Snippet*
 - *Example: World Countries Labels*

Description

Since version 6.2, MapServer has the ability to draw label lines to features where space is an issue for the label (often when the label text is larger than the polygon being labelled). This feature was added through rfc81.

Supported Layer Types

POLYGON

Mapfile Parameters

GRIDSTEP [integer] Specifies the number of pixels between positions that are tested for a label line. You might start with a value of 5, and increase depending on performance (see example below).

MAXDISTANCE [integer] Specifies the maximum distance in pixels from the normal label location that a leader line can be drawn. You might start with a value of 30, and increase depending on the resulting placement (see example below).

STYLE Signals the start of a *STYLE* object. Use this to style the leader line.

Mapfile Snippet

```
LAYER
  NAME "my-labels"
  TYPE POLYGON
  ...
  CLASS
    ...
    LABEL
    ...
  END
  LEADER
    GRIDSTEP 5 # number of pixels between positions that are tested
    MAXDISTANCE 30 # distance in pixels that leader text can be drawn
    STYLE # normal line styles are supported
      COLOR 255 0 0
      WIDTH 1
    END
  END
END
END
```

Example: World Countries Labels

The following example uses a polygon layer to display country labels.

Note: The data and mapfile for this example are available for download at: <http://download.osgeo.org/mapserver/tickets/label-leader.zip> (11MB).

Mapfile Example #1

```

MAP

NAME "leader-test"
STATUS ON
SIZE 800 600
SYMBOLSET "../etc/symbols.txt"
EXTENT -43 10 83 83
UNITS DD
SHAPEPATH "../data"
IMAGECOLOR 255 255 255
FONTSET "../etc/fonts.txt"

WEB
  IMAGEPATH "/ms4w/tmp/ms_tmp/"
  IMAGEURL "/ms_tmp/"
END

#
# Start of layer definitions
#

LAYER
  NAME "continents"
  TYPE POLYGON
  STATUS ON
  DATA "world_countries-dissolve"
  LABELITEM "NA2DESC"
  CLASS
    NAME "World Countries"
    STYLE
      COLOR 200 200 200
      OUTLINECOLOR 120 120 120
    END
    LABEL
      COLOR 0 0 0
      FONT sans
      TYPE truetype
      SIZE 8
      POSITION AUTO
      PARTIALS FALSE
      OUTLINECOLOR 255 255 255
      MINFEATURESIZE 2
      MINDISTANCE 1000
      BUFFER 5
    END
    #####
    # Leader Object
    #####
    LEADER
      GRIDSTEP 40
      MAXDISTANCE 1000
      STYLE
        COLOR 200 100 100
        WIDTH 2
      END
  END

```

```

END
END
END

END # Map File

```

Map Image



Mapfile Example #2

This time use a shorter maximum leader line (MAXDISTANCE) and increase the number of tests (GRIDSTEP).

```

MAP

LAYER
...
CLASS
...
LABEL
...
END
#####
# Leader Object

```

```
#####
LEADER
  GRIDSTEP 10
  MAXDISTANCE 100
  STYLE
    COLOR 200 100 100
    WIDTH 2
  END
END
END
END
END # Map File
```

Map Image



4.1.15 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.

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.

LABEL Signals the start of a *LABEL* object

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

POSITION [**ul|l|ur|ll|lll|llc|lr**] Where to place an embedded legend in the map. Default is lr.

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.

4.1.16 MAP

Note: The map object is started with the word *MAP*, and ended with the word *END*.

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**] This *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**] This *CONFIG* parameter can be used to specify an encryption key that is used with MapServer’s *msencrypt utility*.

MS_ERRORFILE [filename] This *CONFIG* parameter can be used to write MapServer errors to a file (as of MapServer 5.0). With MapServer 5.x, a full path (absolute reference) is required, including the filename. Starting with MapServer 6.0, a filename with relative path can be passed via this *CONFIG* directive, in which case the filename is relative to the mapfile location. Note that setting `MS_ERRORFILE` via an environment variable always requires an absolute path since there would be no mapfile to make the path relative to. For more on this see the *DEBUG* parameter below.

MS_NON SQUARE [yes|no] This *CONFIG* parameter can be used to allow non-square pixels (meaning that the pixels represent non-square regions). For “`MS_NON SQUARE`” “yes” to work, the *MAP*, and each *LAYER* will have to have a *PROJECTION* object.

Note: Has no effect for WMS.

ON_MISSING_DATA [FAIL|LOG|IGNORE] This *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] This *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.layer[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 the *Debugging MapServer* document.

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).

DEFRESOLUTION [int] Sets the reference resolution (pixels per inch) used for symbology. Default is 72.

Used to automatically scale the symbology when *RESOLUTION* is changed, so the map maintains the same look at each resolution. The scale factor is $RESOLUTION / DEFRESOLUTION$.

New in version 5.6.

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* in *OUTPUTFORMAT*) 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 [`jpeg|pdf|png|svg|...|userdefined`] Output format (raster or vector) to generate. The name used here must match the ‘NAME’ of a user defined or internally available *OUTPUTFORMAT*. For a complete list of available *IMAGEFORMAT*s, see the *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). Default is 2048.

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. 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

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 [`on|off`] 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.

Note: The *SYMBOLSET* file must start with the word *SYMBOLSET* and end with the word *END*.

SYMBOL Signals the start of a *SYMBOL* object.

TEMPLATEPATTERN [`regular expression`] This defines a regular expression to be applied to requests to change the *TEMPLATE* parameters via URL requests (i.e. `map.layer[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 [`on|off`]

Deprecated since version 4.6.

Use *TRANSPARENT ON* in the *OUTPUTFORMAT* declaration to specify if the output images should be transparent.

UNITS [*dd*|*feet*|*inches*|*kilometers*|*meters*|*miles*|*nauticalmiles*] Units of the map coordinates. Used for scalebar and scale computations. *Nauticalmiles* was added in MapServer 5.6.

WEB Signals the start of a *WEB* object.

4.1.17 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, SVG, PDF and KML.

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 PDF and SVG depend on building with cairo support.

```
OUTPUTFORMAT
  NAME "png"
  DRIVER AGG/PNG
  MIMETYPE "image/png"
  IMAGEMODE RGB
  EXTENSION "png"
  FORMATOPTION "GAMMA=0.75"
END
OUTPUTFORMAT
  NAME "gif"
  DRIVER GD/GIF
  MIMETYPE "image/gif"
  IMAGEMODE PC256
  EXTENSION "gif"
END
OUTPUTFORMAT
  NAME "png8"
  DRIVER AGG/PNG8
  MIMETYPE "image/png; mode=8bit"
  IMAGEMODE RGB
  EXTENSION "png"
  FORMATOPTION "QUANTIZE_FORCE=on"
  FORMATOPTION "QUANTIZE_COLORS=256"
  FORMATOPTION "GAMMA=0.75"
END
OUTPUTFORMAT
  NAME "jpeg"
  DRIVER AGG/JPEG
  MIMETYPE "image/jpeg"
  IMAGEMODE RGB
  EXTENSION "jpg"
  FORMATOPTION "GAMMA=0.75"
END
OUTPUTFORMAT
  NAME "svg"
  DRIVER CAIRO/SVG
  MIMETYPE "image/svg+xml"
  IMAGEMODE RGB
  EXTENSION "svg"
END
OUTPUTFORMAT
  NAME "pdf"
  DRIVER CAIRO/PDF
  MIMETYPE "application/x-pdf"
```

```

    IMAGEMODE RGB
    EXTENSION "pdf"
END
OUTPUTFORMAT
    NAME "GTiff"
    DRIVER GDAL/GTiff
    MIMETYPE "image/tiff"
    IMAGEMODE RGB
    EXTENSION "tif"
END
OUTPUTFORMAT
    NAME "kml"
    DRIVER KML
    MIMETYPE "application/vnd.google-earth.kml.xml"
    IMAGEMODE RGB
    EXTENSION "kml"
END
OUTPUTFORMAT
    NAME "kmz"
    DRIVER KMZ
    MIMETYPE "application/vnd.google-earth.kmz"
    IMAGEMODE RGB
    EXTENSION "kmz"
END
OUTPUTFORMAT
    NAME "cairopng"
    DRIVER CAIRO/PNG
    MIMETYPE "image/png"
    IMAGEMODE RGB
    EXTENSION "png"
END

```

DRIVER [name] The name of the driver to use to generate this output format. Some driver names include the definition 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” and “GD/PNG”. For output through OGR the OGR driver name is appended, such as “OGR/Mapinfo File”. 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). **TEMPLATE** should be used for template based output. (mandatory)

EXTENSION [type] Provide the extension to use when creating files of this type. (optional)

FORMATOPTION [option] Provides a driver or format specific option. Zero or more **FORMATOPTION** statement may be present within a **OUTPUTFORMAT** declaration. (optional)

- AGG/*: “GAMMA=n” is used to specify the gamma correction to apply to polygon rendering. Allowed values are [0.0,1.0], default is 0.75. This value is used to prevent artifacts from appearing on the border of contiguous polygons. Set to 1.0 to disable gamma correction.
- AGG/JPEG: The “QUALITY=n” option may be used to set the quality of jpeg produced (value from 0-100).
- AGG/PNG: “COMPRESSION=n” is used to determine the ZLIB compression applied to the png creation. n is expected to be an integer value from 0 to 9, with 0 meaning *no* compression (not recommended), 1 meaning fastest compression, and 9 meaning best compression. The compression levels come at a cost (be it in terms of cpu processing or file size, chose the setting that suits you most). The default is COMPRESSION=6.
- AGG/PNG supports quantizing from 24/32 bits to 8bits, in order to reduce the final image size (and there-

fore 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 or RGBA image.
- “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.
- “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 you want to use transparency with a palette, it is important to have these two colors in the palette file: 0,0,0,0 and 255,255,255,255.

Note: 0,0,0,0 is important if you have fully transparent areas. 255,255,255,255 is opaque white. The important colors to have in your palette really depend on your actual map, although 0,0,0,0 , 0,0,0,255 , and 255,255,255,255 are very likely to show up most of the time.

- “PALETTE_FORCE=on” is used to reduce image depth with a predefined palette. This option is incompatible with the previous quantization options. To allow additional colours for anti-aliasing other than those in the predefined palette, use with “QUANTIZE_COLORS”.
- CAIRO/PDF:
 - “GEO_ENCODING=ISO32000” or “GEO_ENCODING=OGC_BP”: Geospatial PDF will be generated. Requires GDAL 1.10 with PDF driver. See the [GDAL Geospatial PDF](#) documentation for requirements.
New in version 6.2.
 - “METADATA_ITEM:option=value”: Additional PDF options can be provided using the METADATA_ITEM prefix. The following options are available: *AUTHOR*, *CREATOR*, *CREATION_DATE*, *KEYWORDS*, *PRODUCER*, *SUBJECT*, *TITLE*.
New in version 6.2.

Example:

```

OUTPUTFORMAT
  NAME pdf
  DRIVER "CAIRO/PDF"
  MIMETYPE "application/x-pdf"
  IMAGEMODE RGB
  EXTENSION "pdf"
  FORMATOPTION "GEO_ENCODING=ISO32000"
  FORMATOPTION "METADATA_ITEM:CREATOR=MapServer, with GDAL trunk"
  FORMATOPTION "METADATA_ITEM:PRODUCER=MapServer, with GDAL trunk"
END

```

- 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.
- GDAL/*: “NULLVALUE=n” is used in raw image modes (IMAGEMODE BYTE/INT16/FLOAT) to pre-initialize the raster and an attempt is made to record this in the resulting file as the nodata value. This is automatically set in WCS mode if rangeset_nullvalue is set.
- OGR/*: See the *OGR Output* document for details of OGR format options.

IMAGEMODE [PC256/RGB/RGBA/INT16/FLOAT32/FEATURE] 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 supports only PC256. (optional)

- PC256: Produced a pseudocolored result with up to 256 colors in the palette (legacy MapServer mode). Only supported for GD/GIF and GD/PNG.
- 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.
- FEATURE: Output is a non-image result, such as features written via templates or OGR.

MIMETYPE [type] Provide the mime type to be used when returning results over the web. (optional)

NAME [name] The name to use in the IMAGETYPE keyword of the map file to select this output format. This name is also used in metadata describing wxs formats allowed, and can be used (sometimes along with mimetype) to select the output format via keywords in OGC requests. (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.

4.1.18 PROJECTION

Background

There are thousands of geographical reference systems. In order to combine datasets with different geographical reference systems into a map, the datasets will have to be transformed (projected) to the chosen geographical reference system of the map. If you want to know more about geographical reference systems and map projections in general, please see the *More Information* links below, or look into Geomatics courses (Geographical Information Systems, Cartography, Geodesy), as projections are an advanced topic for beginners.

Projections with MapServer

To set up projections you must define one projection object for the output image (in the *MAP* object) and one projection object 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
```

Note: For a list of all of the possible PROJ.4 projection parameters, see the [PROJ.4 parameters](#) page.

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/long (“geographic”):

Example 3: Inline Projection Parameters

```
PROJECTION
"proj=latlong"
"ellps=WGS84"
"datum=WGS84"
END
```

Example 4: epsg Projection Use

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

“Web Mercator” or “Google Mercator”

The EPSG code for the commonly used “Web” or “Google” mercator projection is ‘3857’. See <http://spatialreference.org/ref/sr-org/7483/> for more information on this coordinate system. This code was also unofficially referred to as *EPSG:900913*; you are recommended to use the official *EPSG:3857* code instead, such as:

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

PROJECTION AUTO

The following syntax may be used in LAYERs that are OGR connections, shapefile layers or raster layers :

```
PROJECTION
  AUTO
END
```

- In case of a OGR connection, the projection will be retrieved from the OGR layer.
- In case of a shapefile layer, the projection will be retrieved from the associated .prj file.
- In case of raster layers refereing to single raster (DATA keyword), the projection will be retrived from the GDAL datasource. If the raster layer refers to a tile index (OGR layer or shapefile tileindex), the projection will be retrieved according to the above describe rules.

Note: For other layer types, this syntax is invalid.

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.

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.
- A respected author on map projections is John P. Snyder, if you are wishing for printed material to review.

4.1.19 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.

4.1.20 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 it off.

4.1.21 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 [ulluclurllllcllr] 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**|**nauticalmiles**] 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. *Nauticalmiles* was added in MapServer 5.6.

4.1.22 STYLE

Style holds parameters for symbolization and styling. Multiple styles may be applied within a *CLASS* or *LABEL*.

This object appeared in 4.0 and the intention is 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 way of defining the appearance of an object (a *CLASS* or a *LABEL*).

ANGLE [**double**|**attribute**|**AUTO**] Angle, given in degrees, to rotate the symbol (counter clockwise). Default is 0 (no rotation). If you have an attribute that specifies angles in a clockwise direction (compass direction), you have to adjust the angle attribute values before they reach MapServer (360-ANGLE), as it is not possible to use a mathematical expression for *ANGLE*.

- For points, it specifies the rotation of the symbol around its center.
- For decorated lines, the behaviour depends on the value of the *GAP* element.
 - For negative *GAP* values it specifies the rotation of the decoration symbol relative to the direction of the line. An angle of 0 means that the symbol's x-axis is oriented along the direction of the line.
 - For non-negative (or absent) *GAP* values it specifies the rotation of the decoration symbol around its center. An angle of 0 means that the symbol is not rotated.
- For polygons, it specifies the angle of the lines in a *HATCH* symbol (0 - horizontal lines), or it specifies the rotation of the symbol used to generate the pattern in a polygon fill (it does not specify the rotation of the fill as a whole). For its use with hatched lines, see Example #7 in the *symbolology examples*.
- [*attribute*] was introduced in version 5.0, to specify the attribute to use for angle values. The hard brackets [] are required. For example, if your data source has an attribute named "MYROTATE" that holds angle values for each feature, your *STYLE* object for hatched lines might contain:

```
STYLE
  SYMBOL 'hatch-test'
  COLOR 255 0 0
  ANGLE [MYROTATE]
  SIZE 4.0
  WIDTH 3.0
END
```

The associated RFC document for this feature is RFC19.

- The *AUTO* keyword was added in version 5.4, and currently only applies when coupled with the *GEOM-TRANSFORM* keyword.

Note: Rotation using *ANGLE* is not supported for *SYMBOLs* of *TYPE ellipse* with the GD renderer (gif).

ANGLEITEM [**string**] *ANGLE*[*attribute*] must now be used instead.

Deprecated since version 5.0.

ANTIALIAS [**true|false**] Should TrueType fonts be antialiased. Only useful for GD (gif) rendering. Default is false. Has no effect for the other renderers (where anti-aliasing can not be turned off).

BACKGROUND*COLOR* [*r*] [*g*] [*b*] - *deprecated* Color to use for non-transparent symbols.

Note: Multiple *STYLEs* can be used instead:

```
STYLE
  BACKGROUNDCOLOR 0 0 0
  SYMBOL "foo"
  COLOR 255 0 0
END
```

can be replaced with:

```
STYLE
  COLOR 0 0 0
END
STYLE
  SYMBOL "foo"
  COLOR 255 0 0
END
```

Deprecated since version 6.2.

COLOR [*r*] [*g*] [*b*] | [*hex string*] | [*attribute*] Color to use for drawing features.

- *r*, *g* and *b* shall be integers [0..255]. To specify green, the following is used:

```
COLOR 0 255 0
```

- *hex string* can be
 - RGB value - “#rrgbb”. To specify magenta, the following is used:

```
COLOR "#FF00FF"
```

- RGBA value (adding translucence) - “#rrggbbaa”. To specify a semi-translucent magenta, the following is used:

```
COLOR "#FF00FFCC"
```

- [*attribute*] was introduced in version 5.0, to specify the attribute to use for color values. The hard brackets [] are required. For example, if your data set has an attribute named “MYPAIN” that holds color values for each record, use: object for might contain:

```
COLOR [MYPAIN]
```

If *COLOR* is not specified, and it is not a *SYMBOL* of *TYPE pixmap*, then the symbol will not be rendered.

The associated RFC document for this feature is RFC19.

GAP [**double**] *GAP* specifies the distance between *SYMBOLs* (center to center) for decorated lines and polygon fills in layer *SIZEUNITS*. For polygon fills, *GAP* specifies the distance between *SYMBOLs* in both the X and the Y direction. For lines, the centers of the *SYMBOLs* are placed on the line. As of MapServer 5.0 this also applies to PixMap symbols.

When scaling of symbols is in effect (*SYMBOLSCALEDENOM* is specified for the *LAYER*), *GAP* specifies the distance in layer *SIZEUNITS* at the map scale 1:*SYMBOLSCALEDENOM*.

- For lines, if *INITIALGAP* is not specified, the first symbol will be placed *GAP/2* from the start of the line.
- For lines, a negative *GAP* value will cause the symbols' X axis to be aligned relative to the tangent of the line.
- For lines, a positive *GAP* value aligns the symbols' X axis relative to the X axis of the output device.
- For lines, a *GAP* of 0 (the default value) will cause the symbols to be rendered edge to edge
- For polygons, a missing *GAP* or a *GAP* of less than or equal to the size of the symbol will cause the symbols to be rendered edge to edge.

Symbols can be rotated using *ANGLE*.

New in version 6.0: moved from *SYMBOL*

Note: The behaviour of *GAP* has not been stable over time. It has specified the amount of space between the symbols, and also something in between the amount of space between the symbols and the center to center distance. Since 6.2 *GAP* specifies the center to center distance between the symbols.

GEOMTRANSFORM [**bbox|centroid|end|labelpnt|labelpoly|start|vertices**<**expression**>] Used to indicate that the current feature will be transformed before the actual style is applied. Introduced in version 5.4.

- *bbox*: produces the bounding box of the current feature geometry.
- *centroid*: produces the centroid of the current feature geometry.
- *end*: produces the last point of the current feature geometry. When used with *ANGLE AUTO*, it can for instance be used to render arrowheads on line segments.
- *labelpnt*: used for *LABEL* styles. Draws a marker on the geographic position the label is attached to. This corresponds to the center of the label text only if the label is in position CC.
- *labelpoly*: used for *LABEL* styles. Produces a polygon that covers the label plus a 1 pixel padding.
- *start*: produces the first point of the current feature geometry. When used with *ANGLE AUTO*, it can for instance be used to render arrow tails on line segments.
- *vertices*: produces all the intermediate vertices (points) of the current feature geometry (the start and end are excluded). When used with *ANGLE AUTO*, the marker is oriented by the half angle formed by the two adjacent line segments.
- <*expression*>: Applies the given expression to the geometry. Supported expressions:
 - (*buffer*(*[shape]*,*dist*)): Buffer the geometry (*[shape]*) using *dist* pixels as buffer distance. For polygons, a negative *dist* will produce a setback.
 - (*generalize*(*[shape]*,*tolerance*)): simplifies a geometry (*[shape]*) in way comparable to FME's ThinNoPoint algorithm. See <http://trac.osgeo.org/gdal/ticket/966> for more information.

Note: Depends on GEOS.

- (*simplify*(*[shape]*,*tolerance*)): simplifies a geometry (*[shape]*) using the standard Douglas-Peucker algorithm.

- (*simplifypt([shape],tolerance)*): simplifies a geometry (*[shape]*), ensuring that the result is a valid geometry having the same dimension and number of components as the input. *tolerance* must be non-negative.
- (*smoothsia([shape], smoothing_size, smoothing_iteration, preprocessing)*): will smooth a geometry (*[shape]*) using the SIA algorithm

Example (polygon data set) - draw a two pixel wide line 5 pixels inside the boundary of the polygon:

```
STYLE
  OUTLINECOLOR 255 0 0
  WIDTH 2
  GEOMTRANSFORM (buffer([shape],-5))
END
```

There is a difference between STYLE and LAYER GEOMTRANSFORM. LAYER-level will receive ground coordinates (meters, degrees, etc) and STYLE-level will receive pixel coordinates. The argument to methods such as simplify() must be in the same units as the coordinates of the shapes at that point of the rendering workflow, i.e. pixels at the STYLE-level and in ground units at the LAYER-level.

```
LAYER NAME "my_layer"
  TYPE LINE
  STATUS DEFAULT
  DATA "lines.shp"
  GEOMTRANSFORM (simplify([shape], 10)) ## 10 ground units
  CLASS
    STYLE
      GEOMTRANSFORM (buffer([shape], 5)) ## 5 pixels
      WIDTH 2
      COLOR 255 0 0
    END
  END
END
```

See also:

Geometry Transformations

INITIALGAP [double] *INITIALGAP* is useful for styling dashed lines.

If used with *GAP*, *INITIALGAP* specifies the distance to the first symbol on a styled line.

If used with *PATTERN*, *INITIALGAP* specifies the distance to the first dash on a dashed line.

Example 1 - dashed line styled with circles:

```
STYLE
  COLOR 0 0 0
  WIDTH 4
  PATTERN 40 10 END
END
STYLE
  SYMBOL "circlef"
  COLOR 0 0 0
  SIZE 8
  INITIALGAP 20
  GAP 50
END
```

Example 1 - dashed line styled with dashed line overlay:

```

STYLE
  COLOR 0 0 0
  WIDTH 6
  PATTERN 40 10 END
END
STYLE
  COLOR 255 255 255
  WIDTH 4
  INITIALGAP 2
  PATTERN 36 14 END
END

```

New in version 6.2.

LINECAP [butt|round|square] Sets the line cap type for lines. Default is *round*. See *Cartographical Symbol Construction with MapServer* for explanation and examples.

New in version 6.0: moved from *SYMBOL*

LINEJOIN [round|miter|bevel] Sets the line join type for lines. Default is *round*. See *Cartographical Symbol Construction with MapServer* for explanation and examples.

New in version 6.0: moved from *SYMBOL*

LINEJOINMAXSIZE [int] Sets the max length of the *miter* *LINEJOIN* type. The value represents a coefficient which multiplies a current symbol size. Default is 3. See *Cartographical Symbol Construction with MapServer* for explanation and examples.

New in version 6.0: moved from *SYMBOL*

MAXSCALEDENOM [double] Minimum scale at which this *STYLE* 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.

New in version 5.4.

See also:

Map Scale

MAXSIZE [double] Maximum size in pixels to draw a symbol. Default is 500. Starting from version 5.4, the value can also be a decimal value (and not only integer). See *LAYER SYMBOLSCALEDENOM*.

MAXWIDTH [double] Maximum width in pixels to draw the line work. Default is 32. Starting from version 5.4, the value can also be a decimal value (and not only integer). See *LAYER SYMBOLSCALEDENOM*.

MINSCALEDENOM [double] Minimum scale at which this *STYLE* 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.

New in version 5.4.

See also:

Map Scale

MINSIZE [double] Minimum size in pixels to draw a symbol. Default is 0. Starting from version 5.4, the value can also be a decimal value (and not only integer). See *LAYER SYMBOLSCALEDENOM*.

MINWIDTH [double] Minimum width in pixels to draw the line work. Default is 0. Starting from version 5.4, the value can also be a decimal value (and not only integer). See *LAYER SYMBOLSCALEDENOM*.

OFFSET [x][y] Geometry offset values in layer *SIZEUNITS*. In the general case, *SIZEUNITS* will be pixels.

When scaling of symbols is in effect (*SYMBOLSCALEDENOM* is specified for the *LAYER*), *OFFSET* gives offset values in layer *SIZEUNITS* at the map scale 1:*SYMBOLSCALEDENOM*.

An *OFFSET* of 20 40 will shift the geometry 20 *SIZEUNITS* to the left and 40 *SIZEUNITS* down before rendering.

For lines, an *OFFSET* of $y = -99$ will produce a line geometry that is shifted x *SIZEUNITS* perpendicular to the original line geometry. A positive x shifts the line to the right when seen along the direction of the line. A negative x shifts the line to the left when seen along the direction of the line.

For lines, an *OFFSET* of $y = -999$ (added in version 6.4) will produce a multiline geometry corresponding to the borders of a line that is x *SIZEUNITS* wide. This can be used to render only the outlines of a thick line.

OPACITY [*integer*|*attribute*] Opacity to draw the current style (applies to 5.2+, *AGG Rendering Specifics* only, does not apply to pixmap symbols)

- [*attribute*] was introduced in version 5.6, to specify the attribute to use for opacity values.

OUTLINECOLOR [*r*] [*g*] [*b*] | [*attribute*] Color to use for outlining polygons and certain marker symbols (*ellipse*, *vector* polygons and *truetype*). Has no effect for lines. The width of the outline can be specified using *WIDTH*. If no *WIDTH* is specified, an outline of one pixel will be drawn.

If there is a *SYMBOL* defined for the *STYLE*, the *OUTLINECOLOR* will be used to create an outline for that *SYMBOL* (only *ellipse*, *truetype* and polygon *vector* symbols will get an outline). If there is no *SYMBOL* defined for the *STYLE*, the polygon will get an outline.

- r , g and b shall be integers [0..255]. To specify green, the following is used:

```
OUTLINECOLOR 0 255 0
WIDTH 3.0
```

- [*attribute*] was introduced in version 5.0, to specify the attribute to use for color values. The hard brackets [] are required. For example, if your data set has an attribute named "MYPAIN" that holds color values for each record, use: object for might contain:

```
OUTLINECOLOR [MYPAIN]
```

The associated RFC document for this feature is RFC19.

OUTLINEWIDTH [*double*|*attribute*] Width in pixels for the outline. Default is 0.0. The thickness of the outline will not depend on the scale.

New in version 5.4.

PATTERN [*double on*] [*double off*] [*double on*] [*double off*] ... **END** Used to define a dash pattern for line work (lines, polygon outlines, hatch lines, ...). The numbers (doubles) specify the lengths of the dashes and gaps of the dash pattern in layer *SIZEUNITS*. When scaling of symbols is in effect (*SYMBOLSCALEDENOM* is specified for the *LAYER*), the numbers specify the lengths of the dashes and gaps in layer *SIZEUNITS* at the map scale 1:*SYMBOLSCALEDENOM*.

LINECAP, *LINEJOIN* and *LINEJOINMAXSIZE* can be used to control the appearance of the dashed lines.

To specify a dashed line that is 5 units wide, with dash lengths of 5 units and gaps of 5 units, the following style can be used:

```
STYLE
  COLOR 0 0 0
  WIDTH 5.0
  LINECAP BUTT
  PATTERN 5.0 5.0 END
END
```

Since version 6.2, *PATTERN* can be used to create dashed lines for *SYMBOLS* of *TYPE hatch*. Patterns for hatches are always drawn with *LINECAP butt*. The patterns are generated relative to the edges of the bounding box of the polygon (an illustrated example can be found in the *hatch fill section of the symbol construction document*).

New in version 6.0: moved from *SYMBOL*

POLAROFFSET [*doubleattribute*] [*doubleattribute*] Offset given in polar coordinates.

The first parameter is a double value in layer *SIZEUNITS* (or the name of a layer attribute) that specifies the radius/distance.

The second parameter is a double value (or the name of a layer attribute) that specifies the angle (counter clockwise).

When scaling of symbols is in effect (*SYMBOLSCALEDENOM* is specified for the *LAYER*), *POLAROFFSET* gives the distance in layer *SIZEUNITS* at the map scale 1:*SYMBOLSCALEDENOM*.

A *POLAROFFSET* of 20 40 will shift the geometry to a position that is 20 *SIZEUNITS* away along a line that is at an angle of 40 degrees with a line that goes horizontally to the right.

When *POLAROFFSET* is used with layers that have *CONNECTIONTYPE* *uvraster* (*vector field*), the special attributes *uv_length*, *uv_length_2*, *uv_angle* and *uv_minus_angle* are available, making it convenient to specify arrow heads and tails. Example:

```
LAYER
...
TYPE POINT
CONNECTIONTYPE uvraster
...
CLASS
  STYLE
    SYMBOL "arrowbody"
    ANGLE [uv_angle]
    SIZE [uv_length]
    WIDTH 3
    COLOR 100 255 0
  END
  STYLE
    SYMBOL "arrowhead"
    ANGLE [uv_angle]
    SIZE 10
    COLOR 255 0 0
    POLAROFFSET [uv_length_2] [uv_angle]
  END
  STYLE
    SYMBOL "arrowtail"
    ANGLE [uv_angle]
    SIZE 10
    COLOR 255 0 0
    POLAROFFSET [uv_length_2] [uv_minus_angle]
  END
END #class
END #layer
```

New in version 6.2: (rfc78)

SIZE [*doubleattribute*] Height, in layer *SIZEUNITS*, of the symbol/pattern to be used. Default value depends on the *SYMBOL TYPE*. For *pixmap*: the height (in pixels) of the pixmap; for *ellipse* and *vector*: the maximum y value of the *SYMBOL POINTS* parameter, for *hatch*: 1.0, for *truetype*: 1.0.

When scaling of symbols is in effect (*SYMBOLSCALEDENOM* is specified for the *LAYER*), *SIZE* gives the height, in layer *SIZEUNITS*, of the symbol/pattern to be used at the map scale 1:*SYMBOLSCALEDENOM*.

- For symbols of *TYPE hatch*, the *SIZE* is the center to center distance between the lines. For its use with hatched lines, see Example#8 in the *symbolology examples*.

- `[attribute]` was introduced in version 5.0, to specify the attribute to use for size values. The hard brackets `[]` are required. For example, if your data set has an attribute named “MYHIGHT” that holds size values for each feature, your `STYLE` object for hatched lines might contain:

```
STYLE
  SYMBOL 'hatch-test'
  COLOR 255 0 0
  ANGLE 45
  SIZE [MYHIGHT]
  WIDTH 3.0
END
```

The associated RFC document for this feature is RFC19.

- Starting from version 5.4, the value can also be a decimal value (and not only integer).

SYMBOL `[integer|string|filename|url|attribute]` The symbol to use for rendering the features.

- Integer is the index of the symbol in the symbol set, starting at 1 (the 5th symbol is symbol number 5).
- String is the name of the symbol (as defined using the `SYMBOL NAME` parameter).
- Filename specifies the path to a file containing a symbol. For example a PNG file. Specify the path relative to the directory containing the mapfile.
- URL specifies the address of a file containing a pixmap symbol. For example a PNG file. A URL must start with “http”:

```
SYMBOL "http://myserver.org/path/to/file.png"
```

New in version 6.0.

- `[attribute]` allows individual rendering of features by using an attribute in the dataset that specifies the symbol name (as defined in the `SYMBOL NAME` parameter). The hard brackets `[]` are required.

New in version 5.6.

If `SYMBOL` is not specified, the behaviour depends on the type of feature.

- For points, nothing will be rendered.
- For lines, `SYMBOL` is only relevant if you want to style the lines using symbols, so the absence of `SYMBOL` means that you will get lines as specified using the relevant line rendering parameters (`COLOR`, `WIDTH`, `PATTERN`, `LINECAP`, ...).
- For polygons, the interior of the polygons will be rendered using a solid fill of the color specified in the `COLOR` parameter.

See also:

[SYMBOL](#)

WIDTH `[double|attribute]` `WIDTH` refers to the thickness of line work drawn, in layer `SIZEUNITS`. Default is 1.0.

When scaling of symbols is in effect (`SYMBOLSCALEDENOM` is specified for the `LAYER`), `WIDTH` refers to the thickness of the line work in layer `SIZEUNITS` at the map scale 1:`SYMBOLSCALEDENOM`.

- If used with `SYMBOL` and `OUTLINECOLOR`, `WIDTH` specifies the width of the symbol outlines. This applies to `SYMBOL TYPE` *vector* (polygons), *ellipse* and *truetype*.
- For lines, `WIDTH` specifies the width of the line.
- For polygons, if used with `OUTLINECOLOR`, `WIDTH` specifies the thickness of the polygon outline.
- For a symbol of `SYMBOL TYPE` *hatch*, `WIDTH` specifies the thickness of the hatched lines. For its use with hatched lines, see Example #7 in the [symbolology examples](#).

- *[attribute]* was added in version 5.4 to specify the attribute to use for the width value. The hard brackets [] are required.
- Starting from version 5.4, the value can also be a decimal value (and not only integer).

4.1.23 SYMBOL

- Symbol definitions can be included within the main map file 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, Lines and Shadesets.
- 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 *STYLE* objects.
- Line styling was moved to *CLASS STYLE* in MapServer version 5. The mechanisms are no longer available in *SYMBOL*.
- 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.

ANCHORPOINT [*x*] [*y*] Used to specify the location (within the symbol) that is to be used as an anchorpoint when rotating the symbol and placing the symbol on a map. Default is **0.5 0.5** (corresponding to the center of the symbol).

x: A double in the range [0,1] that specifies the location within the symbol along the x axis. 0 specifies the left edge of the symbol, 1 specifies the right edge of the symbol. 0.5 specifies the center of the symbol (in the x direction).

y: A double in the range [0,1] that specifies the location within the symbol along the y axis. 0 specifies the top edge of the symbol, 1 specifies the lower edge of the symbol. 0.5 specifies the center of the symbol (in the y direction).

ANCHORPOINT can be used with *SYMBOLS* of *TYPE ellipse*, *pixmap*, *svg*, *truetype* and *vector*. To ensure proper behaviour for vector symbols, the left and top edges of the bounding box of the symbol should be at 0.

New in version 6.2.

ANTI_ALIAS [*true|false*] Should TrueType fonts be antialiased. Only useful for GD (gif) rendering. Default is *false*. Has no effect for the other renderers (where anti-aliasing can not be turned off).

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*] If *true*, the symbol will be filled with a user defined color (using *STYLE COLOR*). Default is *false*.

If *true*, symbols of *TYPE ellipse* and *vector* will be treated as polygons (fill color specified using *STYLE COLOR* and outline specified using *STYLE OUTLINECOLOR* and *WIDTH*).

If *false*, symbols of *TYPE ellipse* and *vector* will be treated as lines (the lines can be given a color using *STYLE COLOR* and a width using *STYLE WIDTH*).

FONT [*string*] Name of TrueType font to use as defined in the *FONTSET*.

IMAGE [*string*] Image (GIF or PNG) to use as a marker or brush for type *pixmap* symbols.

NAME [**string**] Alias for the symbol. To be used in *CLASS STYLE* objects.

POINTS [*x y*] [*x y*] ... **END**

Signifies the start of a sequence of points that make up a symbol of *TYPE vector* or that define the *x* and *y* radius of a symbol of *TYPE ellipse*. The end of this section is signified with the keyword *END*. The *x* and *y* values can be given using decimal numbers. The maximum *x* and *y* values define the bounding box of the symbol. The size (actually height) of a symbol is defined in the *STYLE*. You can create non-contiguous paths by inserting “-99 -99” at the appropriate places.

x values increase to the right, *y* values increase downwards.

For symbols of *TYPE ellipse*, a single point is specified that defines the *x* and *y* radius of the ellipse. Circles are created when *x* and *y* are equal.

Note: If a *STYLE* 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.

TRANSPARENT [**color index**] Sets a transparent color for the input 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.

Note: The default (AGG) renderer does not support the *TRANSPARENT* parameter. It is supported by the GD renderer (GIF).

TYPE [*ellipse|hatch|pixmap|svg|truetype|vector*]

- *ellipse*: radius values in the *x* and *y* directions define an ellipse.
- *hatch*: produces hatched lines throughout the (polygon) shape.
- *pixmap*: a user supplied image will be used as the symbol.
- *svg*: scalable vector graphics (SVG) symbol. Requires the libsvg-cairo library.
- *truetype*: TrueType font to use as defined in the *MAP FONTSET*.
- *vector*: a vector drawing is used to define the shape of the symbol.

Note: *TYPE cartoline* is no longer used. Dashed lines are specified using *PATTERN*, *LINECAP*, *LINEJOIN* and *LINEJOINMAXSIZE* in *STYLE*. Examples in *Construction of Cartographic Symbols*.

4.1.24 Symbology Examples

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Håvard Tveite

Contact havard.tveite at nmbu.no

Date \$Date\$

Revision \$Revision\$

Last Updated 2011/05/11

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. Using the Symbol Type HATCH (new in 4.6)*
 - *Example 8. Styled lines using GAP*

Example 1. Dashed Line

This example creates a dashed line that is 5 *SIZEUNITS* wide, with 10 *SIZEUNITS* on, 5 off, 5 on, 10 off ...

```
LAYER
...
CLASS
...
STYLE
  COLOR 0 0 0
  WIDTH 5
  LINECAP butt
  PATTERN 10 5 5 10 END
END
END
END
```

Example 2. TrueType font marker symbol

This example symbol is a star, used to represent the national capital, hence the name. The font name is defined in the *FONSET* file. The code number "114" varies, you can use MS Windows' character map to figure it out, or guesstimate.

```
SYMBOL
  NAME "natcap"
  TYPE TRUETYPE
  FONT "geo"
  FILLED true
  ANTIALIAS true # only necessary for GD rendering
  CHARACTER "&#114;"
END
```

Example 3. Vector triangle marker symbol

This example is fairly straight forward. Note that to have 3 sides you need 4 points, hence the first and last points are identical. The triangle is not filled.

```
SYMBOL
  NAME "triangle"
  TYPE vector
  POINTS
```

```
0 4
2 0
4 4
0 4
END
END
```

Example 4. Non-contiguous vector marker symbol (Cross)

This 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.

```
SYMBOL
NAME "cross"
TYPE vector
POINTS
2.0 0.0
2.0 4.0
-99 -99
0.0 2.0
4.0 2.0
END
END
```

Example 5. Circle vector symbol

This example creates a simple filled circle. Using non-equal values for the point will give you an actual ellipse.

```
SYMBOL
NAME "circle"
TYPE ellipse
FILLED true
POINTS
1 1
END
END
```

Example 6. Downward diagonal fill

This example creates a symbol that can be used to create a downward diagonal fill for polygons.

```
SYMBOL
NAME "downwarddiagonalfill"
TYPE vector
TRANSPARENT 0
POINTS
0 1
1 0
END
END
```

Example 7. Using the Symbol Type HATCH (new in 4.6)

As of MapServer 4.6, you can use the symbol type HATCH to produce hatched lines. The following will display hatched lines at a 45 degree angle, 10 *SIZEUNITS* apart (center to center), and 3 *SIZEUNITS* 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: MINSIZE, MAXSIZE, MINWIDTH, and MAXWIDTH.

Example 8. Styled lines using GAP

This example shows how to style lines with symbols.

A 5 *SIZEUNITS* wide black line is decorated with ellipses that are 15 *SIZEUNITS* long (and 7.5 *SIZEUNITS* wide). The ellipses are placed 30 *SIZEUNITS* apart, and the negative *GAP* value ensures that the ellipses are oriented relative to the direction of the line. The ellipses are rotated 30 degrees counter clock-wise from their position along the line.

Symbol definition:

```
SYMBOL
  NAME "ellipse2"
  TYPE ellipse
  FILLED true
  POINTS
    1 2
  END
END
```

Layer definition:

```
LAYER
  ...
  CLASS
    ...
    STYLE
      WIDTH 5
      COLOR 0 0 0
    END
  STYLE
```

```
SYMBOL 'ellipse2'  
COLOR 0 0 0  
ANGLE 30  
SIZE 15  
GAP -30  
END  
END  
END
```

4.1.25 Templating

Author Frank Koormann

Contact frank.koormann at intevation.de

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision\$

Date \$Date\$

Table of Contents

- *Templating*
 - *Introduction*
 - *Format*
 - *Example Template*

Introduction

Templates are used:

- to define the look of a MapServer CGI application interface
- to present the results of a query.
- to create customised output (see *Template-Driven Output*)

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.

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

[date] Outputs the date (as per the web server’s clock). The default format is the same as is used by Apache’s Common Log format, which looks like:

```
01/Dec/2010:17:34:58 -0800
```

Available arguments:

- **format=** A format string as supported by the standard C `strftime()` function. As an example, the default format is defined as:

```
[date format="%d/%b/%Y:%H:%M:%S %z"]
```

- **tz=** timezone to use for the date returned. Default is “local”. Valid values are:
 - “gmt” Output date will be Greenwich time
 - “local” Output the time in the web server’s local time zone.

Additionally or alternatively, the `%z` and `%Z` `strftime` format strings allow the timezone offset or name to be output.

[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):

- **buffer=**, Buffer size, currently the only unit available is pixels. Default is 0.
- **centroid=** Should only the centroid of the shape be used? true or false (case insensitive). Default is false.
- **cs=** Coordinate separator. Default is “,”.
- **irh=**, **irf=**, **orh=**, **orf=**

Characters to be put before (*irh*) and after (*irf*) inner rings, and before (*orh*) and after (*orf*) outer rings of polygons with holes. Defaults are "".

Note: Within each polygon, the outer ring is always output first, followed by the inner rings.

If neither *irh* nor *orh* are set, rings are output as “parts” using *ph/pf/ps*.

- **ph=, pf=, ps=** Characters to put before (*ph*) and after (*pf*) and separators between (*ps*) feature parts (e.g. rings of multigeometries). Defaults are `ph=""`, `pf=""` and `ps=""`.
- **precision=** The number of decimal places to output for coordinates. Default is 0.
- **proj=** The output projection definition for the coordinates, a special value of “image” will convert to image coordinates. Default is none.
- **scale=, scale_x=, scale_y=** Scaling factor for coordinates: Both axes (*scale*), x axis (*scale_x*) and y axis (*scale_y*). Defaults are 1.0.
- **sh=, sf=** Characters to put before (*sh*) and after (*sf*) a feature. Defaults are "".
- **xh=, xf=** Characters to put before (*xh*) and after (*xf*) the x coordinates. Defaults are `xh=""` and `xf=";"`.
- **yh= yf=** Characters to put before (*yh*) and after (*yf*) the y coordinates. Defaults are "".

As a simple example:

```
[shpxy xh="(" yf=")"] will result in: (x1 y1), (x2 y2), (x3 y3)
```

And a more complicated example of outputting KML for multipolygons which may potentially have holes (note that the parameters must all be on one line):

```
<MultiGeometry>
  <Point>
    <coordinates>[shplabel proj=epsg:4326 precision=10],0</coordinates>
  </Point>
  [shpxy ph="<Polygon><tessellate>1</tessellate>" pf=""</Polygon>" xf=","
  xh=" " yh=" " yf=","0 " orh="<outerBoundaryIs><LinearRing><coordinates>"
  orf="</coordinates></LinearRing></outerBoundaryIs>"
  irh="<innerBoundaryIs><LinearRing><coordinates>"
  irf="</coordinates></LinearRing></innerBoundaryIs>" proj=epsg:4326
  precision=10]
</MultiGeometry>
```

[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.

[item formatting options] An attribute table “item”, with list formatting options. The “name” attribute is required.

Available only when processing query results.

Available attributes:

- **name =** The name of an attribute, case insensitive. (required)
- **precision =** The number of decimal places to use for numeric data. Use of this will force display as a number and will lead to unpredictable results with non-numeric data.
- **pattern =** Regular expression to compare the value of an item against. The tag is output only if there is a match.
- **uc =** Set this attribute to “true” to convert the attribute value to upper case.
- **lc =** Set this attribute to “true” to convert the attribute value to lower case.

- **commify** = Set this attribute to “true” to add commas to a numeric value. Again, only useful with numeric data.
- **escape** = Default escaping is for HTML, but you can escape for inclusion in a URL (=url), or not escape at all (=none).
- **format** = A format string used to output the attribute value. The token “\$value” is used to place the value in a more complex presentation. Default is to output only the value.
- **nullformat** = String to output if the attribute value is NULL, empty or doesn’t match the pattern (if defined). If not set and any of these conditions occur the item tag is replaced with an empty string.

As a simple example:

```
[item name="area" precision="2" commify="2" format="Area is $value"]
```

[attribute name],[attribute name_esc],[attribute item name_raw] Attribute name from the data 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 data 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_attribute name],[Join name_attribute name_esc], [Join name_attribute 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.

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" [zoom_2_select]> Zoom in 2 times
47            <option value="1" [zoom_1_select]> Recenter Map
48            <option value="-2" [zoom_-2_select]> 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
60     </td>
61 </tr>
62 </table>
63
64 </form>
65
66 </body>
</html>
```

4.1.26 Union Layer

Author Tamas Szekeres

Contact szekerest at gmail.com

Author Jeff McKenna

Contact jmkenna at gatewaygeomatics.com

Last Updated 2011-04-11

Table of Contents

- *Union Layer*
 - *Description*
 - *Requirements*
 - *Mapfile Configuration*
 - *Feature attributes*
 - *Classes and Styles*
 - *Projections*
 - *Supported Processing Options*
 - *Examples*
 - * *Mapfile Example*
 - * *PHP MapScript Example*

Description

Since version 6.0, MapServer has the ability to display features from multiple layers (called ‘*source layers*’) in a single mapfile layer. This feature was added through rfc68.

Requirements

This is a native MapServer option that doesn’t use any external libraries to support it.

Mapfile Configuration

- The CONNECTIONTYPE parameter must be set to UNION.

- The CONNECTION parameter must contain a comma separated list of the source layer names.
- All of the source layers and the union layer must be the same TYPE (e.g. all must be TYPE POINT, or all TYPE POLYGON etc.)

Note: You may wish to disable the visibility (change their STATUS) of the source layers to avoid displaying the features twice.

For example:

```
LAYER
  NAME "union-layer"
  TYPE POINT
  STATUS DEFAULT
  CONNECTIONTYPE UNION
  CONNECTION "layer1,layer2,layer3" # reference to the source layers
  PROCESSING "ITEMS=itemname1,itemname2,itemname3"
  ...
END
LAYER
  NAME "layer1"
  TYPE POINT
  STATUS OFF
  CONNECTIONTYPE OGR
  CONNECTION ...
  ...
END
LAYER
  NAME "layer2"
  TYPE POINT
  STATUS OFF
  CONNECTIONTYPE OGR
  CONNECTION ...
  ...
END
LAYER
  NAME "layer3"
  TYPE POINT
  STATUS OFF
  CONNECTIONTYPE OGR
  CONNECTION ...
  ...
END
```

Feature attributes

In the LAYER definition you may refer to any attributes supported by each of the source layers. In addition to the source layer attributes the union layer provides the following additional attributes:

1. Combine:SourceLayerName - The name of the source layer the feature belongs to
2. Combine:SourceLayerGroup - The group of the source layer the feature belongs to

During the selection / feature query operations only the 'Combine:SourceLayerName' and 'Combine:SourceLayerGroup' attributes are provided by default. The set of the provided attributes can manually be overridden (and further attributes can be exposed) by using the ITEMS processing option (refer to the example above).

Classes and Styles

We can define the symbology and labelling for the union layers in the same way as for any other layer by specifying the classes and styles. In addition the `STYLEITEM AUTO` option is also supported for the union layer, which provides to display the features as specified at the source layers. The source layers may also use the `STYLEITEM AUTO` setting if the underlying data source provides that.

Projections

For speed, it is recommended to always use the same projection for the union layer and source layers. However MapServer will reproject the source layers to the union layer if requested. (for more information on projections in MapServer refer to *PROJECTION*)

Supported Processing Options

The following processing options can be used with the union layers:

UNION_STATUS_CHECK (TRUE or FALSE) Controls whether the status of the source layers should be checked and the invisible layers (`STATUS=OFF`) should be skipped. Default value is `FALSE`.

UNION_SCALE_CHECK (TRUE or FALSE) Controls whether the scale range of the source layers should be checked and the invisible layers (falling outside of the scale range and zoom range) should be skipped. Default value is `TRUE`.

UNION_SRCLAYER_CLOSE_CONNECTION Override the connection pool setting of the source layers. By introducing this setting we alter the current behaviour which is equivalent to:

```
UNION_SRCLAYER_CLOSE_CONNECTION=ALWAYS
```

Examples

Mapfile Example

The follow example contains 3 source layers in different formats, and one layer (yellow) in a different projection. The union layer uses the **STYLEITEM “AUTO”** parameter to draw the styles from the source layers. (in this case MapServer will reproject the yellow features, in EPSG:4326, for the union layer, which is in EPSG:3978).



```

MAP
...
PROJECTION
  "init=epsg:3978"
END
...
LAYER
  NAME 'unioned'
  TYPE POLYGON
  STATUS DEFAULT
  CONNECTIONTYPE UNION
  CONNECTION "red,green,yellow"
  STYLEITEM "AUTO"
  # Define an empty class that will be filled at runtime from the color and
  # styles read from each source layer.
  CLASS
  END
  PROJECTION
    "init=epsg:3978"
  END
END

LAYER
  NAME 'red'
  TYPE POLYGON
  STATUS OFF
  DATA 'nb.shp'
  CLASS
    NAME 'red'
    STYLE

```

```
        OUTLINECOLOR 0 0 0
        COLOR 255 85 0
    END
END
END

LAYER
    NAME 'green'
    TYPE POLYGON
    STATUS OFF
    CONNECTIONTYPE OGR
    CONNECTION 'ns.mif'
    CLASS
        NAME 'green'
        STYLE
            OUTLINECOLOR 0 0 0
            COLOR 90 218 71
        END
    END
END

LAYER
    NAME 'yellow'
    TYPE POLYGON
    STATUS OFF
    CONNECTIONTYPE OGR
    CONNECTION 'pei.gml'
    CLASS
        NAME 'yellow'
        STYLE
            OUTLINECOLOR 0 0 0
            COLOR 255 255 0
        END
    END
PROJECTION
    "init=epsg:4326"
END
END # Map
```

PHP MapScript Example

```
<?php
// open map
$Map = ms_newMapObj( "D:/ms4w/apps/osm/map/osm.map" );

// create union layer
$Layer = ms_newLayerObj($Map);
$Layer->set("name", "unioned");
$Layer->set("type", MS_LAYER_POLYGON);
$Layer->set("status", MS_ON);
$Layer->setConnectionType(MS_UNION);
$Layer->set("connection", "red,green,yellow");
$Layer->set("styleitem", "AUTO");
```

```

$0Layer->setProjection("init=epsg:3978");
// create empty class
$0Class = ms_newClassObj($0Layer);
...
?>

```

4.1.27 WEB

BROWSEFORMAT [**mime-type**] Format of the interface output, using MapServer CGI. (*added to MapServer 4.8.0*)
The default value is “text/html”. Example:

```
BROWSEFORMAT "image/svg+xml"
```

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.

LEGENDFORMAT [**mime-type**] Format of the legend output, using MapServer CGI. (*added to MapServer 4.8.0*)
The default value is “text/html”. Example:

```
LEGENDFORMAT "image/svg+xml"
```

LOG [**filename**] Since MapServer 5.0 the recommended parameters to use for debugging are the *MAP* object’s **CONFIG** and **DEBUG** parameters instead (see the *Debugging MapServer* document).

File to log MapServer activity in. Must be writable by the user the web server is running as.

Deprecated since version 5.0.

MAXSCALEDENOM [**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 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.

See also:

Map scale

MAXSCALE [**double**] - **deprecated** Since MapServer 5.0 the proper parameter to use is **MAXSCALEDENOM** instead. The deprecated **MAXSCALE** 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 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.

Deprecated since version 5.0.

MAXTEMPLATE [**fileurl**] Template to be used if below the minimum scale for the app (the denominator of the requested scale is larger than **MAXSCALEDENOM**), useful for nesting apps.

METADATA This keyword allows for arbitrary data to be stored as name value pairs.

- Used with OGC services (*WMS Server*, *WFS Server*, *WCS Server*, *SOS Server*, ...) 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.
- If you have XMP support enabled, you can also embed `xmp_metadata` in your output images by specifying XMP tag information here. Example:

```
METADATA
  title "My layer title"
  author "Me!"
  xmp_dc Title "My Map Title"
END
```

- **labelcache_map_edge_buffer**

For tiling, the amount of gutter around an image where no labels are to be placed is controlled by the parameter *labelcache_map_edge_buffer*. The unit is pixels. The value had to be a negative value for 6.0 and earlier versions. From 6.2 the absolute value is taken, so the sign does not matter.

```
METADATA
  "labelcache_map_edge_buffer" "10"
END
```

- **ms_enable_modes**

Enable / disable modes (see rfc90).

Use the asterisk "*" to specify all modes and a preceding exclamation sign "!" to negate the given condition

To disable all CGI modes:

```
METADATA
  "ms_enable_modes" "!*"
END
```

To disable everything but MAP and LEGEND:

```
METADATA
  "ms_enable_modes" "!* MAP LEGEND"
END
```

MINSCALE [**double**] Maximum scale at which this interface is valid. When a user requests a map at a larger 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 **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 maximum scale at which this interface is valid. When a user requests a map at a larger 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 maximum scale for the app (the denominator of the requested scale is smaller than **MINSCALEDENOM**), useful for nesting apps.

QUERYFORMAT [**mime-type**] Format of the query output. (*added to MapServer 4.8.0*) This works for mode=query (using query templates in CGI mode), but not for mode=browse. The default value is “text/html”. Example:

```
QUERYFORMAT "image/svg+xml"
```

TEMPLATE [filename|url]

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 ...).

URL is not a remote file, rather a template. For example:

```
TEMPLATE 'http://someurl/somescript.cgi?mapext=[mapext]'
```

TEMPPATH Path for storing temporary files. If not set, the standard system temporary file path will be used (e.g. tmp for unix). *TEMPPATH* can also be set using the environment variable *MS_TEMPPATH*.

TEMPPATH is used in many contexts (see rfc66).

Make sure that that MapServer has sufficient rights to read and write files at the specified location.

New in version 6.0.

VALIDATION Signals the start of a *VALIDATION* block.

As of MapServer 5.4.0, *VALIDATION* blocks are the preferred mechanism for specifying validation patterns for CGI param runtime substitutions. See *Run-time Substitution*.

4.1.28 XML Mapfile support

MapServer is able to load XML mapfiles automatically, without user XSLT transformations. Basically, MapServer will simply do an XSLT transformation when the mapfile passed to it is an XML one, convert it to a text mapfile in a temporary file on disk, then process the mapfile normally.

New Dependencies

- libxslt
- libexslt

Enabling the support

You can enable the XML mapfile support by adding the following option: `--with-xml-mapfile`. This configure option will enable the libxslt and libexslt check up. If your libxslt/libexslt are not installed in /usr, you'll have to add the following options:

```
--with-xslt=/path/to/xslt/installation
--with-exslt=/path/to/exslt/installation
```

Usage:

In order to enable this feature, set the `MS_XMLMAPFILE_XSLT` environment variable to point to the location of the XSLT to use for the XML->text mapfile conversion. e.g. in Apache:

```
SetEnv MS_XMLMAPFILE_XSLT /path/to/mapfile.xsl
PassEnv MS_XMLMAPFILE_XSLT
```

With this enabled, passing an .xml filename to the CGI *map* parameter will automatically trigger the conversion.

Note: This is a first step to XML mapfile loading support. Obviously, there is a cost to parse and translate the XML mapfile, but this allows easier use of XML mapfiles.

4.1.29 Notes

- The Mapfile is NOT case-sensitive.
- The Mapfile is read from top to bottom by MapServer; this means that LAYERS near the top of your Mapfile will be drawn before those near the bottom. Therefore users commonly place background imagery and other background layer types near the top of their mapfile, and lines and points near the bottom of their mapfile.
- 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 shape data sets 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.

5.1 MapScript

Release 6.4.1

5.1.1 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*

Appendices

Language-specific extensions are described in the following appendices

Python 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.

5.1.2 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\$

Date \$Date\$

Contents

- *SWIG MapScript API Reference*
 - *Introduction*
 - * *Appendices*
 - * *Documentation Elements*
 - * *fooObj*
 - * *Additional Documentation*
 - *MapScript Constants*
 - * *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*
 - *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*

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 might be exposed by objects.

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.

might be exposed byClass 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.

MapScript Constants

The constants are ordered alphabetically within each group.

Version

Name	Type
MS_VERSION	character

Logical Control - Boolean Values

Name	Type
MS_FALSE	integer
MS_NO	integer
MS_OFF	integer
MS_ON	integer
MS_TRUE	integer
MS_YES	integer

Logical Control - Status Values

Name	Type
MS_DEFAULT	integer
MS_DELETE	integer
MS_EMBED	integer

Map Units

Name	Type
MS_DD	integer
MS_FEET	integer
MS_INCHES	integer
MS_METERS	integer
MS_MILES	integer
MS_NAUTICALMILES	integer
MS_PIXELS	integer

Layer Types

Name	Type
MS_LAYER_ANNOTATION (deprecated since 6.2)	integer
MS_LAYER_CIRCLE	integer
MS_LAYER_LINE	integer
MS_LAYER_POINT	integer
MS_LAYER_POLYGON	integer
MS_LAYER_QUERY	integer
MS_LAYER_RASTER	integer
MS_LAYER_TILEINDEX	integer

Font Types

Name	Type
MS_BITMAP	integer
MS_TRUETYPE	integer

Label Positions

Name	Type
MS_AUTO	integer
MS_CC	integer
MS_CL	integer
MS_CR	integer
MS_LC	integer
MS_LL	integer
MS_LR	integer
MS_UC	integer
MS_UL	integer
MS_UR	integer

Label Size (Bitmap only)

Name	Type
MS_GIANT	integer
MS_LARGE	integer
MS_MEDIUM	integer
MS_SMALL	integer
MS_TINY	integer

Shape Types

Name	Type
MS_SHAPE_LINE	integer
MS_SHAPE_NULL	integer
MS_SHAPE_POINT	integer
MS_SHAPE_POLYGON	integer

Measured Shape Types

Name	Type
MS_SHP_ARCM	integer
MS_SHP_MULTIPPOINTM	integer
MS_SHP_POINTM	integer
MS_SHP_POLYGONM	integer

Shapefile Types

Name	Type
MS_SHAPEFILE_ARC	integer
MS_SHAPEFILE_MULTIPPOINT	integer
MS_SHAPEFILE_POINT	integer
MS_SHAPEFILE_POLYGON	integer

Query Types

Name	Type
MS_MULTIPLE	integer
MS_SINGLE	integer

File Types

Name	Type
MS_FILE_MAP	integer
MS_FILE_SYMBOL	integer

Querymap Styles

Name	Type
MS_HILITE	integer
MS_NORMAL	integer
MS_SELECTED	integer

Connection Types

Name	Typ
MS_GRATICULE	integer
MS_INLINE	integer
MS_MYGIS	integer
MS_OGR	integer
MS_ORACLESPATIAL	integer
MS_POSTGIS	integer
MS_RASTER	integer
MS_SDE	integer
MS_SHAPEFILE	integer
MS_TILED_SHAPEFILE	integer
MS_WFS	integer
MS_WMS	integer

DB Connection Types

Name	Type
MS_DB_CSV	integer
MS_DB_MYSQL	integer
MS_DB_ORACLE	integer
MS_DB_POSTGRES	integer
MS_DB_XBASE	integer

Join Types

Name	Type
MS_JOIN_ONE_TO_MANY	integer
MS_JOIN_ONE_TO_ONE	integer

Line Join Types (for rendering)

Name	Type
MS_CJC_BEVEL	integer
MS_CJC_BUTT	integer
MS_CJC_MITER	integer
MS_CJC_NONE	integer
MS_CJC_ROUND	integer
MS_CJC_SQUARE	integer
MS_CJC_TRIANGLE	integer

Image Types

Name	Type
GD/GIF	integer
GD/JPEG	integer
GD/PNG	integer
GD/PNG24	integer
GD/WBMP	integer
GDAL/GTiff	integer
imagemap	integer
pdf	integer
swf	integer

Image Modes

Name	Type
MS_GD_ALPHA	integer
MS_IMAGEMODE_BYTE	integer
MS_IMAGEMODE_FLOAT32	integer
MS_IMAGEMODE_INT16	integer
MS_IMAGEMODE_NULL	integer
MS_IMAGEMODE_PC256	integer
MS_IMAGEMODE_RGB	integer
MS_IMAGEMODE_RGBA	integer
MS_NOOVERRIDE	integer

Symbol Types

Name	Type
MS_SYMBOL_ELLIPSE	integer
MS_SYMBOL_PIXMAP	integer
MS_SYMBOL_SIMPLE	integer
MS_SYMBOL_TRUETYPE	integer
MS_SYMBOL_VECTOR	integer

Return Codes

Name	Type
MS_DONE	integer
MS_FAILURE	integer
MS_SUCCESS	integer

Limiters

Name	Type
MS_IMAGECACHESIZE	long
MS_MAXSTYLELENGTH	long
MS_MAXSYMBOLS	long
MS_MAXVECTORPOINTS	long

Error Return Codes

Name	Type
MESSAGELENGTH	long
MS_CGIERR	long
MS_CHILDERR	long
MS_DBFERR	long
MS_EOFERR	long
MS_GDERR	long
Continued on next page	

Table 5.1 – continued from previous page

MS_HASHERR	long
MS_HTTPERR	long
MS_IDENTERR	long
MS_IMGERR	long
MS_IOERR	long
MS_JOINERR	long
MS_MAPCONTEXTERR	long
MS_MEMERR	long
MS_MISCELLERR	long
MS_NOERR	long
MS_NOTFOUND	long
MS_NUMERRORCODES	long
MS_OGRERR	long
MS_ORACLESPATIALERR	long
MS_PARSEERR	long
MS_PROJERR	long
MS_QUERYERR	long
MS_REGEXERR	long
MS_SDEERR	long
MS_SHPERR	long
MS_SYMERR	long
MS_TTFERR	long
MS_TYPEERR	long
MS_WCSERR	long
MS_WEBERR	long
MS_WFSCONNERR	long
MS_WFSERR	long
MS_WMSCONNERR	long
MS_WMSERR	long
ROUTINELENGTH	long

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.

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_getStdoutBufferString() [string] Fetch the current stdout buffer contents as a string. This method does not clear the buffer.

msIO_installStdinFromBuffer() [void] Installs a mapserver IO handler directing future stdin reading (ie. post request capture) to come from a buffer.

msIO_installStdoutToBuffer() [void] Installs a mapserver IO handler directing future stdout output to a memory buffer.

msIO_resetHandlers() [void] Resets the default stdin and stdout handlers in place of “buffer” based handlers.

msIO_stripStdoutBufferContentHeaders(): void Strip all Content-* headers off the stdout buffer if it has ones.

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).

msResetErrorList() [void] Clears the current error stack.

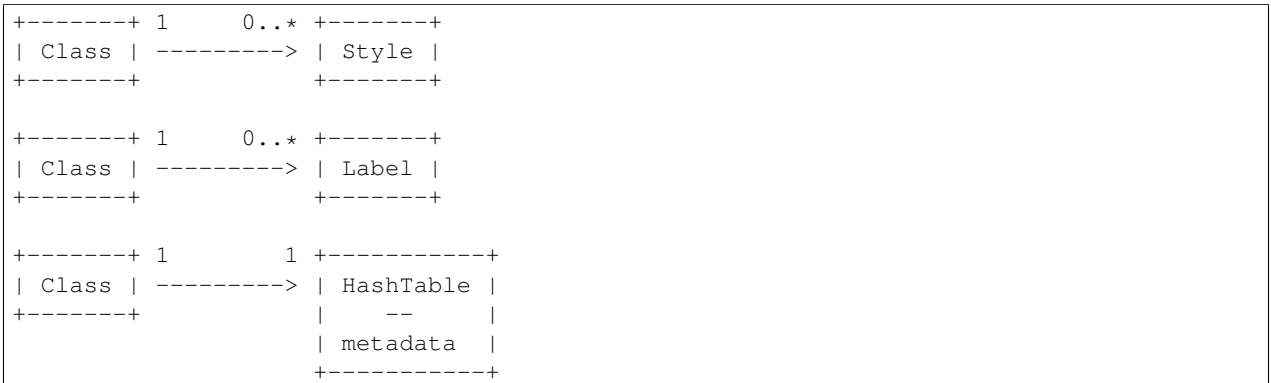
MapScript Classes

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 have been supported since 4.1, and multiple class labels since 6.2. 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. Removed (6.2) - use addLabel, getLabel and removeLabel instead.

layer [*layerObj* immutable] Reference to the parent layer

maxscaledenom [float] The minimum scale at which class is drawn

metadata [*hashTableObj* immutable] class metadata hash table.

minscaledenom [float] The maximum scale at which class is drawn

name [string] Unique within a layer

numlabels [int] Number of labels for class.

New in version 6.2.

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.

addLabel(labelObj) [int] Add a labelObj to the classObj and return its index in the labels array.

New in version 6.2.

clone() [classObj] Return an independent copy of the class without a parent layer.

convertToString() [string] Saves the object to a string. Provides the inverse option for updateFromString. .. version-added:: 6.4

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.

Note: getFirstMetaDataKey(), getMetaData(), and getNextMetaDataKey() are deprecated and will be removed in a future version. Replaced by direct metadata access, see *hashTableObj*.

getLabel(int index) [*labelObj*] Return a reference to the labelObj at *index* in the labels array.

See the *labelObj* section for more details on multiple class labels.

New in version 6.2.

getMetaData(string key) [string] Return the value of the classObj metadata at *key*.

Note: getFirstMetaDataKey(), getMetaData(), and getNextMetaDataKey() are deprecated and will be removed in a future version. Replaced by direct metadata access, see *hashTableObj*.

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.

removeLabel(int index) [*labelObj*] Remove the labelObj at *index* from the labels array and return a reference to the labelObj. numlabels is decremented, and the array is updated.

New in version 6.2.

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]

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().

red [int] Red component of color in range [0-255]

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.

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`.

setRGB(int red, int green, int blue) [int] Set all three RGB components. 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_IMAGE_MAP. 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:

```
+-----+ 0..*      1 +-----+
| LabelCacheMember | <----- | LabelCache |
+-----+                +-----+
```

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.

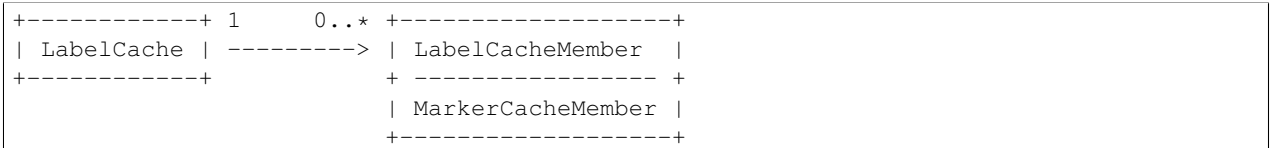
tileindex [int immutable] Tileindex of the layer of the labeled feature.

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.

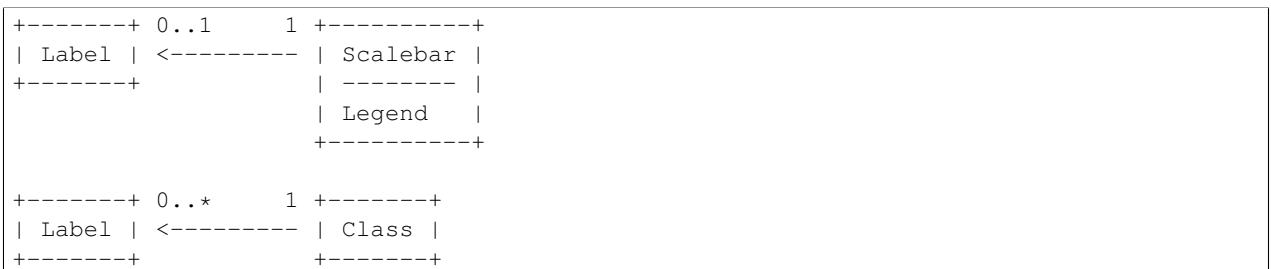
numMarkers [int immutable] Number of marker members.

labelCacheObj Methods

freeCache() [void] Free the labelcache.

labelObj

A labelObj is associated with a classObj, a scalebarObj, or a legendObj:



An instance of labelObj can exist outside of a classObj container and be explicitly inserted into the classObj:

```

new_label = new labelObj()
the_class.addLabel(new_label)
    
```

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 rfc11 for specifics).

autominfeaturesize: int MS_TRUE or MS_FALSE

backgroundcolor [*colorObj*] Color of background rectangle or billboard.

Deprecated since version 6.0: Use *styleObj* and *geomtransform*.

backgroundshadowcolor [*colorObj*] Color of background rectangle or billboard shadow.

Deprecated since version 6.0: Use *styleObj* and *geomtransform*.

backgroundshadowsize [int] Horizontal offset of drop shadow in pixels.

Deprecated since version 6.0: Use *styleObj* and *geomtransform*.

backgroundshadowsizey [int] Vertical offset of drop shadow in pixels.

Deprecated since version 6.0: Use *styleObj* and *geomtransform*.

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.

numstyles [int] Number of label styles

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

convertToString() [string] Saves the object to a string. Provides the inverse option for updateFromString. .. version-added:: 6.4

getBinding(int binding) [string] Get the attribute binding for a specified label property. Returns NULL if there is no binding for this property.

getExpressionString() [string] Returns the label expression string.

getStyle(int index) [*styleObj*] Return a reference to the styleObj at *index* in the styles array.

getTextString() [string] Returns the label text 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.

removeBinding(int binding) [int] Remove the attribute binding for a specified label property.

removeStyle(int index) [*styleObj*] Remove the styleObj at *index* from the styles array and return a copy.

setBinding (int binding, string item) [int] Set the attribute binding for a specified label property. Binding constants look like this: MS_LABEL_BINDING_[attribute name]:

```
setBinding(MS_LABEL_BINDING_COLOR, "FIELD_NAME_COLOR");
```

setExpression(string expression) [int] Set the label expression.

setText(string text) [int] Set the label text.

updateFromString (string snippet) [int] Update a label from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

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] Since 6.0, *dump* is not available anymore. *metadata* is used instead.
Switch to allow mapserver to return data in GML format. MS_TRUE or MS_FALSE. Default is MS_FALSE.
Deprecated since version 6.0: *metadata* is used instead.
- 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.
- labelmaxscaledenom** [float] Minimum scale at which layer will be labeled.
- labelminscaledenom** [float] Maximum scale at which layer will be labeled.
- labelrequires** [string] Logical expression.
- labelsizeitem** [string] Attribute defining label size.
- map** [*mapObj* immutable] Reference to parent map.
- mask** [string] Layer name for masking. (rfc79)
- maxfeatures** [int] Maximum number of layer features that will be drawn. For shapefile data this means the first N features where N = maxfeatures.
- maxscaledenom** [float] Minimum scale at which layer will be drawn.
- metadata** [*hashTableObj* immutable] Layer metadata.
- minscaledenom** [float] Maximum 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.

opacity [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.

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_NAUTICALMILES, MS_METERS, MS_KILOMETERS, MS_DD or MS_PIXELS

status [int] MS_ON, MS_OFF, or MS_DEFAULT.

styleitem [string] Attribute defining styles.

symbolscaledenom [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_NAUTICALMILES, 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.

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.

convertToString() [string] Saves the object to a string. Provides the inverse option for `updateFromString`. .. version-added:: 6.4

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 layer (the layer must have *STATUS on*).

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*.

Note: `getFeature` has been removed as of version 6.0 and replaced by `getShape`

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.

Note: `getFirstMetaDataKey()`, `getMetaData()`, and `getNextMetaDataKey()` are deprecated and will be removed in a future version. Replaced by direct metadata access, see *hashTableObj*.

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.

Note: `getFirstMetaDataKey()`, `getMetaData()`, and `getNextMetaDataKey()` are deprecated and will be removed in a future version. Replaced by direct metadata access, see *hashTableObj*.

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.

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.

getResultsBounds() [*rectObj*] Returns the bounds of the features in the result cache.

getShape(resultCacheMemberObj result) [int] Get a shape from layer data. Argument is a result cache member from `layerObj::getResult(i)`

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 queryable.

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, minscaledenom, and maxscaledenom 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*.

resultsGetShape(int shapeindex [, int tileindex = -1]) [*shapeObj*] Retrieve *shapeObj* from a layer's resultset by index. Tileindex is optional and is used only for tiled shapefiles, Simply omit or pass tileindex = -1 for other data sources. Added in MapServer 5.6.0 due to the one-pass query implementation.

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.

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, MS_FAILURE or MS_DONE. MS_DONE is returned if the layer extent does not overlap rect.

legendObj

legendObj is associated with mapObj:

```
+-----+ 0..1      1 +-----+
| Legend | <-----> | Map |
+-----+                +-----+
```

and with *labelObj*:

```
+-----+ 1          1 +-----+
| Legend | -----> | Label |
+-----+                +-----+
```

legendObj Attributes

height [int] Legend height.

imagecolor [*colorObj*] Legend background color.

keysize_x [int] Width in pixels of legend keys.

keysize_y [int] Pixels.

keyspacing_x [int] Horizontal padding around keys in pixels.

keyspacing_y [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.

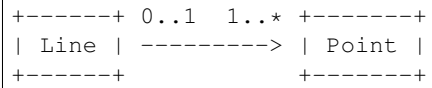
width [int] Label width.

legendObj Methods

convertToString() [string] Saves the object to a string. Provides the inverse option for updateFromString. .. version-added:: 6.4

lineObj

A lineObj is composed of one or more *pointObj* instances:



lineObj Attributes

numpoints [int immutable] Number of points in the line.

lineObj Methods

new lineObj() [lineObj] Create a new instance.

add(*pointObj* point) [int] Add *point* to the line. Returns MS_SUCCESS or MS_FAILURE.

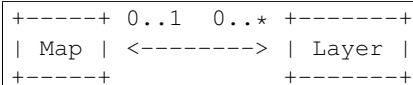
get(int index) [*pointObj*] Return reference to point at *index*.

project(*projectionObj* proj_in, *projectionObj* proj_out) [int] Transform line in place from *proj_in* to *proj_out*. Returns MS_SUCCESS or MS_FAILURE.

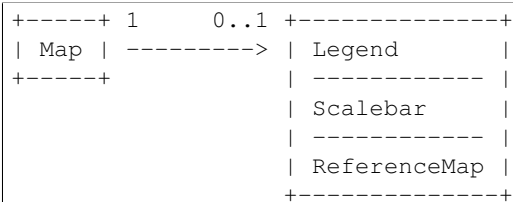
set(int index, *pointObj* point) [int] Set the point at *index* to *point*. Returns MS_SUCCESS or MS_FAILURE.

mapObj

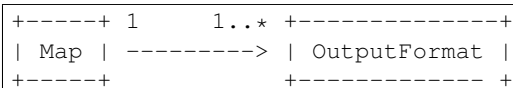
A mapObj is primarily associated with instances of layerObj:



Secondary associations are with legendObj, scalebarObj, referenceMapObj:



outputFormatObj:



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] The number of output formats currently configured on the map object. Can be used to iterate over the list of output formats with the getOutputFormat(idx) method (see below).

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.

Note: As of 6.2 other languages can use the getoutputFormat(idx) and getNumoutputformats() functions to iterate over the format array.

querymap [queryMapObj immutable] **TODO** should this be exposed to mapscript?

reference [*referenceMapObj* immutable] Reference to reference map.

resolution [float] Nominal DPI resolution. Default is 72.

scalebar [*scalebarObj* immutable] Reference to the scale bar.

scaledenom [float] The nominal map scale. A value of 25000 means 1:25000 scale.

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 mapscript.

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'.

convertToString() [string] Saves the object to a string. Provides the inverse option for updateFromString. .. version-added:: 6.4

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 that have *STATUS on*.

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.

getLabel(*int labelindex*) [*labelCacheMemberObj*] Return label at specified index from the map's labelcache.

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.

getOutputFormat(*int i*): outputFormatObj Returns the output format at the specified *i* index from the output formats array or null if *i* is beyond the array bounds. The number of outputFormats can be retrieved by calling *getNumoutputformats*.

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().

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.

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.

processQueryTemplate(string names[], string values[], int numitems) [string] Process MapServer query 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.

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. This method will correct the extents (width/height ratio) before setting the minx,miny,maxx,maxy values. See extent properties to set up a custom extent from *rectObj*.

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.

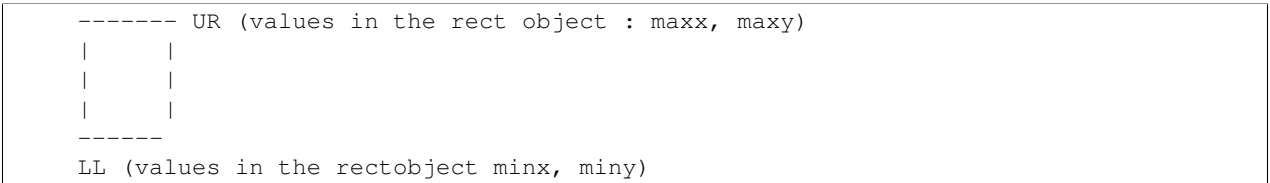
setSize(int width, int height) [int] Set map's image width and height together and carry out the necessary subsequent geotransform computation. Returns MS_SUCCESS or MS_FAILURE.

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.

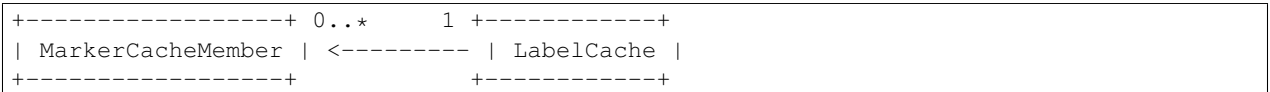
zoomRectangle(*rectObj* imgrect, int width, int height, *rectObj* extent, *rectObj* maxextent) : int Zoom to a pixel coordinate rectangle in the image of *width* and *height* dimensions and georeferencing *extent*. Zooming can be constrained to a maximum *maxextent*. The *imgrect* rectangle contains the coordinates of the LL and UR coordinates in pixel: the maxy in the rect object should be < miny value. 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 formatoption ... 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.

numformatoptions: int The number of option values set on this format. Can be used to iterate over the options array in conjunction with *getOptionAt*

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 defaultvalue=""]) [string] Return the format option at *key* or *defaultvalue* if *key* is not a valid hash index.

getOptionAt(int idx): string Returns the option at *idx* or null if the index is beyond the array bounds. The option is returned as the original KEY=VALUE string. The number of available options can be obtained by calling *getNumformatoptions*.

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.

Note: MapServer's OWSRequest supports only single valued parameters.

addParameter(string name, string value) [void] Add a request parameter, even if the parameter key was previously set. This is useful when multiple parameters with the same key are required. For example:

```
request.addParameter('SIZE', 'x(100)')
request.addParameter('SIZE', 'y(100)')
```

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!

loadParamsFromURL(string url) [int] Initializes the OWSRequest object from the provided URL which is treated like a QUERY_STRING. Note that REQUEST_METHOD=GET and no post data is assumed in this case. This method was added in MapServer 6.0.

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')
```

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.

toShape() [*shapeObj*] Convenience method to quickly turn a point into a shapeObj.

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 }
```

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:

```
+-----+ 0..1      1 +-----+
| ReferenceMap | <-----> | Map |
+-----+                +-----+
```

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

convertToString() [string] Saves the object to a string. Provides the inverse option for updateFromString. .. version-added:: 6.4

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) [*resultCacheMemberObj*] 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.

height [int] Pixels.

imagecolor [*colorObj*] Background color of scalebar.

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

convertToString() [string] Saves the object to a string. Provides the inverse option for `updateFromString`. .. version-added:: 6.4

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 effecient 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.

clone() [shapeObj] Return an independent copy of the shape.

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.

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*.

simplify(double tolerance): shapeObj Given a tolerance, returns a simplified shape object or NULL on error. Requires GEOS support (>=3.0).

symDifference(shapeObj shape) [shapeObj] Returns the computed symmetric difference of the supplied and existing shape. Requires GEOS support. Returns NULL/undef on failure.

topologySimplifyPreservingSimplify(double tolerance): shapeObj Given a tolerance, returns a simplified shape object or NULL on error. Requires GEOS support (>=3.0).

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:



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] Deprecated since version 5.0: Use setBinding.

antialias [int] MS_TRUE or MS_FALSE. Should TrueType fonts be antialiased.

backgroundcolor [*colorObj*] Background pen color.

color [*colorObj*] Foreground or fill pen color.

mincolor [*colorObj*] Attribute for Color Range Mapping (rfc6). 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 (rfc6). 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 (rfc6). 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 (rfc6). 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 (rfc6).

size [int] Pixel width of the style's pen or symbol.

sizeitem [string] Deprecated since version 5.0: Use setBinding.

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 width 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.

convertToString() [string] Saves the object to a string. Provides the inverse option for updateFromString. .. version-added:: 6.4

getBinding(int binding) [string] Get the attribute binding for a specified style property. Returns NULL if there is no binding for this property.

removeBinding(int binding) [int] Remove the attribute binding for a specified style property.

setBinding (int binding, string item) [int] Set the attribute binding for a specified style property. Binding constants look like this: MS_STYLE_BINDING_[attribute name]:

```
setBinding(MS_STYLE_BINDING_SIZE, 'mySizeItem');
```

setSymbolByName(*mapObj* map, string symbolname) [int] Setting the symbol of the styleObj given the reference of the map object and the symbol name.

updateFromString (string snippet) [int] Update a style from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

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] Moved to STYLE

imagepath [string] Path to pixmap file.

inmapfile [int] If set to TRUE, the symbol will be saved inside the mapfile. Added in MapServer 5.6.1

linecap [int] Moved to STYLE

linejoin [int] Moved to STYLE

linejoinmaxsize [float] Moved to STYLE

name [string] Symbol name

numpoints [int immutable] Number of points of a vector symbol.

position [int] No more available?

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, or MS_SYMBOL_TRUETYPE.

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.

maxscaledenom [float] Minimum map scale.

maxtemplate [string] **TODO**

metadata [*hashTableObj* immutable] metadata hash table.

minscaledenom [float] Maximum map scale.

mintemplate [string] **TODO**

queryformat [string] **TODO**

template [string] Path to template document.

webObj Methods

convertToString() [string] Saves the object to a string. Provides the inverse option for `updateFromString`. .. version-added:: 6.4

5.1.3 PHP MapScript

Release 6.4.1

Introduction

This is a [PHP](#) module that makes MapServer's MapScript functionalities available in a PHP Dynamically Loadable Library. In simple terms, this module will allow you to use the powerful PHP scripting language to dynamically create and modify map images in MapServer.

Versions Supported

PHP 5.2.0 or more recent is required; since MapServer 6.0, support for PHP 4, PHP 5.0 and PHP 5.1 have been dropped. PHP MapScript was originally developed for PHP 3.0.14, and after MapServer 3.5 support for PHP 3 was dropped.

The module has been tested and used on Linux, Solaris, *BSD, and Windows.

Note: If you are using MapServer 5.6 and older, please refer to the [PHP MapScript 5.6 documentation](#) instead.

Note: If you are migrating your existing application that is based on MapServer 5.6 or older, to MapServer 6.0 or beyond, please read the [PHP MapScript Migration Guide](#) for important changes.

How to Get More Information on PHP MapScript

- For installation questions regarding the PHP MapScript module, see [PHP MapScript Installation](#).
- The [MapServer Wiki](#) has information on this module, that was contributed by users.
- New PHP MapScript users should read the [By Example](#) document.
- The project's home is the [PHP/MapScript page](#) on MapTools.org.
- Also, see the [MapScript](#), and the [Mapfile](#) sections of this site.
- Refer to the main [PHP site](#) for their official documentation.

Memory Management

Normally, you should not have to worry about the memory management because php has a garbage collector and will free resources for you. If you write only small scripts that don't do a lot of processing, it's not worth to care about that. Everything will be freed at the end of the script.

However, it may be useful to free resources during the execution if the script executes many tasks. To do so, you'll have to call the **free()** method of the mapscript objects and unset the php variables. The purpose of the free methods is to break the circular references between an object and its properties to allow the zend engine to free the resources.

Here's an example of a script that doesn't free things during the execution:

```
$map = new mapObj("mapfile.map");
$of = $map->outputformat;
echo $map->extent->minx." - ".$map->extent->miny." - ".
      $map->extent->maxx." - ".$map->extent->maxy."\n";
echo "Outputformat name: $of->name\n";
unset($of);
unset($map); // Even if we unset the php variables, resources
              // wont be freed. Resources will be only freed
              // at the end of the script
```

and the same script that frees resources as soon as it can

```
$map = new mapObj("mapfile.map");
$of = $map->outputformat;
echo $map->extent->minx." - ".$map->extent->miny." - ".
      $map->extent->maxx." - ".$map->extent->maxy."\n";
echo "Outputformat name: $of->name\n";
unset($of);
$map->free(); // break the circular references
// at this place, the outputformat ($of) and the rect object
// ($map->extent) resources are freed
unset($map);
// the map object is immediately freed after the unset (before the
// end of the script)
```

PHP MapScript API

Author Daniel Morissette

Contact dmorissette at mapgears.com

Author Yewondwossen Assefa

Contact yassefa at dmsolutions.ca

Author Alan Boudreault

Contact aboudreault at mapgears.com

Revision \$Revision\$

Date \$Date\$

Note: If you are using MapServer 5.6 and older, please refer to the PHP MapScript 5.6 documentation instead.

Contents

- *PHP MapScript API*
 - *Important Note*
 - *Constants*
 - *Functions*
 - *Classes*
 - * *classObj*
 - * *clusterObj*
 - * *colorObj*
 - * *errorObj*
 - * *gridObj*
 - * *hashTableObj*
 - * *imageObj*
 - * *labelcacheMemberObj*
 - * *labelcacheObj*
 - * *labelObj*
 - * *layerObj*
 - * *legendObj*
 - * *lineObj*
 - * *mapObj*
 - * *outputformatObj*
 - * *OwsrequestObj*
 - * *pointObj*
 - * *projectionObj*
 - * *querymapObj*
 - * *rectObj*
 - * *referenceMapObj*
 - * *resultObj*
 - * *scalebarObj*
 - * *shapefileObj*
 - * *shapeObj*
 - * *styleObj*
 - * *symbolObj*
 - * *webObj*

Important Note

- Constant names and class member variable names are case-sensitive in PHP.

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, MS_NAUTICALMILES

Layer types MS_LAYER_POINT, MS_LAYER_LINE, MS_LAYER_POLYGON, MS_LAYER_RASTER, MS_LAYER_ANNOTATION (deprecated since 6.2), 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_XY, MS_AUTO, MS_AUTO2, MS_FOLLOW, MS_NONE

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_RASTER, MS_PLUGIN, MS_UNION

Error codes MS_NOERR, MS_IOERR, MS_MEMERR, MS_TYPEERR, MS_SYMERR, 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_WMCONNERR, MS_ORACLESPIATIALERR, MS_WFSERR, MS_WFCONNERR, MS_MAPCONTEXTERR, MS_HTTPERR, MS_WCSERR

Symbol types MS_SYMBOL_SIMPLE, MS_SYMBOL_VECTOR, MS_SYMBOL_ELLIPSE, MS_SYMBOL_PIXMAP, MS_SYMBOL_TRUETYPE

Image Mode types (*outputFormatObj*) MS_IMAGEMODE_PC256, MS_IMAGEMODE_RGB, MS_IMAGEMODE_RGBA, MS_IMAGEMODE_INT16, MS_IMAGEMODE_FLOAT32, MS_IMAGEMODE_BYTE, MS_IMAGEMODE_FEATURE, 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, MS_STYLE_BINDING_WIDTH

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, MS_LABEL_BINDING_POSITION, MS_LABEL_BINDING_SHADOWSIZE_X, MS_LABEL_BINDING_SHADOWSIZE_Y

Alignment MS_ALIGN_LEFT, MS_ALIGN_CENTER, MS_ALIGN_RIGHT

OwsRequest MS_GET_REQUEST, MS_POST_REQUEST

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.

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_iogetstdoutbufferstring() Fetch the current stdout buffer contents as a string. This method does not clear the buffer.

void ms_ioinstallstdinfrombuffer() Installs a mapserver IO handler directing future stdin reading (ie. post request capture) to come from a buffer.

void ms_ioinstallstdouttobuffer() Installs a mapserver IO handler directing future stdout output to a memory buffer.

void ms_ioresethandlers() Resets the default stdin and stdout handlers in place of “buffer” based handlers.

string 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.

void ms_iostripstdoutbuffercontentheaders() Strip all the Content-* headers off the stdout buffer if it has some.

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.

Classes

The following class objects are available through PHP MapScript.

classObj

Constructor Class Objects can be returned by the *layerObj* class, or can be created using:

```
new classObj(layerObj layer [, classObj class])
```

or using the old constructor

```
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.

Type	Name	Note
string	group	
string	keyimage	
labelObj	label	Removed (6.2) - use addLabel, getLabel, ...
double	maxscaledenom	
hashTableObj	metadata	
double	minscaledenom	
string	name	
int	numlabels	read-only (since 6.2)
int	numstyles	read-only
int	status	MS_ON, MS_OFF or MS_DELETE
string	template	
string	title	
int	type	

Members

int addLabel(labelObj label) Add a labelObj to the classObj and return its index in the labels array.

New in version 6.2.

string convertToString() Saves the object to a string. Provides the inverse option for `updateFromString`.

imageObj createLegendIcon(int width, int height) Draw the legend icon and return a new `imageObj`.

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.

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`.

void free() Free the object properties and break the internal references. Note that you have to unset the `php` variable to free totally the resources.

string getExpressionString() Returns the *expression* string for the class object.

labelObj getLabel(int index) Return a reference to the `labelObj` at *index* in the labels array.

See the *labelObj* section for more details on multiple class labels.

New in version 6.2.

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.

styleObj getStyle(int index) Return the style object using an index. `index >= 0 && index < class->numstyles`.

string getTextString() Returns the text string for the class object.

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 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.

labelObj removeLabel(int index) Remove the `labelObj` at *index* from the labels array and return a reference to the `labelObj`. `numlabels` is decremented, and the array is updated.

New in version 6.2.

int removeMetaData(string name) Remove a metadata entry for the class. Returns `MS_SUCCESS/MS_FAILURE`.

int set(string property_name, new_value) Set object property to a new value.

int setExpression(string expression) Set the *expression* string for the class object.

int setMetaData(string name, string value) Set a metadata entry for the class. Returns `MS_SUCCESS/MS_FAILURE`.

int settext(string text) Set the text string for the class object.

int updateFromString(string snippet) Update a class from a string snippet. Returns `MS_SUCCESS/MS_FAILURE`.

```
/*set the color */
$oClass->updateFromString('CLASS STYLE COLOR 255 0 255 END END');
```

clusterObj

Constructor Instance of `clusterObj` is always embedded inside the *layerObj*.

	Type	Name
Members	double	buffer
	double	maxdistance
	string	region

Methods

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

string getFilterString() Returns the *expression* for this cluster filter or NULL on error.

string getGroupString() Returns the *expression* for this cluster group or NULL on error.

int setFilter(string expression) Set layer filter *expression*.

int setGroup(string expression) Set layer group *expression*.

colorObj

Constructor Instances of colorObj are always embedded inside other classes.

	Type	Name
Members	int	red
	int	green
	int	blue

Methods

void setRGB(int red, int green, int blue) Set red, green, blue values.

errorObj 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.

	Type	Name
Members	int	code //See error code constants above
	string	message
	string	routine

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();
}
```

gridObj

Constructor The grid is always embedded inside a layer object defined as a grid (layer->connectiontype = MS_GRATICULE) (for more docs : <https://github.com/mapserver/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
string	labelformat
double	maxacrs
double	maxinterval
double	maxsubdivide
double	minarcs
double	mininterval
double	minsubdivide

Methods

int set(string property_name, new_value) Set object property to a new value.

hashTableObj

Constructor Instance of hashTableObj is always embedded inside the *classObj*, *layerObj*, *mapObj* and *webObj*. 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

void clear() Clear all items in the hashTable (To NULL).

string get(string key) Fetch class metadata entry by name. Returns "" if no entry matches the name. Note that the search is case sensitive.

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

int remove(string key) Remove a metadata entry in the hashTable. Returns MS_SUCCESS/MS_FAILURE.

int set(string key, string value) Set a metadata entry in the hashTable. Returns MS_SUCCESS/MS_FAILURE.

imageObj

Constructor Instances of imageObj are always created by the *mapObj* class methods.

Type	Name	Note
int	width	read-only
int	height	read-only
int	resolution	read-only
int	resolutionfactor	read-only
string	imagepath	
string	imageurl	

Members

Methods

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).

int saveImage([string filename, MapObj oMap]) Writes image object to specified filename. Passing no filename or 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. On success, it returns either MS_SUCCESS 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.

labelcacheMemberObj Accessible only through the *mapObj* (map->getLabel()).

Type	Name	Note
int	classindex	read-only
int	featuresize	read-only
int	layerindex	read-only
int	markerid	read-only
int	numstyles	read-only
int	shapeindex	read-only
int	status	read-only
string	text	read-only
int	tileindex	read-only

Members

Method None

labelcacheObj Accessible only through the *mapObj* (map->labelcache). This object is only used to give the possibility to free the label cache (map->labelcache->freeCache())

Method

boolean freeCache() Free the label cache. Always returns MS_SUCCESS. Ex : map->labelcache->freeCache();

labelObj

Constructor labelObj are always embedded inside other classes.

```
new labelObj()
```

Type	Name
int	align
double	angle
int	anglemode
int	antialias
int	autominfeaturesize
colorObj	backgroundcolor (deprecated since 6.0)
colorObj	backgroundshadowcolor (deprecated since 6.0)
int	backgroundshadowsizeX (deprecated since 6.0)
int	backgroundshadowsizeY (deprecated since 6.0)
int	buffer
colorObj	color
string	encoding
string	font
int	force
int	maxlength
int	maxsize
int	mindistance
int	minfeaturesize
int	minlength
int	minsize
int	numstyles
int	offsetx
int	offsety
colorObj	outlinecolor
int	outlinewidth
int	partials
int	position
int	priority
int	repeatdistance
colorObj	shadowcolor
int	shadowsizeX
int	shadowsizeY
int	size
Continued on next page	

Table 5.2 – continued from previous page

Type	Name
int	type
int	wrap

Members

Methods

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

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.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

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
```

string getExpressionString() Returns the label expression string.

styleObj getStyle(int index) Return the style object using an index. index >= 0 && index < label->numstyles.

string getTextString() Returns the label text string.

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 label->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 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 label->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 removeBinding(const labelbinding) Remove the attribute binding for a specified style property.

Example:

```
$oStyle->removebinding(MS_LABEL_BINDING_COLOR);
```

int set(string property_name, new_value) Set object property to a new value.

int setBinding(const labelbinding, string value) Set the attribute binding for a specified label property.

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")

int setExpression(string expression) Set the label expression.

int setText(string text) Set the label text.

int updateFromString(string snippet) Update a label from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

layerObj

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.

Type	Name	Note
int	annotate	
hashTableObj	bindvals	
string	classgroup	
string	classitem	
clusterObj	cluster	
string	connection	
int	connectiontype	read-only, use <i>setConnectionType()</i> to set it
string	data	
int	debug	
int	dump	deprecated since 6.0
string	filteritem	
string	footer	
gridObj	grid	only available on a layer defined as grid (MS_GRATICULE)
string	group	
string	header	
int	index	read-only
int	labelcache	
string	labelitem	
double	labelmaxscaledenom	
double	labelminscaledenom	
string	labelrequires	
string	mask	
int	maxfeatures	
double	maxscaledenom	
hashTableObj	metadata	
double	minscaledenom	
string	name	
int	num_processing	
int	numclasses	read-only
colorObj	offsite	
int	opacity	
projectionObj	projection	
int	postlabelcache	
string	requires	
int	sizeunits	
int	startindex	
int	status	MS_ON, MS_OFF, MS_DEFAULT or MS_DELETE
string	styleitem	
double	symbolscaledenom	
string	template	

Continued on next page

Table 5.3 – continued from previous page

Type	Name	Note
string	tileindex	
string	tileitem	
double	tolerance	
int	toleranceunits	
int	transform	
int	type	

Members

Methods

int addFeature(shapeObj shape) Add a new feature in a layer. Returns MS_SUCCESS or MS_FAILURE 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.

void clearProcessing() Clears all the processing strings.

void close() Close layer previously opened with open().

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

int draw(imageObj image) Draw a single layer, add labels to cache if required. Returns MS_SUCCESS or MS_FAILURE on error.

int drawQuery(imageObj image) Draw query map for a single layer.

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.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

string generateSLD() Returns an SLD XML string based on all the classes found in the layer (the layer must have *STATUS on*).

classObj getClass(int classIndex) Returns a classObj from the layer given an index value (0=first class)

int getClassIndex(shape [, classgroup, numclasses]) Get the class index of a shape for a given scale. Returns -1 if no class matches. classgroup is an array of class ids to check (Optional). numclasses is the number of classes that the classgroup array contains. By default, all the layer classes will be checked.

rectObj getExtent() Returns the layer's data extents or NULL on error. 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.

string getFilterString() Returns the *expression* for this layer or NULL on error.

array getGridIntersectionCoordinates() Returns an array containing the grid intersection coordinates. If there are no coordinates, it returns an empty array.

array getItems() Returns an array containing the items. Must call open function first. If there are no items, it returns an empty array.

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 getNumResults() Returns the number of results in the last query.

array getProcessing() Returns an array containing the processing strings. If there are no processing strings, it returns an empty array.

string getProjection() Returns a string representation of the *projection*. Returns NULL on error or if no projection is set.

resultObj getResult(int index) Returns a resultObj 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.

rectObj getResultBounds() Returns the bounding box of the latest result.

shapeObj getShape(resultObj result) If the resultObj passed has a valid resultindex, retrieve shapeObj from a layer's resultset. (You get it from the resultObj returned by getResult() for instance). Otherwise, it will do a single query on the layer to fetch the shapeindex

```
$map = new mapObj("gmap75.map");
$l1 = $map->getLayerByName("popplace");
$l1->queryByRect($map->extent);
for ($i=0; $i<$l1->getNumResults();$i++){
    $s = $l1->getShape($l1->getResult($i));
    echo $s->getValue($l1,"Name");
    echo "\n";
}
```

string getWMSFeatureInfoURL(int clickX, int clickY, int featureCount, string infoFormat) Returns 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 queryable.

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 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.

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 open() Open the layer for use with getShape(). Returns MS_SUCCESS/MS_FAILURE.

shapeobj nextShape() Called after msWhichShapes has been called to actually retrieve shapes within a given area. Returns a shape object or NULL on error.

```
$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();

```

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 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 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).

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.

int removeMetaData(string name) Remove a metadata entry for the layer. Returns *MS_SUCCESS/MS_FAILURE*.

int set(string property_name, new_value) Set object property to a new value.

int setConnectionType(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 the 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.

int setFilter(string expression) Set layer filter *expression*.

int setMetaData(string name, string value) Set a metadata entry for the layer. Returns *MS_SUCCESS/MS_FAILURE*.

int setProcessing(string) Add the string to the processing string list for the layer. The layer->num_processing is incremented by 1. Returns MS_SUCCESS or MS_FAILURE on error.

```
$oLayer->setprocessing("SCALE_1=AUTO");
$oLayer->setprocessing("SCALE_2=AUTO");
```

int setProjection(string proj_params) Set layer *projection* and coordinate system. Parameters are given as a single string of comma-delimited PROJ.4 parameters. Returns MS_SUCCESS or MS_FAILURE on error.

int setWKTProjection(string proj_params) Same as setProjection(), but takes an OGC WKT projection definition string as input.

Note: setWKTProjection requires GDAL support

int updateFromString(string snippet) Update a layer from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

```
/*modify the name */
$oLayer->updateFromString('LAYER NAME land_fn2 END');
/*add a new class*/
$oLayer->updateFromString('LAYER CLASS STYLE COLOR 255 255 0 END END END');
```

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, MS_FAILURE or MS_DONE. MS_DONE is returned if the layer extent does not overlap the rectObj.

legendObj

Constructor Instances of legendObj are always are always embedded inside the *mapObj*.

Type	Name	Note
int	height	
colorObj	imagecolor	
int	keysizeX	
int	keysizeY	
int	keyspacingX	
int	keyspacingY	
labelObj	label	
colorObj	outlinecolor	Color of outline of box, -1 for no outline
int	position	for embeded legends, MS_UL, MS_UC, ...
int	postlabelcache	MS_TRUE, MS_FALSE
int	status	MS_ON, MS_OFF, MS_EMBED
string	template	
int	width	

Methods

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

int set(string property_name, new_value) Set object property to a new value.

int updateFromString(string snippet) Update a legend from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

lineObj

Constructor

```
new lineObj()
```

or using the old constructor

```
LineObj ms_newLineObj()
```

Members

Type	Name	Note
int	numpoints	read-only

Methods

int add(pointObj point) Add a point to the end of line. Returns MS_SUCCESS/MS_FAILURE.

int addXY(double x, double y [, double m]) Add a point to the end of line. Returns MS_SUCCESS/MS_FAILURE.

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. Returns MS_SUCCESS/MS_FAILURE.

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.

int project(projectionObj in, projectionObj out) Project the line from “in” projection (1st argument) to “out” projection (2nd argument). Returns MS_SUCCESS/MS_FAILURE.

mapObj

Constructor

```
new mapObj(string map_file_name [, string new_map_path])
```

or using the old constructors

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.

Type	Name	Note
double	cellsize	
int	debug	
double	defresolution	pixels per inch, defaults to 72
rectObj	extent;	
string	fontsetfilename	read-only, set by setFontSet()
int	height	see setSize()
colorObj	imagecolor	
int	keysizeX	
int	keysizeY	
int	keyspacingX	
int	keyspacingY	
labelcacheObj	labelcache	no members. Used only to free the label cache (map->labelcache->free())
legendObj	legend	
string	mapproj	
int	maxsize	
hashTableObj	metadata	
string	name	
int	numlayers	read-only
outputformatObj	outputformat	
int	numoutputformats	read-only
projectionObj	projection	
querymapObj	querymap	
referenceMapObj	reference	
double	resolution	pixels per inch, defaults to 72
scalebarObj	scalebar	
double	scaledenom	read-only, set by drawMap()
string	shapepath	
int	status	
string	symbolsetfilename	read-only, set by setSymbolSet()
int	units	map units type
webObj	web	
int	width	see setSize()

Members

Methods

int applyconfigoptions() Applies the config options set in the map file. For example setting the PROJ_LIB using the setconfigoption 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 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 convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

imageObj draw() Render map and return an image object or NULL on error.

int drawLabelCache(imageObj image) Renders the labels for a map. Returns MS_SUCCESS or MS_FAILURE on error.

imageObj drawLegend() Render legend and return an image object.

imageObj drawQuery() Render a query map and return an image object or NULL on error.

imageObj drawReferenceMap() Render reference map and return an image object.

imageObj drawScaleBar() Render scale bar and return an 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). Returns MS_SUCCESS or MS_FAILURE on error.

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). Returns MS_SUCCESS or MS_FAILURE on error.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

void freeQuery(layerindex) Frees the query result on a specified layer. If the layerindex is -1, all queries on layers will be freed.

string generateSLD() Returns an SLD XML string based on all the classes found in all the layers that have *STATUS on*.

array getAllGroupNames() Return an array containing all the group names used in the layers. If there are no groups, it returns an empty array.

array getAllLayerNames() Return an array containing all the layer names. If there are no layers, it returns an empty array.

colorObj getColorbyIndex(int iCloIndex) Returns a colorObj corresponding to the color index in the palette.

string getConfigOption(string key) Returns the config value associated with the key. Returns an empty sting if key not found.

labelcacheMemberObj getLabel(int index) Returns a labelcacheMemberObj from the map given an index value (0=first label). Labelcache has to be enabled.

```
while ($oLabelCacheMember = $oMap->getLabel($i)) {
    /* do something with the labelcachemember */
    ++$i;
}
```

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 NULL if layer doesn't exist.

array getLayersDrawingOrder() Return an array containing layer's index in the order which they are drawn. If there are no layers, it returns an empty array.

array getLayersIndexByGroup(string groupname) Return an array containing all the layer's indexes given a group name. If there are no layers, it returns an empty array.

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 getNumSymbols() Return the number of symbols in map.

string getProjection() Returns a string representation of the projection. Returns NULL on error or if no projection is set.

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.

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.

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 same name as in the context. The default value is MS_FALSE. Returns MS_SUCCESS/MS_FAILURE.

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 loadQuery(filename) Loads a query from a file. Returns MS_SUCCESS or MS_FAILURE. To be used with save-query.

int moveLayerDown(int layerindex) Move layer down in the hierarchy of drawing. Returns MS_SUCCESS or MS_FAILURE on error.

int moveLayerUp(int layerindex) Move layer up in the hierarchy of drawing. Returns MS_SUCCESS or MS_FAILURE on error.

int offsetExtent(double x, double y) Offset the map extent based on the given distances in map coordinates. Returns MS_SUCCESS or MS_FAILURE.

int owsDispatch(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()*)

imageObj prepareImage() Return a blank image object.

void prepareQuery() Calculate the scale of the map and set map->scaledenom.

string processLegendTemplate(array params) Process legend template files and return the result in a buffer.

See also:

processtemplate

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 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);
```

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 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).

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.

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.

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.

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 saveQuery(string filename[, int results]) Save the current query in a file. Results determines the save format - MS_TRUE (or 1/true) saves the query results (tile index and shape index), MS_FALSE (or 0/false) the query parameters (and the query will be re-run in loadquery). Returns MS_SUCCESS or MS_FAILURE. Either save format can be used with loadquery. See RFC 65 and ticket #3647 for details of different save formats.

int scaleExtent(double zoomfactor, double minscaledenom, double maxscaledenom) Scale the map extent using the zoomfactor and ensure the extent within the minscaledenom and maxscaledenom domain. If minscale-

denom and/or maxscaledenom is 0 then the parameter is not taken into account. Returns MS_SUCCESS or MS_FAILURE.

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 appendOutputFormat(outputFormatObj outputFormat) Appends outputformat object in the map object. Returns the new numoutputformats value.

int removeOutputFormat(string name) Remove outputformat from the map. Returns MS_SUCCESS/MS_FAILURE.

outputFormatObj getOutputFormat(int index) Returns the outputformat at index position.

int set(string property_name, new_value) Set map object property to new value.

int setCenter(pointObj center) Set the map center to the given map point. Returns MS_SUCCESS or MS_FAILURE.

int setConfigOption(string key, string value) Sets a config parameter using the key and the value passed

void setExtent(double minx, double miny, double maxx, double maxy) Set the map extents using the georef extents passed in argument. Returns MS_SUCCESS or MS_FAILURE on error.

int setFontSet(string fileName) Load and set a new *FONTSET*.

boolean setLayersDrawingOrder(array layeryindex) Set the layer's order array. The argument passed must be a valid array with all the layer's index. Returns MS_SUCCESS or MS_FAILURE on error.

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 setProjection(string proj_params, boolean bSetUnitsAndExtents) Set map projection and coordinate system. Returns MS_SUCCESS or MS_FAILURE on error.

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 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.

int setSymbolSet(string fileName) Load and set a symbol file dynamically.

int setWKTProjection(string proj_params, boolean bSetUnitsAndExtents) Same as setProjection(), but takes an OGC WKT projection definition string as input. Returns MS_SUCCESS or MS_FAILURE on error.

Note: setWKTProjection requires GDAL support

int zoomPoint(int nZoomFactor, pointObj oPixelPos, int nImageWidth, int nImageHeight, rectObj oGeorefExt) Zoom to a given XY position. Returns MS_SUCCESS or MS_FAILURE on error.

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.

int zoomRectangle(rectObj oPixelExt, int nImageWidth, int nImageHeight, rectObj oGeorefExt) Set the map extents to a given extents. Returns MS_SUCCESS or MS_FAILURE on error.

Parameters are :

- oPixelExt (rect object) : Pixel Extents
- Width : width in pixel of the current image.
- Height : Height in pixel of the current image.
- Georef extent (rectObj) : current georef extents.

int 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. Returns MS_SUCCESS or MS_FAILURE on error.

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.

outputformatObj

Constructor Instance of outputformatObj is always embedded inside the *mapObj*. It is uses a read only.

No constructor available (coming soon, see ticket 979)

Members

Type	Name	Note
string	driver	
string	extension	
int	imagemode	MS_IMAGEMODE_* value.
string	mimetype	
string	name	
int	renderer	
int	transparent	

Methods

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.

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");
```

int validate() Checks some internal consistency issues, Returns MS_SUCCESS or MS_FAILURE. Some problems are fixed up internally. May produce debug output if issues encountered.

OwsrequestObj

Constructor

```
new OWSRequestObj()
```

or using the old 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 addParameter(string name, string value) Add a request parameter, even if the parameter key was previously set. This is useful when multiple parameters with the same key are required. For example :

```
$request->addparameter('SIZE', 'x(100)');
$request->addparameter('SIZE', 'y(100)');
```

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*.

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');
```

pointObj

Constructor

```
new pointObj()
```

or using the old constructor

```
PointObj ms_newPointObj()
```

Members

Type	Name	Note
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

double distanceToLine(pointObject p1, pointObject p2) Calculates distance between a point and a line defined by the two points passed in argument.

double distanceToPoint(pointObj poPoint) Calculates distance between two points.

double distanceToShape(shapeObj shape) Calculates the minimum distance between a point and a shape.

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. (Optional) Returns MS_SUCCESS/MS_FAILURE.

int project(projectionObj in, projectionObj out) Project the point from “in” projection (1st argument) to “out” projection (2nd argument). Returns MS_SUCCESS/MS_FAILURE.

int setXY(double x, double y [, double m]) Set X,Y coordinate values.

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.

Note: the 4th parameter m is used for measured shape files only. It is not mandatory.

projectionObj**Constructor**

```
new projectionObj(string projectionString)
```

or using the old constructor

```
ProjectionObj 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.

querymapObj

Constructor Instances of querymapObj are always are always embedded inside the *mapObj*.

Type	Name	Note
colorObj	color	
int	height	
int	width	
int	style	MS_NORMAL, MS_HILITE, MS_SELECTED

Members

Methods

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

int set(string property_name, new_value) Set object property to a new value.

int updateFromString(string snippet) Update a queryMap object from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

rectObj

Constructor rectObj are sometimes embedded inside other objects. New ones can also be created with:

```
new rectObj()
```

or using the old constructor

```
RectObj ms_newRectObj()
```

Note: the members (minx, miny, maxx ,maxy) are initialized to -1;

Type	Name
double	minx
double	miny
double	maxx
double	maxy

Members:

Methods

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. (Optional) 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.

int set(string property_name, new_value) Set object property to a new value.

void setextent(double minx, double miny, double maxx, double maxy) Set the rectangle extents.

referenceMapObj

Constructor Instances of referenceMapObj are always embedded inside the *mapObj*.

Type	Name
ColorObj	color
int	height
rectObj	extent
string	image
int	marker
string	markername
int	markersize
int	maxboxsize
int	minboxsize
ColorObj	outlinecolor
int	status
int	width

Members

Methods

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

int set(string property_name, new_value) Set object property to a new value.

int updateFromString(string snippet) Update a referenceMap object from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

resultObj

Constructor

```
new resultObj(int shapeindex)
```

or using the *layerObj*'s getResult() method.

	Type	Name	Note
Members	int	classindex	read-only
	int	resultindex	read-only
	int	shapeindex	read-only
	int	tileindex	read-only

Method None

scalebarObj

Constructor Instances of scalebarObj are always embedded inside the *mapObj*.

	Type	Name	Note
Members	int	align	
	colorObj	backgroundcolor	
	colorObj	color	
	int	height	
	colorObj	imagecolor	
	int	intervals	
	labelObj	label	
	colorObj	outlinecolor	
	int	position	for embeded scalebars, MS_UL, MS_UC, ...
	int	postlabelcache	
	int	status	MS_ON, MS_OFF, MS_EMBED
	int	style	
	int	units	
	int	width	

Methods

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

int set(string property_name, new_value) Set object property to a new value.

int setImageColor(int red, int green, int blue) Sets the imagecolor property (background) of the object. Returns MS_SUCCESS or MS_FAILURE on error.

int updateFromString(string snippet) Update a scalebar from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

shapefileObj

Constructor

```
new shapefileObj(string filename, int type)
```

or using the old 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).

	Type	Name	Note
Members	rectObj	bounds	read-only
	int	numshapes	read-only
	string	source	read-only
	int	type	read-only

Methods

int addPoint(pointObj point) Appends a point to an open shapefile.

int addShape(shapeObj shape) Appends a shape to an open shapefile.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

Note: The shape file is closed (and changes committed) when the object is destroyed. You can explicitly close and save the changes by calling \$shapefile->free(); unset(\$shapefile), which will also free the php object.

rectObj getExtent(int i) Retrieve a shape's bounding box by index.

shapeObj getPoint(int i) Retrieve point by index.

shapeObj getShape(int i) Retrieve shape by index.

shapeObj getTransformed(mapObj map, int i) Retrieve shape by index.

shapeObj

Constructor

```
new shapeObj(int type)
```

or using the old 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.

	Type	Name	Note
Members	rectObj	bounds	read-only
	int	classindex	
	int	index	
	int	numlines	read-only
	int	numvalues	read-only
	int	tileindex	read-only
	string	text	
	int	type	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 add(lineObj line)** Add a line (i.e. a part) to the shape.
- shapeObj boundary()** Returns the boundary of the shape. Only available if php/mapsript is built with GEOS library.
- 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.
- 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 convexhull()** Returns a shape object representing the convex hull of shape. Only available if php/mapsript is built with GEOS library.
- boolean contains(pointObj point)** Returns MS_TRUE if the point is inside the shape, MS_FALSE otherwise.
- 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.
- 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.
- 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.
- int draw(mapObj map, layerObj layer, imageObj img)** Draws the individual shape using layer. Returns MS_SUCCESS/MS_FAILURE.
- 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.
- void free()** Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.
- double getArea()** Returns the area of the shape (if applicable). 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.
- pointObj getLabelPoint()** Returns a point object with coordinates suitable for labelling the shape.
- double getLength()** Returns the length (or perimeter) of the shape. Only available if php/mapsript is built with GEOS library.

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.

pointObj getPointUsingMeasure(double m) Apply only on Measured shape files. Given a measure m, return the corresponding XY location on the shapeobject.

string getValue(layerObj layer, string filename) Returns the value for a given field name.

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.

boolean intersects(shapeObj shape) Returns MS_TRUE if the two shapes intersect, MS_FALSE otherwise.

LineObj line(int i) Returns a reference to line number i.

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 project(projectionObj in, projectionObj out) Project the shape from “in” projection (1st argument) to “out” projection (2nd argument). Returns MS_SUCCESS/MS_FAILURE.

int set(string property_name, new_value) Set object property to a new value.

int setBounds() Updates the bounds property of the shape. Must be called to calculate new bounding box after new parts have been added.

shapeObj simplify(double tolerance) Given a tolerance, returns a simplified shape object or NULL on error. Only available if php/mapsript is built with GEOS library (>=3.0).

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.

shapeObj topologySimplifyPreservingSimplify(double tolerance) Given a tolerance, returns a simplified shape object or NULL on error. Only available if php/mapsript is built with GEOS library (>=3.0).

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.

string toWkt() Returns WKT representation of the shape’s geometry.

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

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.

styleObj

Constructor Instances of styleObj are always embedded inside a *classObj* or *labelObj*.

```
new styleObj(classObj class [, styleObj style])
// or
new styleObj(labelObj label [, styleObj style])
```

or using the old constructor (do not support a labelObj at first argument)

```
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.

Type	Name	Note
double	angle	
int	antialias	
colorObj	backgroundcolor	
colorObj	color	
double	maxsize	
double	maxvalue	
double	maxwidth	
double	minsize	
double	minvalue	
double	minwidth	
int	offsetx	
int	offsety	
int	opacity	only supported for the AGG driver
colorObj	outlinecolor	
string	rangeitem	
double	size	
int	symbol	
string	symbolname	
double	width	

Members

Methods

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

string getBinding(const stylebinding) Get the attribute binding for a specfiled style property. Returns NULL if there is no binding for this property.

```

$oSStyle->setbinding(MS_STYLE_BINDING_COLOR, "FIELD_NAME_COLOR");
echo $oSStyle->getbinding(MS_STYLE_BINDING_COLOR); // FIELD_NAME_COLOR
    
```

string getGeomTransform()

int removeBinding(const stylebinding) Remove the attribute binding for a specfiled style property. Added in MapServer 5.0.

```

$oSStyle->removebinding(MS_STYLE_BINDING_COLOR);
    
```

int set(string property_name, new_value) Set object property to a new value.

int setBinding(const stylebinding, string value) Set the attribute binding for a specfiled style property. Added in MapServer 5.0.

```

$oSStyle->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")

int setGeomTransform(string value)

int updateFromString(string snippet) Update a style from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

symbolObj

Constructor

```
new symbolObj(mapObj map, string symbolname)
```

or using the old constructor

```
int ms_newSymbolObj(mapObj map, string symbolname)
```

Creates a new symbol with default values in the symbolist.

Note: Using the new constructor, the symbol is automatically returned. The old constructor returns the id of the new symbol.

If a symbol with the same name exists, it (or its id) will be returned. To get a symbol object using the old constructor, you need to use a method on the map object:

```
$nId = ms_newSymbolObj($map, "symbol-test");
$oSymbol = $map->getSymbolObjectById($nId);
```

Members

Type	Name	Note
int	antialias	
string	character	
int	filled	
string	font	
string	imagepath	read-only
int	inmapfile	If set to TRUE, the symbol will be saved inside the mapfile.
int	patternlength	read-only
int	position	
string	name	
int	numpoints	read-only
double	sizeX	
double	sizeY	
int	transparent	
int	transparentcolor	

Methods

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

array getPatternArray() Returns an array containing the pattern. If there is no pattern, it returns an empty array.

array getPointsArray() Returns an array containing the points of the symbol. Refer to setpoints to see how the array should be interpreted. If there are no points, it returns an empty array.

int set(string property_name, new_value) Set object property to a new value.

int setImagePath(char filename) Loads a pixmap symbol specified by the filename. The file should be of either Gif or Png format.

int setPattern(array int) Set the pattern of the symbol (used for dash patterns). Returns MS_SUCCESS/MS_FAILURE.

int setPoints(array double) Set the points of the symbol. Note that the values passed is an array containing the x and y values of the points. Returns MS_SUCCESS/MS_FAILURE. 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
....
```

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");
```

2. 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");
```

webObj

Constructor Instances of webObj are always are always embedded inside the *mapObj*.

Members

Type	Name	Note
string	browseformat	
string	empty	read-only
string	error	read-only
rectObj	extent	read-only
string	footer	
string	header	
string	imagepath	
string	imageurl	
string	legendformat	
string	log	
double	maxscaledenom	
string	maxtemplate	
hashTableObj	metadata	
double	minscaledenom	
string	mintemplate	
string	queryformat	
string	template	
string	temppath	

Methods

string convertToString() Saves the object to a string. Provides the inverse option for updateFromString.

void free() Free the object properties and break the internal references. Note that you have to unset the php variable to free totally the resources.

int set(string property_name, new_value) Set object property to a new value.

int updateFromString(string snippet) Update a web object from a string snippet. Returns MS_SUCCESS/MS_FAILURE.

PHP MapScript Migration Guide

Author Alan Boudreault

Contact aboudreault at mapgears.com

Revision \$Revision: 10033 \$

Date \$Date: 2010-03-30 15:58:30 -0400 (Tue, 30 Mar 2010) \$

Table of Contents

- *PHP MapScript Migration Guide*
 - *Introduction*
 - *Migrating 5.6 to 6.0*
 - * *PHP Version Required*
 - * *Error Reporting*
 - * *Manipulating Objects*
 - * *Class Properties*
 - * *Class Methods*
 - * *layerObj*
 - * *mapObj*
 - * *referenceMapObj*
 - * *shapeFileObj*
 - * *labelCacheObj*
 - * *Methods that now return MS_SUCCESS/MS_FAILURE*
 - * *Methods that now return NULL on failure*
 - * *Methods that now return an empty array*

Introduction

This document describes the changes that must be made to PHP MapScript applications when migrating from one MapServer version to another (i.e. backwards incompatibilities), as well as information on some of the new features.

Migrating 5.6 to 6.0

PHP Version Required PHP 5.2.0 or more recent is required. The support for earlier versions has been dropped.

Error Reporting PHP MapScript now uses exceptions for error reports. All errors are catchable. There are no more fatal errors reported via the standard uncatchable PHP system (Only Warnings).

Manipulating Objects

- Object properties can be set like all other PHP objects.

```
$map->scaledenom = 25000;
```

Note: The set/setProperty methods are still available.

- Objects can be created with the PHP “new” operator.

```
$myShape = ms_newShapeObj(MS_SHAPE_LINE); // or  
$myShape = new shapeObj(MS_SHAPE_LINE);
```

Note: All object constructors throw an exception on failure.

Note: *ms_newSymbolObj()* and *new symbolObj()* are different

- *ms_newSymbolObj()* returns the id of the new/existing symbol.
 - *new symbolObj()* returns the symbolObj. You don’t need to get it with *getSymbolObjectById()*.
-

- Cloneable objects should be cloned with the PHP clone keyword. There is no more clone methods.

Class Properties Class properties that have been removed:

- classObj: maxscale, minscale
- layerObj: labelsizetext, labelangle, labelmaxscale, labelminscale, maxscale, minscale, symbolscale, transparency
- legendObj: interlace, transparent
- mapObj: imagetype, imagequality, interlace, scale, transparent
- scalebarObj: interlace, transparent
- symbolObj: gap, stylelength
- webObj: minscale, maxscale

Class Methods Class methods that have been removed:

- imageObj: free
- layerObj: getFilter, getShape
- lineObj: free
- pointObj: free
- projectionObj: free
- rectObj: free
- shapeObj: union_geos
- symbolObj: getstylearray
- classObj: clone
- styleObj: clone
- mapObj: clone
- outputFormatObj: getformatoption, setformatoption

layerObj layerObj->clearProcessing() method now returns void.

mapObj mapObj->queryByIndex(): default behavior for the addToQuery parameter was not ok, now it is.

referenceMapObj referenceMapObj has new properties: marker, markername, markersize, maxboxsize, minboxsize.

shapeFileObj shapeFileObj is automatically closed/written on destroy. (At the end of the script or with an explicit free(), unset())

labelCacheObj To free the cache, you'll have to call the method freeCache() rather than free().

Methods that now return MS_SUCCESS/MS_FAILURE

- layerObj: setProcessing, addFeature, draw
- mapObj: moveLayerUp, moveLayerDown, zoomRectangle, zoomScale, setProjection, setWKTProjection, setLayersDrawingOrder
- outputFormatObj: validate
- scalebarObj: setImageColor
- symbolObj: setPoints, setPattern

Methods that now return NULL on failure

- classObj: clone
- mapObj: clone, draw, drawQuery, getLayerByName, getProjection
- layerObj: nextShape, getExtent
- styleObj: clone

Methods that now return an empty array

- layerObj: getItems, getProcessing, getGridIntersectionCoordinates
- mapObj: getLayersIndexByGroup, getAllGroupNames, getLayersDrawingOrder, getAllLayerNames
- symbolObj: getPatternArray

By Example

Author Vinko Vrsalovic

Contact el at vinko.cl

Revision \$Revision\$

Date \$Date\$

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 API* 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](#)]?

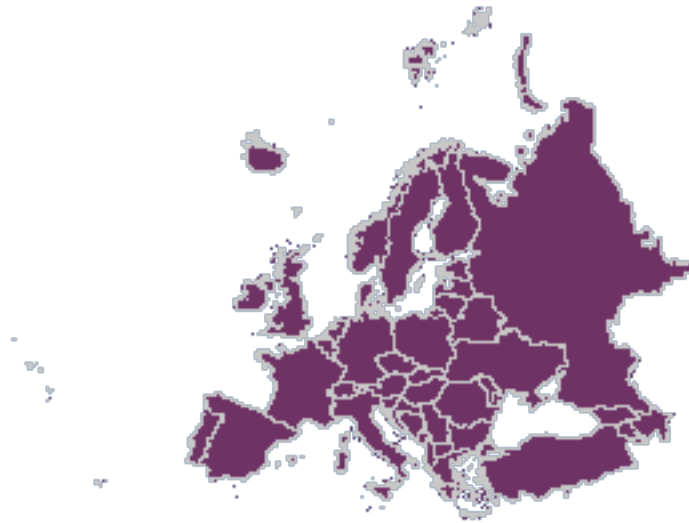
Outline of this Document

- 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:

```

1  NAME "Europe in purple"
2  SIZE 400 400
3  STATUS ON
4  SYMBOLSET "/var/www/html/maps/symbols/symbols.sym"
5  EXTENT -5696501 1923039 5696501 11022882
6  UNITS METERS
7  SHAPEPATH "/var/www/html/maps/data"
8
9  WEB
10     IMAGEPATH "/var/www/html/maps/tmp/"
11     IMAGEURL "/tmp/"
12 END
13
14 LAYER
15     NAME "Europe"
16     TYPE POLYGON
17     STATUS ON
18     DATA "europe"
19     CLASS
20         STYLE
21             COLOR 110 50 100
22             OUTLINECOLOR 200 200 200
23             SYMBOL 0
24         END
25     END
26 END
27
28 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
3  dl('php_mapscript.so');
4
5  $map_path="/var/www/html/ms/map_files/";
6
7  $map = ms_newMapObj($map_path."europe.map");
8  $image=$map->draw();
9  $image_url=$image->saveWebImage();
10
11  ?>
12
13  <HTML>
14  <HEAD>
15      <TITLE>Example 1: Displaying a map</TITLE>
16  </HEAD>
17  <BODY>
18      <IMG SRC=<?php echo $image_url; ?> >
19  </BODY>
20  </HTML>
```

The code I will present through the rest of this document will follow the following rule:

- Every non empty line is numbered

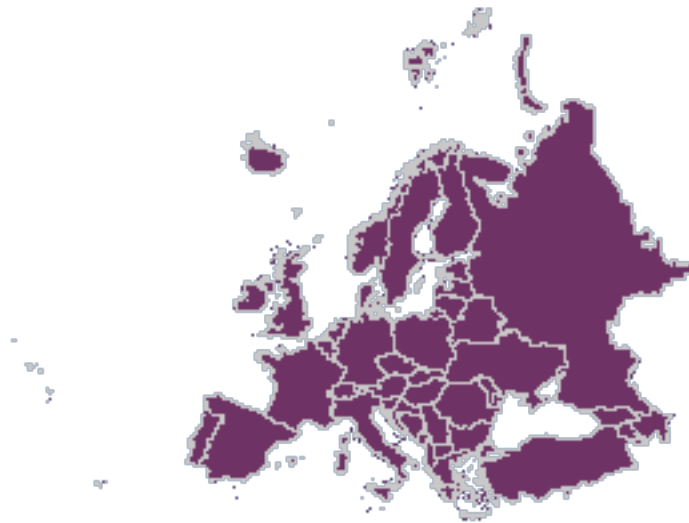
This code will render an image corresponding to the shapefile europe and display it on a HTML page.

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
3  dl('php_mapscript.so');
4
5  // Default values and configuration
6
7  $val_zsize=3;
8  $check_pan="CHECKED";
9  $map_path="/var/www/html/ms/map_files/";
10 $map_file="europe.map";
11
12 $map = ms_newMapObj($map_path.$map_file);
13
14
15 if ( isset($_POST["mapa_x"]) && isset($_POST["mapa_y"])
16     && !isset($_POST["full"]) ) {
17
18     $extent_to_set = explode(" ", $_POST["extent"]);
19
20     $map->setextent($extent_to_set[0], $extent_to_set[1],
21                 $extent_to_set[2], $extent_to_set[3]);
22
23     $my_point = ms_newpointObj();
24     $my_point->setXY($_POST["mapa_x"], $_POST["mapa_y"]);
25
26     $my_extent = ms_newrectObj();
27
28     $my_extent->setextent($extent_to_set[0], $extent_to_set[1],
29                        $extent_to_set[2], $extent_to_set[3]);
30
31     $zoom_factor = $_POST["zoom"]*$_POST["zsize"];
32     if ($zoom_factor == 0) {
33         $zoom_factor = 1;
34         $check_pan = "CHECKED";
35         $check_zout = "";
36         $check_zin = "";
37     } else if ($zoom_factor < 0) {
38         $check_pan = "";
39         $check_zout = "CHECKED";
40         $check_zin = "";
41     } else {
42         $check_pan = "";
43         $check_zout = "";
44         $check_zin = "CHECKED";
45     }
46
47     $val_zsize = abs($zoom_factor);
48
49     $map->zoompoint($zoom_factor, $my_point, $map->width, $map->height,
50                  $my_extent);
```



```

51 }
52
53
54
55 $image=$map->draw();
56 $image_url=$image->saveWebImage();
57
58 $extent_to_html = $map->extent->minx." ".$map->extent->miny." "
59                 .$map->extent->maxx." ".$map->extent->maxy;
60
61 ?>
62 <HTML>
63 <HEAD>
64 <TITLE>Map 2</TITLE>
65 </HEAD>
66 <BODY>
67 <CENTER>
68 <FORM METHOD=POST ACTION=<?php echo $HTTP_SERVER_VARS['PHP_SELF']?>>
69 <TABLE>
70 <TR>
71     <TD>
72         <INPUT TYPE=IMAGE NAME="mapa" SRC="<?php echo $image_url?>">
73     </TD>
74 </TR>
75 <TR>
76     <TD>
77         Pan
78     </TD>
79     <TD>
80         <INPUT TYPE=RADIO NAME="zoom" VALUE=0 <?php echo $check_pan?>>
81     </TD>
82 </TR>
83 <TR>
84     <TD>
85         Zoom In
86     </TD>
87     <TD>
88         <INPUT TYPE=RADIO NAME="zoom" VALUE=1 <?php echo $check_zin?>>
89     </TD>
90 </TR>
91 <TR>
92     <TD>
93         Zoom Out
94     </TD>
95     <TD>
96         <INPUT TYPE=RADIO NAME="zoom" VALUE=-1 <?php echo $check_zout?>>
97     </TD>
98 </TR>
99 <TR>
100    <TD>
101        Zoom Size
102    </TD>
103    <TD>
104        <INPUT TYPE=TEXT NAME="zsize" VALUE="<?php echo $val_zsize?>"
105        SIZE=2>
106    </TD>
107 </TR>
108 <TR>

```

```
109     <TD>
110         Full Extent
111     </TD>
112     <TD>
113         <INPUT TYPE=SUBMIT NAME="full" VALUE="Go"
114         SIZE=2>
115     </TD>
116 </TABLE>
117 <INPUT TYPE=HIDDEN NAME="extent" VALUE="<?php echo $extent_to_html?>">
118 </FORM>
119 </CENTER>
120 </BODY>
121 </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.

5.1.4 Python MapScript Appendix

Author Sean Gillies

Revision \$Revision\$

Date \$Date\$

Contents

- *Python MapScript Appendix*
 - *Introduction*
 - *Classes*
 - *Exception Handling*

Introduction

The Python MapScript module contains some class extension methods that have not yet been implemented for other languages.

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
```

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.

5.1.5 Python MapScript Image Generation

Author Sean Gillies

Revision \$Revision\$

Date \$Date\$

Last Updated 2008/07/15

Table of Contents

- *Python MapScript Image Generation*
 - *Introduction*
 - *Imagery Overview*
 - *The imageObj Class*
 - *Image Output*
 - *Images and Symbols*

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.

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.

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')
```

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.

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.

5.1.6 Mapfile Manipulation

Author Sean Gillies

Revision \$Revision\$

Date \$Date\$

Contents

- *Mapfile Manipulation*
 - *Introduction*
 - *Mapfile Overview*
 - *The mapObj Class*
 - *Children of mapObj*
 - *Metadata*

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.

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.

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.

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)
```

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.

5.1.7 Querying

Author Sean Gillies

Revision \$Revision\$

Date \$Date\$

Contents

- *Querying*
 - *Introduction*
 - *Querying Overview*
 - *Attribute Queries*
 - *Spatial Queries*

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.

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)

        ... # do something with this feature

    # Close when done
    layer.close()
```

Backwards Compatibility

The API changed substantially with version 6.0 and backward compatibility was broken. Scripts will have to be updated to work with the new API.

Attribute Queries

By Attributes

queryByAttributes()

Spatial Queries

By Rectangle

queryByRect()

By Point

queryByRect()

By Shape

`queryByShape()`

By Selection

`queryByFeatures()`

6.1 MapCache

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

MapCache is a server that implements tile caching to speed up access to WMS layers. The primary objectives are to be fast and easily deployable, while offering the essential features (and more!) expected from a tile caching solution.

6.1.1 Compilation & Installation

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

Author Alan Boudreault

Contact aboudreaut at magears.com

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Mathieu Coudert

Contact mathieu.coudert at gmail.com

Table of Contents

- *Compilation & Installation*
 - *Getting the Source*
 - *Linux Instructions*
 - * *Apache Module Specific Instructions*
 - * *Nginx Specific Instructions*
 - * *CGI/FastCGI Specific Instructions*
 - * *Customizing the build, or if something went wrong*
 - *Windows Instructions*
 - * *Dependencies*
 - * *Configure Your Makefile*
 - * *Compilation*
 - * *Move the Module into Apache Directory*
 - * *Configure Your Installed Apache*
 - * *Test Your MapCache Module*

Getting the Source

The MapCache project is located at <https://github.com/mapserver/mapcache>, and can be checked out with either:

```
# readonly
git clone git://github.com/mapserver/mapcache.git
# ssh authenticated
git clone git@github.com:mapserver/mapcache.git
# tarball
wget https://github.com/mapserver/mapcache/zipball/master
```

Linux Instructions

These instructions target a debian/ubuntu setup, but should apply with few modifications to any linux installation.

MapCache requires a number of library headers in order to compile correctly:

- **apache / apr / apr-util / apx2:** these are included in the *apache2-prefork-dev* or *apache2-threaded-dev* packages, depending on what apache mpm you are running. This package will pull in the necessary apr headers, that you would have to manually install if you are not building an apache module (*libaprutil-dev* and *libapr-dev*)
- **png:** *libpng12-dev*
- **jpeg:** *libjpeg62-dev*
- **curl:** *libcurl4-gnutls-dev*

The following libraries are not required, but recommended:

- **pcre:** *libpcre3-dev*. This will give you more powerful regular expression syntax when creating validation expressions for dimensions
- **pixman:** *libpixman-1-dev*. The pixel manipulation library is used for scaling and alpha-compositing images. MapCache ships with some code to perform these tasks, but pixman is generally faster as it includes code optimized for modern cpus (sse2, mmx, etc...)

The following libraries are not required, but needed to enable additional functionalities:

- **fcgi:** *libfcgi-dev*. Needed to build a fastcgi program if you don't want to run mapcache as an apache module.

- **gdal / geos**: *libgdal-dev libgeos-dev*. Needed to enable advanced seeding options (for only seeding tiles that intersect a given geographical feature)
- **sqlite**: *libsqlite3-dev*. For enabling the sqlite backend storages
- **tiff**: *libtiff4-dev*. For enabling the TIFF backend storages
- **berkeley db** *libdb4.8-dev* : For enabling the Berkeley DB backend storages

Note: MapCache now builds with cmake.

For unix users where all packages are in the default locations, the compilation process should resume to:

```
$ cd mapcache
$ mkdir build
$ cd build
$ cmake ..
$ # follow instructions below if missing a dependency
$ make
$ sudo make install
```

Apache Module Specific Instructions

The make install above installs the apache module, but if you need to specifically need to install only the apache module you can do the following

```
$ sudo make install-module
$ sudo ldconfig
```

The installation script takes care of putting the built module in the apache module directory. The process for activating a module is usually distro specific, but can be resumed by the following snippet that should be present in the apache configuration file (e.g. `/usr/local/httpd/conf/httpd.conf` or `/etc/apache2/sites-available/default`):

```
LoadModule mapcache_module    modules/mod_mapcache.so
```

Next, a mapcache configuration is mapped to the server url with the following snippet

For apache < 2.4:

```
<IfModule mapcache_module>
  <Directory /path/to/directory>
    Order Allow,Deny
    Allow from all
  </Directory>
  MapCacheAlias /mapcache "/path/to/directory/mapcache.xml"
</IfModule>
```

For apache >= 2.4:

```
<IfModule mapcache_module>
  <Directory /path/to/directory>
    Require all granted
  </Directory>
  MapCacheAlias /mapcache "/path/to/directory/mapcache.xml"
</IfModule>
```

Before you restart, copy the example mapcache.xml file to where you want it:

```
$ cp mapcache.xml /path/to/directory/mapcache.xml
```

Finally, restart apache to take the modified configuration into account

```
$ sudo apachectl restart
```

If you have not disabled the demo service, you should now have access to it on <http://myserver/mapcache/demo>

Ngix Specific Instructions

Warning: Working with nginx is still somewhat experimental. The following workflow has only been tested on the development version, i.e. nginx-1.1.x

For nginx support you need to build mapcache's nginx module against the nginx source. Download the nginx source code:

```
$ cd /usr/local/src
$ mkdir nginx
$ cd nginx
$ wget http://nginx.org/download/nginx-1.1.19.tar.gz
$ tar -xzf nginx-1.1.19.tar.gz
$ cd nginx-1.1.19/
```

Run the configure command with the flag `--add-module`. This flag must point to mapcache's nginx child directory. Assuming that mapserver source was cloned or un tarred into to `/usr/local/src`, an example configure command for nginx would look like this:

```
$ ./configure --add-module=/usr/local/src/mapcache/nginx
```

Then build nginx:

```
$ make
$ sudo make install
```

Due to nginx's non-blocking architecture, the mapcache nginx module does not perform any operations that may lead to a worker process being blocked by a long computation (i.e.: requesting a (meta)tile to be rendered if not in the cache, proxying a request to an upstream wms server, or waiting for a tile to be rendered by another worker), it will instead issue a 404 error. This behavior is essential so as not to occupy all nginx worker threads, therefore preventing it from responding to all other incoming requests. While this isn't an issue for completely seeded tilesets, this implies that these kinds of requests need to be proxied to another mapcache instance that does not suffer from these starvation issues (i.e. either a fastcgi mapcache, or an internal proxied apache server). In this scenario, both the nginx mapcache instance and the apache/fastcgi mapcache instance should be running with the same `mapcache.xml` configuration file.

Mapcache supplies a `nginx.conf` in its nginx child directory. The conf contains an example configuration to load mapcache. The most relevant part of the configuration is the location directive that points the `^/mapcache` URI to the `mapcache.xml` path. You will need to change this path to point to your own `mapcache.xml` in the mapcache source

The basic configuration without any proxying (which will return 404 errors on unseeded tile requests) is:

```
location ~ ^/mapcache(?<path_info>/.*|$) {
    set $url_prefix "/mapcache";
    mapcache /usr/local/src/mapcache/mapcache.xml;
}
```

If proxying unseeded tile requests to a mapcache instance running on an apache server, we will proxy all 404 mapcache errors to a `mapcache.apache.tld` server listening on port 8080, configured to respond to mapcache requests on the `/mapcache` location.

```
location ~ ^/mapcache(?<path_info>/.*|$) {
    set $url_prefix "/mapcache";
    mapcache /usr/local/src/mapcache/mapcache.xml;
    error_page 404 = @apache_mapcache;
}

location @apache_mapcache {
    proxy_pass http://mapcache.apache.tld:8080;
}
```

If using fastcgi instances of mapcache, spawned with e.g. `spawn-fcgi` or `supervisord` on port 9001 (make sure to enable fastcgi when building mapcache, and to set the `MAPCACHE_CONFIG_FILE` environment variable before spawning):

```
location ~ ^/mapcache(?<path_info>/.*|$) {
    set $url_prefix "/mapcache";
    mapcache /usr/local/src/mapcache/mapcache.xml;
    error_page 404 = @fastcgi_mapcache;
}

location @fastcgi_mapcache {
    fastcgi_pass localhost:9001;
    fastcgi_param QUERY_STRING $query_string;
    fastcgi_param REQUEST_METHOD $request_method;
    fastcgi_param CONTENT_TYPE $content_type;
    fastcgi_param CONTENT_LENGTH $content_length;
    fastcgi_param PATH_INFO $path_info;
    fastcgi_param SERVER_NAME $server_name;
    fastcgi_param SERVER_PORT $server_port;
    fastcgi_param SCRIPT_NAME "/mapcache";
}
```

Copy the relevant sections of `nginx.conf` from mapcache's `nginx` directory into `nginx`'s conf file (in this case `/usr/local/nginx/conf/nginx.conf`), you should now have access to the demo at `http://myserver/mapcache/demo`

CGI/FastCGI Specific Instructions

A binary `cgi/fastcgi` is located in the `mapcache/` subfolder, and is named "mapcache". Activating fastcgi for the mapcache program on your web server is not part of these instructions, more details may be found on the *FastCGI* page or on more general web pages across the web.

The MapCache fastcgi program looks for its configuration file in the environment variable called `MAPCACHE_CONFIG_FILE`, which must be set by the web server before spawning the mapcache processes.

See also:

Configuration File

For apache with `mod_cgi`:

```
SetEnv "MAPCACHE_CONFIG_FILE" "/path/to/mapcache/mapcache.xml"
```

For apache with `mod_fcgid`:

```
FcgidInitialEnv "MAPCACHE_CONFIG_FILE" "/path/to/mapcache/mapcache.xml"
```

If you have not disabled the demo service, you should now have access to it on <http://myserver/fcgi-bin/mapcache/demo> supposing your fcgi processes are accessed under the fcgi-bin alias.

With a working `mod_fcgid` apache instance, the full `httpd.conf` snippet to activate mapcache could be:

```
<IfModule mod_fcgid.c>
  IPCCommTimeout 120
  MaxProcessCount 10
  FcgidInitialEnv "MAPCACHE_CONFIG_FILE" "/path/to/mapcache/mapcache.xml"
  <Location /map.fcgi>
    Order Allow,Deny
    Allow from all
    SetHandler fcgid-script
  </Location>
  ScriptAlias /map.fcgi "/path/to/mapcache/src/mapcache"
</IfModule>
```

The mapcache service would then be accessible at [http://myserver/map.fcgi\[/demo\]](http://myserver/map.fcgi[/demo])

Customizing the build, or if something went wrong

Depending on what packages are available in the default locations of your system, the “`cmake ..`” step will most probably have failed with messages indicating missing dependencies (by default, MapCache has *some* of those). The error message that CMake prints out should give you a rather good idea of what steps you should take next, depending on whether the failed dependency is a feature you require in your build or not.

`mod_mapcache` requires `apache`, `libcurl`, `libjpeg` and `libpng` development headers. The `cmake` script will try to locate them in default system locations, that can be overridden or specified with `-D` switches. For example, if you get a message such as ‘Could NOT find APR’, you can use a command such as (assuming that `apr` is at `/usr/local/apr`):

```
$ cmake -DCMAKE_PREFIX_PATH="/usr/local/apr;" ..
```

Either if you don’t want `fcgi` you can disable the dependency by rerunning `cmake` with `-DWITH_DEPENDENCY=0`, e.g.

```
$ cmake .. -DWITH_FCGI=0
```

Options supported by the MapCache `cmake` builder Here is a list of supported options that can be enabled/disabled at build.

```
option(WITH_PIXMAN "Use pixman for SSE optimized image manipulations" ON)
option(WITH_SQLITE "Use sqlite as a cache backend" ON)
option(WITH_BERKELEY_DB "Use Berkeley DB as a cache backend" OFF)
option(WITH_MEMCACHE "Use memcache as a cache backend (requires recent apr-util)" OFF)
option(WITH_TIFF "Use TIFFs as a cache backend" OFF)
option(WITH_TIFF_WRITE_SUPPORT "Enable (experimental) support for writable TIFF cache backends" OFF)
option(WITH_GEOTIFF "Allow GeoTIFF metadata creation for TIFF cache backends" OFF)
option(WITH_PCRE "Use PCRE for regex tests" OFF)
option(WITH_MAPSERVER "Enable (experimental) support for the mapserver library" OFF)
option(WITH_GEOS "Choose if GEOS geometry operations support should be built in" ON)
option(WITH_OGR "Choose if OGR/GDAL input vector support should be built in" ON)
option(WITH_CGI "Choose if CGI executable should be built" ON)
option(WITH_FCGI "Choose if CGI executable should support FastCGI" ON)
option(WITH_VERSION_STRING "Show MapCache in server version string" ON)
option(WITH_APACHE "Build Apache Module" ON)
```

- **Pixman** (*recommended*, from 0.5 onwards)

```
-DWITH_PIXMAN=[0|1]
```

Pixman is a pixel manipulation library used to assemble image tiles when responding to non-tiled wms requests. Pixman support is recommended as it is highly optimized and will take advantage of recent processor extensions (mms, sse, ...) to speed up blending and resampling operations. In case the pixman library is not found, Mapcache will fall back to internal pixel operations that are slower.

- **Sqlite** (*optional*, from 0.5 onwards)

```
-DWITH_SQLITE=[0|1]
```

Sqlite is used to enable the sqlite and mbtiles cache backend. version 3.5.0 or newer is required.

- **GDAL** (*optional*, from 0.4 onwards, also requires geos)

```
-DWITH_OGR=[0|1]
```

Gdal (actually ogr) is used by the seeding utility to allow the seeding of tiles only intersecting a given polygon, e.g. to preseed all the tiles of a given country.

- **GEOS** (*optional*, from 0.5 onwards)

```
-DWITH_GEOS=[0|1]
```

Along with gdal/ogr, geos is needed by the seeder to test for the intersection of tiles with geographical features. A sufficiently recent version of geos (with support for prepared geometries) is required (but not enforced by the configure script, so you'll end up with compilation errors if a too old geos version is used)

- **PCRE** (*optional*)

```
-DWITH_PCRE=[0|1]
```

Pcre (perl compatible regular expressions) can be used instead of posix regular expressions for validating WMS dimensions. they are more powerfull than posix REs (and might be slower). You don't need this if you aren't planning on using WMS dimension support with regex validation, or if your validation needs are covered by posix REs.

See also:

[Tilset Dimensions](#)

- **FastCGI Support** (*optional*)

```
-DWITH_FCGI=[0|1]
```

MapCache can run as a fastcgi executable. Note that the overhead of fastcgi is non-negligeable with respect to the throughput you may obtain with a native apache module. The fastcgi build is less tested, and may lag behind compared to the apache module version on some minor details, YMMV.

- **TIFF read/write Cache Support** (*optional*)

Use TIFFs as a cache backend (READONLY) :

```
-DWITH_TIFF=[0|1]
```

TIFF write support (for creating new TIFF files and adding tiles to existing TIFF files) is still experimental and disabled by default. There is a risk in ending up with corrupt TIFF files if they are placed on a filesystem that does not honour file locking, as in that case multiple processes might end up writing to the same file. File locking across concurrent threads is also problematic, although

MapCache tries to detect this situation and apply sufficient locking workarounds. To stay on the safe side, write support should for now only be enabled on local filesystems, with a prefork mpm or fastcgi MapCache install.

```
-DWITH_TIFF_WRITE_SUPPORT=[0|1]
```

When writing TIFF files, MapCache can also optionally add georeferencing information if compiled with libtiff support. GeoTiff writing does not produce the full tags needed for defining which projection the grid is in, but will only produce those defining the pixel scale and the tiepoints (i.e. the equivalent information found in the accompanying .tfw files)

```
-DWITH_GEOTIFF=[0|1]
```

See also:

(Geo)TIFF Caches

- **Memcached Cache Support** (*optional*)

```
-DWITH_MEMCACHE=[0|1]
```

The memcached cache backend is disabled by default. You can optionally enable it as it does not depend on other external libraries (support is obtained through apr-util).

See also:

Memcache Caches

- **Apache Module Options**

You can disable the apache module building if you only plan on using the fastcgi executable or the seeder.

```
-DWITH_APACHE=[0|1]
```

MapCache adds itself to the version string reported by the apache server. This can be disabled with

```
-DWITH_VERSION_STRING=[0|1]
```

- **Native MapServer Mode** (*experimental options*)

MapCache is by default not linked to MapServer in any way, and communicates through the WMS protocol only. For performance reasons, there is a possibility to directly use the mapserver C library and avoid an http request and an image compression/decompression. This integration is still experimental and should be used cautiously

```
-DWITH_MAPSERVER=[0|1]
```

This will use the libmapserver.so from mapserver's install directory. MapServer itself should be compiled with thread-safety enabled, unless you plan use the prefork mpm or fastcgi, **and** you do not plan to use the seeder. For thread safety on the mapserver side, you might want to have a look at tickets #4041 and #4044.

- **Debug Mode** (*work in progress*)

Note: Since the cmake migration, this has to be done.

It enables some extra tests inside the code, and prints out many more debugging messages to the server logs. you should probably not enable this unless you want to track down a problem happening inside MapCache.

Windows Instructions

Warning: The following instructions are outdated, Windows builds are now handled identically to the Unix ones with `cmake`.

These instructions target a Windows 7 setup with an Apache httpd compiled from source. The Apache MapCache module has been successfully built with with Microsoft Visual Studio C++ versions 2003, 2008 and 2010.

Dependencies

Required:

- **Apache / APR / APR-UTIL:** included with apache httpd installation

Those can be installed manually, or using the appropriate Windows SDK from: <http://www.gisinternals.com/sdk/>

- **PNG**
- **JPEG**
- **CURL**

Recommended:

- **PCRE:** <ftp://ftp.gnu.org/pub/gnu/regex/regex-0.12.tar.gz>

Optional:

- **FCGI:** Needed to build a fastcgi program if you don't want to run mapcache as an apache module.
- **GDAL / GEOS:** Needed to enable advanced seeding options (for only seeding tiles that intersect a given geographical feature)
- **SQLITE:** For enabling the sqlite backend storages
- **TIFF:** For enabling the TIFF backend storages

Configure Your Makefile

Open `nmake.opt` and modify the paths to point to the various libraries.

Compilation

```
$ nmake /f Makefile.vc
```

If successful, the resulting libraries and executables will be generated in their associated directories:

apache/ Apache module (`mod_mapcache.dll`)

cgi/ FastCGI MapCache executable (`mapcache.exe`)

util/ MapCache utilities (`mapcache_seed.exe`)

Move the Module into Apache Directory

Copy the `mod_mapcache.dll` file into one of your Apache subdirectories.

Note: Although other modules are installed into `/Apache/modules/`, you should place `mod_mapcache.dll` wherever its required dll files (`libcurl.dll`, `zlib.dll`, etc.) live, to avoid any loading issues later on.

Configure Your Installed Apache

- Modify your `httpd.conf` file to load the module:

```
LoadModule mapcache_module "D:/ms4w/Apache/cgi-bin/mod_mapcache.dll"
```

- Next, configure your mapcache directory with the following snippet

```
<IfModule mapcache_module>
  <Directory "D:/ms4w/apps/mapcache/">
    Order Allow,Deny
    Allow from all
  </Directory>
  MapCacheAlias /mapcache "D:/ms4w/apps/mapcache/mapcache.xml"
</IfModule>
```

- Configure your `mapcache.xml` file (see the *Configuration* section for help)

Warning: If you receive an error such as “cache disk: host system does not support file symbolic linking” you should comment the line “`<symlink_blank/>`” in your `mapcache.xml` file, such as:

```
<cache name="disk" type="disk">
  <base>D:/ms4w/tmp/ms_tmp/cache</base>
  <!--<symlink_blank/>-->
</cache>
```

- Finally, restart your Apache. You should see a message in Apache’s error.log with a message similar to:

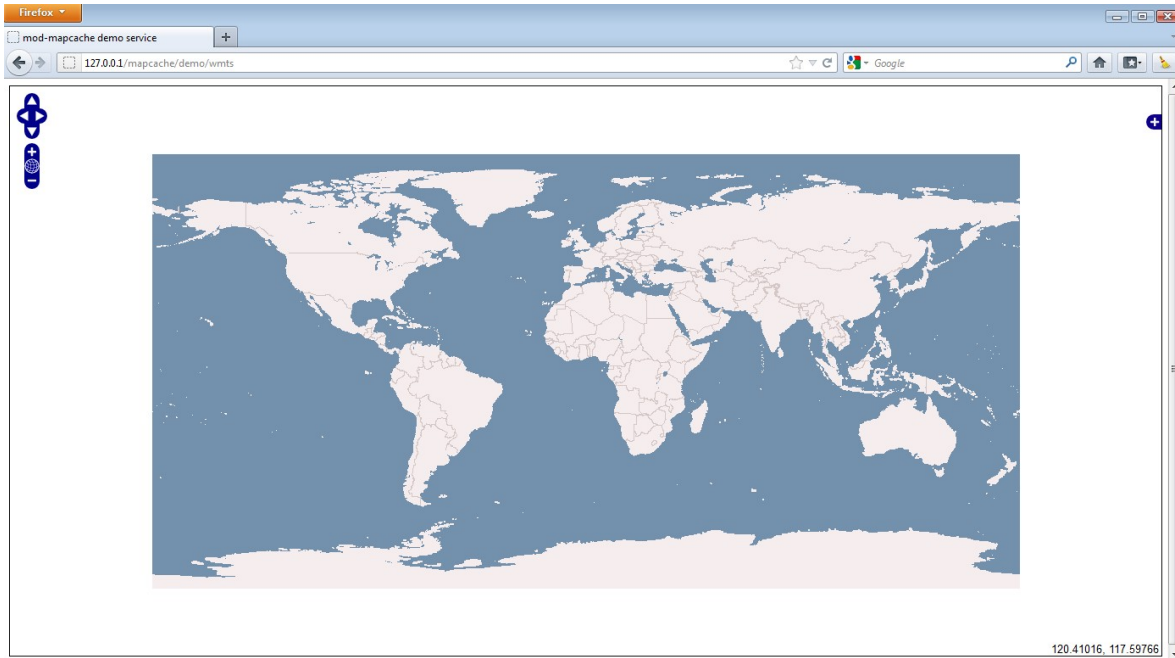
```
[notice] Apache/2.2.21 (Win32) mod-mapcache/0.5-dev configured -- resuming normal operations
```

Test Your MapCache Module

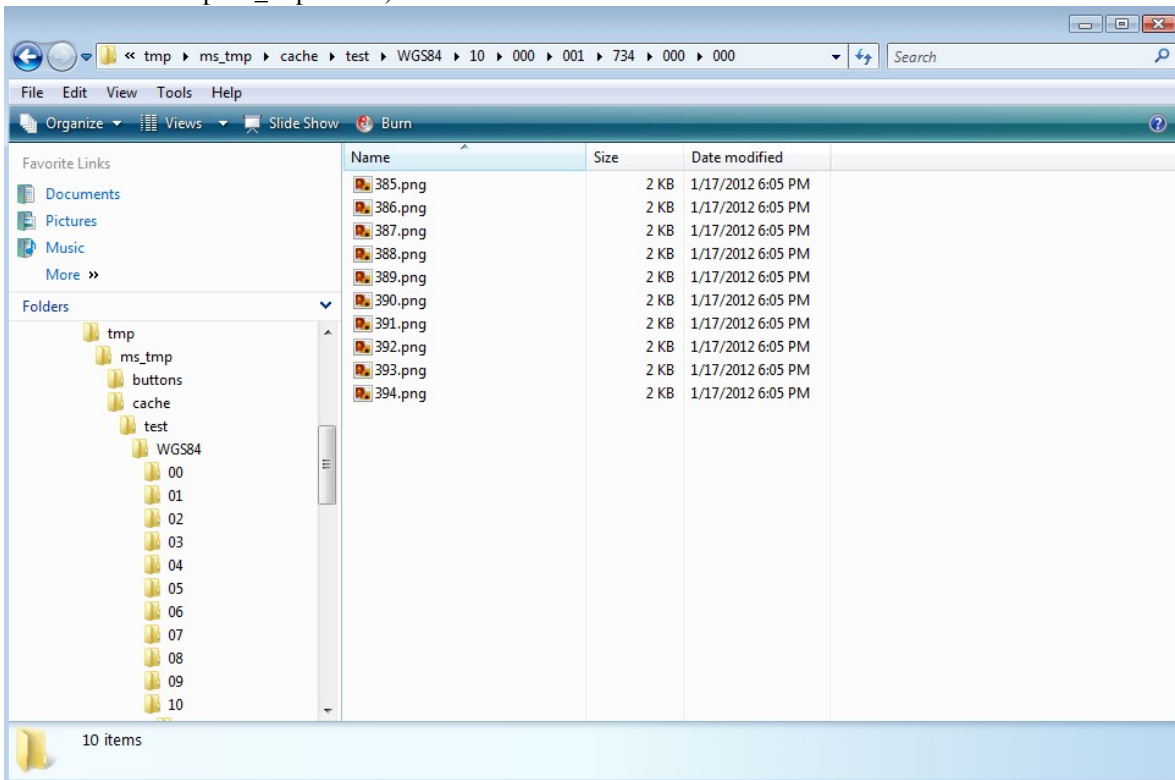
- In your web browser, goto the local MapCache demo page: <http://127.0.0.1/mapcache/demo/> You should see a clickable list of demo links:

```
tms
wmts
gmaps
kml
ve
wms
```

- Click on one of the demos (such as <http://127.0.0.1/mapcache/demo/wmts>), a map viewer should load, similar to the image below.



- Zoom in a few times, and your configured cache location should be generating tiles (in this case inside `D:/ms4w/tmp/ms_tmp/cache/`).



6.1.2 Configuration File

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

The configuration files determines what and how mod-mapcache will serve incoming requests. It is an xml file that comprises a list of entries, as outlined here:

```
<mapcache>
  <grid>...</grid>
  <source>...</source>
  <cache>...</cache>
  <format>...</format>
  <tileset>...</tileset>
  <services>...</services>
</mapcache>
```

Source

A source is a service mod-mapcache can query to obtain image data. This is typically a WMS server accessible by a url (there are currently no other sources than WMS implemented, others may be added later if the need arises)

```
<source name="vmap0" type="wms">

  <!--
    extra parameters that will be added to the GetMap request. you can specify any
    parameter here, e.g. VERSION if you want to override the version of the WMS
    request.
    the LAYERS parameter is mandatory.
    usual parameters here are FORMAT , or MAP if using mapserver
  -->
  <getmap>
    <params>
      <FORMAT>image/png</FORMAT>
      <LAYERS>basic</LAYERS>
    </params>
  </getmap>

  <!-- http url and parameters that will be used when making WMS requests -->
  <http>

    <!-- url of the wms service, without any parameters -->
    <url>http://vmap0.tiles.osgeo.org/wms/vmap0</url>

    <!--
      http headers added to request. make sure you know what you are
      doing when adding a header here, as they take precedence over any
      default headers curl will be adding to the request.
      typical headers that can be added here are User-Agent and Referer.

      when adding a <key>value</key> element here, the request to the
      wms source will contain the

      key: value\r\n

      HTTP header.
    -->
    <headers>
      <User-Agent>mod-mapcache/r175</User-Agent>
      <Referer>http://www.mysite.com?param=2&par=4</Referer>
    </headers>
```

```

    <!-- timeout in seconds before bailing out from a request -->
    <connection_timeout>30</connection_timeout>
  </http>
</source>

```

- The name and type attributes are straightforward: type is “wms”, and name is
- the key by which this source will be referenced `<url>` is the http location
- where the service can be accessed `<wmsparams>` is a list of parameters that
- will be added to the wms request. You should probably at the very least add
- the FORMAT and LAYERS parameters. By convention(?), WMS parameters are
- uppercase, and you should respect this convention in your configuration file.
- This is where you can also override some default WMS parameters if needed. By
- default, the parameters that will be used are: `<REQUEST>GetMap</REQUEST>`
- `<SERVICE>WMS</SERVICE>` `<STYLES></STYLES>` `<VERSION>1.1.0</VERSION>`

Cache

A cache is a location where received tiles will be stored.

```

<cache name="disk" type="disk">

  <!-- base

  absolute filesystem path where the tile structure will be stored.
  this directory needs to be readable and writable by the user running
  apache
  -->
  <base>/tmp</base>

  <!-- symlink_blank

  enable blank (i.e. uniform color) tile detection. blank tiles will be
  detected at creation time and linked to a single blank tile on disk to
  preserve disk space.
  -->
  <symlink_blank/>
</cache>

<cache name="tpl" type="disk" layout="template">
  <!-- template

  string template that will be used to map a tile (by tileset, grid name, dimension,
  format, x, y, and z) to a filename on the filesystem.
  the following replacements are performed:
  - {tileset} : the tileset name
  - {grid} : the grid name
  - {dim} : a string that concatenates the tile's dimension
  - {ext} : the filename extension for the tile's image format
  - {x},{y},{z} : the tile x,y,z values
  - {inv_x}, {inv_y}, {inv_z} : inverted x,y,z values (inv_x = level->maxx - x - 1). This
    is mainly used to support grids where one axis is inverted (e.g. the google schema)
    and you want to create on offline cache.

```

```
* note that this type of cache does not support blank-tile detection and symlinking.

* warning: it is up to you to make sure that the template you chose creates a unique
filename for your given tilesets. e.g. do not ommit the {grid} parameter if your
tilesets reference multiple grids. Failure to do so will result in filename
collisions !

-->
<template>/tmp/template-test/{tileset}#{grid}#{dim}/{z}/{x}/{y}.{ext}</template>
</cache>

<!-- memcache cache
entry accepts multiple <server> entries
requires a fairly recent apr-util library and headers
-->
<cache name="memcache" type="memcache">
  <server>
    <host>localhost</host>
    <port>11211</port>
  </server>
</cache>

<!-- sqlite cache
requires building with "with-sqlite"
-->
<cache name="sqlite" type="sqlite3">
  <!-- dbfile

  absolute filename path of the sqlite database file to use.
  this file needs to be readable and writable by the user running
  tha mapcache instance
  -->
</cache>

<cache name="mbtiles" type="mbtiles">
  <dbfile>/path/to/MapBox/tiles/natural-earth-1.mbtiles</dbfile>
</cache>
```

Format

A format is an image format that will be used to return tile data to clients, and to store tile data on disk.

```
<format name="PNGQ_FAST" type="PNG">

  <!-- compression

  png compression: best or fast
  note that "best" compression is cpu intensive for little gain over the default
  default compression is obtained by leving out this tag.
  -->
  <compression>fast</compression>

  <!-- colors

  if supplied, this enables png quantization which reduces the number of colors
  in an image to attain higher compression. this operation is destructive, and can
  cause artifacts in the stored image.
```

```

    the number of colors can be between 2 and 256
-->
  <colors>256</colors>
</format>
<format name="myjpeg" type="JPEG">
  <!-- quality

    JPEG compression quality, ranging from 0 to 100
    95 produces high quality images with few visual artifacts
    values under around 80 produce small images but with visible artifacts.
    YMMV
-->
  <quality>75</quality>

  <!-- photometric

    photometric interpretation of the bands created in the jpeg image.
    default is ycbcr and produces the smallest images. can also be "rgb"
    which ususally results in x2 or x3 image sizes.
-->
  <photometric>ycbcr</photometric>
</format>
<format name="PNG_BEST" type="PNG">
  <compression>best</compression>
</format>
<format name="mixed" type="MIXED">
  <transparent>PNG_BEST</transparent>
  <opaque>JPEG</opaque>
</format>

```

Grid

A grid is the matrix that maps tiles on an area, and consists of a spatial reference, a geographic extent, resolutions, and tile sizes.

Mandatory Configuration Options

- **<size>**: The width and height of an individual tile, in pixels. Must be specified as to positive integers separated by a space character. The most common entry for this is:

```
<size>256 256</size>
```

- **<extent>**: The geographical extent covered by the grid, in ground units (e.g. meters, degrees, feet, ...). Must be specified as 4 floating point numbers separated by spaces, order like minx, miny, maxx, maxy. Mapcache expects all of its extents to be given in lonlat, and does the translation to latlon at request time if needed. The (minx,miny) point defines the origin of the grid, i.e. the pixel at the bottom left of the bottom-left most tile is always placed on the (minx,miny) geographical point.

The (maxx,maxy) point is used to determine how many tiles there are for each zoom level.

```
<extent>-180 -90 180 90</extent>
```

- **<srs>**: The projection of the grid, usually given by it epsg identifier. The actual meaning of the value put here isn't used directly by mapcache to compute reprojections, it is only used to lookup which grid to use when

receiving WMS requests.

```
<srs>epsg:4326</srs>
```

Note: This is the value that is passed on to the *source* when requesting a tile that is not already cached for the current grid. You must make sure that the source that is queried is capable of returning image data for this srs.

- **<units>**: The ground units used by the grid's projection. This entry is not used directly by mapcache aside from calculating scales for the WMTS capabilities document. Allowed values are:
 - **m**: meters
 - **dd**: decimal degrees
 - **ft**: feet

```
<units>dd</units>
```

- **<resolutions>**: This is a list of resolutions for each of the zoom levels defined by the grid. This must be supplied as a list of positive floating point values, separated by a space and ordered from largest to smallest. The largest value will correspond to the grid's zoom level 0. Resolutions are expressed in "units-per-pixel", depending on the unit used by the grid (e.g. resolutions are in meters per pixel for most grids used in webmapping).

```
<resolutions>0.703125000000000 0.351562500000000 0.175781250000000 8.78906250000000e-2 4.3945312
```

Optional Configuration Options

- **<srsalias>**: This tag can be specified multiple times, and allows the user to add multiple srs entries for a given grid. This is especially useful if the epsg id for a given projection has evolved over time, or to support other catalogues than the epsg one (which is the only catalog supported by the wms specification).

```
<srs>EPSG:310024802</srs>  
<srsalias>IGNF:GEOPORTALFXX</srsalias>  
<srsalias>EPSG:310024001</srsalias>
```

- **<metadata>**:
 - **<title>**: The name of the grid, in human readable form. Appears in the capabilities documents.

```
<title>This grid covers the area blah blah blah</title>
```

- **<WellKnownScaleSet>**: see the WMTS keyword. This will add a WellKnownScaleSet entry to the WMTS capabilities document. It is up to the user to make sure that the supplied resolutions for the grid actually match the pre-defined WellKnownScaleSet.

```
<WellKnownScaleSet>urn:ogc:def:wkss:OGC:1.0:GoogleCRS84Quad</WellKnownScaleSet>
```

- **<origin>**: Specifies the origin of the grid. Valid values are *top-left*, *bottom-left*, *top-right* and *bottom-right*. If not used, the grid will have the bottom left corner as reference point.

```
<origin>top-left</origin>
```

Preconfigured Grids

There are three predefined grids you can use without referencing them in the mapcache.xml file:

- the “WGS84” grid corresponds to a grid where the whole world is rendered on 2 times 1 256x256 pixel tiles at level 0 (i.e. the (-180,-90,180,90) extent fits on a 512x256 image). It goes down to zoom level 17.

```
<grid name="WGS84">
  <metadata>
    <title>GoogleCRS84Quad</title>
    <WellKnownScaleSet>urn:ogc:def:wkss:OGC:1.0:GoogleCRS84Quad</WellKnownScaleSet>
  </metadata>
  <extent>-180 -90 180 90</extent>
  <srs>EPSG:4326</srs>
  <units>dd</units>
  <size>256 256</size>
  <resolutions>0.703125000000000 0.351562500000000 0.175781250000000 8.78906250000000e-2 4.3945
</grid>
```

- the “g” grid corresponds to the case when you wish to overlay tiles on top of googlemaps, and is the default tiling scheme used in webmapping applications. This grid goes down to zoom level 18. Level 0 is a single 256x256 tile. This grid’s default srs is EPSG:900913 which is non-standard, but in wider use than than its official EPSG:3857 entry.

```
<grid name="g">
  <metadata>
    <title>GoogleMapsCompatible</title>
    <WellKnownScaleSet>urn:ogc:def:wkss:OGC:1.0:GoogleMapsCompatible</WellKnownScaleSet>
  </metadata>
  <extent>-20037508.3427892480 -20037508.3427892480 20037508.3427892480 20037508.3427892480</ex
  <srs>EPSG:900913</srs>
  <srsalias>EPSG:3857</srsalias>
  <units>m</units>
  <size>256 256</size>
  <resolutions>156543.0339280410 78271.51696402048 39135.75848201023 19567.87924100512 9783.939
</grid>
```

- the “GoogleMapsCompatible” grid is nearly identical to the “g” grid, except its default srs is EPSG:3857 instead of EPSG:900913.

```
<grid name="GoogleMapsCompatible">
  <metadata>
    <title>GoogleMapsCompatible</title>
    <WellKnownScaleSet>urn:ogc:def:wkss:OGC:1.0:GoogleMapsCompatible</WellKnownScaleSet>
  </metadata>
  <extent>-20037508.3427892480 -20037508.3427892480 20037508.3427892480 20037508.3427892480</ex
  <srs>EPSG:3857</srs>
  <srsalias>EPSG:900913</srsalias>
  <units>m</units>
  <size>256 256</size>
  <resolutions>156543.0339280410 78271.51696402048 39135.75848201023 19567.87924100512 9783.939
</grid>
```

Tileset

A tileset is the essential configuration item for mod-mapcache, and corresponds to a set of tiles coming from a *source*, stored in a *cache*, and returned to the client in a given *format*.

```
<tileset name="test">

  <!-- source: the "name" attribute of a preconfigured <source>
  If the tileset does not contain a <source> element, then it is
```

```

    considered read only and the caches will never be updated. In that
    sense you would have a slightly different mapcache.xml file for your
    seeder than for your webserver.
    As for returning blank tiles, you have the <errors> directive that does that (set it to empty.
<source>vmap0</source>

<!-- cache: the "name" attribute of a preconfigured <cache> -->
<cache>sqlite</cache>

<!-- grid: the "name" attribute of a preconfigured <grid>
    you can also use the following notation to limit the area that will be cached and served to client.
    <grid restricted_extent="-10 40 10 50">WGS84</grid>
    this way is better than using a grid with a limited extent, as in this way the tiles that are added
    cached are not invalidated should you want to modify the restricted extent in the future. When
    the restricted_extent attribute, you should give the corresponding information to the client to
    be using the service.

    You can also limit the zoom levels that are cached/accessible by using the minzoom, maxzoom attribute.

    NOTE: when adding a <grid> element, you *MUST* make sure that the source you have selected is able to
    return images in the grid's srs.
-->
<grid restricted_extent="-10 40 10 50" minzoom="4" maxzoom="17">WGS84</grid>
<grid>g</grid>

<!-- metadata
    optional metadata tags used for responding to GetCapabilities request.
    you can put anything in here, although only the title and abstract tags
    are currently used to populate the GetCapabilities document.
-->
<metadata>
  <title>vmap0 map</title>
  <abstract>blabla</abstract>
</metadata>

<!-- watermark
    optional tag to add a watermark to the tiles *before* storing them to cache
    the supplied image MUST be exactly the same size as the size of the tiles
    configured in the <grid>
    the supplied image is read when the configuration is loaded.
    if you make changes to the image, they will NOT be reflected on tiles already
    stored in the cache, nor on newly stored tiles until the server is restarted
<watermark>/path/to/static/watermark.png</watermark>
-->

<!-- format
    (optional) format to use when storing a tile. this should be a format with high
    compression, eg. png with compression "best", as the compression operation is only
    done once at creation time.
    if left out, no recompression is applied to the image, mod-mapcache will store the
    exact image received from the <source>
    note that the <format> tag is mandatory if metatile, metabuffer or watermark are
    supplied, as in those cases a recompression has to be done.
-->
<format>PNG</format>

<!-- metatile

```



```

    number of columns and rows to use for metatiling, see http://geowebcache.org/docs/current/conce
-->
<metatile>5 5</metatile>

<!-- metabuffer
    area around the tile or metatile that will be cut off to prevent some edge artifacts.
    if using this, the configured source must be instructed not to put any labels inside
    this area, as otherwise this will result in truncated labels (for mapserver, this is
    the "labelcache_map_edge_buffer" "-10" metadata entry, along with label PARTIALS FALSE
-->
<metabuffer>10</metabuffer>

<!-- expires
    optional expiration value in seconds for a tile. this is expressed in a number of seconds
    after the creation date of the tile
    This is the value that will be set in the HTTP Expires and Cache-Control headers, and has
    no effect on the actual expiration of tiles on the caches. See <auto_expire> for that.
-->
<expires>3600</expires>

<!-- auto_expire
    automatically re-request tiles and update the cache once they are older than the given number
    of seconds after their creation.
    Note that this will only delete tiles from the cache when they are accessed, you cannot
    use this configuration to limit the size of the created cache.
    Note that if set, this value overrides the value given by <expires>
-->
<auto_expire>86400</auto_expire>

<!-- dimensions
    optional dimensions that should be cached
    the order of the <dimension> tags inside the <dimensions> is important as it is used
    to create the directory structure for the disk cache. i.e. if you change the order of these
    values, any tiles that have been previously cached are invalidated (but not removed from the
    cache, it's just they don't exist anymore for mod-mapcache
-->
<dimensions>
  <!-- values dimension
    the example here creates a DIM1 dimension
    * WMS and WMTS clients can now add a &DIM1=value to their request string. If they don't
    specify this key/value, the default will be to use DIM1=foobar
    * the allowed values for DIM1= are foobar (it is important to add the default value to the
    allowed values entry), foobarbaz, foo and bar.
    * the value specified for DIM1 will be forwarded to the WMS source
    * the produced tile will be stored in base/gridname/DIM1/value/xx/xx/xx/xx/xx/xx.png
    file. i.e. there are as many different caches created as there are values in the
    <values> tag.
  -->
  <dimension type="values" name="DIM1" default="foobar">foobar, foobarbaz, foo, bar</dimension>

  <!-- regex dimension
    the following creates a MAPFILE dimension, for using the same mod-mapcache tileset with dif.
    mapserver mapfiles. the name of the mapfiles need not be known to mod-mapcache, and can be
    created even after mod-mapcache has been started.
    when a user passes a MAPFILE=/path/to/mapfile, the string "/path/to/mapfile" is validated against
    the supplied (PCRE) regular expression. The one in this example allows a name composed of any
    characters separated by slashes (/) and finishing with ".map" ( [a-zA-Z0-9\.\.]*\.map$ ), but will fail if
    there are two consecutive dots (..) in the path, to prevent filesystem traversal ( (?!\.\.\.)).

```

```

-->
<dimension type="regex" name="MAPFILE" default="/path/to/mapfile.map">^(?!.*\.\.)([a-zA-Z0-9\.\.])
<!-- intervals dimension
the syntax is the same as common-ows, i.e. a comma separated list of "min/max/resolution" etc
eg:
* 0/5000/1000 allows the values 0,1000,2000,3000,4000 and 5000
* 0/100/0 allows any values between 0 and 100
* both values can be combined: 0/5000/1000,0/100/0
-->
<dimension name="ELEVATION" type="intervals" default="0">0/5000/1000</dimension>

<!-- coming in a future version: support for ISO8601 date/time dimensions -->

</dimensions>
</tileset>

```

Services

Services are the type of request that mod-mapcache will respond to. You should of course enable at least one.

```

<service type="wms" enabled="true">
  <!-- this service should actually be called "ogc". It is different from the other
  services as it does not listen on the /wms endpoint, but directly on /.
  It will intercept wms getmap requests that can be treated from configured
  tilesets, and can optionally forward all the rest to (an)other server(s)
  TODO: this needs way more documenting
  <forwarding_rule name="foo rule">
    <append_pathinfo>true</append_pathinfo>
    <http>
      <url>http://localhost/mapcacheproxy</url>
    </http>
  </forwarding_rule>
  -->
  <!-- full_wms
  configure response to wms requests that are not aligned to a tileset's grids.
  responding to requests that are not in the SRS of a configured grid is not supported, but
  this should never happen as only the supported SRSs are publicized in the capabilities
  document.

  allowed values are:
  - error: return a 404 error (default)
  - assemble: build the full image by assembling the tiles from the cache
  - forward: forward the request to the configured source.
  -->
  <full_wms>assemble</full_wms>
  <!-- resample mode
  filter applied when resampling tiles for full wms requests.
  can be either:
  - nearest : fastest, poor quality
  - bilinear: slower, higher quality
  -->
  <resample_mode>bilinear</resample_mode>

  <!-- format
  image format to use when assembling tiles
  -->

```

```

<format>myjpeg</format>

</service>
<service type="wmts" enabled="true"/>
<service type="tms" enabled="true"/>
<service type="kml" enabled="true"/>
<service type="gmaps" enabled="true"/>
<service type="ve" enabled="true"/>
<service type="demo" enabled="true"/>

```

Miscellaneous

```

<!-- default_format
  format to use when a client asks for an image that is dynamically created from multiple
  tiles from the cache.
  note that using a png format with "best" compression is not recommended
  here as it comes with a very compression overhead in terms of cpu processing. it is
  recommended to use a png format with "fast"compression here, unless you have plenty
  of server cpu power and or limited bandwidth
-->
<default_format>JPEG</default_format>

<!-- services
  services that will be responded to by mod-mapcache
  each service is accessible at the url http://host/path/to/mapcache/{service},
  eg http://myhost/mapcache/wms for OGC WMS.
-->

<!-- errors
  configure how error will be reported back to a client:
  - log : no error is reported back, except an http error code.
  - report : return the error message to the client in textual format
  - empty_img : return an empty image to the client. the actual error code is in the X-Mapcache
  - report_img : return an image with the error text included inside. not implemented yet.

  the default setting is to report the error message back to the user. In production, you might want
  if you're paranoid, or to "empty_img" if you want to play nice with non-conforming clients.
-->
<errors>report</errors>

<!--
  location to put lockfiles (to block other clients while a metatile is being rendered.
  defaults to /tmp
  this location should be writable by the apache user
-->
<lock_dir>/tmp</lock_dir>

<!--
  interval in microseconds to sleep before checking that a lockfile is still present.
  default is 1/100th of a second (i.e. 10000 microseconds)
-->
<lock_retry>10000</lock_retry>

<!-- log_level
  For CGI/FastCGI only - For the apache module use the httpd.conf
  LogLevel key.

```

```
    Defines the verbosity of the what is sent to the logs.
    - debug
    - info
    - notice
    - warn (default)
    - error
    - crit
    - alert
    - emerg
-->
<log_level>warn</log_level>

<!-- auto_reload
    For FastCGI only. If set to true, the configuration will be automatically
    reloaded if the configuration file has changed.
    default is false.
-->
<auto_reload>true</auto_reload>
```

6.1.3 Supported Tile Services

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

Author Stephen Woodbridge

MapCache has the ability to serve tiles using a variety of different request protocols and tile naming conventions. This document is to describe these. The various services must be turned on in the mapcache.xml file for MapCache to respond to those specific requests.

All services are available on the demo interface, from which you are highly encouraged to copy/paste the javascript code to get started when creating your own pages accessing the MapCache tiles.

The following notation is used on this page and refer to object names in the mapcache.xml configuration file.

- <tileset_name> - name of a configured tileset
- <grid_name> - name of explicitly or implicitly defined grid
- <quadkey> - specific to the Virtual Earth Tile service
- <z> - Zoom level in zxy naming scheme
- <y> - row number in zxy naming scheme
- <x> - column number in zxy naming scheme

TMS service

The TMS service uses a z/x/y tile naming scheme where:

- z is the zoom level
- x is the column number
- y is the row number

To activate the TMS service, add these lines to the mapcache.xml configuration file:

```
<service type="tms" enabled="true"/>
```

A “capabilities” document can be fetched via:

```
http://myhost.com/mapcache/tms/1.0.0/
```

Tiles are requested using this scheme:

```
http://myhost.com/mapcache/tms/1.0.0/<tileset_name>@<grid_name>/<z>/<x>/<y>.png
```

For epsg:3857 or epsg:900913 or GoogleMapsCompatible grids, with cell = [z/x/y]:

z0:

```
[0/0/0]
```

z1:

```
[1/0/0] [1/1/0]
[1/0/1] [1/1/1]
```

z2:

```
[2/0/0] [2/1/0] [2/2/0] [2/3/0]
[2/0/1] [2/1/1] [2/2/1] [2/3/1]
[2/0/2] ...
[2/0/3] ...
```

etc...

For epsg:4326 or WGS84 grids:

Note: The OGC WMTS specification rather absurdly requires the GoogleCRS84Quad WellKnownScaleSet to have a level 0 who’s extent is -180,-180,180,180. The default “WGS84” MapCache grid honors this, which may cause some incompatibilities with software that expects level 0 to be 2x1 tiles with extent -180,-90,180,90

z0:

```
[0/0/0]
```

z1:

```
[1/0/0] [1/1/0]
```

z2:

```
[2/0/0] [2/1/0] [2/2/0] [2/3/0]
[2/0/1] [2/1/1] [2/2/1] [2/3/1]
```

etc...

KML Service

The KML service produces [Super-Overlays](#) for tilesets that are aligned to the WGS84 / epsg:4326 grids. A Super-Overlay is a KML file that links to an image url, and to a set of other KML urls corresponding to neighbouring resolutions. The KML service uses a z/x/y tile naming scheme where:

- *z* is the zoom level
- *x* is the column number

- *y* is the row number

Note: For the KML service to be functional, the TMS service must also be activated, as the KML super-overlays link to images using this spec.

To activate the KML service, add these lines to the mapcache.xml configuration file:

```
<service type="tms" enabled="true"/>
<service type="kml" enabled="true"/>
```

Tiles are requested using this scheme:

```
http://myhost.com/mapcache/kml/<tileset_name>@<grid_name>/<z>/<x>/<y>.kml
```

OGC WMTS Service

To activate the WMTS service, add these lines to the mapcache.xml configuration file:

```
<service type="wmts" enabled="true"/>
```

This service follows the standard OGC WMTS requests and supports both the classical OGC style KVP encoded and REST style requests.

```
http://myhost.com/mapcache/wmts?SERVICE=WMTS&VERSION=1.0.0&...
http://myhost.com/mapcache/wmts/1.0.0/....
```

The capabilities are obtained through:

```
http://myhost.com/mapcache/wmts?service=wmts&request=getcapabilities&version=1.0.0
http://myhost.com/mapcache/wmts/1.0.0/WMTSCapabilities.xml
```

See also:

FeatureInfo Requests

See also:

Tileset Dimensions

OGC WMS Service

MapCache responds to WMS version 1.1.1 requests, and has limited support for version 1.3.0 ones.

```
<service type="wms" enabled="true"/>
```

Note: Note that the WMS service is a little different than the other MapCache services, as it listens on the root of the configured instance instead of on an additional endpoint (i.e. the service replies on <http://server/mapcache/?> and not on <http://server/mapcache/wms?>). This behavior is required in order to enable *proxying of unsupported requests* while offering a single endpoint for all OGC services.

Note: MapCache primarily supports version 1.1.1 WMS requests, but has limited support for the newer version 1.3.0 ones. For the 1.3.0 requests, MapCache will determine which *grid* to use by using the CRS= parameter instead of the SRS= one, and will correctly honor axis ordering for the epsg reference systems that switch the usual x/y ordering of the BBOX parameter.

See also:

*FeatureInfo Requests***See also:***Tileset Dimensions***See also:***Tile Assembling*

WMS requests follow the classical KVP encoded style:

```
http://myhost.com/mapcache?SERVICE=WMS&VERSION=1.1.1&REQUEST=....
```

The capabilities document is returned by:

```
http://myhost.com/mapcache?service=wms&request=getcapabilities
```

GoogleMaps XYZ Service

```
<service type="gmaps" enabled="true"/>
```

Prerequisites: your WMS should be capable of producing images in the EPSG:900913 or EPSG:3857 SRS, i.e. it should reference the “g” or “GoogleMapsCompatible” grid

This is the minimal html page that should get you going. The important bits are in the urlTemplate (for V2) and getTileURL (for V3) variables:

- /mapcache is the apache path where MapCache handles requests
- test@g is the tileset and grid name to use, joined by a ‘@’ - the {Z}/{X}/{Y} should be left alone
- the final extension should be changed to “jpg” if you are using a jpeg format with your tileset.

V2 API

```
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8"/>
<title>Google/MapServer Tile Example</title>
<script src="http://maps.google.com/maps?file=api&v=2&key=ABQIAAAAnfs7bKE82qgb3Zc2YyS-oBT2yXp_ZAY8_u"
  type="text/javascript"></script>
<script type="text/javascript">

function load() {
  if (GBrowserIsCompatible()) {
    var urlTemplate = '/mapcache/gmaps/test@g/{Z}/{X}/{Y}.png';
    var myLayer = new GTileLayer(null, 0, 18, {
      tileUrlTemplate:urlTemplate,
      isPng:true,
      opacity:0.8 });
    var map = new GMap2(document.getElementById("map"));
    map.addControl(new GLargeMapControl());
    map.addControl(new GMapTypeControl());
    map.setCenter(new GLatLng(0, 0), 1);
    map.addOverlay(new GTileLayerOverlay(myLayer));
```

```
    }  
  }  
  
</script>  
</head>  
<body onload="load()" onunload="GUnload()">  
  <div id="map" style="width: 500px; height: 500px"></div>  
</body>  
</html>
```

V3 API

The previous javascript for the V2 example should be slightly changed to:

```
var map = new google.maps.Map("<element-id>", { /*options*/ });  
var layerOptions = {  
  getTileUrl: function(coord, zoom) {  
    return "/mapcache/gmaps/test@g/" + zoom + "/" + coord.x + "/" + coord.y + ".png";  
  },  
  tileSize: new google.maps.Size(256,256) // or whatever  
};  
map.overlayMapTypes.insertAt(0, new google.maps.ImageMapType(layerOptions));
```

Virtual Earth Tile service

Tiles are organized in one of two different layouts depending on whether they are using a Spherical Mercator project, like epsg:3857 or epsg:900913, or if they are using the geographical projection, like epsg:4326.

Tiles are requested using this scheme:

```
http://myhost.com/mapcache/ve?LAYER=<tileset_name>@<grid_name>&tile=<quadkey>
```

For epsg:3857 or epsg:900913 or GoogleMapsCompatible grids, <quadkey> are arranged:

z0:

```
[0]  
  
http://myhost.com/mapcache/ve?LAYER=osm@GoogleMapsCompatible&tile=0
```

z1:

```
[00] [01]  
[02] [03]  
  
http://myhost.com/mapcache/ve?LAYER=osm@GoogleMapsCompatible&tile=00  
http://myhost.com/mapcache/ve?LAYER=osm@GoogleMapsCompatible&tile=01  
http://myhost.com/mapcache/ve?LAYER=osm@GoogleMapsCompatible&tile=02  
http://myhost.com/mapcache/ve?LAYER=osm@GoogleMapsCompatible&tile=03
```

```
etc...
```

For epsg:4326 or WGS84 grids, <quadkey> are arranged:

z1:


```
[0] [1]
```

```
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=0
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=1
```

z2:

```
[00] [01] [10] [11]
```

```
[02] [03] [12] [13]
```

```
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=00
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=01
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=02
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=03
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=10
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=11
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=12
http://myhost.com/mapcache/ve?LAYER=osm@WGS84&tile=13
```

```
etc...
```

6.1.4 Seeder

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

Author Mathieu Coudert

Contact mathieu.coudert at gmail.com

Mod-mapcache ships with an advanced seeding tool, whose main features are:

- configurable number of seeding threads, to speed up the rendering.
- ability to reseed tiles older than a certain timestamp
- ability to seed tiles given a shapefile/ogr datasource

Usage

The seeding utility is named `mapcache_seed`, and is located under your install directory (default is `/usr/local/bin`).

Commandline options

Options are available in a short or long version (i.e. `-c` or `--config`).

-c | --config [file]: path to the `mapcache.xml` configuration file that contains the tilesets that need to be seeded.

-D | --dimension "DIMENSION=VALUE": used to specify which dimension to use if the tileset supports dimensions. Can be used multiple times to set multiple dimensions, e.g. `-D "DIM1=VAL1" -D "DIM2=VAL2"`.

-e | --extent minx,miny,maxx,maxy: bounding box of the area to seed.

-f | --force: force tile recreation even if it already exists.

-g | --grid [grid]: name of the grid that must be seeded (the selected tileset must reference the given grid).

-h | --help: show help

-i | --iteration-mode: either “*drill-down*” or “*level-by-level*”. Default is to use drill-down for g, WGS84 and GoogleMapsCompatible grids, and level-by-level for others. Use this flag to override.

-m | --mode: the mode to use the seeder : seed, delete or transfer. Default is seed (*mode: seed*).

-M | --metasize: override metatile size while seeding, eg. 8,8.

-n | --nthreads: number of parallel threads that should be used to request tiles from the wms source. The default is 1, but can be set higher if the WMS server can withstand parallel requests (as a rule of thumb, the value chosen here should never be much higher than the number of cpus on the wms server).

Note: this option is incompatible with the **-p | --nprocesses** option.

-o | --older [timestamp|now]: only seed tiles that are older than the given value. The value can either be the string “now”, or a date formatted like year/month/day hour:minute, eg: “2011/01/31 20:45”.

Note: a full timestamp should be quoted.

-p | --nprocesses: number of parallel processes that should be used to request tiles from the wms source.

Note: this option is incompatible with the **-n | --nthreads** option.

Warning: When working with multiple processes (-p switch) and sqlite cache backends, some errors may appear under high concurrency when writing to the sqlite database (error: SQL logic error or missing database (1)). Upgrading to sqlite >= 3.7.15 seems to resolve this issue.

-q | --quiet: don't print progress messages to the standard output.

-t | --tileset [tileset]: name of the tileset that must be seeded.

-v | --verbose: print verbose debugging info (if compiled in).

-x | --transfer: the tileset to transfer when seeder is use into transfer mode.

-z | --zoom minzoom,maxzoom: start and end zoom levels that must be seeded.

Optional Commandline options when using OGR/GEOS

At compile time, if ogr and geos where found on the system, the seeder tool supports additional options to seed only the tiles that cover an arbitrary geographical area. *Important:* Note that for the time being, the OGR datasource should be in the same projection as the grid you are seeding, as there is no automatic reprojection from the datasource projection to the grid projection.

-d | --ogr-datasource [ogr_datasource]: ogr connection to the spatial source. Consult the OGR documentation for all that is supported. In the simplest case (e.g. a Shapefile), this is just the full filename of the shapefile.

-l | --ogr-layer [ogr_layer]: for datasources that contain multiple layers (e.g. postgis, with multiple tables), determines which layer will be used.

-s | --ogr-sql [ogr_sql]: OGR sql expression that can be applied (see [OGR SQL](#)).

-w | --ogr-where [ogr_where]: sql “where” expression to filter out returned values. This would typically be used to select only the geometry of a given country if the datasource contains all the world contours.

Important Note

The seeding utility must be run under the same user account as the user running the webserver. This is required so the permissions on the tiles created by the seeder are accessible by the webserver, and conversely so the seeder has the rights to write files to directories created by the webserver.

A sample seeding session goes like this:

```
[user@host]$ sudo www-data
[www-data@host]$ /path/to/mapcache/src/mapcache_seed -c /path/to/www/conf/mapcache.xml [options]
[www-data@host]$ logout
[user@host]$
```

Examples

Seed the “osm” tileset with the “g” (google/web-mercator) grid:

```
./src/mapcache_seed -c mapcache.xml -t osm -g g
```

Seed levels 0 through 12:

```
./src/mapcache_seed -c mapcache.xml -t osm -g g -z 0,12
```

Given a shapefile that contains the world country countours, seed only the areas that are covered by land (i.e. skip the oceans). Also use 4 request threads in parallel:

```
./src/mapcache_seed -c mapcache.xml -t osm -g g -z 0,12 -n 4 -d /path/to/seed.shp
```

Same as beforehand, but only seed the USA (notice the quote usage, required to create valid sql with a single-quoted ‘US’:

```
./src/mapcache_seed -c mapcache.xml -t osm -g g -z 0,12 -n 4 -d /path/to/seed.shp -w "FIPS_A2='US'"
```

Reseed levels 0 to 12 (this could also be done by deleting the cache for levels 0 to 12 and doing a classic seed, but doing so this way does not slow down the access from web clients):

```
./src/mapcache_seed -c mapcache.xml -t osm -g g -z 0,12 -o now
```

6.1.5 Cache Types

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

This document details the different cache backends that can be used to store tiles

Disk Caches

The disk based cache is the simplest cache to configure, and the one with the the fastest access to existing tiles. It is ideal for small tile repositories, but may cause trouble for sites hosting millions of tiles, as the number of files or directory may rapidly overcome the capabilities of the underlying filesystem. Additionally, the block size chosen for the filesystem must closely match the mean size of a stored tile: ideally, any given tile should just fit inside a filesystem block, so as not to waste storage space inside each block, and not have to use up multiple blocks per tile.

The location of the files/directories has to be readable and writable by the user running the tile server.

There are two types of disk caches, that create a different hierarchy of files:

Default Structure

The default disk cache stores tiles in a structure nearly identical to the file/directory hierarchy used by TileCache. The only change is that a top level directory corresponding to the name of the grid is added (as MapCache supports multiple grids per tileset)

This cache is capable of detecting blank (i.e. uniform color) tiles and using a symbolic link to a single blank tile to gain disk space.

```
<cache name="disk" type="disk">
  <base>/tmp</base>
  <symlink_blank/>
</cache>
```

The two only configuration keys are the root directory where the tiles will be stored, and a key to activate the symbolic linking of blank tiles

Template Structure

The template based disk cache allows you to create (or reuse an existing) tile structure that you define in advance. The <template> parameter takes a string argument where various template entries will be replaced at runtime by the correct value for each tile to store.

```
<cache name="tpl" type="disk">
  <!-- template

  string template that will be used to map a tile (by tileset, grid name, dimension,
  format, x, y, and z) to a filename on the filesystem.
  the following replacements are performed:
  - {tileset} : the tileset name
  - {grid} : the grid name
  - {dim} : a string that concatenates the tile's dimension
  - {ext} : the filename extension for the tile's image format
  - {x},{y},{z} : the tile x,y,z values
  - {inv_x}, {inv_y}, {inv_z} : inverted x,y,z values (inv_x = level->maxx - x - 1). This
    is mainly used to support grids where one axis is inverted (e.g. the google schema)
    and you want to create on offline cache.

  * note that this type of cache does not support blank-tile detection and symlinking.

  * warning: it is up to you to make sure that the template you chose creates a unique
  filename for your given tilesets. e.g. do not ommit the {grid} parameter if your
  tilesets reference multiple grids. Failure to do so will result in filename
  collisions !

  -->
  <template>/tmp/template-test/{tileset}#{grid}#{dim}/{z}/{x}/{y}.{ext}</template>
</cache>
```

BerkeleyDB Caches

The BerkeleyDB cache backend stores tiles in a key-value flat-file database. and therefore does not have the disadvantages of disk caches with regards to the number of files stored on the filesystem. As the image blobs are stored contiguously, the block size chosen for the filesystem has no influence on the storage capacity of the volume.

Note that for a given bdb cache, only a single database file is created, which will store the tiles of its associated tilesets (i.e. there is not a database file created per tileset, grid, and or dimension). If you need to store different tilesets to different files, then use multiple dbd cache entries. It is not possible to use multiple database files for tileset grids or dimensions.

The BerkeleyDB based caches are a bit faster than the disk based caches during reads, but may be a bit slower during concurrent writes if a high number of threads all try to insert new tiles concurrently.

```
<cache name="bdb" type="bdb">
  <!-- base (required)
    absolute filesystem path where the Berkeley db database file is to be stored.
    this directory must exist, and be writable
  -->
  <base>/tmp/foo/</base>

  <!-- key_template (optional)
    string template used to create the key for a tile entry in the database.
    defaults to the value below. you should include {tileset}, {grid} and {dim} here
    unless you know what you are doing, or you will end up with mixed tiles
  <key_template>{tileset}-{grid}-{dim}-{z}-{y}-{x}.{ext}</key_template>
  -->
</cache>
```

Sqlite Caches

There are two different sqlite caches that vary by the database schema they create and query. Sqlite caches have the advantage that they store tiles as blobs inside a single database file, and therefore do not have the disadvantages of disk caches with regards to the number of files stored. As the image blobs are stored contiguously, the block size chosen for the filesystem has no influence on the storage capacity of the volume.

The sqlite based caches are a bit slower than the disk based caches, and may have write-locking issues at seed time if a high number of threads all try to insert new tiles concurrently.

Default Schema

Tiles are stored in the configured sqlite file created by mapcache with

```
create table if not exists tiles(
  tileset text,
  grid text,
  x integer,
  y integer,
  z integer,
  data blob,
  dim text,
  ctime datetime,
  primary key(tileset, grid, x, y, z, dim)
);
```

```
<cache name="sqlite" type="sqlite3">
</cache>
```

You may also add custom sqlite pragmas that will be executed when first connecting to a sqlite db, e.g. to override some compiled in sqlite defaults

```
<cache name="sqlitetemplate" type="sqlite3">
  <dbfile>/tmp/sqlitefile.db</dbfile>
  <pragma name="max_page_count">1000000</pragma>
</cache>
```

<pragma> entries will result in a call to

```
PRAGMA max_page_count = 1000000;
```

Custom Schema

This cache can use any database schema, it is up to you to supply the SQL that will be executed to select or insert a new tile.

This cache type is not fully implemented yet (there is no way to configure it from the mapcache.xml configuration file yet)

MBTiles Schema

This cache type is a shortcut to the previous custom schema sqlite cache, with pre-populated SQL queries that correspond to the MBTiles specification.

Although the default mbtiles schema is very simple, mapcache uses the same multi-table schema found in most downloadable mbtiles file, to notably enable storing blank (i.e. uniform) tiles without duplicating the encoded image data (in the same way the disk cache supports tile symlinking).

The mbtiles schema is created with:

```
create table if not exists images (
  tile_id text,
  tile_data blob,
  primary key(tile_id));
create table if not exists map (
  zoom_level integer,
  tile_column integer,
  tile_row integer,
  tile_id text,
  foreign key(tile_id) references images(tile_id),
  primary key(tile_row,tile_column,zoom_level));
create table if not exists metadata(
  name text,
  value text); -- not used or populated yet
create view if not exists tiles
as select
  map.zoom_level as zoom_level,
  map.tile_column as tile_column,
  map.tile_row as tile_row,
  images.tile_data as tile_data
from map
join images on images.tile_id = map.tile_id;
```

```
<cache name="mbtiles" type="mbtiles">
  <dbfile>/Users/XXX/Documents/MapBox/tiles/natural-earth-1.mbtiles</dbfile>
</cache>
```

Note: Contrarily to the standard sqlite mapcache schema, the mbtiles db file only supports a single tileset per cache.

The behavior if multiple tilesets are associated to the same mbtiles cache is undefined, and will definitely produce incorrect results.

Warning: When working with multiple processes (-p switch) and sqlite cache backends, some errors may appear under high concurrency when writing to the sqlite database (error: SQL logic error or missing database (1)). Upgrading to sqlite >= 3.7.15 seems to resolve this issue.

Memcache Caches

This cache type stores tile to an external memcached server running on the local machine or accessible on the network. This cache type has the advantage that memcached takes care of expiring tiles, so the size of the cache will never exceed what has been configured in the memcache instance.

Memcache support requires a rather recent version of the apr-util library. Note that under very high loads (usually only attainable on synthetic benchmarks on localhost), the memcache implementation of apr-util may fail and start dropping connections for some intervals of time before coming back online afterwards.

You can add multiple <server> entries.

```
<cache name="memcache" type="memcache">
  <server>
    <host>localhost</host>
    <port>11211</port>
  </server>
</cache>
```

Note: Tiles stored in memcache backends are configured to expire in 1 day by default. This can be overridden at the tileset level with the <auto_expire> keyword.

(Geo)TIFF Caches

TIFF caches are the most recent addition to the family of caches, and use the internal tile structure of the TIFF specification to access tile data. Tiles can be stored in JPEG only (TIFF does not support PNG tiles).

As a single tiff file may contain many tiles, there is a drastic reduction in the number of files that have to be stored on the filesystem, which solves the major shortcomings of the disk cache. Another advantage is that the same TIFF files can be used by programs or WMS servers that only understand regular GIS raster formats, and be served up with high performance for tile access.

The TIFF cache should be considered read-only for the time being. Write access is already possible but should be considered experimental as there might be some file corruption issues, notably on network file-systems. Note that until all the tiles in a given tiff file have been seeded/created, the TIFF file is said to be “sparse” in the sense that it is missing a number of jpeg tiles. As such most non-gdal based programs will have problems opening these incomplete files.

Note that the tiff tile structure must exactly match the structure of the grid used by the tileset, and the tif file names must follow strict naming rules.

Defining the TIFF file sizes

The number of tiles stored in each the horizontal and vertical direction must be defined:

- <xcount> the number of tiles stored along the x (horizontal) direction of the TIFF file.

- <ycount> the number of tiles stored along the y (vertical) direction of the TIFF file.

```
<cache name="tiff" type="tiff">
  <xcount>64</xcount>
  <ycount>64</ycount>
  ...
</cache>
```

Setting up the file naming convention

The <template> tag sets the template to use when looking up a TIFF file name given the x,y,z of the requested tile

```
<cache name="tiff" type="tiff">
  <template>/data/tiffs/{tileset}/{grid}/L{z}/R{inv_y}/C{x}.tif</template>
  ...
</cache>
```

The following template keys are available for operating on the given tile's x,y, and z:

- {x} is replaced by the x value of the leftmost tile inside the TIFF file containing the requested tile
- {inv_x} is replaced by the x value of the rightmost tile.
- {y} is replaced by the y value of the bottommost tile.
- {inv_y} is replaced by the y value of the topmost tile.
- {div_x} is replaced by the index of the TIFF file starting from the left of the grid (i.e. {div_x} = {x}/<countx>)
- {inv_div_x} same as {div_x} but starting from the right.
- {div_y} is replaced by the index of the TIFF file starting from the bottom of the grid (i.e. {div_y} = {y}/<county>)
- {inv_div_y} same as {div_y} but starting from the top.

Note: {inv_x} and {inv_div_x} will probably be rarely used, whereas {inv_y} and {inv_div_y} will find some usage for people who prefer to index their TIFF files from top to bottom rather than bottom to top.

Customizing the template keys

In some occurrences, it may be desirable to have a precise hand on the filename to use for a given x,y,z, tile lookup, e.g. to look for a file named "Z03-R00003-C000009.tif" instead of just "Z3-R3-C9.tif". The <template> entry supports formatting attributes, following the unix printf syntax (c.f.: <http://linux.die.net/man/3/printf>), by suffixing each template key with "_fmt", e.g.:

```
<cache name="tiff" type="tiff">
  <template
    x_fmt="%08d"
    inv_y_fmt="%08d"
  >/data/tiffs/{tileset}/{grid}/L{z}/R{inv_y}/C{x}.tif</template>
</cache>
```

Note: If not specified, the default behavior is to use "%d" for formatting.

Setting JPEG compression options

An additional optional parameter defines which JPEG compression should be applied to the tiles when saved into the TIFF file:

- `<format>` the name of the (jpeg) format to use

See also:

JPEG Format

```
<cache name="tiff" type="tiff">
  ...
  <format>myjpeg</format>
</cache>
```

In this example, assuming a grid using 256x256 tiles, the files that are read to load the tiles are tiled TIFFs with jpeg compression, whose size are 16384x16384. The number of files to store on disk is thus reduced 4096 times compared to the basic disk cache.

GeoTIFF Support

If compiled with GeoTiff and write support, MapCache will add referencing information to the TIFF files it creates, so that the TIFF files can be used in any geotiff-enabled software. Write support does not produce full geotiffs with the definition of the projection used, but only the pixel scale and tie-points, i.e. what is usually found in .tfw files.

For reference, here is the gdalinfo output on a TIFF file created by MapCache when compiled with geotiff support:

```
LOCAL_CS["unnamed",
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"]]]
Origin = (-20037508.342789247632027,20037508.342789247632027)
Pixel Size = (156543.033928040997125,-156543.033928040997125)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  COMPRESSION=YCbCr JPEG
  INTERLEAVE=PIXEL
  SOURCE_COLOR_SPACE=YCbCr
Corner Coordinates:
Upper Left  (-20037508.343,20037508.343)
Lower Left  (-20037508.343,-20037508.343)
Upper Right (20037508.343,20037508.343)
Lower Right (20037508.343,-20037508.343)
Center      ( 0.0000000, 0.0000000)
```

6.1.6 Image Formats

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

MapCache allows you to configure how the image should be saved to a cache once it has been requested from a source. The JPEG format should mostly be used for raster imagery, whereas the PNG format is most useful for vector based imagery where there are large uniform areas.

JPEG Format

The JPEG format saves tiles to jpeg, you can configure the JPEG compression level (from 1 to 100) and the colorspace that should be used (rgb or ycbcr)

```
<format name="myjpeg" type="JPEG">
  <quality>85</quality>
  <photometric>ycbcr</photometric>
</format>
```

- *quality*: this is the typical jpeg quality setting. Values under 50 produce lighter images but with notable compression artifacts. 100 should be avoided as it produces very heavy images
- *photometric*: By default the YCbCr colorspace is used as it produces images that tend to be 2 to 3 times lighter. Use rgb if you don't want the default.

PNG Format

The PNG format creates PNG images, with optional quantization (reduction of the number of colors to create an 8bit paletted PNG)

```
<format name="mypng" type="PNG">
  <compression>fast</compression>
  <colors>256</colors>
</format>
```

- *compression*: choose which zlib compression to apply to the image data. Recognized values are "fast" and "best". Omit the key to use the default zlib compression.
- *colors*: number of colors to use for quantization. Omit this key to produce 24 or 32 bit RGB/RGBA pngs, or set to a value between 2 and 256 to create an 8-bit paletted png. The quantization step is destructive, there is no guarantee that pixels will not have a noticeable shift in color in the case when the tile contains many colors.

Mixed Format

There is a third special format which mixes JPEG and PNG compression depending on the contents of the image. This format allows to create caches for raster imagery using JPEG compression (which is more efficient) on zones with imagery data, and PNG compression (which supports transparency) on zones with no imagery or on a boundary between imagery and emptiness.

```
<format name="mymixed" type="MIXED">
  <opaque>myjpeg</opaque>
  <transparent>mypng</transparent>
</format>
```

- *opaque*: the format to use when the image has only fully opaque pixels
- *transparent*: the format to use if when the image has some transparent or semi-opaque pixels.

6.1.7 Tileset Dimensions

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

6.1.8 FeatureInfo Requests

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

TBD

6.1.9 Proxying Unsupported Requests

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

Note: This page is a work in progress

MapCache has the ability to forward any incoming request that it cannot natively respond to (either by returning a tile directly, by *merging* multiple tiles, etc...).

This setup allows mapcache to be placed transparently in front of an existing OGC-service supplying server to accelerate tiled or getmap requests for a selected number of *grids*, while maintaining service compatibility for, e.g., unsupported grids, WFS requests, ...

Note: The proxying of requests is configured inside the *WMS* MapCache service, which is semantically awkward.

The configuration for this behavior is activated by a succession of entries inside the `<forwarding_rule>` element of the `wms <service>`. Rules are tested for in the order in which they appear in the `mapcache.xml` configuration file, and the first one that matches is used. If no rules are defined, or if no rule matches the incoming request, an error is returned to the user.

```
<service type="wms" enabled="true">
  <forwarding_rule name="first rule">
    <!-- rule tests -->
    <!-- proxy destination -->
  </forwarding_rule>
  <forwarding_rule name="second rule">
    <!-- rule tests -->
    <!-- proxy destination -->
  </forwarding_rule>
</service>
```

A `<forwarding_rule>` consists of a set of matching rules and an `<http>` block defining where the request should be forwarded to.

Parameter Filtering

The rules apply to the KVP parameters that were passed in the incoming request, and are added with the `<param>` keyword:

```
<forwarding_rule name="first rule">
  <param name="SERVICE" type="values">WFS,WCS</param>
  <!-- ... !>
</forwarding_rule>
```

The “type” attribute is the same that what is allowed for *dimensions*, i.e. allowed values are “values”, “regex”, and “intervals”. In the previous example, the rule would match any incoming request having ...&SERVICE=WFS&... or ...&SERVICE=WCS&... in its request parameters.

```
<forwarding_rule name="first rule">
  <param name="SERVICE" type="values">WFS,WCS</param>
  <param name="LAYERS" type="values">somelayername</param>
  <!-- ... !>
</forwarding_rule>
```

Multiple rules can be used if the filtering has to be done on mutiple parameters. In the previous example, the rule would match a WFS or WCS request that concerns the “somelayername” layer only.

A <forwarding_rule> that has no <param> child will match any incoming request that could not be serviced by MapCache directly from its cache, and can be used to forward all unsupported request to a full OGC compliant server so that an un-cached response can be returned to the client.

See also:

Tileset Dimensions

Proxy Destination

Once a <forwarding_rule> matches, its <http> child will be used to proxy the request to another server.

```
<forwarding_rule name="first rule">
  <!-- ... !>
  <http>
    <url>http://wmserver/ogc.cgi?</url>
  </http>
</forwarding_rule>
```

See also:

HTTP Service Definition

6.1.10 Data Sources

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

Note: This page is a work in progress

MapCache uses the concept of a “source” as a service that is able to return image data given a set of parameters (namely an extent, an image size, and a projection). Typically, a source is the third party WMS server that you want to put a tilecache in front of.

HTTP Service Definition

WMS Sources

MapFile Sources

6.1.11 Tile Assembling

Author Thomas Bonfort

Contact tbonfort at terriscope.fr

TBD

See also:

[MapCache presentation slides at FOSS4G2011](#)

6.1.12 Features

- services WMS, WMTS, TMS, VirtualEarth/Bing and Googlemaps requests: *Supported Tile Services*
- ability to respond to untiled WMS requests by merging tiles from the cache or forwarding them to the wms source: *Tile Assembling*
- responds to WMS/WMTS *GetFeatureInfo* requests (forwarded to source service)
- *KML* superoverlay generation
- data provided by WMS backends (GDAL supported sources planned)
- cache types:
 - *Disk*
 - *BerkeleyDB*
 - *Sqlite*
 - *Memcached*
 - *Tiled TIFFs*
- configurable metatiling
- on-the-fly tile merging for combining multiple tiles into a single image
- image post-processing (recompression and quantization) when arriving from a backend
- interprets and produces cache control headers: Last-Modified, If-Modified-Since, Expires
- multithreaded *seeding utility*, that can seed specific zoom levels or specific areas (e.g.: seed levels 0 to 12 of all tiles intersecting Colorado)
- ability to add a custom watermark on stored tiles
- produces a CGI/fastCGI executable for using with other webservers than apache
- configurable symbolic linking of blank tiles to gain disk storage
- configurable error reporting: plain http error code, textual message, or empty (blank) image
- ability to specify vendor params or dimensions to be forwarded to the WMS backend (and build a cache that takes these parameters into account): *Tileset Dimensions*

7.1 Data Input

7.1.1 Vector Data

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Tyler Mitchell

Contact tmitchell at osgeo.org

Last Updated 2013-10-01

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://www8.nos.noaa.gov/coris_glossary/index.aspx?letter=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:

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.

File names 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 discussed 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.

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
```

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!

- **HOBUSTATES_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.

Contour

1. Overview

Mapserver can compute and render a contour layer on the fly from a raster source. The raster source is one band of raster data, which represents a digital elevation model (DEM). More info about DEMs at: http://en.wikipedia.org/wiki/Digital_elevation_model

2. How it works

CONNECTIONTYPE CONTOUR. The new type is a hybrid layer, which has a raster data source as input and vector features as output. Initially, only the line representation of those vector features will be supported.

Since the internal layer is of type vector, queries will be supported and operate on the vectors (not on the raw raster source). In the future we might see a need to add a query mode that queries the raster source, but this is not included in this phase of work.

To render a contour layer, we need to define a layer in the mapfile with the following options:

- Set the layer TYPE to LINE.
- Set CONNECTIONTYPE to CONTOUR.
- Set the DATA to the raster file that contains the elevation band.
- Specify the band to use as elevation using PROCESSING “BANDS”, same as regular raster.
- Specify one or more classes and styles to render the line features.

PROCESSING settings:

These options should be specified at layer level:

- CONTOUR_INTERVAL: elevation interval between contours
- CONTOUR_LEVELS: comma-separated list of one or more ‘fixed levels’ to extract
- CONTOUR_ITEM: provides a name for the item (attribute) in which to put the elevation. (optional)

You can also provide explicit min/max scaledenom in the CONTOUR_INTERVAL or CONTOUR_LEVELS values if you wish to use scale-dependent contour spacing. This is done by adding an optional “miscaledenom,maxscaledenom:” prefix to the value or list of values. See the example below.

Example of a simple layer definition:

```
LAYER NAME "my_contour_layer"
  TYPE LINE
  STATUS DEFAULT
  CONNECTIONTYPE CONTOUR
  DATA /mnt/data/raster/grib/dem.grib
  PROCESSING "BANDS=1"
  PROCESSING "CONTOUR_ITEM=elevation"
  PROCESSING "CONTOUR_INTERVAL=10"
  CLASS
    STYLE
      WIDTH 2
      COLOR 255 0 0
    END
  LABEL
  END
END
```

Example of a layer definition with scale-dependent contour ranges:

```
LAYER NAME "my_contour_layer"
  TYPE LINE
  STATUS DEFAULT
  CONNECTIONTYPE CONTOUR
  DATA /mnt/data/raster/grib/dem.grib
  PROCESSING "BANDS=1"
  PROCESSING "CONTOUR_ITEM=elevation"
  PROCESSING "CONTOUR_INTERVAL=0,25000:5" # interval of 5 for scales of 25000 or less
  PROCESSING "CONTOUR_INTERVAL=25000,500000:10" # interval of 10 for scales in the 25000 to 500000 range
  PROCESSING "CONTOUR_LEVELS=500000,0:10,25,50,100" # explicit list of levels for scales of 500000 and above
  LABELITEM "elevation"
  CLASS
    STYLE
      WIDTH 2
      COLOR 255 0 0
    END
  LABEL
  ...
  END
END
```

2.1 Data cellsize

The data produced by the gdal contour algorithm are generally in high resolution. A lot of point are used to generated contours with precision. You might want to generalize/simplify the line in some cases (ie. Shape Smoothing). The `[data_cellsize]` attribute binding represents the cellsize of the extend fetched from the raster file. This is different than the map cellsize.

In the following example, I generalize my shape with a tolerance of 25% of the data cellsize to produce smooth contours at all scales:

```
LAYER
  NAME "MyContourLayer"
  STATUS DEFAULT
  DATA "wind.tif"
  TYPE LINE
  CONNECTIONTYPE CONTOUR
  PROJECTION AUTO END
  PROCESSING "BANDS=1"
  PROCESSING "CONTOUR_ITEM=elevation"
```

```

PROCESSING "CONTOUR_INTERVAL=0,0:1"
GEOMTRANSFORM (smoothsia(generalize([shape], 0.25*[data_cellsize])))
CLASS
  EXPRESSION ([elevation] >= 0)
  STYLE
    COLOR 0 0 255
  END # STYLE
END # CLASS
END # LAYER

```

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
```

ESRI File Geodatabase

ESRI File Geodatabases exist in a file folder and offer improved performance and size limitations. For more information see the [ESRI description page](#).

Note: Only file geodatabases created by ArcGIS 10.0 and above can be read by GDAL/MapServer.

File listing

File geodatabases are made up of a set of files within a folder. The files are made up of geographic data, attribute data, index files, and lock files. A better description of the file contents can be found [here](#).

Data Access / Connection Method

File geodatabase access is available through OGR. See the OGR [driver page](#) for specific driver information. The driver is available for GDAL \geq 1.9.0.

The CONNECTION parameter must be used to point to the name of the file folder, and the DATA parameter should be the name of the spatial table (or OGR layer).

```
CONNECTIONTYPE ogr
CONNECTION "filegdb-folder"
DATA "layername"
```

Note: The CONNECTION path is relative to the mapfile (SHAPEPATH is not used here). Full paths can also be used.

OGRINFO Examples

First you should make sure that your GDAL/OGR build contains the file geodatabase “FileGDB” driver, by using the ‘-formats’ command:

```
>ogrinfo --formats
Supported Formats:
...
"FileGDB" (read/write)
"ESRI Shapefile" (read/write)
"MapInfo File" (read/write)
"UK .NTF" (readonly)
"SDTS" (readonly)
"TIGER" (read/write)
...
```

If you don't have the driver, see GDAL's [BuildHints](#) page for compiling the driver.

Once you have the FileGDB driver you are ready to try an ogrinfo command on your database to get a list of spatial tables. In the example below our folder is named *us_states.gdb*:

```
ogrinfo us_states.gdb
INFO: Open of `us_states.gdb'
using driver `FileGDB' successful.
1: statesp020 (Multi Polygon)
```

Now use ogrinfo to get information on the structure of the *statesp020* table:

```
ogrinfo us_states.gdb statesp020 -summary
INFO: Open of `us_states.gdb'
using driver `FileGDB' successful.

Layer name: statesp020
Geometry: Multi Polygon
Feature Count: 2895
Extent: (-179.000000, 17.000000) - (179.000000, 71.000000)
Layer SRS WKT:
GEOGCS["GCS_North_American_1983",
  DATUM["North_American_Datum_1983",
    SPHEROID["GRS_1980",6378137.0,298.257222101]],
  PRIMEM["Greenwich",0.0],
  UNIT["Degree",0.017453292519943295]]
```

```
FID Column = OBJECTID
Geometry Column = SHAPE
AREA: Real (0.0)
PERIMETER: Real (0.0)
STATESP020: Real (0.0)
STATE: String (0.0)
STATE_FIPS: String (0.0)
```

Mapfile Example

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

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 PGeo:testDSN counties -summary
  INFO: Open of `testDSN'
  using driver `PGeo' successful.

  1: counties

```

```
2: counties_Shape_Index
...
```

(where “testDSN” is the name of your System DSN)

Mapfile Example

Direct Access to MDB

```
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
```

Through an ODBC Connection

```
LAYER
  NAME my_geodatabase
  TYPE POLYGON
  CONNECTIONTYPE ogr
  CONNECTION "PGeo:testDSN"
  DATA "counties"
  STATUS ON
  CLASS
    NAME "counties"
    STYLE
      COLOR 255 255 120
    END
  END
END
```

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)
TITLE: 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
```

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
```

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

```

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.

KML - Keyhole Markup Language

Table of Contents

- *KML - Keyhole Markup Language*
 - *Links to KML-Related Information*
 - *Data Access / Connection Method*
 - *Example 1: Displaying a .KML file*
 - *Example 2: Displaying a .KMZ file*

Keyhole Markup Language (KML) is an XML-based language for managing the display of 3D geospatial data. KML is a standard maintained by the Open Geospatial Consortium (OGC).

Links to KML-Related Information

- [Google's KML Reference](#)
- [OGC's KML Specification](#)
- [KML Validator](#)
- [KML Validator](#) (against OGC KML 2.2)

Data Access / Connection Method

KML access in MapServer 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. A more complete KML reader was added to GDAL/OGR in version 1.8.0, through the [libKML driver](#) (including the ability to read multigeometry, and KMZ files).

The CONNECTION parameter must include the kml or kmz extension, and the DATA parameter should be the name of the layer.

```
CONNECTIONTYPE OGR
CONNECTION "filename.kml"
DATA "layername"
```

Example 1: Displaying a .KML file

OGRINFO 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_example"
  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
```

Example 2: Displaying a .KMZ file

OGRINFO First you should make sure that your GDAL/OGR build contains the “LIBKML” driver, by using the ‘-formats’ command:

```
>ogrinfo --formats
  Loaded OGR Format Drivers:
  ...
  -> "GML" (read/write)
  -> "GPX" (read/write)
  -> "LIBKML" (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. Or you can follow the [compiling notes](#) for libKML and GDAL/OGR.

Once you have the LIBKML driver you are ready to try an ogrinfo command on your file to get a list of available layers:

```
>ogrinfo Lunenburg_Municipality.kmz
  INFO: Open of `Lunenburg_Municipality.kmz'
  using driver `LIBKML' successful.
  1: Lunenburg_Municipality
```

Now use ogrinfo to get information on the structure of the layer:

```
>ogrinfo Lunenburg_Municipality.kmz Lunenburg_Municipality -summary
  INFO: Open of `Lunenburg_Municipality.kmz'
  using driver `LIBKML' successful.

  Layer name: Lunenburg_Municipality
  Geometry: Unknown (any)
  Feature Count: 1
  Extent: (-64.946433, 44.133207) - (-64.230281, 44.735125)
  Layer SRS WKT:
  GEOGCS["WGS 84",
    DATUM["WGS_1984",
      SPHEROID["WGS 84",6378137,298.257223563,
        AUTHORITY["EPSG","7030"]],
      TOWGS84[0,0,0,0,0,0,0],
      AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
      AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
      AUTHORITY["EPSG","9108"]],
    AUTHORITY["EPSG","4326"]]
  Name: String (0.0)
  description: String (0.0)
  timestamp: DateTime (0.0)
  begin: DateTime (0.0)
  end: DateTime (0.0)
  altitudeMode: String (0.0)
  tessellate: Integer (0.0)
  extrude: Integer (0.0)
  visibility: Integer (0.0)
```

Mapfile Example

```
LAYER
  NAME "lunenburg"
  TYPE POLYGON
  STATUS DEFAULT
  CONNECTIONTYPE OGR
  CONNECTION "Lunenburg_Municipality.kmz"
  DATA "Lunenburg_Municipality"
  CLASS
    NAME "Lunenburg"
    STYLE
      COLOR 244 244 16
      OUTLINECOLOR 199 199 199
    END
  END
END # layer
```

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

```

MSSQL

Author Tamas Szekeres

Contact szekerest at gmail.com

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2012-09-26

Contents

- *MSSQL*
 - *Introduction*
 - *Creating Spatial Data Tables in MSSQL 2008*
 - *Connecting to Spatial Data in MSSQL 2008*
 - * *OPTION 1: Connect Through OGR*
 - *Verify Local Support for MSSQLSpatial*
 - *Test OGR Connection Parameters*
 - *Create MapServer Layer using CONNECTIONTYPE OGR*
 - * *OPTION 2: Connect Through MapServer Plugin*
 - *Create MapServer Layer*
 - *Selecting the Type of the Geometry Column*
 - *Expected Location of the MSSQL Plugin*
 - *Binaries Containing the MSSQL Plugin*
 - * *Using Spatial Indexes*
 - * *Layer Processing Options*
 - *More Information*

Introduction

Microsoft SQL Server 2008+ supports storing spatial data by using the built-in geometry/geography data types. MapServer can connect to MSSQL through either: 1) an OGR connectiontype, or 2) a driver that accesses these tables containing spatial columns, which is compiled as a plugin (“msplugin_mssql2008.dll”).

Creating Spatial Data Tables in MSSQL 2008

There are several ways to create spatial data tables in MSSQL 2008. You can easily upload existing data to an MSSQL table by using the `ogr2ogr` commandline tool and the OGR’s [MSSQL Spatial driver](#) Here is an example that uploads a shapefile (province.shp) into an MSSQL 2008 instance:

```
ogr2ogr -f MSSQLSpatial -a_srs EPSG:4326 "MSSQL:server=.\SQLEXPRESS;database=geo;trusted_connection=y
```

Connecting to Spatial Data in MSSQL 2008

In order to connect to the MSSQL 2008 spatial database you should set up a valid connection string to the database like the following examples:

```
Server=.\MSSQLSERVER2008;Database=Maps;Integrated Security=true
```

```
Server=55.55.55.55,1433;uid=a_user;pwd=a_password;database=a_database;  
Integrated Security=True
```

```
Server=55.55.55.55\SQLEXPRESS,1433;uid=a_user;pwd=a_password;  
database=a_database;Integrated Security=True
```

OPTION 1: Connect Through OGR GDAL/OGR (and therefore MapServer) can read spatial tables in MSSQL 2008 through the [MSSQLSpatial driver](#).

Verify Local Support for MSSQLSpatial Use the command “ogrinfo –formats” to verify that your local GDAL is built with support for MSSQL; the response should contain “MSSQLSpatial” such as:

```
Supported Formats:
-> "OCI" (read/write)
-> "ESRI Shapefile" (read/write)
-> "MapInfo File" (read/write)
...
-> "MSSQLSpatial" (read/write)
...
```

Test OGR Connection Parameters Use the ogrinfo commandline utility to test your connection through the MSSQLSpatial driver, such as:

```
ogrinfo "MSSQL:server=.\SQLEXPRESS;database=geo;trusted_connection=yes" province -summary
```

Create MapServer Layer using CONNECTIONTYPE OGR Your layer should contain a CONNECTIONTYPE OGR statement, as well as a CONNECTION. The connection should also contain a “tables=” parameter, and also the name of the geometry column in brackets. You do not need to specify the DATA parameter unless you define an sql select statement starting with the ‘WHERE’ keyword. For example:

```
LAYER
  NAME "provinces"
  TYPE POLYGON
  STATUS ON
  ####
  CONNECTIONTYPE OGR
  CONNECTION "MSSQL:server=.\SQLEXPRESS;uid=xx;pwd=xxx;database=geo;trusted_connection=yes;tables=provinces"
  ####
  PROJECTION
    "init=epsg:4326"
  END
  CLASS
    NAME "Land"
    STYLE
      COLOR 240 240 240
      OUTLINECOLOR 199 199 199
    END
  END
  PROCESSING 'CLOSE_CONNECTION=DEFER'
END # layer
```

Note: The usual CONNECTIONTYPE terms ‘using unique’ and ‘using srid’ are not meaningful for the OGR driver in this case, as these parameters are automatically retrieved from the ‘geometry_columns’ metadata table.

OPTION 2: Connect Through MapServer Plugin

Create MapServer Layer Once the connection can be established to the server the layer can be configured to access MSSQL2008 as follows:

```
LAYER
  NAME "rivers_mssql_spatial"
  TYPE POLYGON
  STATUS DEFAULT
```

```
CONNECTIONTYPE PLUGIN
PLUGIN "msplugin_mssql2008.dll"
CONNECTION "Server=.\MSSQLSERVER2008;Database=Maps;Integrated Security=true"
DATA "ogr_geometry from rivers USING UNIQUE ogr_fid USING SRID=4326"
...
END
```

The DATA parameter is used to perform the SQL select statement to access your table in MSSQL. The geometry column is required in the select statement; in the above example the ogr_geometry column is the geometry column in the rivers table. The table should also have an unique column (ogr_fid) which is provided for random access to the features in the feature query operations.

The DATA section should also contain the spatial reference id (SRID) of the features in the data table The SRID is used when specifying the search shapes during the intersect operations which should match with the SRID of the features otherwise no features are returned in a particular query. if you omit specifying the SRID value in the DATA section the driver will use SRID=0 when defining the search shapes.

Selecting the Type of the Geometry Column For the geometry columns MSSQL supports 2 data types: “geometry” and “geography”. By default the driver considers the type of the geometry column is “geometry”. In case if the type of the geometry column is “geography” we must specify the data type in the DATA section explicitly, like:

```
DATA "ogr_geometry(geography) from rivers USING UNIQUE ogr_fid USING SRID=4326"
```

Expected Location of the MSSQL Plugin On Windows platforms the DLLs needed by the program are searched for in the following order:

1. The directory from which the application loaded.
2. The current directory.
3. The system directory. Use the `GetSystemDirectory` function to get the path of this directory.
4. The 16-bit system directory.
5. The Windows directory. Use the `GetWindowsDirectory` function to get the path of this directory.
6. The directories that are listed in the PATH environment variable.

Binaries Containing the MSSQL Plugin Currently the following binary distributions contain msplugin_mssql2008.dll:

- [MapServer and GDAL binary and SDK packages](#)
- [MS4W distributions](#)

Using Spatial Indexes In order to speed up the access to the features a spatial index should be created to the geometry column which could easily be done with the OGR MSSQL Spatial driver like:

```
ogrinfo -sql "create spatial index on rivers"
"MSSQL:server=.\MSSQLSERVER2008;database=Maps;
Integrated Security=true"
```

In general we can safely rely on the query optimizer to select the most appropriate index in the sql query operations. In some cases - however - we should force the optimizer to use the spatial index by specifying the index hint in the DATA section like:

```
DATA "ogr_geometry from rivers using index ogr_geometry_sidx
      USING UNIQUE ogr_fid USING SRID=4326"
```

Layer Processing Options We can control the behaviour of the MSSQL driver by using the following PROCESSING options:

- **CLOSE_CONNECTION=DEFER** - 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.
- **MSSQL_READ_WKB=TRUE** - Uses WKB (Well Known Binary) format instead of native format when fetching geometries.

More Information

- [OGR MSSQL Spatial driver page](#) (describes the OGR MSSQL support)
- [ogr2ogr application](#) (describes the ogr2ogr commandline application)
- [Vector Data](#) (MapServer Vector Data Access Guide)

MySQL

Author David Fawcett

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

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2010-07-29

Contents

- *MySQL*
 - *Introduction*
 - *Connecting to Spatial Data in MySQL*
 - * *Requirements*
 - * *Verify MySQL Support in OGR Build*
 - * *Test Connection with ogrinfo*
 - * *Create MapServer Layer*
 - *Connecting to non-Spatial Data in MySQL*
 - * *Requirements*
 - * *Create .ovf file*
 - * *Test Connection with ogrinfo*
 - * *Create MapServer Layer*
 - *More Information*

Introduction

The following methods connect to MySQL through OGR's [MySQL driver](#), thus avoiding the need to set up an ODBC connection.

Connecting to Spatial Data in MySQL

This section describes how to display a spatial MySQL table (meaning that the table has a column of type geometry) in MapServer. OGR's [MySQL driver](#) was expanded in OGR version 1.3.2 to support access to MySQL spatial tables.

Requirements

- MapServer compiled with OGR support
- OGR/GDAL version 1.3.2 or more recent compiled with MySQL support

Verify MySQL Support in OGR Build You can verify that your local build of OGR contains MySQL support by using the ogrinfo commandline utility, and making sure that “MySQL” is returned:

```
ogrinfo --formats

Supported Formats:
-> "ESRI Shapefile" (read/write)
-> "MapInfo File" (read/write)
...
-> "PostgreSQL" (read/write)
-> "MySQL" (read/write)
...
```

Test Connection with ogrinfo MySQL connection strings in OGR are in the following format:

```
MYSQL:database,host=yourhost,user=youruser,password=yourpass,tables=yourtable
```

Therefore an example ogrinfo command would be:

```
> ogrinfo MYSQL:test,user=root,password=mysql,port=3306
```

which should return a list of all of your tables in the ‘test’ database:

```
INFO: Open of `MYSQL:test,user=root,password=mysql,port=3306'
      using driver `MySQL' successful.
1: province (Polygon)
```

and you can return a summary of the MySQL spatial layer:

```
> ogrinfo MYSQL:test,user=root,password=mysql,port=3306 province -summary

INFO: Open of `MYSQL:test,user=root,password=mysql,port=3306'
      using driver `MySQL' successful.

Layer name: province
Geometry: Polygon
Feature Count: 48
Extent: (-13702.315770, 3973784.599548) - (1127752.921471, 4859616.714055)
Layer SRS WKT:
PROJCS["ED50_UTM_zone_30N",
...
FID Column = OGR_FID
Geometry Column = SHAPE
id: Real (2.0)
...
```

Create MapServer Layer

```
LAYER
  NAME "spain_provinces_mysql_spatial"
  TYPE POLYGON
  STATUS DEFAULT
  CONNECTIONTYPE OGR
  CONNECTION "MySQL:test,user=root,password=mysql,port=3306"
  DATA "SELECT SHAPE,admin_name from province"
  LABELITEM "admin_name"
  CLASS
    NAME "Spain Provinces"
    STYLE
      COLOR 240 240 240
      OUTLINECOLOR 199 199 199
    END
    LABEL
      COLOR 0 0 0
      FONT sans
      TYPE truetype
      SIZE 8
      POSITION AUTO
      PARTIALS FALSE
      OUTLINECOLOR 255 255 255
    END
  END
END # layer
```

The `DATA` parameter is used to perform the SQL select statement to access your table in MySQL. The geometry column is required in the select statement; in the above example the `SHAPE` column is the geometry column in the `province` table.

Connecting to non-Spatial Data in MySQL

This section describes how to display a non-spatial MySQL table (meaning the table does not have a column of type geometry) in MapServer.

Support for this functionality is found in GDAL/OGR 1.2.6 and older on Windows and GDAL/OGR 1.3.2 on Linux.

Requirements

- 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](#) (MapServer OGR document)
- [Vector Data](#) (MapServer Vector Data Access Guide)
- [MySQL wiki page](#) (describes the deprecated mygis support)

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
END
```

OGR

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2010-10-16

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
- OGDl Vectors
- Oracle Spatial
- PostgreSQL
- SDTS
- SQLite
- UK.NTF (National Transfer Format)
- US Census TIGER/Line
- VRT - Virtual Datasource
- WFS
- X-Plane/Flighgear aeronautical data

Note: Some of the above formats (e.g. OGDl) 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 OGD I 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, idl.Name FROM pol LEFT JOIN idl ON pol.eas_id = idl.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 ogrindex. 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, your 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 label-cache 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://demo.mapserver.org/ogr-demos/yk_demo/etc/symbols_mapinfo.txt

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 line thickness is mapped to CLASS.STYLE.WIDTH. The default will be 1 pixel line (as it always is with MapServer).

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).

If the OGR symbol id is a web reference (<http://.../mysymbol.png>), the symbol will be downloaded and a new symbol entry will be created referring to it.

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. The font styles *bold* and *italic* are supported as follows: Arial bold fontname maps to *arial-bold*. Arial italic fontname maps to *arial-italic*. Arial bold italic fontname maps to *arial-bold-italic*. If the styles are not available, arial 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.

Transparency If the color parameter from the OGR style contains an alpha value, the value will be used to set the *OPACITY* parameter in the *STYLE* object.

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://demo.mapserver.org/ogr-demos/yk_demo/demo_init.html
- http://demo.mapserver.org/ogr-demos/nfld_demo/demo_init.html

The following site uses OGR, as well as MapInfo's 'Seamless Map Layers' feature:

- http://demo.mapserver.org/ogr-demos/ro_demo/demo_init.html

The following site uses OGR to display TIGER 2000 files:

- http://demo.mapserver.org/ogr-demos/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://demo.mapserver.org/ogr-demos/yk_demo/etc/symbols_mapinfo.txt

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.

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"
  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.

PostGIS/PostgreSQL

Table of Contents

- *PostGIS/PostgreSQL*
 - *PostGIS/PostgreSQL*
 - *Data Access /Connection Method*
 - *OGRINFO Examples*
 - *Mapfile Example*
 - *Support for SQL/MM Curves*
 - * *Example#1: CircularString in MapServer*
 - * *Example#2: CompoundCurve in MapServer*
 - * *Example#3: CurvePolygon in MapServer*
 - * *Example#4: MultiCurve in MapServer*
 - * *Example#5: MultiSurface in MapServer*
 - * *Using MapServer < 6.0*

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 when drawing features, but using them improves performance. If you want to make MapServer query calls to a PostGIS layer, your DATA parameter must include "using unique". Omitting it will cause the query to fail.

Here is a simple generic example:

```
CONNECTIONTYPE POSTGIS
CONNECTION "host=yourhostname dbname=yourdatabasename user=yourdbusername
           password=yourdbpassword port=yourpgport"
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.aid) as subquery
      using unique gid using srid=4326"

```

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 ST_Area(g.the_geom) desc
      limit 10) as subquery using unique gid using srid=4326"

```

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_Intersects(g.geom,!BOX!)) as
      subquery using unique gid using srid=4326"

```

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.net/documentation/>

Support for SQL/MM Curves

PostGIS is able to store circular interpolated curves, as part of the SQL Multimedia Applications Spatial specification (read about the [SQL/MM specification](#)).

For more information about PostGIS' support, see the *SQL-MM Part 3* section in the PostGIS documentation, such as [here](#).

As of MapServer 6.0, the PostGIS features CircularString, CompoundCurve, CurvePolygon, MultiCurve, and MultiSurface can be drawn through MapServer directly.

Example#1: CircularString in MapServer The following is the Well Known Text of the feature loading into PostGIS:

```
INSERT INTO test ( g, id ) VALUES ( ST_GeomFromText('CIRCULARSTRING(0 0,
4 0, 4 4, 0 4, 0 0)', -1), 2);
```

An example MapServer layer might look like:

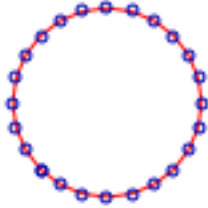
```
LAYER
  NAME "curves_poly"
  STATUS DEFAULT
  TYPE POLYGON
  CONNECTIONTYPE postgis
  CONNECTION "user=postgres password=postgres dbname=curves host=localhost port=5432"
```

```

DATA "g from test using SRID=-1 using unique id"
CLASS
  STYLE
    COLOR 128 128 128
    ANTIALIAS true
  END
END
END

```

And testing with *shp2img* should produce a map image of:



Example#2: CompoundCurve in MapServer The following is the Well Known Text of the feature loading into PostGIS:

```

INSERT INTO test ( g, id ) VALUES ( ST_GeomFromText('COMPOUNDCURVE(
    CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1)'), -1), 3);

```

An example MapServer layer might look like:

```

LAYER
  NAME "curves_poly"
  STATUS DEFAULT
  TYPE POLYGON
  CONNECTIONTYPE postgis
  CONNECTION "user=postgres password=postgres dbname=curves host=localhost port=5432"
  DATA "g from test using SRID=-1 using unique id"
  CLASS
    STYLE
      COLOR 128 128 128
      ANTIALIAS true
    END
  END
END

```

And testing with *shp2img* should produce a map image of:



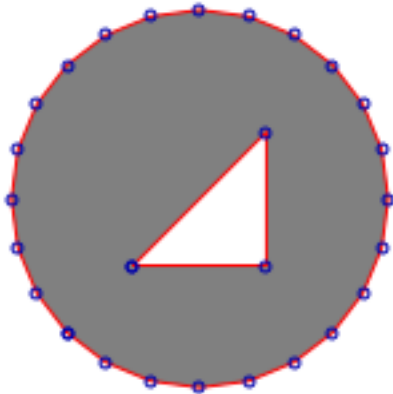
Example#3: CurvePolygon in MapServer The following is the Well Known Text of the feature loading into PostGIS:

```
INSERT INTO test ( g, id ) VALUES ( ST_GeomFromText('CURVEPOLYGON(
    CIRCULARSTRING(0 0, 4 0, 4 4, 0 4, 0 0),(1 1, 3 3,
    3 1, 1 1)'), -1), 4);
```

An example MapServer layer might look like:

```
LAYER
  NAME "curves_poly"
  STATUS DEFAULT
  TYPE POLYGON
  CONNECTIONTYPE postgis
  CONNECTION "user=postgres password=postgres dbname=curves host=localhost port=5432"
  DATA "g from test using SRID=-1 using unique id"
  CLASS
    STYLE
      COLOR 128 128 128
      ANTIALIAS true
    END
  END
END
```

And testing with *shp2img* should produce a map image of:



Example#4: MultiCurve in MapServer The following is the Well Known Text of the feature loading into PostGIS:

```
INSERT INTO test ( g, id ) VALUES ( ST_GeomFromText('MULTICURVE((0 0,
    5 5),CIRCULARSTRING(4 0, 4 4, 8 4)'), -1), 6);
```

An example MapServer layer might look like:

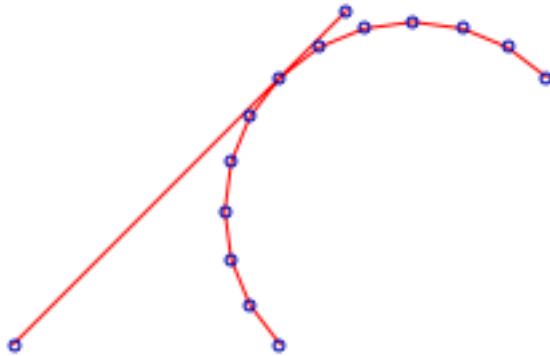
```
LAYER
  NAME "curves_poly"
  STATUS DEFAULT
  TYPE POLYGON
  CONNECTIONTYPE postgis
  CONNECTION "user=postgres password=postgres dbname=curves host=localhost port=5432"
  DATA "g from test using SRID=-1 using unique id"
```

```

CLASS
  STYLE
    COLOR 128 128 128
    ANTI_ALIAS true
  END
END
END
END

```

And testing with *shp2img* should produce a map image of:



Example#5: MultiSurface in MapServer The following is the Well Known Text of the feature loading into PostGIS:

```

INSERT INTO test ( g, id ) VALUES ( ST_GeomFromText('MULTISURFACE(
    CURVEPOLYGON(CIRCULARSTRING(0 0, 4 0, 4 4, 0 4,
    0 0),(1 1, 3 3, 3 1, 1 1)),((10 10, 14 12, 11 10,
    10 10),(11 11, 11.5 11, 11 11.5, 11 11)))', -1), 7);

```

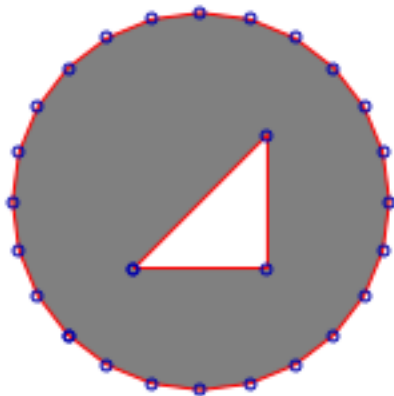
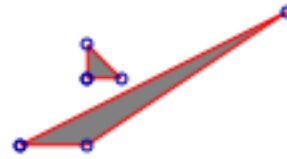
An example MapServer layer might look like:

```

LAYER
  NAME "curves_poly"
  STATUS DEFAULT
  TYPE POLYGON
  CONNECTIONTYPE postgis
  CONNECTION "user=postgres password=postgres dbname=curves host=localhost port=5432"
  DATA "g from test using SRID=-1 using unique id"
  CLASS
    STYLE
      COLOR 128 128 128
      ANTI_ALIAS true
    END
  END
END

```

And testing with *shp2img* should produce a map image of:



Using MapServer < 6.0 If you cannot upgrade to MapServer 6.0, then you can use the PostGIS function *ST_CurveToLine()* in your MapServer LAYER to draw the curves (note that this is much slower however):

```
LAYER
  NAME "curves_poly"
  STATUS DEFAULT
  TYPE POLYGON
  CONNECTIONTYPE postgis
  CONNECTION "user=postgres password=postgres dbname=curves host=localhost port=5432"
  DATA "wkb_geometry from (select c.id, ST_CurveToLine(c.g) as
                                wkb_geometry from c) as subquery using
                                unique id using SRID=-1"

  CLASS
    STYLE
      COLOR 128 128 128
      ANTIALIAS true
    END
  END
END
```


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 sdts_maryland
  TYPE LINE
  CONNECTIONTYPE OGR
  CONNECTION "data/sdts/MD/MDTRCATD.DDF"
  DATA "LE01"
  STATUS DEFAULT
  CLASS
```

```

STYLE
  COLOR 0 0 0
END
END
END

```

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: LNDGRN
10: LNDMRK
11: LIGHTS (Point)
12: OBSTRN
13: RDOSTA (Point)
14: SEAARE
15: SBDARE
16: SLCONS
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)
NTXTDS: 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/uslbs02m.000"
  DATA "DEPARE"
  CLASS
    STYLE
      COLOR 247 237 219
      OUTLINECOLOR 120 120 120
    END
  END
END # Layer
```

Spatialite

Spatialite spatially enables the file-based SQLite database. For more information see the [Spatialite description page](#).

File listing

Similar to other database formats, the .sqlite file consists of several tables. The geometry is held in a BLOB table column.

Data Access / Connection Method

Spatialite access is available through OGR. See the [OGR driver page](#) for specific driver information. The driver is available in GDAL/OGR version 1.7.0 or later.

OGR uses the names of spatial tables within the Spatialite database (tables with a geometry column that are registered in the geometry_columns table) as layers.

The CONNECTION parameter must include the sqlite extension, and the DATA parameter should be the name of the spatial table (or OGR layer).

```
CONNECTIONTYPE OGR
CONNECTION "spatialite_db.sqlite"
DATA "layername"
```

OGRINFO Examples

First you should make sure that your GDAL/OGR build contains the spatialite “SQLite” driver, by using the ‘--formats’ command:

```
>ogrinfo --formats
  Loaded OGR Format Drivers:
  ...
  -> "GMT" (read/write)
  -> "SQLite" (read/write)
  -> "ODBC" (read/write)
  ...
```

If you don't have the driver, you might want to try the [MS4W](#) or [OSGeo4W](#) packages, which include the driver.

Once you have confirmed that you have the SQLite driver you are ready to try an ogrinfo command on your database to get a list of spatial tables:

```
>ogrinfo counties.sqlite
INFO: Open of `counties.sqlite'
using driver `SQLite' successful.
1: mn_counties (Polygon)
```

Now use ogrinfo to get information on the structure of the spatial table:

```
>ogrinfo counties.sqlite county -summary
INFO: Open of `counties.sqlite'
using driver `SQLite' successful.

Layer name: mn_counties
Geometry: Polygon
Feature Count: 87
Extent: (189783.560000, 4816309.330000) - (761653.524114, 5472346.500000)
Layer SRS WKT:
PROJCS["NAD83 / UTM zone 15N",
  GEOGCS["NAD83",
    DATUM["North_American_Datum_1983",
      SPHEROID["GRS 1980",6378137,298.257222101,
        AUTHORITY["EPSG","7019"]],
      TOWGS84[0,0,0,0,0,0,0],
        AUTHORITY["EPSG","6269"]],
    PRIMEM["Greenwich",0,
      AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
      AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4269"]],
  UNIT["metre",1,
    AUTHORITY["EPSG","9001"]],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0],
  PARAMETER["central_meridian",-93],
  PARAMETER["scale_factor",0.9996],
  PARAMETER["false_easting",500000],
  PARAMETER["false_northing",0],
  AUTHORITY["EPSG","26915"],
  AXIS["Easting",EAST],
  AXIS["Northing",NORTH]]
FID Column = PK_UID
Geometry Column = Geometry
AREA: Real (0.0)
PERIMETER: Real (0.0)
COUNTY_ID: Integer (0.0)
FIPS: String (0.0)
...
```

Mapfile Example

Standard connection

```
LAYER
  NAME my_counties_layer
  TYPE POLYGON
```

```

CONNECTIONTYPE ogr
CONNECTION "counties.sqlite"
DATA "mn_counties"
STATUS ON
CLASS
  NAME "mncounties"
  STYLE
    COLOR 255 255 120
  END
END
END

```

Connection utilizing SQL syntax

```

LAYER
  NAME my_counties_layer
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "counties.sqlite"
  DATA "select geometry from mn_counties"
  STATUS ON
  CLASS
    NAME "mncounties"
    STYLE
      COLOR 255 255 120
    END
  END
END
END

```

Connection utilizing joined table for additional attributes

```

LAYER
  NAME my_counties_layer
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "counties.sqlite"
  DATA "SELECT mn.geometry, c.fips FROM mn_counties mn inner
        join county_data c on mn.county_id = c.county_id"
  STATUS ON
  CLASS
    NAME "mncounties"
    STYLE
      COLOR 255 255 120
    END
  END
END
END

```

Standard Connection with a filter

```

LAYER
  NAME my_counties_layer
  TYPE POLYGON
  CONNECTIONTYPE OGR
  CONNECTION "counties.sqlite"
  DATA "mn_counties"
  FILTER ('[fips]' = '27031')
  STATUS ON
  CLASS

```

```
NAME "mncounties"  
STYLE  
  COLOR 255 255 120  
END  
END  
END
```

Filter utilizing SQL syntax

```
LAYER  
  NAME my_counties_layer  
  TYPE POLYGON  
  CONNECTIONTYPE OGR  
  CONNECTION "counties.sqlite"  
  DATA "select geometry from mn_counties where fips = '27031"  
  STATUS ON  
  CLASS  
    NAME "mncounties"  
    STYLE  
      COLOR 255 255 120  
    END  
  END  
END
```

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)
FEDIRP: 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

```

Vector field rendering - UVraster

Vector fields are used for instance in meteorology to store/display wind direction and magnitude.

The source is two bands of raster data, the first band represents the U component of the vector, and the second band the V component. Using the u,v values at a given location we can compute a rotation and magnitude and use that to draw an arrow of a size proportional to the magnitude and pointing in the direction of the phenomenon (wind, current, etc.)

For more details about vector fields, refer to: [Vector field](#)

A vector field *LAYER* is a hybrid layer, which has a raster data source as input and vector features as output. The output features are represented as points. Queries are not supported.

Since the data source is a raster, all raster processing options can be used (e.g. RESAMPLE). RESAMPLE=AVERAGE generally gives a good result, and the default. This can be overridden by explicitly specifying the type of resampling.

Vector field layers are of *TYPE point*, and have *CONNECTIONTYPE uvraster*. The raster data set is specified in *DATA*. The two bands that define the vector field are specified using *PROCESSING BANDS* (U first, V second).

The UVraster connection type offers the following attributes:

- [u]: the raw u value
- [v]: the raw v value
- [uv_angle]: the vector angle
- [uv_minus_angle]: the vector angle - opposite direction
- [uv_length]: the vector length (scaled with the *UV_SIZE_SCALE* optional value)
- [uv_length_2]: half the vector length

Optional *PROCESSING* settings:

- *UV_SPACING*: The spacing is simply the distance, in pixels, between arrows to be displayed in the vector field. Default is 32.
- *UV_SIZE_SCALE*: The uv size scale is used to convert the vector lengths (magnitude) of the raster to pixels for a better rendering. Default is 1.

Example of a layer definition:

```

SYMBOL
  NAME "horizline"
  TYPE VECTOR
  POINTS
    0 0

```

```

    1 0
    END # points
END # symbol
SYMBOL
    NAME "arrowhead"
    TYPE vector
    FILLED true
    #ANCHORPOINT 0 0.5
    POINTS
        0 2
        4 1
        0 0
    END # points
END # symbol
SYMBOL
    NAME "arrowtail"
    TYPE vector
    FILLED true
    ANCHORPOINT 1 0.5 # to shift the arrowtail
    POINTS
        0 2
        4 1
        0 0
        -99 -99
        0 1
        4 1
    END # points
END # symbol
LAYER
    NAME "my_uv_test"
    TYPE POINT
    STATUS DEFAULT
    CONNECTIONTYPE uvraster
    DATA /path/wind.grib2
    PROCESSING "BANDS=1,2"
    PROCESSING "UV_SPACING=40"
    PROCESSING "UV_SIZE_SCALE=0.2"
    CLASS
        STYLE
            SYMBOL "horizline"
            ANGLE [uv_angle]
            SIZE [uv_length]
            WIDTH 3
            COLOR 100 255 0
        END # style
        STYLE
            SYMBOL "arrowhead"
            ANGLE [uv_angle]
            SIZE 10
            COLOR 255 0 0
            POLAROFFSET [uv_length_2] [uv_angle]
        END # style
        STYLE
            SYMBOL "arrowtail"
            ANGLE [uv_angle]
            SIZE 10
            COLOR 255 0 0
            POLAROFFSET [uv_length_2] [uv_minus_angle]
        END # style
    END # class
END # layer

```

```
END # style
END # class
END # layer
```

New in version 6.2: (rfc78)

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
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
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.

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://demo.mapserver.org/cgi-bin/wfs?SERVICE=WFS&VERSION=1.0.0&REQUEST=GetCapabilities
```

Example of the Resulting XML from GetCapabilities:

```
...
<FeatureTypeList>
  <Operations>
    <Query/>
  </Operations>
  <FeatureType>
    <Name>continents</Name>
    <Title>World continents</Title>
    <SRS>EPSG:4326</SRS>
    <LatLongBoundingBox minx="-180" miny="-90" maxx="180" maxy="83.6274"/>
  </FeatureType>
  <FeatureType>
    <Name>cities</Name>
    <Title>World cities</Title>
    <SRS>EPSG:4326</SRS>
    <LatLongBoundingBox minx="-178.167" miny="-54.8" maxx="179.383" maxy="78.9333"/>
  </FeatureType>
</FeatureTypeList>
...
```

Data Access / Connection Method

- WFS access is a core MapServer feature. MapServer currently supports WFS version 1.0.0.
- WFS access is also supported through OGR.

- CONNECTIONTYPE WFS or CONNECTIONTYPE OGR must be used (see the *OGR* documentation for details on how to use WFS through OGR).
- 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 continents. (see the GetCapabilities above for the possible layers available on this test server) . The URL is all one line, broken up here for readability (click [here](#) for a working link).

```
http://demo.mapserver.org/cgi-bin/wfs?
SERVICE=WFS&
VERSION=1.0.0&
REQUEST=getfeature&
TYPENAME=continents&
MAXFEATURES=100
```

Map File Example

```
LAYER
  NAME "continents"
  TYPE POLYGON
  STATUS ON
  CONNECTION "http://demo.mapserver.org/cgi-bin/wfs?"
  CONNECTIONTYPE WFS
  METADATA
    "wfs_typename"          "continents"
    "wfs_version"          "1.0.0"
    "wfs_connectiontimeout" "60"
    "wfs_maxfeatures"      "10"
  END
  PROJECTION
    "init=epsg:4326"
  END
  CLASS
    NAME "Continents"
    STYLE
      COLOR 255 128 128
      OUTLINECOLOR 96 96 96
    END
  END
END # Layer
```

7.1.2 Raster Data

Author Frank Warmerdam

Contact warmerdam at pobox.com

Revision \$Revision\$

Date \$Date\$

Last Updated 2013/07/04

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*

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.

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 considerably 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 values 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
```

Supported Formats

Since version 6.2, Mapserver raster input support is through the GDAL raster library only.

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 if GDAL support is built into a particular MapServer executable, use the `-v` flag 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 6.2.0 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG
SUPPORTS=PROJ SUPPORTS=GD SUPPORTS=AGG SUPPORTS=FREETYPE
SUPPORTS=CAIRO SUPPORTS=OPENGL 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 INPUT=JPEG INPUT=POSTGIS INPUT=OGR
INPUT=GDAL INPUT=SHAPEFILE
```

```
warmerda@gdal2200[18]% gdalinfo --formats
Supported Formats:
  VRT (rw+v): Virtual Raster
  GTiff (rw+vs): GeoTIFF
  NITF (rw+vs): National Imagery Transmission Format
  RPFTOC (rovs): Raster Product Format TOC format
  ECRGTOC (rovs): ECRG TOC format
  ...
```

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 Mapfile 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).

Starting with MapServer 6.4, the files can have different coordinate system (projection). This requires specifying the *TILESRS* keyword and generating the tileindex with a few additional options. See *Tileindexes with tiles in different projections*.

```
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 must be generated in the same coordinate system as the raster layer. If the files in the tileindex are not in the same projection as the raster layer, or are in heterogeneous projections, the `TILESRS` keyword must be specified in the `LAYER` definition. See *Tileindexes with tiles in different projections*
- 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

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.

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. Rendering of 24bit imagery is supported via the GDAL renderer.

If the output is still 8bit pseudo-colored (the `IMAGEMODE` is `PC256` in the associated `OUTPUTFORMAT` 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 substantial 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.

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.

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"
```

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"
```

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"
```

EXTENT_PRIORITY=WORLD Override GDAL with a world file.

Example:

```
PROCESSING "EXTENT_PRIORITY=WORLD"
```

LOAD_FULL_RES_IMAGE=YES/NO This option affects how image data is loaded for the resampler when reprojecting or otherwise going through complex resampling (as opposed to the fast default image decimation code path). This forces the source image to be loaded at full resolution if turned on (default is NO). This helps work around problems with default image resolution selection in when radical warping is being done. It can result in very slow processing if the source image is large.

LOAD_WHOLE_IMAGE=YES/NO This option affects how image data is loaded for the resampler (as above). This option, if turned on, will cause the whole source image to be loaded and helps make up for problem identifying the area required, usually due to radical image reprojection near a dateline or projection “horizon”. The default is NO. Turning this on can dramatically affect rendering performance and memory requirements.

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"
```

OVERSAMPLE_RATIO=double Default is 2.5. Rendering time will increase with increasing OVERSAMPLE_RATIO.

Example:

```
PROCESSING "OVERSAMPLE_RATIO=1.0"
```

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 rfc4.

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"
```

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"
```

WORLDFILE=<file> Specifies an alternative world file (for georeferencing). If a path only is specified, the base name of the dataset will be appended. The suffix (*.wld* / *.tfw* / ...) can be omitted.

Example:

```
PROCESSING "WORLDFILE=/path/"
or
PROCESSING "WORLDFILE=/path/file.wld"
or
PROCESSING "WORLDFILE=/path/file"
```

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 alot 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

    s = layer.getShape( result )
    for i in range(layer.numitems):
        print '%s: %s' % (layer.getItem(i), s.getValue(i))

layer.close()
```

This 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.

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 *MINSCALEDENOM*, and *MAXSCALEDENOM* 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 `msResampleGDALToMap` 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.

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 *MINSCALEDENOM* / *MAXSCALEDENOM* 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.

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 are 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.

Since the GDAL/OGR library is used for vector and raster access in MapServer, many more formats are supported, so please see the [OGR](#) (vector) and [GDAL](#) (raster) formats pages.

8.1 Output Generation

8.1.1 AGG Rendering Specifics

Author Thomas Bonfort

Contact thomas.bonfort at gmail

Revision \$Revision\$

Date \$Date\$

Last Updated 2008/11/24

Table of Contents

- *AGG Rendering Specifics*
 - *Introduction*
 - *Setting the OutputFormat*
 - *New Features*
 - *Modified Behavior*

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*.

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
```

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.

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).

8.1.2 AntiAliasing with MapServer

Author Pericles Nacionales

Contact naci0002 at umn.edu

Revision \$Revision\$

Date \$Date\$

Last Updated 2009/01/17

Warning: This document is outdated. Since version 6.0, MapServer will produce aliased output for the “gd” drivers, and antialiased output for the “agg/” and “cairo/” ones

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*

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
```

2. Add `TRANSPARENCY` to the `LAYER` object and set value to `ALPHA`

```
MAP
...
IMAGETYPE PNG24
...

LAYER
...
TRANSPARENCY ALPHA
...

END
END
```

3. 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 ELLIPSE—it supports antialiasing.

Here's an example of a real-world mapfile:

Note: From MapServer 6, symbol type CARTOLINE is no longer supported. You have to use AGG rendering and STYLE PATTERN to achieve dashed lines. Therefore, the following example does not work anymore.

```

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:

Note: From MapServer 6, symbol type CARTOLINE is not available. You have to use AGG rendering and STYLE PATTERN to achieve dashed lines. Therefore, the following symbol can not be used anymore.

```

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.

..index:: single: Dynamic charting

8.1.3 Dynamic Charting

Author Thomas Bonfort

Contact thomas.bonfort at gmail.com

Last Updated 2012/05/24

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 *Mapfile*s. You can also check out the *Vector Data Access Guide*, which has lots of examples of how to access specific data sources.

Setup

Supported Renderers

Dynamic charts are supported solely with the GD and *AGG* renderers.

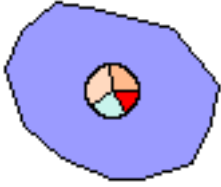
Attempting to add a chart layer with any other renderer (e.g. PDF or SVG) will result in undefined behavior. Rendering quality with the GD renderer is less than optimal, especially with small graphs, due to the lack of subpixel rendering functions.

Output from AGG and GD Renderers

MapServer AGG Rendering



MapServer GD Rendering



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

..index:: triple: PROCESSING; CHART_SIZE; LAYER

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
```

..index:: triple: PROCESSING; CHART_SIZE_RANGE; LAYER

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 exponent"
```

or just

```
PROCESSING "CHART_SIZE_RANGE = itemname"
```

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).
- exponent (optional) applies an exponential factor to the diameter, calculated with:

```
diameter=mindiameter +
  pow((attribute_value-minvalue)/(maxvalue-minvalue),1.0/exponent)*
  (maxdiameter-mindiameter);
```

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 [attribute]
      ...
    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 ;)).

..index:: triple: PROCESSING; CHART_TYPE; LAYER

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 [attribute]
  # 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

```

..**index::** single: Pie chart

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 [v1006]
14         COLOR 255 244 237
15     END
16  END
17  CLASS
18     NAME "Population Age 20-39"
19     STYLE
20         SIZE [v1007]
21         COLOR 255 217 191
22     END
23  END
24  CLASS
25     NAME "Population Age 40-59"
26     STYLE
27         SIZE [v1008]
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.

..**index::** single: Bar graph

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.

..index:: triple: PROCESSING; CHART_BAR_MINVAL; LAYER

..index:: triple: PROCESSING; CHART_BAR_MAXVAL; LAYER

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).

..index:: single: Stacked bar graph

Stacked bar Graphs

Stacked bar graphs can be drawn using:

```
PROCESSING "CHART_TYPE=VBAR"
```

8.1.4 Flash Output

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Yewondwossen Assefa

Contact assefa at dmsolutions.ca

Revision \$Revision\$

Date \$Date\$

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*

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.

Note: SWF is no longer supported in version 6.0.

Links to Flash-Related Information

- [Open Source Flash Viewer](#)
- [Flash maps demo](#)

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"
```

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.

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*

8.1.5 HTML Legends with MapServer

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2012-03-23

Table of Contents

- *HTML Legends with MapServer*
 - *Introduction*
 - * *Implementation*
 - * *Legend Object of Mapfile*
 - * *CGI [legend] tag*
 - * *HTML Legend Template File*
 - *Sample Site Using the HTML Legend*

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)

Deprecated since version 6.2.

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)

Deprecated since version 6.2.

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)

Deprecated since version 6.2.

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]`
- `[if name=layer_queryable value=...] ... [if]`
value is the queryability of the first layer in the group: 0=NOT QUERYABLE, 1=QUERYABLE
New in version 5.6.

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=AN-
NOTATION (deprecated since 6.2), 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]`
- `[if name=layer_queryable value=...] ... [if]`

value is the queryability of the layer: 0=NOT QUERYABLE, 1=QUERYABLE

New in version 5.6.

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 (deprecated since 6.2), 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]`
- `[if name=layer_queryable value=...] ... [/if]`
value is the queryability of the layer: 0=NOT QUERYABLE, 1=QUERYABLE
New in version 5.6.

Sample Site Using the HTML Legend

http://demo.mapserver.org/itasca_legend/

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]
```

8.1.6 HTML Imagemaps

Author David Fawcett

Contact david.fawcett at gmail.com

Last Updated 2008-10-08

Contents

- *HTML Imagemaps*
 - *Introduction*
 - *Mapfile Layer Definition*
 - *Templates*
 - *Request URL*
 - *Additional Notes*
 - *More Information*

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

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.

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/mypa
```

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]" >
```

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
```

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.

More Information

[Steve Lime's SHPXY Example](#)

8.1.7 OGR Output

Author Frank Warmerdam

Contact warmerdam at pobox.com

Last Updated 2011-11-15

Table of Contents

- *OGR Output*
 - *Introduction*
 - *OUTPUTFORMAT Declarations*
 - *LAYER Metadata*
 - *MAP / WEB Metadata*
 - *Geometry Types Supported*
 - *Attribute Field Definitions*
 - *Return Packaging*
 - *Test Suite Example*

Introduction

OGR output support was added to MapServer 6.0. It provides an output driver to produce feature style output suitable as a return result from WMS GetFeatureInfo or WFS GetFeature requests. OGR feature output depends on MapServer being built against the GDAL/OGR library. The OGR output driver should be enabled in MapServer 6.0 or newer when INPUT=OGR appears in the version string.

OUTPUTFORMAT Declarations

Details of OGR output formats allowed are controlled by an *OUTPUTFORMAT* declaration. The declarations define the OGR format driver to be used, creation options specific to that driver, and more general instructions to MapServer on how to package multi-file results and whether to try and build the result on disk or in memory.

Examples:

```

OUTPUTFORMAT
  NAME "CSV"
  DRIVER "OGR/CSV"
  MIMETYPE "text/csv"
  FORMATOPTION "LCO:GEOMETRY=AS_WKT"
  FORMATOPTION "STORAGE=memory"
  FORMATOPTION "FORM=simple"
  FORMATOPTION "FILENAME=result.csv"
END

OUTPUTFORMAT
  NAME "OGRGML"
  DRIVER "OGR/GML"
  FORMATOPTION "STORAGE=filesystem"
  FORMATOPTION "FORM=multipart"
  FORMATOPTION "FILENAME=result.gml"
END

OUTPUTFORMAT
  NAME "SHAPEZIP"
  DRIVER "OGR/ESRI Shapefile"
  FORMATOPTION "STORAGE=memory"
  FORMATOPTION "FORM=zip"
  FORMATOPTION "FILENAME=result.zip"
END

```

The OGR format driver to be used is determined by the name appearing after “OGR/” in the DRIVER argument. This name should match one of the formats listed as supported for the “-f” argument to ogr2ogr in the ogr2ogr usage message.

The IMAGEMODE for OGR output is FEATURE, but this is implicit and does not need to be explicitly stated for OGR output driver declarations.

The OGR renderer will support the following FORMATOPTION declarations:

DSCO:* Anything prefixed by DSCO: is used as a dataset creation option with the OGR driver. See the OGR web page for the particular format driver to see layer creation options available.

LCO:* Anything prefixed by LCO: is used as a layer creation option. See the OGR web page for the particular format driver to see layer creation options available.

FORM=simple/zip/multipart Indicates whether the result should be a simple single file (single), a mime multipart attachment (multipart) or a zip file (zip). “zip” is the default.

STORAGE=memory/filesystem/stream Indicates where the datasource should be stored while being written. “file” is the default.

If “memory” then it will be created in /vsimem/ - but this is only suitable for drivers supporting VSI*L which we can’t easily determine automatically.

If “filesystem”, then a directory for temporary files (specified using *WEB_TEMPPATH* or *MS_TEMPPATH*) will be used for writing and reading back the file(s) to stream to the client.

If “stream” then the datasource will be created with a name “/vsistdout” as an attempt to write directly to stdout. Only a few OGR drivers will work properly in this mode (ie. CSV, perhaps kml, gml).

FILENAME=name Provides a name for the datasource created, default is “result.dat”.

LAYER Metadata

The OGR output driver utilizes several items from the LAYER level METADATA object. Some of these were originally intended for GML output or are primarily intended to support WFS.

wfs_getfeature_formatlist (Optional) A comma delimited list of formats supported for WFS GetFeature responses. The OUTPUTFORMAT NAME values should be listed.

```
"wfs_getfeature_formatlist" "OGRGML,SHAPEZIP,CSV"
```

gml_include_items (Optional) A comma delimited list of items to include, or keyword “all”. You can enable full exposure by using the keyword “all”.

```
"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 (Optional) An alias for an attribute’s name. The resulting file will refer to this attribute by the alias. Here is an example:

```
"gml_province_alias" "prov"
```

gml_[item name]_type (Optional) Specifies the type of the attribute. Valid values are Integer|Real|Character|Date|Boolean.

gml_[item name]_width (Optional) Specifies the width of the indicated field for formats where this is significant, such as Shapefiles.

gml_[item name]_precision (Optional) Specifies the precision of the indicated field for formats where this is significant, such as Shapefiles. Precision is the number of decimal places, and is only needed for “Real” fields.

gml_types (Optional) If this field is “auto” then some input feature drivers (ie. OGR, and native shapefiles) will automatically populate the type, width and precision metadata for the layer based on the source file.

```
"gml_types" "auto"
```

ows/wfs_geomtype (Optional) Set the geometry type of OGR layers created from this MapServer LAYER. One of “Point”, “LineString”, “Polygon”, “MultiPoint”, “MultiLineString”, “MultiPolygon”, “GeometryCollection”, “Geometry”, or “None”. Most are fairly obvious, but “Geometry” can be used to represent a mix of geometry types, and “None” is sometimes suitable for layers without geometry. Note that layers which are a mix of polygon and multipolygon would normally have to be described as “Geometry”. To produce 2.5D output append “25D” to the geometry type (ie. “Polygon25D”). Note that Z values are only carried by MapServer if built with USE_POINT_Z_M support.

```
"ows_geomtype" "Polygon"
```

MAP / WEB Metadata

wms_feature_info_mime_type In order for WMS GetFeatureInfo to allow selection of OGR output formats, the mime type associated with the OUTPUTFORMAT must be listed in this metadata item.

```
"wms_feature_info_mime_type" "text/csv"
```

Geometry Types Supported

In MapServer we have POINT, LINE and POLYGON layers which also allow for features with multiple points, lines or polygons. However, in the OGC Simple Feature geometry model used by OGR a point and multipoint layer are quite distinct. Likewise for a LineString and MultiLineString and Polygon an MultiPolygon layer type.

To work around the mismatches between the MapServer and OGR geometry models, there is a mechanism to specify the geometry type to be used when exporting through OGR. This is the “wfs/ows_geomtype” metadata item on the layer. It may be one of one of “Point”, “LineString”, “Polygon”, “MultiPoint”, “MultiLineString”, “MultiPolygon”, “GeometryCollection”, “Geometry”, or “None”.

If this item is not specified, then “Point”, “LineString” or “Polygon” will be used depending on the TYPE of the LAYER. In cases of mixed geometry types (ie. polygons and multipolygons) the geometry type should be set to “Geometry” which means any geometry type.

```
"ows_geomtype" "Geometry"
```

In order 2.5D support (geometries with Z coordinates) to be enabled, the “25D” suffix must be added to the value of the “ows_geomtype” metadata item (i.e. “Polygon25D”), and MapServer must be built with USE_POINT_Z_M support.

Attribute Field Definitions

For OGR output it is highly desirable to be able to create the output fields with the appropriate datatype, width and precision to reflect the source feature definition.

It is possible to set the gml_[item]_type, gml_[item]_width and gml_[item]_precision metadata on the layer to provide detailed field definitions:

```
METADATA
  "gml_ID_type"          "Integer"
  "gml_ID_width"        "8"
  "gml_AREA_type"       "Real"
  "gml_AREA_width"      "15"
  "gml_AREA_precision"  "6"
  "gml_NAME_type"       "Character"
  "gml_NAME_width"      "64"
  ...
```

However, doing this manually is tedious and error prone. For that reason some feature sources (at least OGR, Shapefiles, POSTGIS and ORACLESPATIAL) support a mechanism to automatically populate this information from the source datastore. To accomplish this specify:

```
"gml_types"            "auto"
```

If no effort is made to set type, width and precision information for attribute fields, they will all be treated as variable length character fields when writing through OGR.

Return Packaging

One of the challenges returning generalized feature formats is that many such formats consists of multiple files which must be returned in the result. There are three approaches taken to this based on the FORM FORMATOPTION in the OUTPUTFORMAT declaration.

simple In this case a single result is returned. This is suitable for format drivers that produce a single file. The return result will have the mimetype listed in the OUTPUTFORMAT declaration. Note that if the OGR driver actually returns multiple files, only the primary one (the one with a name matching the filename passed into the OGR CreateDataSource call) will be returned. The return result will have a suggested filename based on the FILENAME FORMATOPTION.

multipart In this case all the files produced are returned as a multipart mime result. In this case the MIMETYPE of the OUTPUTFORMAT is ignored. All component files are returned with a mime type of “application/binary” and the whole package is “multipart/mixed”.

zip In this case all the files produced are bundled into one .zip file and this zip file is returned with a mimetype of “application/zip”. The OUTPUTFORMAT MIMETYPE is ignored.

One caveat with “zip” results is that this option is only available if the GDAL/OGR version is 1.8 or newer (or a 1.8 development later than approximately Oct 15, 2010). Earlier versions of GDAL/OGR lacked the zipping capability needed.

Test Suite Example

The MSAutoTest test suite contains a test case for use of OGR Output from WFS. The mapfile is at:

https://github.com/mapserver/msautotest/blob/master/wxs/wfs_ogr.map

The comments at the start of the file have a variety of sample requests that can be run against the map, as long as [MAP-FILE] is replaced with the mapfile name. They requests should be run against mapserv sitting in the msautotest/wxs directory.

8.1.8 PDF Output

Author Yewondwossen Assefa

Contact yassefa at dmsolutions.ca

Revision \$Revision\$

Date \$Date\$

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*

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.

Note: From version 6.0, PDF output is supported through Cairo. This is not reflected in the current documentation.

What is currently supported and not supported

1. Vector Layers

- Layer Point: supported
- Layer Line: supported
- Layer Polygon: supported
- Layer Circle : not supported

Note: Note: Dashed lines are supported with PDFlib version 6 or greater.

Note: Polygons filled with symbols are not supported.

2. 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.

3. WMS Layers

Not yet supported

4. Surround components

Legend, scalebar are not supported.

5. 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](#)).

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.pdflib.com/products/pdflib/>. 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.pdflib.com/products/pdflib/>
- open the project PDFlib.dsw in MS Visual C++
- build the project `pdflib_dll`
- after a successful build, you should have a `pdflib.lib` and `pdplib.dll` under the `pdflib` directory
- copy the `pdflib.dll` under your system directory (ex : `c:/winnt/system32`)
- the `pdflib.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.

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.

8.1.9 SVG

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision\$

Date \$Date\$

Last Updated 2005/12/13

Table of Contents

- *SVG*
 - *Introduction*
 - *Feature Types and SVG Support Status*
 - *Testing your SVG Output*
 - *goSVG*

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.

Note: From version 6.0, SVG output is supported through Cairo. This is not reflected in the current documentation.

Links to SVG-Related Information

- [SVG 1.1 specification](#)
- [SVG Discussion Paper](#)
- [G-XML Project Page](#)
- [SVG Tiny Profile](#)
- [MapFile Reference Doc](#)

Feature Types and SVG Support Status

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.

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-
<text x="43" y="121" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="34" y="205" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="164" y="258" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="316" y="190" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="334" y="258" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="249" y="230" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="241" y="242" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="223" y="260" font-family="fritqat-italic" font-size="8pt" fill="#ff0000" stroke="#ffffff" stroke-
<text x="210" y="279" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="82" y="234" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="40" y="223" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
<text x="214" y="125" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-
</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.

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/crs" 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=" 267,263 270,263 271,260 272,263 275,263 273,265 273,263 271,260 270,263 267,263" />
<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="fritgat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="43" y="121" font-family="fritgat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="34" y="205" font-family="fritgat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="164" y="258" font-family="fritgat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="316" y="190" font-family="fritgat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="334" y="258" font-family="fritgat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="249" y="230" font-family="fritgat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="241" y="242" font-family="fritgat" font-size="8pt" fill="#000000" stroke="ffffff" stroke-width="1" />
<text x="223" y="260" font-family="fritgat-italic" font-size="8pt" fill="ff0000" stroke="ffffff" stroke-width="1" />
<text x="210" y="279" font-family="fritgat" 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-  
<text x="40" y="223" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" stroke-  
<text x="214" y="125" font-family="fritqat" font-size="8pt" fill="#000000" stroke="#ffffff" strok  
</svg>
```

8.1.10 Tile Mode

Author Paul Ramsey

Contact pramsey at cleverelephant.ca

Revision \$Revision\$

Date \$Date\$

Last Updated 2008/04/30

Table of Contents

- *Tile Mode*
 - *Introduction*
 - *Configuration*
 - *Utilization*

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.

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*

As of MapServer 6.0, there are two extra parameters available for configuring tile mode.

- *tile_map_edge_buffer* renders the tile into a buffered rendering frame, then clips out the final tile. This will reduce edge effects when large symbols or wide lines are drawn. Recommended value: the size of the largest symbol or line width in your map file.
- *tile_metatile_level* renders the tiles into a fixed metatile, then clips out the final tile. This will reduce label repetition, at the expense of much higher rendering cost. Recommended value: 1 if you are doing labelling of large features in your layer. 0 otherwise.

If you use both *tile_map_edge_buffer* and *tile_metatile_level* at the same time, the buffer will be applied at the meta-tile level.

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 which 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                                     tileUrlTemplate: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>
```

8.1.11 Template-Driven Output

Author Chris Hodgson

Contact chodgson at refractions.net

Last Updated 2011-04-13

Table of Contents

- *Template-Driven Output*
 - *Introduction*
 - *OUTPUTFORMAT Declarations*
 - *Template Substitution Tags*
 - *Examples*

Introduction

RFC 36 added support for defining template-driven OUTPUTFORMATs for use with feature queries, including WMS GetFeatureInfo and WFS GetFeature. This allows for custom text-oriented output such as GeoJSON, KML, or XML. The templates are essentially the same as with the standard MapServer query *Templating*, however there are some additional tags to allow for template definition in a single file instead of the standard header/template/footer.

Note: There are other, simpler, ways to output some of these formats using MapServer. However, template-driven output provides maximal flexibility and customization of the output, at the cost of additional complexity and configuration.

Note: In order for template-driven output to work, layers that are to be output need to have the *TEMPLATE* key word included:

```
TEMPLATE "dummy"
```

Note: In order for template-driven output to work through WFS, the format needs to be listed in `wfs_getfeature_formatlist` in the *WEB METADATA* section or the *LAYER METADATA* section (the geojson format from the example below):

```
"wfs_getfeature_formatlist" "gml,geojson"
```

OUTPUTFORMAT Declarations

Details of template-driven output formats are controlled by an *OUTPUTFORMAT* declaration. The declarations define the template file to be used, as well as other standard OUTPUTFORMAT options.

Examples:

```
OUTPUTFORMAT
  NAME "kayml"
  DRIVER "TEMPLATE"
  MIMETYPE "application/vnd.google-earth.kml+xml"
  FORMATOPTION "FILE=myTemplate.kml"
  FORMATOPTION "ATTACHMENT=queryResults.kml"
END

OUTPUTFORMAT
  NAME "geojson"
  DRIVER "TEMPLATE"
  FORMATOPTION "FILE=myTemplate.js"
END

OUTPUTFORMAT
  NAME "customxml"
  DRIVER "TEMPLATE"
  FORMATOPTION "FILE=myTemplate.xml"
END
```

The template file to be used is determined by the “FILE=...” *FORMATOPTION*. The template filename is relative to the mapfile’s path. As is standard with MapServer template files, the file must contain the magic string ‘mapserver template’ in the first line of the file, usually within a comment, but this line is not output to the client.

Note: Valid suffixes for the template file are: .xml, .wml, .html, .htm, .svg, .kml, .gml, .js, .tmpl.

The MIMETYPE and FORMATOPTION “ATTACHMENT=...” parameters are very useful for controlling how a web browser handles the output file.

Template Substitution Tags

These tags only work in query result templates, and their purpose is primarily to simplify the templating to a single file for custom output formats.

[include src=“otherTemplate.txt”] Includes another template file; the path to the template file is relative to the map-file path.

Attributes:

- src: The file to be included.

[resultset layer=layername]...[/resultset] Defines the location of the results for a given layer.

Attributes:

- layer: The layer to be used
- no data: (optional) A string to return if no results are returned.

[feature]...[/feature] Defines the loop around the features returned for a given layer.

Attributes:

- limit: (optional) Specifies the maximum number of features to output for this layer.

- trimlast: (optional) Specifies a string to be trimmed off of the end of the final feature that is output. This is intended to allow for trailing record delimiters to be removed. See the examples below.

[join name=join1]...[/join] defines the loop around the features join from another layer.

See also:

Templating

Examples

This example shows how to emulate the old 3-file system using the new system, to compare the usage:

```
<!-- mapserver template -->
[include src="templates/header.html"]
[resultset layer=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 layer=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:

```
<!-- mapserver template -->
<?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" xmlns:gml="http://www.opengis.net/gml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://localhost/ms_ogc_workshop ./mums.xsd">
  <gml:description>This is a GML document which provides locations of
    all MapServer User Meeting that have taken place</gml:description>
  <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.

Could be called using ...&layer=mums&mode=nquery&qformat=geojson

Or by adding &outputformat=geojson to a WFS getfeature request:

```

// mapserver template
[resultset layer=mums]
{
  "type": "FeatureCollection",
  "features": [
    [feature trimlast=", "]
    {
      "type": "Feature",
      "id": "[myuniqueid]",
      "geometry": {
        "type": "PointLineString",
        "coordinates": [
          {
            "type": "Point",
            "coordinates": [[x], [y]]
          }
        ]
      },
      "properties": {
        "description": "[description]",
        "venue": "[venue]",
        "year": "[year]"
      }
    },
    [/feature]
  ]
}
[/resultset]

```

A more complicated KML example. Note the use of [shpxy] to support multipolygons with holes, and also that a point placemark is included with each feature using [shplabel]:

```

<!--MapServer Template-->
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2"
  xmlns:gx="http://www.google.com/kml/ext/2.2"
  xmlns:kml="http://www.opengis.net/kml/2.2"
  xmlns:atom="http://www.w3.org/2005/Atom">
<Document>
  <Style id="parks_highlight">

```

```

<IconStyle>
  <scale>1.4</scale>
  <Icon>
    <href>http://maps.google.com/mapfiles/kml/shapes/parks.png</href>
  </Icon>
  <hotSpot x="0.5" y="0" xunits="fraction" yunits="fraction"/>
</IconStyle>
<LineStyle>
  <color>ffff5500</color>
  <width>4.2</width>
</LineStyle>
<PolyStyle>
  <color>aaaaaaaa</color>
</PolyStyle>
<BalloonStyle>
  <text>
    <![CDATA[
      <p ALIGN="center"><b>${name}</b></p>
      ${description}
    ]]>
  </text>
</BalloonStyle>
</Style>
<Style id="parks_normal">
  <IconStyle>
    <scale>1.2</scale>
    <Icon>
      <href>http://maps.google.com/mapfiles/kml/shapes/parks.png</href>
    </Icon>
    <hotSpot x="0.5" y="0" xunits="fraction" yunits="fraction"/>
  </IconStyle>
  <LineStyle>
    <color>ffff5500</color>
    <width>4.2</width>
  </LineStyle>
  <PolyStyle>
    <color>ff7fff55</color>
  </PolyStyle>
  <BalloonStyle>
    <text>
      <![CDATA[
        <p ALIGN="center"><b>${name}</b></p>
        ${description}
      ]]>
    </text>
  </BalloonStyle>
</Style>
<StyleMap id="parks_map">
  <Pair>
    <key>normal</key>
    <styleUrl>#parks_normal</styleUrl>
  </Pair>
  <Pair>
    <key>highlight</key>
    <styleUrl>#parks_highlight</styleUrl>
  </Pair>
</StyleMap>
[resultset layer=parks]

```

```

<Folder>
  <name>Parks</name>
[feature trimlast="," limit=1]
  <Placemark>
    <name>[NAME]</name>
    <Snippet/>
    <description>
      <![CDATA[
        <p>Year Established: [YEAR_ESTABLISHED]</p>
        <p>Area: [AREA_KILOMETERS_SQUARED] sq km</p>
      ]]>
    </description>
    <styleUrl>#parks_map</styleUrl>
    <ExtendedData>
      <Data name="Year Established">[YEAR_ESTABLISHED]</Data>
      <Data name="Area">[AREA_KILOMETERS_SQUARED]</Data>
    </ExtendedData>
    <MultiGeometry>
      <Point>
        <coordinates>[shplabel proj=epsg:4326 precision=10],0</coordinates>
      </Point>
[shpxy
      ph="<Polygon><tessellate>1</tessellate>" pf="</Polygon>"
      xf="," xh=" " yh=" " yf=","0 "
      orh="<outerBoundaryIs><LinearRing><coordinates>"
      orf="</coordinates></LinearRing></outerBoundaryIs>"
      irh="<innerBoundaryIs><LinearRing><coordinates>"
      irf="</coordinates></LinearRing></innerBoundaryIs>"
      proj=epsg:4326 precision=10]
    </MultiGeometry>
  </Placemark>
[/feature]
</Folder>
[/resultset]
</Document>
</kml>

```

Warning: For templates (*Templating*), there are a number of reserved words. If you have want to expose an attribute with a name that is equal to a reserved word, you can not use the shorthand [attribute_name], but will have to use construct [item name=attribute_name] instead. For example, in a template, [id] is a system generated unique session id (see *Templating*). So if you have an attribute named “id” that you want to expose, you will either have to rename it or use the construct [item name=id].

8.1.12 Kml Output

Last Updated 2010/11/26

Authors Dvaid Kana (david.kana at gmail.com)

Authors Thomas.Bonfort (thomas.bonfort at gmail.com)

Authors Yewondwossen Assefa (yassefa at dmsolutions.ca)

Authors Michael Smith (michael.smith at usace.army.mil)

Version MapServer 6.0

Id \$

Introduction

This purpose of this document is to describe the KML/KMZ output support in MapServer 6.0.

The main goal of the KML driver is to generate KML output used mainly by Google Earth application.

General Functionality

Kml support is provided by using a kml or kmz image type in the map file. Output can then be generated using MapServer cgi (example mode=map) or through a WMS request.

Output format

The default name of the output format is kml or kmz, and this name can be used to set the imagetype parameter in the map file.

The format can also be defined in the map file:

```
OUTPUTFORMAT
  NAME kml
  DRIVER "KML"
  MIMETYPE "application/vnd.google-earth.kml+xml"
  IMAGEMODE RGB
  EXTENSION "kml"
  FORMATOPTION 'ATTACHMENT=gmap75.kml' #name of kml file returned
  FORMATOPTION "maxfeaturestodraw=100"
END

OUTPUTFORMAT
  NAME kmz
  DRIVER "KMZ"
  MIMETYPE "application/vnd.google-earth.kmz"
  IMAGEMODE RGB
  EXTENSION "kmz"
  FORMATOPTION 'ATTACHMENT=gmap75.kmz' #name of kmz file returned
END
```

Build

- On windows: there is a flag KML in nmake.opt
- On Linux: `-with-kml`
- AGG driver is necessary for the kml driver
- To be able to get kmz support, MapServer needs to be build against GDAL 1.8

Limiting the number of features

The number of vector features drawn by default is set to 1000 per layer. To control the number of features, users can set:

- layer level metadata that only applies to the layer: `"maxfeaturestodraw" "100"`
- map level metadata that applies to all layers: `"maxfeaturestodraw" "100"`
- output format option that applies to all layers: `FORMATOPTION "maxfeaturestodraw=100"`

Map

In terms for Kml object, the MapServer KML output will produce a <Document> element to include all the layers that are part of the MapServer map file. Features supported for the Document are:

Document element	Supported	MapServer equivalence/Notes
name	Yes	Name in the map file
visibility	No	Can be supported if needed. Default is 1
open	No	Can be supported if needed. Default is 0
address	No	Could be supported for example using ows_address if available
AdressDe-tails	No	
pho-neNumber	No	Could be supported using ows_contactvoicetelephone is available
Snippet	No	
description	No	
Ab-stractView	No	
TimePrim-itive	No	
styleURL	No	
StyleSelec-tor	Yes	Style element will be supported. All different styles from the layers will be stored here and referenced from the folders using a styleUrl. In addition to the Styles related to features, there is a ListStyle element added at the document level. This allows to control the way folders are presented. See Layers section (styleUrl) setting for more details.
Region	No	
Metadata	No	
Extended-Data	No	

Layers

Each layer of the MapServer map file will be inside a Kml <Folder> element. Supported Folder elements are:

Folder element	Supported	MapServer equivalence/Notes
name	Yes	Name of the layer. If not available the name will be Layer concatenated with the layer's index (Layer1)
visibility	Yes	Always set to 1
open	No	Default is 0
atom:authoratom:linkaddressAddressDetailsphoneNumberSnippet	No	
description	No	Could be supported using ows_description
AbstarctView	No	
TimePrimitive	No	
styleUrl	Yes	The user can use the kml_folder_display layer or map level metedata to choose a setting. Possible values are 'check' (default), 'radioFolder', 'checkOffOnly', 'checkHideChildren'.
RegionMetadataExtended-Data	No	

Each element in the Layer will be inside a Kml <Placemark> element. As described in the Kml reference : "A Placemark is a Feature with associated Geometry. In Google Earth, a Placemark appears as a list item in the Places panel. A Placemark with a Point has an icon associated with it that marks a point on the Earth in the 3D viewer. (In

the Google Earth 3D viewer, a Point Placemark is the only object you can click or roll over. Other Geometry objects do not have an icon in the 3D viewer. To give the user something to click in the 3D viewer, you would need to create a MultiGeometry object that contains both a Point and the other Geometry object.)”

For Polygon and Line layers, when a feature is associated with a label, a MultiGeometry element containing a point geometry and the geometry of the feature is created. The point feature will be located in the middle of the polygon or line

```
<Folder>
  <name>park</name>
  <visibility>1</visibility>
  <styleUrl>#LayerFolder_check</styleUrl>
  <Placemark>
    <name>Ellesmere Island National Park Reserve</name>
    <styleUrl>#style_line_ff787878_w4.0_polygon_ff00ffc8_label_ff0000ff</styleUrl>
    <MultiGeometry>
      <Polygon>
        <outerBoundaryIs>
          <LinearRing>
            <coordinates>
              ...
            </coordinates>
          </LinearRing>
        </outerBoundaryIs>
        <Point>
          <coordinates>
            -70.86810858,82.12291871
          </coordinates>
        </Point>
      </MultiGeometry>
    </Placemark>
  </Folder>
```

Supported Features in the Placemark element are:

Place-mark element	Supported	MapServer equivalence/Notes
name	Yes	Label attached with the feature. If there is no label a default name is assigned using the layer name and the shape id (ex. park.1)
visibility	No	Is is by default set to true
open	No	
address	No	
Address-Details	No	
phoneNumber	No	
Snippet	No	This is a short description the feature. If needed It could be supported.
description	Yes	This information is what appears in the description balloon when the user clicks on the feature. The <description> element supports plain text as well as a subset of HTML formatting elements. Used when KML/OWS_DESCRIPTION is defined
AbstractView	No	
TimePrimitive	No	
styleUrl	Yes	Refers to a Style defined in the Document
StyleSelector	No	
Region	No	
Metadata	No	
Extended-Data	Yes	Used when KML/OWS_INCLUDE_ITEMS is defined
Geometry	Yes	Depends on the layer type

General notes on layers

- Labelcache is turned off on each layer
- Projection block should be set. If not set It will be assumed that the data is in lat/lon projection (a debug message will be sent to the user: Debug Message 1)
- Possible to output vector layers as raster using metadata: “KML_OUTPUTASRASTER” “true”
- The user can use the KML_FOLDER_DSIPLAY layer or map level metedata to choose a setting. Possible values are ‘check’ (default), ‘radioFolder’, ‘checkOffOnly’, ‘checkHideChildren’.
- The user can use metadata KML/OWS_DESCRIPTION or KML/OWS_INCLUDE_ITEMS to define the description attached to each feature. If KML/OWS_DESCRIPTION are defined, the <description> tag of the Placemark will be used. If KML/OWS_INCLUDE_ITEMS, the <ExtendedData> tag will be used.
- The user can use the metadata KML_NAME_ITEM to indicate the field name to be used a a name tag for each feature.
- The user can use metadata KML_ALTITUDEMODE to specify how altitude components in the <coordinates> element are interpreted. Possible values are: absolute, relativeToGround, clampToGround. <http://code.google.com/apis/kml/documentation/kmlreference.html#altitudemode>
- The user can use metedata KML_EXTRUDE to specify whether to connect the LinearRing to the ground. <http://code.google.com/apis/kml/documentation/kmlreference.html#tessellate>

- The user can use metadata KML_TESSELLATE to specify whether to allow the LinearRing to follow the terrain. <http://code.google.com/apis/kml/documentation/kmlreference.html#extrude>
- The user can specify an attribute to be used as the elevation value using a layer level metadata: 'kml_elevation_attribute' '<Name of attribute>'. The value will be used as the z value when the coordinates are written.

Point Layers

- Each layer will be inside a Folder element.
- Each feature will be represented by a Placemark.
- The Geometry element for a Point layer would be represented as a Point geometry element in Kml. Supported elements are:

Line Layers

- Each layer will be inside a Folder element.
- Each feature in the layer would be represented by a Placemark.
- If a label is attached to the line, the Geometry element would be represented as a MultiGeometry that includes a LineString element and a Point element representing the position of the label. If no label is attached to a feature, the Geometry element will be a LineString.

Polygon Layers

- Each layer will be inside a Folder element.
- Each feature will be represented by a Placemark.
- If a label is attached to the polygon, the Geometry element would be represented as a MultiGeometry that includes a Polygon element and a Point element representing the position of the label.

Raster Layers

- Each layer will be inside a Folder element.
- A GroundOverlay feature is created for the layer, that includes an href link to the raster image generated and LatLongBox for the extents (map extents).
- The href is generated using the imagepath and imageurl settings in the map file.

Styling

As described in Section 4, all different styles from the layers will be stored at the Document level and referenced from the folders using a styleUrl.

Point Layers

Point layers will be styled using the `IconStyle` styling element of kml. An image representing the symbol will be created and referenced from the `IconStyle` object. If a label is attached to the point, a `LabelStyle` element will also be used. The `LabelStyle` will have the `color` parameter set.

```
<Style id="style_label_ff0000ff_symbol_star_13.0_ff000000">
  <IconStyle>
    <Icon>
      <href>>http://localhost/ms_tmp/4beab862_19bc_0.png</href>
    </Icon>
  </IconStyle>
  <LabelStyle>
    <color>ff0000ff</color>
  </LabelStyle>
</Style>
```

Line Layers

Line layers will be styled using the `LineStyle` styling element of kml. Color and width parameters of the `LineStyle` will be used. If a label is attached to the layer, a `LabelStyle` element will also be used.

Polygon Layers

Polygon layers will be styled using the `PolyStyle` styling element of kml. Color parameter of the `PolyStyle` will be used. If an outline is defined in the map file, an additional `LineStyle` will be used. If a label is attached to the layer, a `LabelStyle` element will also be used.

Attributes

As described in section on Layers, two ways of defining the description:

- `kml/ows_description`
- `kml/ows_include_items`

Coordinate system

The map level projection should be set to `epsg:4326`. If not set, the driver will automatically set it. Layers are expected to have projection block if their projection is different from `epsg:4326`.

Warning and Error Messages

- When the projection of the map file is not set or is different from a lat/lon projection, the driver automatically sets the projection to `epsg:4326`. If the map is in debug mode, the following message is sent: “KmlRenderer::checkProjection: Mapfile projection set to `epsg:4326`”
- If `imagepath` and `imageurl` are not set in the web object, the following message is sent in debug mode: “KmlRenderer::startNewLayer: `imagepath` and `imageurl` should be set in the web object”

9.1 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.

9.1.1 MapServer OGC Specification support

- Web Map Service (OGC:WMS)
 - Server: 1.0.0, 1.0.7, 1.1.0, 1.1.1, 1.3.0
 - Client: 1.0.0, 1.0.7, 1.1.0, 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, 2.0.0, 2.0.1
- Geography Markup Language (OGC:GML) 2.1.2, 3.1.0 Level 0 Profile, 3.2.1
- GML Application Schema - Coverages (OGC:GMLCOV) 1.0.0, 1.0.1
- 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, 2.0.0

9.1.2 WMS Server

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2013-06-20

Table of Contents

- *WMS Server*
 - *Introduction*
 - *Setting Up a WMS Server Using MapServer*
 - *Changing the Online Resource URL*
 - *WMS 1.3.0 Support*
 - *Reference Section*
 - *FAQ / Common Problems*

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 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 (see *Templating*) specified in the *CLASS* TEMPLATE parameter (the filename has to have an .html extension). 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.

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 6.3-dev OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=KML
SUPPORTS=PROJ SUPPORTS=GD SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=CAIRO
SUPPORTS=ICONV SUPPORTS=FRIBIDI SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=GEOS INPUT=JPEG INPUT=POSTGIS INPUT=OGR
INPUT=GDAL INPUT=SHAPEFILE
```

Example 2. On Windows:

```
C:\apache\cgi-bin> mapserv -v
MapServer version 6.3-dev OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=KML
SUPPORTS=PROJ SUPPORTS=GD SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=CAIRO
SUPPORTS=ICONV SUPPORTS=FRIBIDI SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=GEOS INPUT=JPEG INPUT=POSTGIS INPUT=OGR
INPUT=GDAL INPUT=SHAPEFILE
```

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:...”)
 - wms_enable_request

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 - see *Templating*)

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 [The EPSG web page](#) 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. BoundingBox is a mandatory element of WMS capabilities for WMS 1.3.0 (for WMS 1.1.0 it is optional, 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 advertize 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. BoundingBox is a mandatory element of WMS capabilities for WMS 1.3.0 (for WMS 1.1.0 it is optional, 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 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.

- **"wms_enable_request" metadata:**

Specify which requests to enable. If not specified, no requests will be enabled! See the explanation below.

- **Configuring for GetFeatureInfo Requests:**

You must set the layer `TEMPLATE` parameter for the layer to be queryable by `GetFeatureInfo` requests (see [Templating](#)). For requests of type "text/html" you should also set the layer `HEADER` and `FOOTER` parameters.

As of MapServer 4.6 you must set the `gml_*` metadata for the layer attributes to be served (see the Layer Object metadata in the Reference Section later in this document). To include geometry, `gml_geometries` and `gml_[geometry name]_type` has to be specified.

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://demo.mapserver.org/cgi-bin/wms?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. Set “STYLES=” with an empty value to use default style(s). Named styles are also supported and are controlled by CLASS GROUP names in the mapfile.

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=prov_bound&STYLES=&SRS=EPSG:4326
&BBOX=-173.537,35.8775,-11.9603,83.8009&WIDTH=400&HEIGHT=300
&FORMAT=image/png
```

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.

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.

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. If the requested layername is a GROUP-name, all included layers will be returned in the legend-icon.

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>class1</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).
- **SLD_VERSION** - (Optional) The SLD version.
- **SCALE** - (Optional) Specify a scale so that only layers that fall into that scale will have a legend.
- **STYLE** - (Optional) The style.
- **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
```

Changing 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:

Apache ReWrite rules (using Apache mod_rewrite)

One can use Apache’s mod_rewrite to avoid specifying the map, or any other default parameter in the mapserver URL. This task consist of three steps, specifying the mod_rewrite module to be loaded, enabling the mod_rewrite module for the selected directories and at last to write a .htaccess file to do the rewriting.

In the httpd.conf file, the mod_rewrite module is per default disabled. To enable it, remove the opening # in the line

```
#LoadModule rewrite_module modules/mod_rewrite.so
```

To be able to use the module, it must be enabled, using the directive AllowOverride. This can be done per server or per directory. If you just have one server, add an “AllowOverride All” line in the httpd.conf file (see the Apache documentation to be sure about the security implications of this). Per directory is the easiest way to make it work on virtual hosts. Within the <virtualHost> section of the httpd.conf insert:

```
<Directory myhtdocsdire>
  AllowOverride All
</Directory>
```

Where myhtdocsdire is the directory defined as documentroot for the actual virtual server.

When the directives are set to load and enable the mod_rewrite module, Apache has to be restarted.

In a web-accessible directory make a .htaccess file like the following:

```
RewriteEngine on
RewriteRule wmsmap?(.*) /cgi-bin/mapserv?map=/home/www/mapserverstuff/mymapfile.map&$1
```

The rewriteRule says: given a webpage starting with wmsmap, pick out the query parameters, make a new page request starting with /cgi-bin/mapserv?map=(...)? and add on whatever was the query parameters in the original page request.

e.g, the URL wmsmap?mode=map will be rewritten as

```
:: /cgi-bin/mapserv?map=/home/www/mapserverstuff/mymapfile.map&mode=map
```

If just the URL wmsmap is given (without any parameters) a page not found error will show up as that does not match the wmsmap? expression.

Apache environment variables - MS_MAPFILE

A default mapfile can be specified using the MS_MAPFILE environment variable:

```
Alias /mywms /usr/lib/cgi-bin/mapserver
<Location /mywms>
  SetHandler cgi-script
  Options ExecCGI
  SetEnv MS_MAPFILE /path/to/mymapfile.map
</Location>
```

Apache SetEnvif

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
```

ASP script (IIS - Microsoft Windows)

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&" & strRequest

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
%>
```

Mapscript wrapper

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)
```

Wrapper script (Unix)

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
```

Note: Using a `/bin/sh` wrapper script causes an overhead on system resources as two processes have to be spawned instead of one, and is therefore not recommended.

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. MapServer 5.x 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.

MapServer 6.0 and up holds a list of epsg codes with inverted axis order. It is currently based on EPSG database version 7.6. It is also possible to define the axis order at build time for a specific EPSG code(see #3582). This allows for example to use the “normal” axis order for some of 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&... (example request)

Users can also use the EPSG:4326 coordinates and use the axis ordering of lat/long:

- ...&EPSG:4326&BBOX=-90.0,-180.0,90,180.0&... (example request)

Other notable changes

- valid values for the EXCEPTIONS parameter in a GetMap request are XML, INIMAGE, BLANK
- valid value for the EXCEPTIONS parameter in a GetFeatureInfo request is XML
- LayerLimit is introduced, allowing a server to advertise and limit the number of layers a client is allowed to include in a GetMap request

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.

OGC 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.

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_*' 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.)

Web Object Metadata

ows_allowed_ip_list (or wms_allowed_ip_list)

- *Description:* (Optional) A list of IP addresses that will be allowed access to the service.

Example:

```
METADATA
  "ows_allowed_ip_list" "123.45.67.89 11.22.33.44"
END
```

ows_denied_ip_list (or wms_denied_ip_list)

- *Description:* (Optional) A list of IP addresses that will be denied access to the service.

Example:

```
METADATA
  "ows_denied_ip_list" "123.45.67.89 11.22.33.44"
END
```

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_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.opengis.net>.

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_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.

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_bbox_extended:

- *Description:* (Optional) “true” or “false”. If true, bounding boxes are reported for all supported SRS / CRS in the capabilities document. If false, only the bounding box of the first SRS / CRS is reported.
- Introduced in 6.0.

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_enable_request (or **ows_enable_request**)

- *Description:* Space separated list of requests to enable. The default is none. The following requests can be enabled: *GetCapabilities*, *GetMap*, *GetFeatureInfo* and *GetLegendGraphic*. A "!" in front of a request will disable the request. "*" enables all requests.

- *Examples:*

To enable only *GetMap* and *GetFeatureInfo*:

```
"wms_enable_request" "GetMap GetFeatureInfo"
```

To enable all requests except *GetFeatureInfo*

```
"wms_enable_request" "* !GetFeatureInfo"
```

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 (see *Templating*). The client can then call the server with `INFO_FORMAT=text/html`.

 - If not specified, MapServer by default has text/plain and GML implemented.

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_getcapabilities_version

- *Description:* (Optional) Default version to use for *GetCapabilities* requests that do not have a version parameter. If not set, the latest supported version will be returned.

wms_getlegendgraphic_formatlist

- *Description:* (Optional) A comma-separated list of valid formats for a WMS *GetLegendGraphic* request.

wms_getmap_formatlist

- *Description:* (Optional) A comma-separated list of valid formats for a WMS *GetMap* request.

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_keywordlist_vocabulary

- *WMS Attribute Name:* vocabulary of KeywordList -> Keyword
- *Description:* (Optional) Name of vocabulary used in **wms_keywordlist_[vocabulary's name]_items** as described below.

wms_keywordlist_[vocabulary's name]_items

- *WMS TAG Name:* KeywordList -> Keyword
- *Description:* (Optional) A comma-separated list of keywords or keyword phrases to help catalog searching for given vocabulary.

wms_languages

- *Description:* (Optional) A comma-separated list of supported languages. For details please refer to the section *Multi-language support for certain capabilities fields* in the INSPIRE View Service documentation.

wms_layerlimit

- *WMS TAG Name:* LayerLimit (WMS1.3.0, sect. 7.2.4.3)
- *Description:* (Optional) The maximum number of layers a WMS client can specify in a GetMap request. If not set, then no limit is imposed.

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_remote_sld_max_bytes

- *Description:* (Optional) Maximum size in bytes authorized when fetching a remote SLD through http. Defaults to 1 MegaByte (1048596).

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_rootlayer_abstract

- *WMS TAG Name:* Abstract (WMS1.1.1, sect. 7.1.4.2)
- *Description:* (Optional) Same as wms_abstract, applied to the root Layer element. If not set, then wms_abstract will be used.

wms_rootlayer_keywordlist

- *WMS TAG Name:* KeywordList (WMS1.1.1, sect. 7.1.4.2)
- *Description:* (Optional) Same as wms_keywordlist, applied to the root Layer element. If not set, then wms_keywordlist will be used.

wms_rootlayer_title

- *WMS TAG Name:* Title (WMS1.1.1, sect. 7.1.4.1)

- *Description:* (Optional) Same as `wms_title`, applied to the root Layer element. If not set, then `wms_title` will be used.

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_geometries

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) Provides a name for geometry elements. The value is specified as a string to be used for geometry element names. By default, GML geometries are not written in GML GetFeatureInfo output, unless `gml_geometries` and `gml_[geometry name]_type` are both set. By default, only the bounding box is written. If `gml_geometries` is set to “none”, neither the bounding box nor the geometry are written.

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) Specifies the type of the attribute. Valid values are the OGR data types: Integer|Real|Character|Date|Boolean. Mapserver translates these to valid GML data types.

gml_[geometry name]_type

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) 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_[geometry name]_type, where [geometry name] is the string value specified for gml_geometries, and a value which is one of:

- point
- multipoint
- line
- multiline
- polygon
- multipolygon

gml_xml_items

- *Description:* (Optional, applies only to GetFeatureInfo GML requests) A comma delimited list of items that should not be XML-encoded.

ows_allowed_ip_list Same as ows_allowed_ip_list in the Web Object.

ows_denied_ip_list Same as ows_denied_ip_list in the Web Object.

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_authorityurl_name, wms_authorityurl_href

- *Description:* (Optional) AuthorityURL is used in tandem with Identifier values to provide a means of linking identifier information back to a web service. The `wms_identifier_authority` should provide a string that matches a declared `wms_authorityurl_name`. Both `wms_authorityurl_name` and `wms_authorityurl_href` must be present for an AuthorityURL tag to be written to the capabilities.
- refer to section 7.1.4.5.12 of the WMS 1.1.1 spec.

wms_bbox_extended:

- *Description:* (Optional) “true” or “false”. If true, bounding boxes are reported for all supported SRS / CRS in the capabilities document. If false, only the bounding box of the first SRS / CRS is reported.
- Introduced in 6.0.

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_enable_request (or ows_enable_request)

- *Description:* Space separated list of requests to enable. The default is none. The following requests can be enabled: *GetCapabilities*, *GetMap*, *GetFeatureInfo* and *GetLegendGraphic*. A "!" in front of a request will disable the request. "*" enables all requests.
- *Examples:*

To enable only *GetMap* and *GetFeatureInfo*:

```
"wms_enable_request" "GetMap GetFeatureInfo"
```

To enable all requests except *GetFeatureInfo*

```
"wms_enable_request" "* !GetFeatureInfo"
```

wms_exclude_items

- *Description:* (Optional, applies only to *GetFeatureInfo* text/plain requests) A comma delimited list of items to exclude, or keyword "all".

See *gml_exclude_items* above.

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_getfeatureinfo_formatlist

- *Description:* (Optional) Comma-separated list of formats that could be valid for a *GetFeatureInfo* request. If defined, only these formats are advertised through in the *Capabilities* document.

wms_getlegendgraphic_formatlist

- *Description:* (Optional) Comma-separated list of image formats that could be valid for a *GetLegendGraphic* request. If defined, only these formats are advertised through in the *Capabilities* document.

wms_getmap_formatlist

- *Description:* (Optional) Comma-separated list of image formats that could be valid for a *GetMap* request. If defined, only these formats are advertised through in the *Capabilities* document.

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_identifier_authority, wms_identifier_value

- *Description:* (Optional) Identifier is used in tandem with AuthorityURL end points to provide a means of linking identifier information back to a web service. The wms_identifier_authority should provide a string that matches a declared wms_authorityurl_name. Both wms_identifier_authority and wms_identifier_value must be present for an Identifier tag to be written to the capabilities.
- refer to section 7.1.4.5.12 of the WMS 1.1.1 spec.

wms_include_items

- *Description:* (Optional, applies only to GetFeatureInfo text/plain requests) A comma delimited list of items to include, or keyword “all”.
See gml_include_items above.

wms_keywordlist Same as wms_keywordlist in the Web Object.

wms_keywordlist_vocabulary Same as wms_keywordlist_vocabulary in the Web Object.

wms_keywordlist_[vocabulary’s name]_items Same as wms_keywordlist_[vocabulary’s name]_items 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 necessarily publish the name of the group in the capabilities, only the title is always published. If a layer with the same name as used in WMS_LAYER_GROUP is found it is treated as named group and if no layer with this name is found as unnamed group.

As a consequence the groups set with WMS_LAYER_GROUP can not always 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_metadataurl_format

- *Description:* (Optional) The file format MIME type of the metadata record (e.g. “text/plain”). The layer metadata wms_metadataurl_type and wms_metadataurl_href must also be specified.
- refer to section 7.1.4.5.10 of the WMS 1.1.1 spec.

wms_metadataurl_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_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_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_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`, `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.

Vendor specific WMS parameters**angle**

- Angle (in degrees) to rotate the map.

Note: The angle value is in degrees clockwise.

radius

- This parameter accepts two types of input:
 - An integer that specifies the search radius in pixels.
 - The special value *bbox* that will change the query into a bbox query based on the bbox given in the request parameters.

bbox_pixel_is_point

- If this parameter is "TRUE", MapServer will treat the BBOX received in WMS GetMap requests as if it was provided in pixel_is_point mode. Essentially disabling the conversion from pixel_is_area (WMS model) to pixel_is_point that is present in mapwms.c for that specific mapfile.

Cascading WMS Requests

Currently, there are 3 requests that support WMS cascading:

- GetMap
- GetFeatureInfo
- GetLegendGraphic

Before MapServer 6.2, a GetLegendGraphic request was not cascaded. A legend was returned using the layer classes. To preserve that behavior, a GetLegendGraphic request will be cascaded only **if**:

1. The GetLegendGraphic request is enabled via the *_enable_request* metadata.
2. The layer does not contain any class with the name property set. ie:

```
CLASS
  NAME "Parks"
  STYLE
    COLOR 0 255 0
```

```
END
END
```

This layer won't be cascaded because it contains at least a class with the property NAME set.

Note: If you know that the remote WMS server *does not* support a given WMS request, you should disable this request explicitly for your layer using the (ows/wms)_enable_request metadata. Otherwise, you will simply get the XML exception from the cascaded server.

Sample WMS Server Mapfile

The following is a very basic WMS Server mapfile:

```

1  MAP
2  NAME "WMS-test"
3  STATUS ON
4  SIZE 400 300
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  "wms_enable_request" "*" ##necessary
19  END
20  END # Web
21
22  PROJECTION
23  "init=epsg:42304" ##required
24  END
25
26  SYMBOL
27  NAME "circle"
28  TYPE ellipse
29  POINTS 1 1 END
30  END # Symbol
31
32  #
33  # Start of layer definitions
34  #
35
36  LAYER
37  NAME "park"
38  METADATA
39  "wms_title" "Parks" ##required
40  END
41  TYPE POLYGON
42  STATUS OFF
43  DATA park

```

```

44 PROJECTION
45     "init=epsg:42304"    ##recommended
46 END
47 CLASS
48     NAME "Parks"
49     STYLE
50         COLOR 200 255 0
51         OUTLINECOLOR 120 120 120
52     END # Style
53 END # Class
54 END # Layer
55
56 LAYER
57     NAME popplplace
58     METADATA
59         "wms_title"      "Cities"    ##required
60     END
61     TYPE POINT
62     STATUS ON
63     DATA popplplace
64     PROJECTION
65         "init=epsg:42304"    ##recommended
66     END
67     CLASS
68         NAME "Cities"
69         STYLE
70             SYMBOL "circle"
71             SIZE 8
72             COLOR 0 0 0
73         END # Style
74     END # Class
75 END # Layer
76
77 END # Map File

```

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"
  "wms_enable_request" "*"    ##necessary
END # METADATA
END # WEB
```

The above server advertises the first two auto projections.

9.1.3 INSPIRE View Service

Author Stephan Meissl

Contact stephan.meissl at eox.at

Last Updated 2012-03-19

Table of Contents

- *INSPIRE View Service*
 - *Introduction*
 - *Activation of INSPIRE support*
 - *Multi-language support for certain capabilities fields*
 - *Provision of INSPIRE specific metadata*
 - *Named group layers*
 - *Style section for root layer and possibly existing group layers*

Introduction

INSPIRE is the name of an [European directive](#), establishing an infrastructure for spatial information in Europe to support Community environmental policies, and policies or activities which may have an impact on the environment.

The INSPIRE View Service is an implementation of the [INSPIRE Technical Guidance](#) document on top of the [WMS Server](#) implementation explained in the previous chapter.

In order to achieve INSPIRE View Service compliance, the following enhancements have been implemented in MapServer:

- Activation of INSPIRE support (two scenarios)
- Multi-language support for certain capabilities fields
- Provision of INSPIRE specific metadata
- Named group layers
- Style section for root layer and possibly existing group layers

Activation of INSPIRE support

INSPIRE specific metadata can either be referenced in an external INSPIRE service metadata document (scenario 1) or can be directly embedded in the capabilities document (scenario 2). MapServer supports both scenarios.

Activation of the corresponding scenario for INSPIRE support takes place in the `WEB.METADATA` section of the mapfile through `wms_inspire_capabilities`. If activated, the corresponding INSPIRE namespace as well as appropriate validation warnings are generated in the capabilities document.

Scenario 1 - activate INSPIRE support using a reference to external service metadata:

```
WEB
METADATA
  "wms_inspire_capabilities" "url"
  ...
END
END
```

Scenario 2 - activate INSPIRE support using embedded service metadata:

```
WEB
METADATA
  "wms_inspire_capabilities" "embed"
  ...
END
END
```

Multi-language support for certain capabilities fields

INSPIRE requires multi-language support and requests a list of all supported languages as well as the default language in the capabilities document. Based on the language parameter in the GetCapabilities request, certain specific metadata values, namely

- `wms_title`
- `wms_abstract`
- `wms_rootlayer_title`
- `wms_rootlayer_abstract`
- `wms_group_title`
- `wms_group_abstract`
- `wms_style_title`

- `wms_style_<name>_title`

as well as language dependent reference data like

- `DATA "road_eng"`
- `CONNECTION "db_ger"`

need to be provided in the requested language. If the language is not supported (or no language parameter is present), the default language has to be used.

All supported languages have to be specified as comma separated list (first language is default) through `wms_languages` in the `WEB.METADATA` section of the mapfile. This language parameter is also added to the `OnlineResource` in the `GetCapabilities` output:

```
WEB
METADATA
...
"wms_languages" "eng,ger"          #first default, values according ISO 639-2/B
...
END
END
```

For language specific metadata values, a key extension method is applied:

```
WEB
METADATA
...
"wms_title.eng" "myservicetitle"
"wms_title.ger" "myservicetitleger"
"wms_abstract" "mylayerabstract"    #fallback
"wms_abstract.ger" "mylayerabstractger"
...
END
END
```

For language dependent reference data, a similar approach like the *run-time substitution* feature of MapServer has been followed (only `DATA` and `CONNECTION` values with `%language%` are substituted):

```
...
LAYER
  NAME TN.RoadTransportNetwork.RoadLink
  DATA "road_%language%"
  ...
END
...
```

If the language is not supported (or no language parameter is present), the default language is substituted.

Provision of INSPIRE specific metadata

Depending on the scenario, additional metadata information is required to support the specification. The INSPIRE related fields are provided below.

Scenario 1 - INSPIRE related fields using referenced external service metadata:

```
WEB
METADATA
"wms_inspire_capabilities" "url"
"wms_languages" "eng,ger"          #first default, values according ISO 639-2/B
"wms_inspire_metadataurl_href" "http://INSPIRE.service/metadata"
```

```

"wms_inspire_metadataurl_format" "application/vnd.ogc.csw.capabilities.response_xml"
"wms_keywordlist_ISO_items" "infoMapAccessService" #value according "classification of spatial data
"wms_keywordlist_vocabulary" "ISO"
"wms_title" "myservicetitle"
"wms_abstract" "myabstract"
"wms_fees" "conditions unknown" #value either "no conditions apply"|default "conditions un
"wms_accessconstraints" "None" #value according ISO 19115 (MD_RestrictionCode codelist) o
"wms_contactorganization" "MapServer" #responsible organization
"wms_contactposition" "owner" #responsible organization, value according "INSPIRE Metadata
...
END
END

```

Scenario 2 - INSPIRE related fields using embedded service metadata:

```

WEB
METADATA
"wms_inspire_capabilities" "embed"
"wms_languages" "eng,ger" #first default, values according ISO 639-2/B
"wms_inspire_temporal_reference" "2011-09-19" #date of last revision, value according YYYY-MM-DD
"wms_inspire_mpoc_name" "mym pocname" #point of contact
"wms_inspire_mpoc_email" "mym poc@e.mail" #point of contact
"wms_inspire_metadatadate" "2011-09-19" #value according YYYY-MM-DD
"wms_inspire_resourcelocator" "http://myinspireresource" #URL for ResourceLocator
"wms_inspire_keyword" "infoMapAccessService" #value according "classification of spatial data serv
"wms_keywordlist_ISO_items" "infoMapAccessService"
"wms_keywordlist_vocabulary" "ISO"
"wms_title" "myservicetitle"
"wms_abstract" "myabstract"
"wms_fees" "conditions unknown" #value either "no conditions apply"|default "conditions un
"wms_accessconstraints" "None" #value according ISO 19115 (MD_RestrictionCode codelist) o
"wms_contactorganization" "MapServer" #responsible organization
"wms_contactposition" "owner" #responsible organization, value according "INSPIRE Metadata
...
END
END

```

Notes:

- several fields require certain values, these values are not validated by MapServer itself, instead a manual validation against the [INSPIRE schemas](#) and the [WMS INSPIRE tester](#) is recommended
- as suggested in this [document](#) regarding scenario 2, `<inspire_common:ResourceType>` is always set to service and `<inspire_common:SpatialDataServiceType>` is always set to view, both values can't be altered through the mapfile
- conformity is always set to not evaluated, based on the latest [INSPIRE Metadata Implementing Rules](#) (page 7), a specification document, the specification date and a specification URI or URL need to be provided for degree conformant/not conformant, which is currently not implemented

Named group layers

INSPIRE mandates usage of named group layers. Thus the functionality of `wms_layer_group` is extended to support named group layers. If a layer with the same name as used in `wms_layer_group` is found it is treated as named group and if no layer with this name is found as unnamed group as before.

Provided that ability, a hierarchy of any level can be achieved. See for example the grouping used in the [wms_inspire.map](#) mapfile in `msautotest`:

```

TN
+--- TN.CommonTransportElements
    +--- TN.CommonTransportElements.TransportArea
    +--- TN.CommonTransportElements.TransportLink
    +--- TN.CommonTransportElements.TransportNode
+--- TN.RoadTransportNetwork
    +--- TN.RoadTransportNetwork
    +--- TN.RoadTransportNetwork.VehicleTrafficArea
    +--- TN.RoadTransportNetwork.RoadServiceArea
    +--- TN.RoadTransportNetwork.RoadArea
+--- TN.RailTransportNetwork
    +--- TN.RailTransportNetwork.RailwayLink
    +--- TN.RailTransportNetwork.RailwayStationArea
    +--- TN.RailTransportNetwork.RailwayYardArea
    +--- TN.RailTransportNetwork.RailwayArea

```

Style section for root layer and possibly existing group layers

For regular layers, the concept of GROUP and CLASSGROUP can be used to set the layer style name to the according value. Additionally the layer style titles can be overwritten through `wms_style_<stylename>_title` and the layer style legendURLs through `wms_style_<stylename>_legendurl_*` (width, height, format, and href need to be provided):

```

...
LAYER
  NAME TN.RoadTransportNetwork.RoadLink
  DATA "road"
  METADATA
    "wms_title.eng" "Transport networks: Road Link"
    "wms_title.ger" "Verkehrsnetze: Strassensegment"
    ...
    "wms_style_inspire_common:DEFAULT_title" "mylayerstyletitle" #style title
    "wms_style_inspire_common:DEFAULT_legendurl_width" "85" #override style legendurl (mandat
    "wms_style_inspire_common:DEFAULT_legendurl_height" "40"
    "wms_style_inspire_common:DEFAULT_legendurl_format" "image/png"
    "wms_style_inspire_common:DEFAULT_legendurl_href" "http://path/to/onlineresource...roadlink"
  END
  ...
END
...
CLASSGROUP "inspire_common:DEFAULT"
CLASSITEM "NAME_E"

CLASS
  NAME "myclass1"
  GROUP "inspire_common:DEFAULT"
  EXPRESSION "Trans-Canada Highway"
  COLOR 255 0 0
END

CLASS
  NAME "myclass2"
  GROUP "inspire_common:DEFAULT"
  COLOR 0 255 0
END
...

```

The following method is implemented to support (customizable) style sections in the root layer:

- use `wms_style_name` in the `WEB.METADATA` section to add a style section to the root layer
- use `wms_style_title` to override the style title (optional)
- use `wms_style_legendurl_*` to override width, height, format and href of the legendURL (optional)

and possibly existing group layers:

- use `wms_group_style_name` in the first corresponding `LAYER.METADATA` section to add a style section to the group layer
- use `wms_group_style_title` to override the style title (optional)
- use `wms_group_style_legendurl_*` to override width, height, format and href of the legendURL (optional)

```

...
WEB
METADATA
...
"wms_style_name" "inspire_common:DEFAULT"           #style name
"wms_style_title" "myroadarealayerstyletitle"      #style title
"wms_style_legendurl_width" "85"                   #override style legendurl (mandatory: width, height)
"wms_style_legendurl_height" "40"
"wms_style_legendurl_format" "image/png"
"wms_style_legendurl_href" "http://path/to/onlineresource...roadarea"
END
END

LAYER
NAME TN.RailTransportNetwork.RailwayLink
GROUP TN.CommonTransportElements.TransportLink
DATA "road"
METADATA
"wms_group_title.eng" "Transport networks: Generic Transport Link"
"wms_group_title.ger" "Verkehrsnetze: Generisches Verkehrssegment"
"wms_group_abstract" "mygenerictransportlinklayerabstract" #fallback
"wms_group_abstract.ger" "mygenerictransportlinklayerabstract"
"wms_group_style_name" "inspire_common:DEFAULT" #style name
"wms_group_style_title" "mygenerictransportlinklayerstyletitle" #style title
"wms_group_style_legendurl_width" "85"             #override style legendurl (mandatory: width, height)
"wms_group_style_legendurl_height" "40"
"wms_group_style_legendurl_format" "image/png"
"wms_group_style_legendurl_href" "http://path/to/onlineresource...generictransportlink"
"wms_title.eng" "Transport networks: Railway Link"
"wms_title.ger" "Verkehrsnetze: Eisenbahnverbindung"
"wms_abstract" "myrailwaylinklayerabstract" #fallback
"wms_abstract.ger" "myrailwaylinklayerabstractger"
...
END
...
END
...

```

Provided that ability, 3 levels of hierarchy can be achieved as done in the example mapfiles `wms_inspire_scenario1.map` and `wms_inspire_scenario2.map` in `msautotest`:

```

TN.RoadTransportNetwork.RoadArea
+--- TN.RoadTransportNetwork.RoadLink
+--- TN.CommonTransportElements.TransportLink
    +--- TN.RailTransportNetwork.RailwayLink
    +--- TN.AirTransportNetwork.AirLink

```

9.1.4 WMS Client

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2013-06-20

Table of Contents

- *WMS Client*
 - *Introduction*
 - *Compilation / Installation*
 - *MapFile Configuration*
 - *Limitations/TODO*

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.

MapServer supports the following WMS versions when acting as client: 1.0.0, 1.0.7, 1.1.0, 1.1.1 (see *MapServer OGC Specification support* for an updated list).

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

Compilation / Installation

The WMS connection type is enabled by the `-with-wmsclient` configure switch. It requires PROJ4, GDAL and libcurl version 7.10.1 or more recent. 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)

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 6.3-dev OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=KML
SUPPORTS=PROJ SUPPORTS=GD SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=CAIRO
SUPPORTS=ICONV SUPPORTS=FRIBIDI SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=GEOS INPUT=JPEG INPUT=POSTGIS INPUT=OGR
INPUT=GDAL INPUT=SHAPEFILE
```

Example 2. Mapserv Version Info on Windows:

```
C:\ms4w\apache\cgi-bin> mapserv -v
MapServer version 6.3-dev OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=KML
SUPPORTS=PROJ SUPPORTS=GD SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=CAIRO
SUPPORTS=ICONV SUPPORTS=FRIBIDI SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=GEOS INPUT=JPEG INPUT=POSTGIS 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.

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

Before version 6.0, and in version 6.0 when wms_cache_to_disk metadata is turned on, 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.

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 "country_bounds"
  TYPE RASTER
  STATUS ON
  CONNECTION "http://demo.mapserver.org/cgi-bin/wms?"
  CONNECTIONTYPE WMS
  METADATA
    "wms_srs"           "EPSG:4326"
    "wms_name"          "country_bounds"
    "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. Including one may allow MapServer to avoid looking up a definition in the PROJ.4 init files.
- **"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.opengis.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”
- **“wms_cache_to_disk” metadata** - set this to “1” to force MapServer to write fetched images to disk. Writing to disk is necessary to take advantage of MapServer’s caching logic to avoid refetching WMS requests made previously. This feature is new to MapServer 6.0 - previously results were always written to disk.
- **“wms_nonsquare_ok” metadata** - set this to “0” to indicate that the remote WMS only supports requests for square pixels. In this case MapServer will be careful to only make square pixel requests even if it means oversampling in one dimension compared to the resolution of image data required. This feature is new to MapServer 6.0.
- **“wms_extent” metadata** - If there is exactly one SRS supported by this layer (as listed in the wms_srs metadata), and if the wms_extent metadata item (or an extent specified via the EXTENT keyword) is set then MapServer will take care to only making requests within this area. This can short circuit requests completely outside the layer, reduce processing for layers that only partially overlap the target map area and avoid poor behaviors with reprojection in some areas. The contents of this metadata item should be of the form “minx miny maxx maxy”. This feature is new to MapServer 6.0.

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_elevation"
TYPE RASTER
STATUS ON
CONNECTIONTYPE WMS
CONNECTION "http://demo.org/cgi-bin/wms?VERSION=1.1.0&LAYERS=bm&FORMAT=image/png"
PROJECTION
  "init=epsg:4326"
END
END
```

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.

9.1.5 WMS Time

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision\$

Date \$Date\$

Last Updated 2006/06/26

Table of Contents

- *WMS Time*
 - *Introduction*
 - *Enabling Time Support in MapServer*
 - *Future Additions*
 - *Limitations and Known Bugs*

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](#)

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.

It is also recommended to set a *LAYER FILTER* for the time layer to provide a default time also for non-WMS requests. If the time item is *mytime*, and the time format is “YYYYMMDD” the following layer filter could be used:

```
FILTER ([mytime] = '2004-01-01 14:10:00')
```

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"
    "wms_enable_request" "*"
  END
TYPE POINT
STATUS ON
```

```

DATA "quakes"
FILTER ([TIME]='2004-01-01 14:10:00')
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-t.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/PT5M"
12  "wms_timeitem" "datetime" #column in postgis table of type timestamp
13  "wms_timedefault" "2006-06-23T03:10:00Z"
14  "wms_enable_request" "*"
15 END
16 CONNECTION "dbname=postgis host=10.10.10.20"
17 CONNECTIONTYPE postgis
18 END
```

19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42

```
# raster layer
LAYER
  NAME "nexrad-n0r-wmst"
  TYPE RASTER
  STATUS ON
  DEBUG ON
  PROJECTION
    "init=epsg:4326"
  END
  METADATA
    "wms_title" "NEXRAD BASE REF WMS-T"
    "wms_srs" "EPSG:4326"
    "wms_extent" "-126 24 -66 50"
    "wms_timeextent" "2003-08-01/2006-12-31/PT5M"
    "wms_timeitem" "datetime" #datetime is a column in postgis table of type timestamp
    "wms_timedefault" "2006-06-23T03:10:00Z"
    "wms_enable_request" "*"
  END
  OFFSITE 0 0 0
  TILEITEM "filepath" #filepath is a column in postgis table with varchar of the filepath to each image
  TILEINDEX "time_idx"
  FILTER ([datetime] = "2006-06-23T03:10:00Z")
END
```

You can find more information on Time and tileindexes in the [WCS documentation](#).

Future Additions

- Support for a special time value: “current”.

Limitations and Known Bugs

- Pattern “YYYYMMDD” does not work on Windows. (Bug#970)

9.1.6 WMS Dimension

Author Yewondwossen Assefa

Contact yassefa at dmsolutions.ca

Last Updated 2013/10/08

Table of Contents

- *WMS Dimension*
 - *Introduction*
 - *Enabling Dimension Support in MapServer*
 - *GetCapabilities Output*
 - *Supported Dimension Requests*
 - *Processing Dimension Requests*

Introduction

A WMS server can provide support for several type of dimensions such as time, elevation or other types of dimensions (for example, satellite images in different wavelength bands). For temporal dimension, please refer to [WMS Time Support](#). This document describes support for the elevation dimension and other type of dimensions

Links to WMS-Related Information

- [MapServer WMS Server HowTo](#)
- [WMS Time Support HowTo](#)

Enabling Dimension Support in MapServer

Setting Up a WMS Layer with dimension support

To have a valid WMS layer with dimension support, the user has to define the following metadata at the layer level:

- *wms_dimensionlist*: (Mandatory) comma separated list of dimension names available for the layer
- *wms_[dimensionname]_item*: (Mandatory) this is the name of the field in the DB that contains the dimension values.
- *wms_[dimensionname]_units*: (Mandatory) Attribute indicating units of dimensional axis. If the dimensional quantity has no units (e.g. band number in a multi-wavelength sensor), use the null string: "". If the dimensional quantity has units, unit names should be taken from the Unified Code for Units of Measure (UCUM) if UCUM has an appropriate entry. When UCUM is used, the mandatory units attribute shall be an appropriate entry from the UCUM "name" column.
- *wms_[dimensionname]_extent*: (Mandatory) defines a valid set of values for the dimension
- *wms_[dimensionname]_default*: (Optional) this value is used if it is defined and the dimension value is missing in the request.

Specifying Dimension Extents Dimension Extents can be declared with the following syntax for the *wms_[dimensionname]_extent* metadata (see Annex C.3 in the [WMS 1.1.1 specification](#) document for a full description):

1. *value* - a single value.
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 "lakes_elev"
  METADATA
    "wms_title"          "Lakes"
    "wms_description"   "Lakes"
    "wms_dimensionlist" "elevation, text_dimension"
    "wms_elevation_item" "ELEV"
    "wms_elevation_extent" "500, 490, 480"
```

```

"wms_elevation_units" "meters"
"wms_elevation_default" "500"
"wms_text_dimension_item" "text_dimen"
"wms_text_dimension_extent" "first, second, third"
  "wms_text_dimension_units" "my_units"
  "wms_enable_request" "*"
END
TYPE POLYGON
..
END

```

GetCapabilities Output

If your layer is set up properly, requesting the capabilities on the server outputs one or several Dimension elements. Here is an example of a GetCapabilities result for a layer configured for two dimensions (wms 1.3.0):

```

<Layer queryable="0" opaque="0" cascaded="0">
  <Name>lakes_elev</Name>
  <Title>Lakes</Title>
  <CRS>EPSG:4326</CRS>
  <EX_GeographicBoundingBox>
    <westBoundLongitude>0.000178263</westBoundLongitude>
    <eastBoundLongitude>0.0034202</eastBoundLongitude>
    <southBoundLatitude>-0.002134</southBoundLatitude>
    <northBoundLatitude>0.000313775</northBoundLatitude>
  <BoundingBox CRS="EPSG:4326" minx="-0.002134" miny="0.000178263" maxx="0.000313775" maxy="0.0034202">
    <Dimension name="elevation" units="meters" default="500" multipleValues="1" nearestValue="0">500,
    <Dimension name="text_dimension" units="my_units" multipleValues="1" nearestValue="0">first, second, third
  </Layer>

```

Supported Dimension Requests

A request parameter name is constructed by concatenating the prefix “dim_” with the sample dimension Name (the value of the name attribute of the corresponding <Dimension> and <Extent> elements in the Capabilities XML). The resulting “dim_name” is case-insensitive. The use of the “dim_” prefix is to avoid clashes between server-defined dimension names and current or future OGC Web Service specifications. (Time and Elevation, being predefined, do not use the prefix.) (section C.4.2)

- *single value*: for example: ...&elevation=500&...
- *multiple values*: for example: ...&dim_text_dimension=first,second&...
- *single range value*: for example: ...&elevation=480/490&...
- *multiple range values*: for example: ...&elevation=480/490,490/500&...

Processing Dimension Requests

When MapServer process a valid dimension wms parameter, It will process it into expressions and set it on the *LAYER FILTER* object. If there was already a Logical “MapServer expressions”, It will be concatenated with it.

For example a request such as &elevation=490/500&... on a MapServer layer (with an empty FILTER) would give .. code-block:: mapfile

```
FILTER (([ELEV] >= 490 AND [ELEV] <= 500))
```

For example a request such as `&elevation=600&...` on a postgis layer with an existing FILTER would give .. code-block:: guess

- FILTER (elev > 500) #before request
- FILTER ((elev > 500) and ((ELEV = 600))) #after request

9.1.7 Map Context

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2010-10-07

Contents

- *Map Context*
 - *Introduction*
 - *Implementing a Web Map Context*

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](#)

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.opengis.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  MAP
2
3  NAME "mapcontext"
4  STATUS ON
5  SIZE 400 300
6  SYMBOLSET "../etc/symbols.txt"
7  EXTENT -180 -90 180 90
8  UNITS DD
9  SHAPEPATH "../data"
10 IMAGECOLOR 255 255 255
11 FONTSET "../etc/fonts.txt"
12
13
14 #
15 # Start of web interface definition
16 #
17 WEB
18   IMAGEPATH "/ms4w/tmp/ms_tmp/"
19   IMAGEURL  "/ms_tmp/"
20   METADATA
21     "wms_abstract" "Demo for map context document. Blah blah..."
22     "wms_title" "Map Context demo"   #### REQUIRED
23   END
24 END
25
26 PROJECTION
27   "init=epsg:4326"
28 END
29

```

```

30 #
31 # Start of layer definitions
32 #
33
34     LAYER
35         NAME "country_bounds"
36         TYPE RASTER
37         STATUS ON
38         CONNECTION "http://demo.mapserver.org/cgi-bin/wms?"
39         CONNECTIONTYPE WMS
40         METADATA
41             "wms_title"                "World Country Boundaries" ##### REQUIRED
42             "wms_onlineresource"       "http://demo.mapserver.org/cgi-bin/wms?" ##### REQUIRED
43             "wms_srs"                  "EPSG:4326"
44             "wms_name"                 "country_bounds"
45             "wms_server_version"       "1.1.1"
46             "wms_format"               "image/gif"
47             "wms_dimensionlist"        "time,width"
48             "wms_dimension"           "time"
49             "wms_dimension_time_unitsymbol" "t"
50             "wms_dimension_time_units"  "ISO8601"
51             "wms_dimension_time_uservalue" "1310"
52             "wms_dimension_time_default" "1310"
53             "wms_dimension_time_multiplevalues" "1310,1410"
54             "wms_dimension_time_nearestvalue" "0"
55         END
56     END
57
58 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
    if (!extension_loaded("MapScript")) dl(MODULE);
    $oMap = ms_newMapObj("mapcontext.map");
    $oMap->saveMapContext("mapcontext_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="mapcontext" 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:4326" minx="-180.000000" miny="-90.000000" maxx="180.000000" maxy="90.000000" />
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="World Country Boundaries">
16       <OnlineResource xlink:type="simple" xlink:href="http://demo.mapserver.org/cgi-bin/wms?"/>
17     </Server>
18     <Name>country_bounds</Name>
19     <Title>World Country Boundaries</Title>
20     <SRS>EPSG:4326</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" />
26     </DimensionList>
27   </Layer>
28 </LayerList>
29 </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 layers_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.

9.1.8 WFS Server

Author Jean-François Doyon

Contact jdoyon at nrcan.gc.ca

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2013-05-16

Contents

- *WFS Server*
 - *Introduction*
 - *Configuring your MapFile to Serve WFS layers*
 - *Reference Section*
 - *To-do Items and Known Limitations*

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.

Versions 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](#).

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
- 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.

- The "wfs_enable_request" metadata (see below).

Example WFS Server Mapfile

The following is an example of a bare minimum WFS Server mapfile. Note the comments for the required parameters.

MAP

```

NAME "WFS_server"
STATUS ON
SIZE 400 300
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/"
  METADATA
    "wfs_title" "WFS Demo Server for MapServer" ## REQUIRED
    "wfs_onlineresource" "http://demo.mapserver.org/cgi-bin/wfs?" ## Recommended
    "wfs_srs" "EPSG:4326 EPSG:4269 EPSG:3978 EPSG:3857" ## Recommended
    "wfs_abstract" "This text describes my WFS service." ## Recommended
    "wfs_enable_request" "*" # necessary
  END
END

PROJECTION
  "init=epsg:4326"
END

#
# Start of layer definitions
#

#####
# World Continents
#####
LAYER
  NAME "continents"
  METADATA
    "wfs_title" "World continents" ##REQUIRED
    "wfs_srs" "EPSG:4326" ## REQUIRED
    "gml_include_items" "all" ## Optional (serves all attributes for layer)
    "gml_featureid" "ID" ## REQUIRED
    "wfs_enable_request" "*"
  END
  TYPE POLYGON
  STATUS ON
  DATA 'shapefile/countries_area'
  PROJECTION
    "init=epsg:4326"
  END
  CLASS
    NAME 'World Continents'
    STYLE
      COLOR 255 128 128
      OUTLINECOLOR 96 96 96

```

```
END
END
END #layer
END #mapfile
```

Rules for Handling SRS in MapServer WFS

The OGC WFS 1.0 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. However, it is possible to have every layer in a WFS server advertised in a different SRS.

The OGC WFS 1.1 specification allows more than one SRS to be advertised, and one of the SRSs will be advertised as the default SRS (the default SRS will be the first in the list specified in the *METADATA wfs_srs / ows_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 is set.
- If no top-level map SRS is defined, then each layer is advertised in its own SRS in the capabilities.

Note: By “SRS is defined”, we mean either the presence of a *PROJECTION* object defined using an EPSG code, or of a *wfs_srs / ows_srs* metadata at this level.

Note: At the map top-level the *wfs_srs / ows_srs* metadata value takes precedence over the contents of the *PROJECTION* block.

Note: The first advertised *wfs_srs / ows_srs* metadata value will be used as the default projection of the bbox requested (unless explicitly set) and will be using in spatial filtering the data for those layer types that support spatial filtering. If the data is in a different projection, the spatial filter BBOX will be projected to match the source data.

At the layer level, if both the *wfs_srs / ows_srs* metadata and the *PROJECTION* object are set to different values, then the *wfs_srs / ows_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 / ows_srs* metadata before being served to WFS clients.

Confusing? As a rule of thumb, simply set the *wfs_srs / ows_srs* at the map level (in web metadata) and never set the *wfs_srs / ows_srs* metadata at the layer level and things will work fine for most cases.

Axis Orientation in WFS 1.1

The axis order in previous versions of the WFS specifications was to always use easting (x or lon) and northing (y or lat). WMS 1.1 specifies that, depending on the particular SRS, 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 WFS 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 returned to the server for the GetFeature request reflect the inverse axis orders for EPSG codes between 4000 and 5000.

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://demo.mapserver.org/cgi-bin/wfs?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: 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://demo.mapserver.org/cgi-bin/wfs?SERVICE=WFS&VERSION=1.0.0&REQUEST=getfeature&TYPENAME=continents&MAXI>

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*.
- [Quantum GIS](#) is a full GIS package which includes WFS client support. (recommended)
- [Deegree](#) provides a WFS client.
- [uDig](#) can add layers from WMS/WFS servers.

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*.

MapServer WFS Extensions

STARTINDEX In addition to the MAXFEATURES=n keyword, MapServer also supports a STARTINDEX=n keyword in WFS GetFeature requests. This can be used to skip some features in the result set and in combination with MAXFEATURES provides for the ability to use WFS GetFeature to page through results. Note that STARTINDEX=0 means start with the first feature, skipping none.

OUTPUTFORMAT Normally OUTPUTFORMAT should be GML2 for WFS 1.0 and either "text/xml; subtype=gml/2.1.2" or "text/xml; subtype=gml/3.1.1" for WFS 1.1. However as an extension to the specification, it is also possible to configure MapServer for a variety of other feature output formats. This is discussed in some detail in the *OGR Output* document.

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_*’ 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.

Web Object Metadata

ows_allowed_ip_list (or wfs_allowed_ip_list)

- *Description:* (Optional) A list of IP addresses that will be allowed access to the service.

Example:

```
METADATA
  "ows_allowed_ip_list" "123.45.67.89 11.22.33.44"
END
```

ows_denied_ip_list (or wfs_denied_ip_list)

- *Description:* (Optional) A list of IP addresses that will be denied access to the service.

Example:

```
METADATA
  "ows_denied_ip_list" "123.45.67.89 11.22.33.44"
END
```

ows_schemas_location (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.opengis.net>. See <http://ogc.dmsolutions.ca> for an example of a valid schema tree.

ows_updatesequence (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.

wfs_abstract (Optional) Descriptive narrative for more information about the server.

WFS TAG Name: Abstract (WFS 1.0.0, sect. 12.3.3)

wfs_accessconstraints (Optional) Text describing any access constraints imposed by the service provider on the WFS or data retrieved from this service.

WFS TAG Name: Accessconstraints (WFS 1.0.0, sect. 12.3.3)

wfs_enable_request (or **ows_enable_request**) Space separated list of requests to enable. The default is none. The following requests can be enabled: *GetCapabilities*, *GetFeature* and *DescribeFeatureType*. A “!” in front of a request will disable the request. “*” enables all requests.

Examples:

To enable only *GetCapabilities* and *GetFeature*:

```
"wfs_enable_request" "GetCapabilities GetFeature"
```

To enable all requests except *GetCapabilities*

```
"wfs_enable_request" "*" !GetCapabilities"
```

wfs_encoding (Optional) XML encoding for all XML documents returned by the server. The default is ISO-8859-1.

wfs_feature_collection Replaces the default name of the feature-containing element (<msFeatureCollection>) with a user-defined value.

wfs_fees (Optional) Any fees imposed by the service provider for usage of this service or for data retrieved from the WFS.

WFS TAG Name: Fees (WFS 1.0.0, sect. 12.3.3)

wfs_getcapabilities_version (Optional) Default version to use for GetCapabilities requests that do not have a version parameter. If not set, the latest supported version will be returned.

wfs_keywordlist (Optional) List of words to aid catalog searching.

WFS TAG Name: Keyword (WFS 1.0.0, sect. 12.3.3)

wfs_maxfeatures (Optional) The number of elements to be returned by the WFS server. This has priority over the 'maxfeatures' parameter passed by the user. If the not set the current behaviour is not changed. Sensible values are integers greater than 0. If 0 is specified, no features will be returned.

wfs_namespace_prefix (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 (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 (Recommended) The URL prefix for HTTP GET requests.

WFS TAG Name: Onlineresource (WFS 1.0.0, sect. 12.3.3)

wfs_service_onlineresource (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_srs (Recommended) The SRS to use for all layers in this server. (e.g. EPSG:4326) See the note about the SRS rules in WFS.

wfs_title (Required) Human readable title to identify server.

WFS TAG Name: Title (WFS 1.0.0, sect. 12.3.3)

Layer Object

gml_constants (Optional) A comma delimited list of constants. This option allows you to define data that are not part of the underlying dataset and add them to the GML output. Many application schemas require constants of one form or another. To specify the value and type of the constants use gml_[item name]_value and gml_[item name]_type.

```
"gml_constants" "const1,const2"
"gml_const1_type" "Character"
"gml_const1_value" "abc"
"gml_const2_type" "Integer"
"gml_const2_value" "999"
```

gml_exclude_items (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 (Required for MapServer 4.10) Field to be used for the ID of the feature in the output GML. wfs_featureid or ows_featureid can be specified instead.

gml_geometries 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_[geometry name]_occurrences 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_[geometry name]_occurrences layer metadata item, where again [geometry name] is the string value specified for gml_geometries, and the value is a comma-delimited pair containing the respective lower and upper bounds.

gml_[geometry name]_type 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_[geometry name]_type, where [geometry name] is the string value specified for gml_geometries, and a value which is one of:

- point
- multipoint
- line
- multiline
- polygon
- multipolygon

gml_groups (Optional) A comma delimited list of group names for the layer.

gml_[group name]_group (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 (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 (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]_precision (Optional) Specifies the precision of the indicated field for formats where this is significant, such as Shapefiles. Precision is the number of decimal places, and is only needed for “Real” fields. Currently this is only used for OGR based output formats, not the WFS GML2/GML3 output.

gml_[item name]_type (Optional) Specifies the type of the attribute. Valid values are the OGR data types: Integer|Real|Character|Date|Boolean. Mapserver translates these to valid GML data types.

gml_[item name]_value Used to specify values for gml_constants.

gml_[item name]_width (Optional) Specifies the width of the indicated field for formats where this is significant, such as Shapefiles.

gml_types (Optional) If this field is “auto” then some input feature drivers (ie. OGR, POSTGIS, ORACLESPATIAL and native shapefiles) will automatically populate the type, width and precision metadata for the layer based on the source file. Currently this is only used for OGR based output formats, not the WFS GML2/GML3 output.

```
"gml_types" "auto"
```

gml_xml_items (Optional) A comma delimited list of items that should not be XML-encoded.

ows_allowed_ip_list Same as ows_allowed_ip_list in the Web Object.

ows_denied_ip_list Same as ows_denied_ip_list in the Web Object.

wfs_abstract Same as wfs_abstract in the Web Object.

wfs_enable_request (or ows_enable_request) Space separated list of requests to enable. The default is none. The following requests can be enabled: *GetCapabilities*, *GetFeature* and *DescribeFeatureType*. A “!” in front of a request will disable the request. “*” enables all requests.

Examples:

To enable only *GetCapabilities* and *GetFeature*:

```
"wfs_enable_request" "GetCapabilities GetFeature"
```

To enable all requests except *GetCapabilities*

```
"wfs_enable_request" "* !GetCapabilities"
```

wfs_extent (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 (Required for MapServer 4.10) Field to be used for the ID of the feature in the output GML. gml_featureid or ows_featureid can be specified instead.

wfs_getfeature_formatlist (Optional) Comma-separated list of formats that should be valid for a GetFeature request. If defined, only these formats are advertised in the Capabilities document.

wfs_keywordlist Same as wfs_keywordlist in the Web Object.

wfs_metadataurl_format (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 (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 (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 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 about the SRS rules in WFS.

wfs_title Same as wfs_title in the Web Object.

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](#) or [TinyOWS](#) 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 mailing list).

9.1.9 WFS Client

Author Jean-François Doyon

Contact jdoyon at nrcan.gc.ca

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2010-11-07

Contents

- [WFS Client](#)
 - [Introduction](#)
 - [Setting up a WFS-client Mapfile](#)
 - [TODO / Known Limitations](#)

Introduction

MapServer can retrieve and display data from a WFS server. The following document explains how to display data from a WFS server using the MapServer *CONNECTIONTYPE WFS*.

WFS can also be accessed through OGR (*CONNECTIONTYPE OGR*). See the [OGR](#) documentation for details.

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.

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. **Windows** users must specify a full path for *IMAGEPATH*, such as: *IMAGEPATH "C:/tmp/ms_tmp/"*

```
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://demo.mapserver.org/cgi-bin/wfs?> 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_geometryname* (optional): The name of the geometry column used for spatial filtering in the filter parameter (Geometry by default). This parameter is used for ArcGIS or GeoServer WFS services as several geometry column can be chosen (or with a different default name to Geometry).
- *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. Sensible values are integers greater than 0. If 0 is specified, no features will be returned.
- *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_request_method"  "GET"
END
...

```

- *wfs_typedname* (required): the <Name> of the layer found in the GetCapabilities. An example GetCapabilities request is: <http://demo.mapserver.org/cgi-bin/wfs?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 "continents"
  TYPE POLYGON
  STATUS ON
  CONNECTION "http://demo.mapserver.org/cgi-bin/wfs?"
  CONNECTIONTYPE WFS
  METADATA
    "wfs_typedname"      "continents"
    "wfs_version"        "1.0.0"
    "wfs_connectiontimeout" "60"
    "wfs_maxfeatures"    "10"
  END
  PROJECTION
    "init=epsg:4326"

```

```

END
CLASS
  NAME "Continents"
  STYLE
    COLOR 255 128 128
    OUTLINECOLOR 96 96 96
  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://demo.mapserver.org/cgi-bin/wfs?SERVICE=WFS&VERSION=1.0.0&TYPENAME=continents"

```

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.

9.1.10 WFS-T Server

Contents

- *WFS-T Server*
 - *WFS-T*

WFS-T

MapServer does not support the WFS-T specification. A companion program, *TinyOWS*, is a fast implementation of WFS-T. TinyOWS and MapServer share the same Mapfile format, and you can use the same Mapfile and datasources for a dual installation of MapServer and TinyOWS on your server.

9.1.11 WFS Filter Encoding

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Yewondwossen Assefa

Contact assefa at dmsolutions.ca

Last Updated 2010-10-07

Date \$Date\$

Table of Contents

- *WFS Filter Encoding*
 - *Introduction*
 - *Currently Supported Features*
 - *Get and Post Requests*
 - *Use of Filter Encoding in MapServer*
 - *Limitations*
 - *Tests*

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](#).

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)

Units of measure

The following units of measure are supported:

m or meters	meters
km or kilometers	kilometers
NM	nauticalmiles
mi or miles	miles
in or inches	inches
ft or feet	feet
deg or dd	degree
px	pixels

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”.

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 (note the “Filter_Capabilities” section):

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <WFS_Capabilities version="1.0.0" updateSequence="0"
3   xmlns="http://www.opengis.net/wfs" xmlns:ogc="http://www.opengis.net/ogc"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://www.opengis.net/wfs
6   http://schemas.opengis.net/wfs/1.0.0/WFS-capabilities.xsd">
7
8 <!-- MapServer version 5.6.5 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
9   OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=AGG SUPPORTS=FREETYPE
10  SUPPORTS=ICONV SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
11  SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
12  SUPPORTS=GEOS SUPPORTS=RGBA_PNG INPUT=EPPL7 INPUT=POSTGIS INPUT=OGR
13  INPUT=GDAL INPUT=SHAPEFILE -->
14
15 <Service>
16   <Name>MapServer WFS</Name>
17   <Title>WMS Demo Server for MapServer</Title>
18   <Abstract>This demonstration server showcases MapServer (www.mapserver.org)
19     and its OGC support</Abstract>
20   <OnlineResource>http://demo.mapserver.org/cgi-bin/wfs?</OnlineResource>
21 </Service>
22
23 <Capability>
24   <Request>
25     <GetCapabilities>
26       <DCPType>
27         <HTTP>
28           <Get onlineResource="http://demo.mapserver.org/cgi-bin/wfs?" />
29         </HTTP>
30       </DCPType>
31       <DCPType>
32         <HTTP>
33           <Post onlineResource="http://demo.mapserver.org/cgi-bin/wfs?" />
34         </HTTP>
35       </DCPType>
36     </GetCapabilities>
37     ...
38   </Request>
39 </Capability>
```

```

40 ...
41 <ogc:Filter_Capabilities>
42   <ogc:Spatial_Capabilities>
43     <ogc:Spatial_Operators>
44       <ogc:Equals/>
45       <ogc:Disjoint/>
46       <ogc:Touces/>
47       <ogc:Within/>
48       <ogc:Overlaps/>
49       <ogc:Crosses/>
50       <ogc:Intersects/>
51       <ogc:Contains/>
52       <ogc:DWithin/>
53       <ogc:BBOX/>
54     </ogc:Spatial_Operators>
55   </ogc:Spatial_Capabilities>
56   <ogc:Scalar_Capabilities>
57     <ogc:Logical_Operators/>
58     <ogc:Comparison_Operators>
59       <ogc:Simple_Comparisons/>
60       <ogc:Like/>
61       <ogc:Between/>
62     </ogc:Comparison_Operators>
63   </ogc:Scalar_Capabilities>
64 </ogc:Filter_Capabilities>
65
66 </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 "cities"
  TYPE POINT
  STATUS ON
  CONNECTION "http://demo.mapserver.org/cgi-bin/wfs?"
  CONNECTIONTYPE WFS
  METADATA
    "wfs_typename" "cities"
    "wfs_version" "1.0.0"
    "wfs_connectiontimeout" "60"
    "wfs_maxfeatures" "100"
    "wfs_filter" "<PropertyIsGreaterThan<<PropertyName>POPULATION</PropertyName>
                  <Literal>10000000</Literal></PropertyIsGreaterThan>"
  END
  PROJECTION
    "init=epsg:4326"
  END
  LABELITEM 'NAME'
  CLASS
    NAME 'World Cities'
    STYLE
      COLOR 255 128 128
      OUTLINECOLOR 128 0 0
      SYMBOL 'circle'

```

```

    SIZE      9
  END
  LABEL
    COLOR      0 0 0
    OUTLINECOLOR 255 255 255
    TYPE      TRUETYPE
    FONT      sans
    SIZE      7
    POSITION    UC
    PARTIALS  FALSE
  END
END
END

```

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.

Limitations

- A limited set of spatial operators are supported.

Tests

Here are some test URLs for the different supported filters:

- PropertyIsEqualTo

```

http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<PropertyIsEqualTo><PropertyName>NAME</PropertyName>
<Literal>Halifax</Literal></PropertyIsEqualTo></Filter>

```

- PropertyIsNotEqualTo

```

http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<PropertyIsNotEqualTo><PropertyName>NAME</PropertyName>
<Literal>Halifax</Literal></PropertyIsNotEqualTo></Filter>

```

- PropertyIsLessThan

```

http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<PropertyIsLessThan><PropertyName>POPULATION</PropertyName>
<Literal>1000</Literal></PropertyIsLessThan></Filter>

```

- PropertyIsGreaterThan

```

http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<PropertyIsGreaterThan><PropertyName>POPULATION</PropertyName>
<Literal>10000000</Literal></PropertyIsGreaterThan></Filter>

```

- PropertyIsLessThanOrEqualTo


```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<PropertyIsLessThanOrEqualTo><PropertyName>POPULATION</PropertyName>
<Literal>499</Literal></PropertyIsLessThanOrEqualTo></Filter>
```

- **PropertyIsGreaterThanOrEqualTo**

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<PropertyIsGreaterThanOrEqualTo><PropertyName>POPULATION</PropertyName>
<Literal>10194978</Literal></PropertyIsGreaterThanOrEqualTo></Filter>
```

- **PropertyIsBetween**

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<PropertyIsBetween><PropertyName>POPULATION</PropertyName>
<LowerBoundary>10194978</LowerBoundary>
<UpperBoundary>12116379</UpperBoundary></PropertyIsBetween></Filter>
```

- **PropertyIsLike**

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<PropertyIsLike wildcard='*' singleChar='.' escape='!'>
<PropertyName>NAME</PropertyName><Literal>Syd*</Literal></PropertyIsLike>
</Filter>
```

- **Logical operator OR**

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<OR><PropertyIsEqualTo><PropertyName>NAME</PropertyName>
<Literal>Sydney</Literal></PropertyIsEqualTo><PropertyIsEqualTo>
<PropertyName>NAME</PropertyName><Literal>Halifax</Literal>
</PropertyIsEqualTo></OR></Filter>
```

- **Logical operator AND**

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<AND><PropertyIsLike wildcard='*' singleChar='.' escape='!'>
<PropertyName>NAME</PropertyName><Literal>Syd*</Literal></PropertyIsLike>
<PropertyIsEqualTo><PropertyName>POPULATION</PropertyName>
<Literal>4250065</Literal></PropertyIsEqualTo></AND></Filter>
```

- **Logical operator NOT**

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<AND><NOT><PropertyIsEqualTo><PropertyName>POPULATION</PropertyName>
<Literal>0</Literal></PropertyIsEqualTo></NOT><NOT><PropertyIsEqualTo>
<PropertyName>POPULATION</PropertyName><Literal>12116379</Literal>
</PropertyIsEqualTo></NOT></AND></Filter>
```

- **Spatial operator BBOX**

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<BBOX><PropertyName>Name</PropertyName><Box%20srsName='EPSG:42304'>
```

```
<coordinates>135.2239,34.4879 135.8578,34.8471</coordinates></Box></BBOX>
</Filter>
```

- Spatial operator Dwithin

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<DWithin><PropertyName>Geometry</PropertyName><gml:Point>
<gml:coordinates>135.500000,34.666667</gml:coordinates>
</gml:Point><Distance units='m'>10000</Distance></DWithin></Filter>
```

- Spatial operator Intersects

```
http://demo.mapserver.org/cgi-bin/wfs?&VERSION=1.0.0&SERVICE=WFS
&REQUEST=GetFeature&TYPENAME=cities&Filter=<Filter>
<Intersects><PropertyName>Geometry</PropertyName>
<gml:Polygon><gml:outerBoundaryIs><gml:LinearRing>
<gml:coordinates>135.5329,34.6624 135.4921,34.8153 135.3673,34.7815
135.3800,34.6216 135.5361,34.6210 135.5329,34.6624</gml:coordinates>
</gml:LinearRing></gml:outerBoundaryIs></gml:Polygon></Intersects></Filter>
```

- The OGC conformance tests (http://cite.opengeospatial.org/test_engine) 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](http://cite.opengeospatial.org).

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 (`geoma-try.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.

9.1.12 SLD

Author Jeff McKenna

Contact `jmckenna at gatewaygeomatics.com`

Author Yewondwossen Assefa

Contact `assefa at dmsolutions.ca`

Last Updated 2011-01-14

Contents

- *SLD*
 - *Introduction*
 - *Server Side Support*
 - *Client Side Support*
 - *Named Styles support*
 - *Other Items Implemented*
 - *Issues Found During Implementation*

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.](#)

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://demo.mapserver.org/cgi-bin/wms?SERVICE=wms&VERSION=1.1.1&REQUEST=GetMap
&LAYERS=country_bound
s&SLD=http://demo.mapserver.org/ogc-demos/map/sld/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>country_bounds</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>country_bounds</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	
Method GET : SLD_BODY	Yes	Additional item
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](#)

This 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 for 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	Yes	Only for AGG driver and mapserver version >=5.2
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: 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

Features	Supported	Notes
Geometry	No	
Stroke	Yes	Strokes are the same as for the LineSymbolizer
Fill	Yes	Was developed to support symbol fill polygons in addition to solid fill
Fill-opacity	Yes	Only available for AGG driver and mapserver version >=5.2
PerpendicularOffset	No	SLD 1.1.0 parameter
Displacement	Yes	SLD 1.1.0 parameter. Sets offsetx/y in MapServer

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: 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	If true-type fonts are not used, default bitmap sizes are given
LabelPlacement	Yes	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 on a Point, Line, or Polygon layer - in addition to other symbolizers used for these layers.
- PointPacement: 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 (MapServer version >=5.4): 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:

- point layer : test for label, font, point placement, color, angle: [sld 10a / full request 10a](#)
- point 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: The ColorMapEntry quantity parameters should be in increasing order.

The following Features are available in SLD 1.1

Features	Supported	Notes
Categorize	Yes	

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.opengis.net/sld/1.1.0/StyledLayerDescriptor.xsd"
xmlns="http://www.opengis.net/sld" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:se="http://www.opengis.net/se" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<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 fffffff
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

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 // force visibility of the layer
17 $oLayer->set('status', MS_ON);
18
```

```

19 // generate the sld for that layer
20 $SLD = $oLayer->generateSLD();
21
22 // save sld to a file
23 $fp = fopen("parcel-sld.xml", "a");
24 fputs( $fp, $SLD );
25 fclose($fp);
26
27 ?>

```

Named Styles support

Named styles support are introduced in MapServer 5.2. The support is base on rfc39.

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

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.

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.

9.1.13 WCS Server

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Stephan Meissl

Contact stephan.meissl at eox.at

Author Fabian Schindler

Contact fabian.schindler at eox.at

Last Updated 2012-08-30

Table of Contents

- *WCS Server*
 - *Introduction*
 - *Configuring Your Mapfile to Serve WCS Layers*
 - *Test Your WCS 1.0 Server*
 - *WCS 1.1.0+ Issues*
 - *WCS 2.0*
 - *HTTP-POST support*
 - *Reference Section*
 - *Rules for handling SRS in a MapServer WCS*
 - *Spatio/Temporal Indexes*
 - *WCS 2.0 Application Profile - Earth Observation (EO-WCS)*
 - *To-do Items and Known Limitations*

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 Geospatial Consortium’s \(OGC\) Web Coverage Service Interfaces Implementation Specification version 1.0.0](#).

Links to WCS-Related Information

- [OGC’s WCS Standard page](#)
- [WCS 1.0.0 specification](#)
- [WCS 1.1.1c1 specification](#)
- [WCS 2.0](#)
 - [GML 3.2.1 Application Schema Coverages](#)
 - [WCS 2.0 Specification - Core](#)

- [WCS 2.0 Specification - KVP Protocol Binding Extension](#)
- [WCS 2.0 Specification - XML/POST Protocol Binding Extension](#)
- [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 \geq 4.4 (tested with 5.0.2 while updating this document)

For WCS 1.1.x (MapServer 5.2) and WCS 2.0 (MapServer 6.0) support 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 with WCS Server support.

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
- WEB metadata or LAYER metadata “wcs_enable_request” must be set
- 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
- LAYER is enabled to be served via WCS (see [MS RFC 67](#))
- LAYER PROJECTION must be set, even if PROJECTION is set at the MAP level (a bug?)

Example WCS Server Mapfile

The following is an example of a simple WCS Server mapfile. Note the comments for the required parameters.

```
MAP
  NAME WCS_server
  STATUS ON
  SIZE 400 300
  SYMBOLSET "../etc/symbols.txt"
  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?"
    "wcs_enable_request" "*"
  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 bath_mapserv.tif
  PROJECTION
    "init=epsg:42304"
  END
END
END # Map File

```


Output Formats

The raster formats supported by MapServer WCS are determined by the `wcs_formats` metadata item on the `LAYER`. This should contain a space separated list of `OUTPUTFORMAT` driver names separated by spaces. If absent, all raster `OUTPUTFORMAT`s are allowed.

WCS is a “raw data” oriented format. So it is often most suitable to use it with format using the `BYTE`, `INT16` and `FLOAT32` `IMAGEMODE`s with GDAL related output formats rather than the built in “rendering oriented” output formats. By default the only GDAL format driver defined is the `GTiff` driver. The following are example output format declarations utilizing the raw image modes:

```
OUTPUTFORMAT
  NAME GEOTIFF_16
  DRIVER "GDAL/GTiff"
  MIMETYPE "image/tiff"
  IMAGEMODE FLOAT32
  EXTENSION ".tif"
END

OUTPUTFORMAT
  NAME AAIGRID
  DRIVER "GDAL/AAIGRID"
  MIMETYPE "image/x-aaigrid"
  IMAGEMODE INT16
  EXTENSION ".grd"
  FORMATOPTION "FILENAME=result.grd"
END
```

The `FORMATOPTION FILENAME` defines the preferred name of the result file when returned WCS `GetCoverage` results.

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:

[WCS GetCapabilities live example](#)

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&COVERAGE=bathymetry
```

Here is a working example of a DescribeCoverage request:

[WCS DescribeCoverage live example](#)

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 DescribeCoverage response.

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 350KB tif is being requested, so this may take a second):

[WCS GetCoverage live example](#)

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.

WCS 2.0

Overview

Version 6.0 introduces support for the new version 2.0 of the WCS specification. This section documents the usage of the new WCS version.

Web Coverage Service (WCS) 2.0 Interface Standard This specification adopts the new OGC Core and Extension model and at the moment the following documents are available from the [OGC's WCS Standard page](#):

- [GML 3.2.1 Application Schema Coverages](#)
- [WCS 2.0 Specification - Core](#)
- [WCS 2.0 Specification - KVP Protocol Binding Extension](#)
- [WCS 2.0 Specification - XML/POST Protocol Binding Extension](#)

Technical changes from WCS version 1.1.2 include entirely building on the [GML 3.2.1 Application Schema Coverages](#) and adoption of [OWS Common 2.0](#). Another major change is the introduction of trim and slice concepts which is explained in more detail below.

There are [WCS 2.0 Schemas](#) defined against which all requests and responses should validate.

WCS 2.0 KVP request parameters The following KVP request parameters are available in WCS 2.0:

COVERAGEID=id: This parameter is technically the same as the **COVERAGE** parameter for WCS 1.0 or the **IDENTIFIER** parameter for **WCS 1.1**. In DescribeCoverage requests, multiple IDs can be requested by concatenating them with commas.

SUBSET=axis[,crs](low,high): This parameter subsets the coverage on the given axis. This parameter can be given multiple times, but only once for each axis. The optional sub-parameter **crs** can either be an EPSG definition (like EPSG:4326), an URN or an URI or 'imageCRS' (which is the default). All **crs** sub-parameters from all **SUBSET** parameters must be equal. (e.g: you cannot subset one axis in imageCRS and another in EPSG:4326).

Note: The syntax of the **crs** sub-parameter is likely to need to be changed when new specification documents become available (see *To-do Items and Known Limitations*).

SIZE=axis(value): This parameter sets the size of the desired axis to the desired value (pixels).

RESOLUTION=axis(value): This parameter sets the resolution of the desired axis to the desired value (pixels/unit).

Note: The **SIZE** and **RESOLUTION** are mutually exclusive on one axis, but can be mixed on different axes (e.g: **SIZE** on x-axis and **RESOLUTION** on y-axis). Also axis names in **SUBSET**, **SIZE** and **RESOLUTION** parameters cannot be mixed. E.g: ...&SUBSET=x(0,100)&SIZE=lon(200)&... is not legal although the axis names logically refer to the same axis.

Note: Recognized values for the axis sub-parameter are: "x", "xaxis", "x-axis", "x_axis", "long", "long_axis", "long-axis", "lon", "lon_axis", "lon-axis", "y", "yaxis", "y-axis", "y_axis", "lat", "lat_axis" and "lat-axis".

OUTPUTCRS=crs: This parameter defines in which crs the output image should be expressed in.

MEDIATYPE=mediatype: This parameter is relevant to GetCoverage requests, when multipart XML/image output is desired. It should be set to 'multipart/related' (which is currently the only possible value for this parameter).

INTERPOLATION=interpolation_method: This defines the interpolation method used, for rescaled images. Possible values are "NEAREST", "BILINEAR" and "AVERAGE".

RANGESUBSET=band1[,band2[,...]]: With this parameter, a selection of bands can be made. Also the bands can be reordered. The bands can be referred to either by name (which can be retrieved using the DescribeCoverage request) or by index (starting with '1' for the first band).

Note: The parameter names **SIZE**, **RESOLUTION**, **OUTPUTCRS=crs:**, **INTERPOLATION**, and **RANGESUBSET** might need to be changed when new specification documents become available (see *To-do Items and Known Limitations*).

Unchanged KVP parameters The following parameters have not (or just slightly) changed since the last version of the WCS standard.

VERSION=version: For **WCS 2.0**, this should be set to '2.0.1'. This parameter is deprecated for GetCapabilities related to OGC Web Services Common Standard 2.0.0.

SERVICE=service

REQUEST=request

ACCEPTVERSIONS=versions

SECTIONS=sections

UPDATESEQUENCE=updatesequence

ACCEPTFORMATS=formats: This parameter is currently ignored.

ACCEPTLANGUAGES=languages: This parameter is currently ignored.

FORMAT=format: The desired format can now also be set with the name of the outputformat object defined in the mapfile. In contrast to previous versions of WCS this parameter is optional when the native format is either specified or can be determined via GDAL.

MAP=mapfile

KVP request examples The below sample request outline the new KVP request syntax:

```
# GetCapabilities
http://www.yourserver.com/wcs?SERVICE=WCS&ACCEPTVERSIONS=1.0.0,2.0.1
  &REQUEST=GetCapabilities
http://www.yourserver.com/wcs?SERVICE=WCS&REQUEST=GetCapabilities
# DescribeCoverage 2.0
http://www.yourserver.com/wcs?SERVICE=WCS&VERSION=2.0.1
  &REQUEST=DescribeCoverage&COVERAGEID=grey
# GetCoverage 2.0 image/tiff full
http://www.yourserver.com/wcs?SERVICE=WCS&VERSION=2.0.1
  &REQUEST=GetCoverage&COVERAGEID=grey&FORMAT=image/tiff
# GetCoverage 2.0 multipart/related (GML header & image/tiff) full
http://www.yourserver.com/wcs?SERVICE=WCS&VERSION=2.0.1
  &REQUEST=GetCoverage&COVERAGEID=grey&FORMAT=image/tiff
  &MEDIATYPE=multipart/related
# GetCoverage 2.0 image/tiff trim x y both
http://www.yourserver.com/wcs?SERVICE=WCS&VERSION=2.0.1
  &REQUEST=GetCoverage&COVERAGEID=grey&FORMAT=image/tiff
  &SUBSET=x(10,200)&SUBSET=y(10,200)
# GetCoverage 2.0 reproject to EPSG 4326
http://www.yourserver.com/wcs?SERVICE=WCS&VERSION=2.0.1
  &REQUEST=GetCoverage&COVERAGEID=grey&FORMAT=image/tiff
  &SUBSET=x,http://www.opengis.net/def/crs/EPSSG/0/4326(-121.488744,-121.485169)
```

Please refer to the [WCS 2.0 tests in msautotest](#) for further sample requests.

Changes to previous versions

The layer name must be a valid NCName, i.e: must not start with a number and can only contain alphanumerical characters. This constraint derives of the gml:id property which has to be a NCName, that relates to the coverage ID which is itself taken from the layers name.

Specifying coverage specific metadata

For WCS enabled layers in MapServer, there are different possibilities for declaring coverage metadata for WCS 2.0. In the simplest case, all of the required metadata can be retrieved from the source image.

For some reason this may not be desirable, maybe because the source image does not provide these metadata. Not every input image format has geospatial metadata attached. In this case, the layer metadata can be used to provide this information.

The convention is that once `(wcslows)_extent` and one of `(wcslows)_size` and `(wcslows)_resolution` is set in the layer metadata, all the coverage specific metadata will be retrieved from there. Otherwise the source image is queried via GDAL, if possible.

The relevant layer metadata fields are `(wcslows)_bandcount`, `(wcslows)_imagemode`, `(wcslows)_native_format`, and all *New band related metadata entries*.

New band related metadata entries

In this section new WCS 2.0 specific layer metadata entries are discussed.

The following layer metadata fields can be used to return a more detailed description for the range type of a “virtual dataset” coverage. A coverage is considered as a “virtual dataset” if the `(wcslows)_extent` metadata entry and one of the `(wcslows)_size` or `(wcslows)_resolution` metadata entries are set.

First of all, the used version of metadata has to be identified. To identify the bands of a coverage, one of the following fields must be present:

- `(wcslows)_band_names` (corresponding to WCS 2.0)
- `(wcslows)_rangeset_axes` (corresponding to WCS 1.1)

The type of these fields is a space delimited list of names, whereas the count of the names has to match the “bandcount” metadata field. These names are then used as a prefix for other metadata fields only concerning this band. The possible metadata keys are the following:

- WCS 2.0:
 - `{band_name}_band_interpretation`
 - `{band_name}_band_uom`
 - `{band_name}_band_definition`
 - `{band_name}_band_description`
 - `{band_name}_interval`
- WCS 1.1
 - `{band_name}_semantic`
 - `{band_name}_values_types`
 - `{band_name}_values_semantic`
 - `{band_name}_description`
 - `{band_name}_interval`

All values are interpreted as strings, only “interval” is interpreted as 2 double precision float values separated with a space.

Also default values can be configured for every key. These have the same suffix as the band specific keys but start with `(wcslows)` instead of the bands name:

- WCS 2.0:
 - `(wcslows)_band_interpretation`
 - `(wcslows)_band_uom`

- (wcslows)_band_definition
- (wcslows)_band_description
- (wcslows)_interval
- WCS 1.1
 - (wcslows)_semantic
 - (wcslows)_values_types
 - (wcslows)_values_semantic
 - (wcslows)_description
 - (wcslows)_interval

If no specific or default value is given, the output is dependant on the metadata key. The UOM, for example will be set to 'W.m-2.Sr-1', interval and significant figures will be determined according to the image type and definition, description, and interpretation will not be visible in the output at all.

This example demonstrates the use of the band-specific metadata fields with their default values:

```

METADATA
"ows_srs" "EPSG:4326"
"wcs_extent" "47.5070762077246 16.038578977182 49.0103258976982 17.2500586851354"

"wcs_size" "1200 1100"
"wcs_imagemode" "BYTE"

"wcs_bandcount" "3"
"wcs_band_names" "BandA BandB BandC"

#default values
"wcs_band_interpretation" "This is default interpretation"
"wcs_band_uom" "DefaultUOM"
"wcs_band_definition" "DefaultDefinition"
"wcs_band_description" "This is default description"
"wcs_interval" "0 125"
"wcs_significant_figures" "3"

#specific band values
"BandA_band_interpretation" "This is a specific interpretation"
"BandA_band_uom" "SpecificUOM"
"BandA_band_definition" "SpecificDefinition"
"BandA_band_description" "This is a specific description"
"BandA_interval" "0 255"
END

```

The above example would result in having BandA a more specific description, and BandB and BandC having the default description. It would also be possible to only use some of the specific values for BandA and others from the default.

If no default and specific values are given for the interval or significant figures metadata field, the a default is generated from the "imagemode" field, which itself defaults to FLOAT32.

The new metadata fields also contain the (wcslows)_nilvalues and (wcslows)_nilvalues_reasons

- (wcslows)_nilvalues

With this field, specific nilvalues can be set. The values have to be delimited by a space.

- (wcslows)_nilvalues_reasons

This field defines the reasons for the specific nilvalues. The reasons are also space delimited and reference the nilvalue with the same index. The values for the reasons should be URIs or URNs.

The following example demonstrates the use of both metadata fields:

```
METADATA
"ows_srs" "EPSG:4326"
"wcs_extent" "47.5070762077246 16.038578977182 49.0103258976982 17.2500586851354"

"wcs_size" "1200 1100"
"wcs_imagemode" "BYTE"
"wcs_bandcount" "3"

"wcs_nilvalues" "0 255"
"wcs_nilvalues_reasons"
  "urn:ogc:def:nil:OGC::BelowDetectionLimit urn:ogc:def:nil:OGC::AboveDetectionLimit"
END
```

HTTP-POST support

Since version 6.0 MapServer also supports HTTP-POST XML requests. All requests possible via HTTP GET can also be sent via POST. POST requests are possible for WCS 1.1 or WCS 2.0 which adhere to the according standard.

This is an example GetCapabilities request:

```
<?xml version="1.0" encoding="UTF-8"?>
<wcs:GetCapabilities
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation="http://www.opengis.net/wcs/2.0
    http://schemas.opengis.net/wcs/2.0/wcsAll.xsd"
  xmlns="http://www.opengis.net/wcs/2.0"
  xmlns:wcs='http://www.opengis.net/wcs/2.0'
  xmlns:ows="http://www.opengis.net/ows/2.0"
  service="WCS">
  <ows:AcceptVersions>
    <ows:Version>2.0.1</ows:Version>
  </ows:AcceptVersions>
  <ows:Sections>
    <ows:Section>OperationsMetadata</ows:Section>
    <ows:Section>ServiceIdentification</ows:Section>
  </ows:Sections>
</wcs:GetCapabilities>
```

This is an example DescribeCoverage request, which is only valid for WCS 2.0:

```
<?xml version="1.0" encoding="UTF-8"?>
<wcs:DescribeCoverage
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation="http://www.opengis.net/wcs/2.0
    http://schemas.opengis.net/wcs/2.0/wcsAll.xsd"
  xmlns="http://www.opengis.net/wcs/2.0"
  xmlns:wcs="http://www.opengis.net/wcs/2.0"
  service="WCS"
  version="2.0.1">
  <wcs:CoverageId>SOME_ID</wcs:CoverageId>
</wcs:DescribeCoverage>
```

This example demonstrates the usage of a WCS 2.0 POST-XML GetCoverage request:

```
<?xml version="1.0" encoding="UTF-8"?>
<wcs:GetCoverage
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation="http://www.opengis.net/wcs/2.0
    http://schemas.opengis.net/wcs/2.0/wcsAll.xsd"
  xmlns="http://www.opengis.net/wcs/2.0"
  xmlns:wcs="http://www.opengis.net/wcs/2.0"
  service="WCS"
  version="2.0.1">
  <wcs:CoverageId>SOME_ID</wcs:CoverageId>
  <wcs:DimensionTrim>
    <wcs:Dimension crs="http://www.opengis.net/def/crs/EPSSG/0/4326">x
    </wcs:Dimension>
    <wcs:TrimLow>16.5</wcs:TrimLow>
    <wcs:TrimHigh>17.25</wcs:TrimHigh>
  </wcs:DimensionTrim>
  <wcs:DimensionTrim>
    <wcs:Dimension crs="http://www.opengis.net/def/crs/EPSSG/0/4326">y
    </wcs:Dimension>
    <wcs:TrimLow>47.9</wcs:TrimLow>
  </wcs:DimensionTrim>
  <wcs:format>image/tiff</wcs:format>
  <wcs:mediaType>multipart/related</wcs:mediaType>
  <wcs:Resolution dimension="x">0.01</wcs:Resolution>
  <wcs:Size dimension="y">50</wcs:Size>
</wcs:GetCoverage>
```

Please refer to the [WCS 2.0 Specification - XML/POST Protocol Binding Extension](#) and the [WCS 2.0 Schemas](#) for further information on POST request in WCS 2.0.

Reference Section

To avoid confusion only “wcs_*” and “ows_*” prefixed metadata entries are evaluated in OGC WCS services. Previous versions used “wms_*” prefixed entries as fallback which is dropped in version 6.0 in favor of forcing explicit decisions. The module will look for the “wcs_*” and “ows_*” metadata prefixes in this order.

The following metadata are available in the setup of the mapfile:

Web Object Metadata

ows_allowed_ip_list (or wcs_allowed_ip_list)

- *Description:* (Optional) A list of IP addresses that will be allowed access to the service.

Example:

```
METADATA
  "ows_allowed_ip_list" "123.45.67.89 11.22.33.44"
END
```

ows_denied_ip_list (or wcs_denied_ip_list)

- *Description:* (Optional) A list of IP addresses that will be denied access to the service.

Example:

```
METADATA
  "ows_denied_ip_list" "123.45.67.89 11.22.33.44"
END
```

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_contactfacilelephone, wcs_contactorganization, wcs_contactperson, wcs_contactposition, wcs_contactvoicetelephone, wcs_country, wcs_postcode, 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_enable_request (or ows_enable_request)

- *Description:* Space separated list of requests to enable. The default is none. The following requests can be enabled: *GetCapabilities*, *GetCoverage* and *DescribeCoverage*. A "!" in front of a request will disable the request. "*" enables all requests.

- *Examples:*

To enable only *GetCapabilities* and *GetCoverage*:

```
"wcs_enable_request" "GetCapabilities GetCoverage"
```

To enable all requests except *GetCapabilities*

```
"wcs_enable_request" "* !GetCapabilities"
```

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.

wcs_service_onlineresource

- *Description:* (Optional) Top-level onlineresource URL. MapServer uses the onlineresource metadata (if provided) in the following order:
 1. `wcs_service_onlineresource`
 2. `ows_service_onlineresource`
 3. `wcs_onlineresource` (or automatically generated URL, see the onlineresource section of this document)

Layer Object Metadata

ows_allowed_ip_list Same as `ows_allowed_ip_list` in the Web Object.

ows_denied_ip_list Same as `ows_denied_ip_list` in the Web Object.

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_enable_request (or **ows_enable_request**)

- *Description:* Space separated list of requests to enable. The default is none. The following requests can be enabled: *GetCapabilities*, *GetCoverage* and *DescribeCoverage*. A “!” in front of a request will disable the request. “*” enables all requests.
- *Examples:*

To enable only *GetCapabilities* and *GetCoverage*:

```
"wcs_enable_request" "GetCapabilities GetCoverage"
```

To enable all requests except *GetCapabilities*

```
"wcs_enable_request" "*" !GetCapabilities"
```

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, separated by a space. (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”) (used for WCS 1.0)

wcs_native_format

- *Description:* (Optional) **The mime-type of the current format of the** served raster layer (e.g. “image/tiff”). This field is used when coverage metadata is provided by the layer metadata only (when `wcs_extent` and `wcs_size/wcs_resolution` are set). When set, WCS 2.0 GetCoverage requests will use this format when no other format is specified (the format parameter is optional then).

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.

Rules for handling SRS in a MapServer WCS

TODO!

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.

WCS 2.0 Application Profile - Earth Observation (EO-WCS)

OGC is currently discussing the adoption of an Earth Observation (EO) Application Profile for WCS 2.0 (EO-WCS) (see [public RFC on EO-WCS](#)). For an implementation please refer to the Open Source project [EOxServer](#) which already implements this proposed EO-WCS based on MapServer.

To-do Items and Known Limitations

- 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.
- The available set of WCS 2.0 specification documents is not yet complete. Thus, for some implementation details, the content of some forthcoming extensions had to be anticipated based on the approaches taken for WCS 1.1 and 1.0. The implementation will be adjusted as soon as new specification documents become available.
- If you want to use libxml2 or its derived tools (like xmllint) for validation be aware that there is a currently bug in libxml2 that breaks the validation of GML 3.2.1.

9.1.14 WCS Use Cases

Author Norman Barker

Contact nbarker at ittvis.com

Author Gail Millin

Contact nbarker at ittvis.com

Revision \$Revision\$

Date \$Date\$

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

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
```

```
LAYER
NAME "Landsat7"
STATUS OFF
TYPE RASTER
PROCESSING "SCALE=AUTO"
UNITS Meters
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"
  "wcs_enable_request" "*"
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).

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
  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"
    "wcs_enable_request" "*"
  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

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).

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 the corresponding layer

```

LAYER
NAME "srtm"
STATUS OFF
TYPE RASTER
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"
  "wcs_enable_request" "*"
END
END

```

Performance gains can be made by using the `gdaladdo` utility described at http://www.gdal.org/gdal_utilities.html#gdaladdo

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
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 'tif'
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  
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'  
  "wcs_enable_request" "*"   
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.

9.1.15 SOS Server

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Revision \$Revision\$

Date \$Date\$

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*

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 rfc13.

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).

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 SUPPORTS=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 SUPPORTS=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`
 - `sos_enable_request`
 - 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 sos_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.

```
MAP
  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/mapserv?map=/sos/sos_test.map" ## REQUIRED
    "sos_title"          "My SOS Demo Server" ## Recommended
    "sos_srs"            "EPSG:4326" ## REQUIRED
    "sos_enable_request" "*" # Necessary
  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
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 result-Model 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=SOS&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=SOS&Offering=WQ1289&
observedproperty=Water Quality&eventtime=<ogc:TM_Equals><gml:TimePeriod>
<gml:beginPosition>1991-05-01</gml:beginPosition><gml:endPosition>
1993-02-02</gml:endPosition></gml:TimePeriod></ogc:TM_Equals>
&result=<Filter><Or><PropertyIsEqualTo><PropertyName>COLOUR
</PropertyName><Literal>180</Literal></PropertyIsEqualTo>
<PropertyIsEqualTo><PropertyName>COLOUR</PropertyName><Literal>200
</Literal></PropertyIsEqualTo></or></Filter>&version=1.0.0
&responseFormat=text/xml; subtype="om/1.0.0"
```

Example 3:

```
http://127.0.0.1/cgi-bin/mapserv.exe?map=D:/ms4w/apps/sos/sos_test.map&
Request=GetObservation&service=SOS&Offering=WQ1289&
observedproperty=Water Quality&featureofinterest=<gml:Envelope>
<gml:lowerCorner srsName='EPSG:4326'>-66 43</gml:lowerCorner>
<gml:upperCorner srsName='EPSG:4326'>-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=SOS&Offering=WQ1289&
observedproperty=Water Quality&version=1.0.0&
responseFormat=text/xml; subtype="om/1.0.0"&resultModel=om:Observation
```

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="sensorML/1.0.0"
```

Limitations / TODO

1. Have MapServer generate the SensorML document, instead of redirecting the request to an existing SensorML document.

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_*’ instead of ‘sos_*’. MapServer tries the ‘sos_*’ 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.

Web Object Metadata

ows_allowed_ip_list (or sos_allowed_ip_list)

- *Description:* (Optional) A list of IP addresses that will be allowed access to the service.

Example:

```
METADATA
  "ows_allowed_ip_list" "123.45.67.89 11.22.33.44"
END
```

ows_denied_ip_list (or sos_denied_ip_list)

- *Description:* (Optional) A list of IP addresses that will be denied access to the service.

Example:

```
METADATA
  "ows_denied_ip_list" "123.45.67.89 11.22.33.44"
END
```

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.

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.

sos_abstract

- *Description:* (Optional) Descriptive narrative for more information about the server.

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_country, sos_postcode, sos_stateorprovince

- *Description:* Optional contact address information. If provided then all six metadata items are required.

sos_contactelectronicmailaddress

- *Description:* Optional contact Email address.

sos_contactfacsimiletelephone

- *Description:* Optional contact facsimile telephone number.

sos_contactinstructions

- *Description:* (Optional) Supplemental instructions on how or when to contact the individual or organization.

sos_contactorganization, sos_contactperson, sos_contactposition

- *Description:* Optional contact information. If provided then all three metadata items are required.

sos_contactvoicetelephone

- *Description:* Optional contact voice telephone number.

sos_enable_request (or ows_enable_request)

- *Description:* Space separated list of requests to enable. The default is none. The following requests can be enabled: *GetCapabilities*, *GetObservation* and *DescribeSensor*. A “!” in front of a request will disable the request. “*” enables all requests.

- *Examples:*

To enable only *GetCapabilities* and *GetObservation*:

```
"sos_enable_request" "GetCapabilities GetObservation"
```

To enable all requests except *GetCapabilities*

```
"sos_enable_request" "* !GetCapabilities"
```

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 ‘,’

sos_fees

- *Description:* (Optional) Fees information. Use the reserved word “none” if there are no fees.

sos_hoursofservice

- *Description:* (Optional) Time period (including time zone) when individuals can contact the organization or individual.

sos_keywordlist

- *Description:* (Optional) A comma-separated list of keywords or keyword phrases to help catalog searching.

sos_maxfeatures

- *Description:* (Optional) The number of elements to be returned by the SOS server. If the not set all observations are returned

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_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_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.

Layer Object Metadata

ows_allowed_ip_list Same as `ows_allowed_ip_list` in the Web Object.

ows_denied_ip_list Same as `ows_denied_ip_list` in the Web Object.

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_enable_request (or **ows_enable_request**)

- *Description:* Space separated list of requests to enable. The default is none. The following requests can be enabled: *GetCapabilities*, *GetObservation* and *DescribeSensor*. A “!” in front of a request will disable the request. “*” enables all requests.
- *Examples:*

To enable only *GetCapabilities* and *GetObservation*:

```
"sos_enable_request" "GetCapabilities GetObservation"
```

To enable all requests except *GetCapabilities*

```
"sos_enable_request" "* !GetCapabilities"
```

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_authority

- *Description:* (Optional) An associated authority for a given component of an observed property

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_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"
```

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.

9.1.16 How to set up MapServer as a client to access a service over https

Revision \$Revision: 12521 \$

Date \$Date: 2011-09-06 19:48:20 +0200 (Tue, 06 Sep 2011) \$

Table of Contents

- *How to set up MapServer as a client to access a service over https*
 - *Introduction*
 - *Requirements*
 - *Default Installation (with apt-get install, rpm, manual, etc)*
 - *Non-Standard Installation (common with ms4w and fgs)*
 - *Remote Server with a Self-Signed SSL Certificate*

Introduction

The following documentation explains how to set up MapServer as a client to access a WMS/WFS server through a secure SSL connection using the HTTPS protocol. It describes the common problems a user could encounter and how to solve them.

Requirements

MapServer 5.4.1 and up, compiled with Curl. Curl must be built with SSL support.

Default Installation (with apt-get install, rpm, manual, etc)

The Curl CA bundle file should be located in the default directory.

Verify your connection with the Curl command line:

```
curl https://targethostname:port/gmap-demo/gmap75.phtml
```

Edit your map file to add the WMS connection URL. For example:


```
CONNECTION "https://domainname:port/cgi-bin/mapserv?map=/path/to/wms.map"
CONNECTIONTYPE WMS
```

If the layer is displayed correctly you do not need to read on.

Non-Standard Installation (common with ms4w and fgs)

If you get the following error, it means that your CA bundle is not found.

```
curl https://localhost:port/gmap-demo/gmap75.phtml
curl: (77) error setting certificate verify locations:
  CAfile: /home/nsavard/fgsfull/share/curl/cacert.pem
  CApath: none
```

It may be caused by the `CURL_CA_BUNDLE` environment variable pointing to the wrong location or the CA bundle file not being present. Follow the steps below to correct either case.

Set the `CURL_CA_BUNDLE` environment variable to point to the bundle file (e.g. `export CURL_CA_BUNDLE=/path/to/my-ca-bundle.ext` where `my-ca-bundle.ext` could be `cacert.pem` or `ca-bundle.crt`).

Download the CA bundle file “`cacert.pem`” found at <http://curl.haxx.se/docs/caextract.html> or if you have the Curl source you could create the CA bundle by executing “`make ca-bundle`” or “`make ca-firefox`” (if you have Firefox and the `certutil` tool installed). If you used the second choice, the bundle file will be named `ca-bundle.crt` and will be found in the `lib` directory under the Curl root directory. See <http://curl.haxx.se/docs/caextract.html> for more details. Store this file in the location pointed to by the `URL_CA_BUNDLE` environment variable.

Verify your connection using the Curl command line:

```
curl https://targethostname:port/gmap-demo/gmap75.phtml
```

Note: If you use `ms4w`, `osgeo4w` or `fgs` installation, these installers should take care of this problem for you.

Remote Server with a Self-Signed SSL Certificate

If you get the following error, it means that your remote server probably use a self-signed SSL certificate and the server certificate is not included in your CA bundle file.

```
curl: (60) SSL certificate problem, verify that the CA cert is OK. Details:
error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
More details here: http://curl.haxx.se/docs/sslcerts.html
```

curl performs SSL certificate verification by default, using a “bundle” of Certificate Authority (CA) public keys (CA certs). If the default bundle file isn't adequate, you can specify an alternate file using the `--cacert` option.

If this HTTPS server uses a certificate signed by a CA represented in the bundle, the certificate verification probably failed due to a problem with the certificate (it might be expired, or the name might not match the domain name in the URL).

If you'd like to turn off curl's verification of the certificate, use the `-k` (or `--insecure`) option.

To get the remote server certificate you have to execute this command:

```
openssl s_client -connect domainname:port
```

Copy everything from the “—BEGIN CERTIFICATE—” tag to “—END CERTIFICATE—” tag. Paste it at the end of the my-ca-bundle.ext file.

Verify your connection with the Curl command line:

```
curl https://targethostname:port/gmap-demo/gmap75.phtml
```

Note: If you get the following error, it means that the domain name in the URL request is not corresponding to the one that was declared when creating the remote server certificate.

```
curl: (51) SSL: certificate subject name 'domainname' does not match target host name 'domainname'
```

You have to use the exact same domain name as the one appearing in the “Common Name” prompt used when generating the remote server certificate. You cannot use the remote server ip for instance. It means that the following URL is not acceptable.

```
CONNECTION "https://xxx.xxx.xxx.xxx:port/cgi-bin/mapserv?map=/path/to/wms.map"  
CONNECTIONTYPE WMS
```

9.1.17 MapScript Wrappers for WxS Services

Author Frank Warmerdam

Contact warmerdam at pobox.com

Revision \$Revision\$

Date \$Date\$

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*

Introduction

With the implementation of MapServer rfc16 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.

Warning: *MapServer CGI Controls* and *Run-time Substitution* are **not** applied when using mapscript WxS wrappers. It is up to the mapscript code you are writing to apply any mapfile modifications.

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 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.

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 mapsript;
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 mapsript::OWSRequest();

# pick up CGI paramters passed
$req->loadParams();

# init mapfile
my $map = new mapsript::mapObj($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 = mapsript::msIO_installStdoutToBuffer();

```

```

$dispatch = $map->OWSDispatch($req);

# at this point, the client's request is sent

# pull out the HTTP headers
my $ct = mapsript::msIO_stripStdoutBufferContentType();

# and then pick up the actual content of the response
my $content = mapsript::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);
}

```

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.mapsript.mapObj;
import edu.umn.gis.mapsript.OWSRequest;
import edu.umn.gis.mapsript.mapsript;

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 = mapscript.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:" +
                }
                else
                    success++;
            }

            System.out.println( "Successes: " + success );
            System.out.println( "Failures: " + failure );

        } catch( Exception e ) {
            e.printStackTrace();
        }
    }
}

```

PHP Example

Most of the same capabilities are accessible in php mapscript. 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();

?>

```

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.

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)
```

10.1 TinyOWS

Author Olivier Courtin

Contact olivier dot courtin at oslandia.com

TinyOWS is a lightweight and fast implementation of the OGC WFS-T specification. Web Feature Service (WFS) allows to query and to retrieve features. The transactional profile (WFS-T) allows then to insert, update or delete such features.

From a technical point of view WFS-T is a Web Service API in front of a spatial database. TinyOWS is so deeply tighed to PostgreSQL/PostGIS.

TinyOWS is already safely used in quite big GIS infrastructure arch, for instance, to allow European farmers to report the locations and crops on their fields (and then to be paid by EEC accordingly). <http://2012.foss4g-cee.org/wp-content/uploads/2012/04/IPA-Online.pdf>

TinyOWS implement strictly OGC standards and pass successfully all WFS OGC CITE tests (and even beta ones).

TinyOWS is part of MapServer Suite, but provided as a distinct module (i.e you could use it in conjunction with MapServer and MapCache, or as a standalone app) But both MapServer and TinyOWS could use the same configuration file, if you want to (or native TinyOWS XML config file).

10.1.1 TinyOWS Installation

Requires

TinyOWS need following libraries/applications:

- LibXML2 (2.8 version or later)
- PostGIS (1.5.x version or later)
- PostgreSQL (with libpq headers)
- A working Web Server with cgi-bin support
- Fast-CGI is recommended

Installing from a stable source release

An example of a typical download, configure, make, make install:

```
$ wget http://download.osgeo.org/mapserver/tinyows-1.1.0.tar.bz2
$ tar xvjf tinyows-1.1.0.tar.bz2
$ cd mapserver-tinyows
$ ./configure
$ make
$ sudo make install
$ sudo make install-demo
```

Then copy the `tinyows` binary to your `cgi-bin` directory.

Installing the Current Trunk from GIT

To build from git, you must first install the `autoconf` utility, and git application .

```
$ git clone git://github.com/mapserver/tinyows.git
$ cd tinyows
$ autoconf
$ ./configure
$ make
$ sudo make install
$ sudo make install-demo
```

Then copy the `tinyows` binary to your `cgi-bin` directory.

Installing from source on Windows (oldies so dunno if still up to date)

- From GIT with Visual C++ for Win32 (written by Alexander Bruy)

10.1.2 Configuring TinyOWS with an XML File

The simplest way to configure TinyOWS is with a single XML file called `tinyows.xml`.

The default path is `/etc/tinyows.xml`. You can also use `TINYOWS_CONFIG_FILE` environment variable to set your own path.

Configuration file simple Example

An example `config.xml` file is in the `demo` directory:

```
<tinyows online_resource="http://127.0.0.1/cgi-bin/tinyows"
  schema_dir="/usr/local/tinyows/schema/">

  <pg host="127.0.0.1" user="postgres" password="postgres" dbname="tinyows_demo" port="5432"/>

  <metadata name="TinyOWS Server"
    title="TinyOWS Server - Demo Service" />

  <layer retrievable="1"
    writable="1"
    ns_prefix="tows"
    ns_uri="http://www.tinyows.org/"
    name="world"
    title="World Administrative Boundaries" />
```

```
<layer retrievable="1"
  writable="1"
  ns_prefix="tows"
  ns_uri="http://www.tinyows.org/"
  name="france"
  title="French Administrative Sub Boundaries (IGN - GeoFLA Departements)" />
</tinyows>
```

Testing your config.xml file

Once you have a config.xml file related to your service, launch TinyOWS with the `--check` option to validate your configuration file, test your database connection, and list the layers to be used:

```
./tinyows --check
TinyOWS version:    1.1.0
FCGI support:      Yes
Config File Path:  /etc/tinyows.xml (TinyOWS XML)
PostGIS Version:   2.1.0
PostGIS dsn:       host=127.0.0.1 user=postgres password=postgres dbname=foo port=5432
Output Encoding:   UTF-8
Database Encoding: UTF8
Schema dir:        /usr/local/share/tinyows/schema/
Display bbox:      Yes
Estimated extent:  No
Check schema:      Yes
Check valid geoms: No
Available layers:
- public.commune (2154) -> tows.commune [RW]
- public.world (4326) -> tows.world [RW]
```

Structure of the config.xml file

TinyOWS Element

TinyOWS is the root element. He is mandatory, and must contains some system informations about the service itself. Some globals service options could also be switched on or off at this level.

Attribute	Re-quired?	De-fault	Description
on-line_resource	mandatory		URL where the service is located, e.g: <code>http://127.0.0.1/cgi-bin/tinyows</code>
schema_dir	mandatory		Path where TinyOWS schema dir is located e.g: <code>/usr/local/tinyows/schema/</code>
log	optional		Path where TinyOWS logs input requests. e.g: <code>/var/log/tinyows.log</code> . This file must be writable by the user that owns the TinyOWS process.
log_level	optional	0	Bit field value to indicate what to log: 1: ERROR, 2: EVENT, 4: HTTP QUERY, 8: SQL. e.g: 15 to log all.
degree_precision	optional	6	Indicate how many digits of decimal precision when coordinates are express in latitude/longitude.
meter_precision	optional	0	Indicate how many digits of decimal precision to use when coordinates are projected (so meter unit).
display_bbox	optional	1	Flag to indicate if bounding box should be computed for WFS GML <code>GetFeature</code> output. It's mandatory in WFS specification. But as it's time consuming it could be interesting to be able to deactivate it.
estimated_extent	optional	0	Flag to indicate if TinyOWS should use <code>estimated_extent</code> (faster but slightly less accurate).
check_schema	optional	1	Flag to indicate if input data must be checked against schema before to be executed. Caution, schema validation is an important part of security. Disable this attribute at your peril.
check_valid_geom	optional	1	Flag to indicate if OGC SFS 1.1 geometry validation should be done prior to execute a transaction.
encoding	optional	UTF-8	Output encoding. Other values could be ISO-8859-1 for instance. No encoding conversion is done on data; this attribute is declarative.
expose_pk	optional	0	Flag to indicate if TinyOWS should expose PK in schema (and so require them in Transaction query).
wfs_default_version	optional		String version to indicate WFS default version, 1.0.0 or 1.1.0 for instance.

Limits Element

Limits Element provides a maximum for the server output. It could help to prevent a denial of service attack, or an abnormally large user query, from crashing your server. This element is optional.

Limits attributes

Attribute	Re-quired?	De-fault	Description
features	optional		Use to set maximum number of features returned to WFS client, on <code>GetFeature</code> request
geobbox	optional		Geographic bounding bbox, used to indicate maximum extent: East,West,North,South

```
<tinyows>
...
<limits features="10000" />
...
</tinyows>
```

PostgreSQL Connection

PostgreSQL connection element. This element is mandatory.

Attribute	Required?	Default	Description
host	optional	localhost	Name (or IP) to PostgreSQL server (default is localhost)
user	optional		PostgreSQL user to connect (default is the system user used to run the server)
password	optional		PostgreSQL password connection
dbname	optional		PostGIS database (by default, same as the system user used to run the server)
port	optional	5432	PostgreSQL port number
encoding	optional	UTF-8	PostgreSQL DB encoding, as specified in http://www.postgresql.org/docs/9.0/static/multibyte.html#CHARSET-TABLE

```
<tinyows>
...
<pg host="127.0.0.1"
    user="postgres"
    password="postgres"
    dbname="gis_data"
    port="5432" />
...
</tinyows>
```

Metadata and Contact Elements

Used to provide information about the service itself. These two elements are mandatory.

	Attribute	Required?	Default	Description
Metadata attributes	name	mandatory		Web Service Name
	title	mandatory		Web Service Title
	keywords	optional		Web Service Keywords list (comma separated list)
	fees	optional		Web Service Fees
	access_constraints	optional		Web Service Access Constraints

Abstract Element The Abstract element is an optional child element of Metadata. It is a place for a free-formatted text description of the service.

Contact attributes

Attribute	Required?	Default	Description
name	mandatory		Web Service Contact Name
site	mandatory		Web Service Contact URL
email	mandatory		Web Service Contact Email
individual_name	optional		Web Service Contact Individual Name
position	optional		Web Service Contact Position
phone	optional		Web Service Contact Phone
fax	optional		Web Service Contact Fax
online_resource	optional		Web Service Contact URL (e.g additional Metadatas)
address	optional		Web Service Contact Postal Address
postcode	optional		Web Service Contact Postcode
city	optional		Web Service Contact City
administrative_area	optional		Web Service Contact Administrative Area
country	optional		Web Service Contact Country
hours_of_service	optional		Web Service Contact Hours of Services
contact_instructions	optional		Web Service Contact Instructions ll

Contact and Metadata example with only mandatory attributes:

```
<metadata name="TinyOWS Server"
  title="TinyOWS Server - Demo Service" />

<contact name="TinyOWS Server"
  site="http://www.tinyows.org/"
  email="tinyows-users@lists.maptools.org" />
```

Layer Element

Layer element is used to set all layers provided by the service. Although this element is technically optional, omitting it will cause no layer at all to be provided.

Attribute	Re-quired?	De-fault	Inher-its?	Description
ns_prefix	manda-tory		Yes	Layer's Namespace Prefix used in WFS
ns_uri	manda-tory		Yes	Layer's Namespace URI used in WFS
name	manda-tory		No	Layer's Name
title	optional		No	Layer's Title
retriev-able	optional	false	Yes	If true, layer is retrievable on WFS <code>GetFeature</code> request
writable	optional	false	Yes	If true, layer is editable with WFS Transaction request
schema	optional	'pub-lic'	Yes	PostgreSQL Schema name.
table	optional		No	PostgreSQL table name (default is to use layer's name).
abstract	optional		No	Abstract text
keywords	optional		Yes	Keywords (comma separated list)
srid	optional		Yes	Comma separated list of output SRID
geobbox	optional		Yes	WGS-84 bbox of max extent: East,West,North,South
in-clude_items	optional		Yes	Comma separated list of columns to retrieve (only)
ex-clude_items	optional		Yes	Comma separated list of columns to not retrieve
pkey	optional		Yes	Column name to use as a Primary Key, when there's no PostgreSQL one (e.g usefull to use with VIEW)

```

<tinyows>
...
  <layer retrievable="1"
    writable="1"
    ns_prefix="tows"
    ns_uri="http://www.tinyows.org/"
    name="world"
    title="World Administrative Boundaries" />
...
</tinyows>

```

Nested Layers Layer entities could be nested, properties in this case are inherited. A Layer without title is then considered as a 'virtual' layer.

```

<tinyows>
...
  <layer name="root"
    retrievable="1" writable="1"
    ns_prefix="tows"
    ns_uri="http://www.tinyows.org/"
    schema="my_db_schema">

    <layer name="foo" title="foo" />
    <layer name="bar" title="bar" />

  </layer>

...
</tinyows>

```

10.1.3 Configuring TinyOWS with a standard Mapfile

Mapfile Config File support for TinyOWS

TinyOWS supports as a configuration file a standard MapServer Mapfile. This allow a single file to configure both MapServer and TinyOWS. (e.g could be usefull if you use them both, as one for WMS and the other as WFS-T)

TinyOWS does not handle all of the parameters in a Mapfile, but will ignore, without error, any extra parameters that are not implemented in TinyOWS.

If you prefer, you can configure TinyOWS using an XML file (*Configuring TinyOWS with an XML File*).

To indicate where your Mapfile is located, to TinyOWS binary, use the “TINYOWS_MAPFILE” environment variable.

Here an example of a single Mapfile:

```
MAP
  NAME "TinyOWS"

  WEB
    METADATA
      "tinyows_schema_dir" "/usr/local/share/tinyows/schema/"
      "tinyows_onlineresource" "127.0.0.1/cgi-bin/tinyows.fcgi"
      "wfs_title" "TinyOWS service provided by a MapFile"
      "wfs_contact" "foo@bar.net"
    END
  END

  LAYER
    NAME 'France'
    CONNECTIONTYPE postgis
    CONNECTION "host=127.0.0.1 user=postgres password=postgres dbname=tinyows_demo port=5432"
    METADATA
      'wfs_title' 'France'
      'wfs_namespace_prefix' 'tows'
      'wfs_namespace_uri' 'http://www.tinyows.org/'
      'wfs_srs' 'EPSG:27582'
      'tinyows_table' 'france'
      'tinyows_writable' '1'
      'tinyows_retrievable' '1'
    END
    DUMP TRUE
  END
END
```

Current concepts and limitations:

- Only the PostGIS CONNECTIONTYPE is handled
- TinyOWS does not support all of the WFS parameters available in a Mapfile. But on the other hand, you *are* able to configure every part of TinyOWS with a Mapfile.
- The CONNECTION string value in each layer must be the same.
- Mapfile PROJECTION content is not parsed, so use explicit wfs_srs.
- Mapfile LAYER and FILTER are not parsed.
- Default values are TinyOWS ones, even for common properties shared by both TinyOWS and MapServer.

- TinyOWS does not use DATA element from Mapfile, so you have to use tinyows_table (and tinyows_schema if needed) in each layer.
- If DUMP is not set to TRUE on a layer, both read and write access are disabled for the layer.

Mapfile path of each TinyOWS config element

Original TinyOWS XML Config File	Mapfile counterpart
/tinyows@online_resource	/map/metadata@tinyows_onlineresource
/tinyows@schema_dir	/map/metadata@tinyows_schema_dir
/tinyows@log	/map/metadata@tinyows_log
/tinyows@log_level	/map/metadata@tinyows_log_level
/tinyows@degree_precision	/map/metadata@tinyows_degree_precision
/tinyows@meter_precision	/map/metadata@tinyows_meter_precision
/tinyows@display_bbox	/map/metadata@tinyows_display_bbox
/tinyows@estimated_extent	/map/metadata@tinyows_estimated_extent
/tinyows@check_schema	/map/metadata@tinyows_check_schema
/tinyows@check_valid_geom	/map/metadata@tinyows_check_valid_geom
/tinyows@encoding	/map/metadata@wfs_encoding
/tinyows@db_encoding	/map/metadata@tinyows_db_encoding
/tinyows@expose_pk	/map/metadata@tinyows_expose_pk
/tinyows/limits@features	/map/metadata@wfs_maxfeatures
/tinyows/limits@geobbox	/map/metadata@tinyows_geobbox
/tinyows/pg@host	/map/layer@connection
/tinyows/pg@user	/map/layer@connection
/tinyows/pg@password	/map/layer@connection
/tinyows/pg@dbname	/map/layer@connection
/tinyows/pg@port	/map/layer@connection
/tinyows/pg@encoding	/map/metadata@tinyows_db_encoding
/tinyows/metadata@name	/map@name
/tinyows/metadata@title	/map/metadata@wfs_title
/tinyows/metadata@keywords	/map/metadata@wfs_keywordlist
/tinyows/metadata/abstract	/map/metadata@wfs_abstract
/tinyows/metadata@fees	/map/metadata@wfs_fees
/tinyows/metadata@access_constraints	/map/metadata@wfs_accessconstraints
/tinyows/layer@ns_prefix	/map/layer/metadata@wfs_namespace_prefix or /map/metadata@wfs_namespace_prefix
/tinyows/layer@ns_uri	/map/layer/metadata@wfs_namespace_uri or /map/metadata@wfs_namespace_uri
/tinyows/layer@name	/map/layer@name
/tinyows/layer@title	/map/layer/metadata@wfs_title
/tinyows/layer@retrievable	/map/layer/metadata@tinyows_retrievable and /map/layer@dump
/tinyows/layer@writable	/map/layer/metadata@tinyows_writable and /map/layer@dump
/tinyows/layer@schema	/map/layer/metadata@tinyows_schema
/tinyows/layer@keywords	/map/layer/metadata@wfs_keywordlist
/tinyows/layer/abstract	/map/layer/metadata@wfs_abstract
/tinyows/layer@srid	/map/metadata@wfs_srs and /map/layer/metadata@wfs_srs
/tinyows/layer@geobbox	/map/layer/metadata@tinyows_geobbox
/tinyows/layer@include_items	/map/layer/metadata@include_items
/tinyows/layer@exclude_items	/map/layer/metadata@exclude_items
/tinyows/layer@pkey	/map/layer/metadata@pkey
/tinyows/contact@name	/map/metadata@ows_contactorganization
/tinyows/contact@site	

Continued on next page

Table 10.1 – continued from previous page

Original TinyOWS XML Config File	Mapfile counterpart
/tinyows/contact@email	/map/metadata@ows_contactelectronicmailaddress
/tinyows/contact@individual_name	/map/metadata@ows_contactperson
/tinyows/contact@position	/map/metadata@ows_contactposition
/tinyows/contact@phone	/map/metadata@ows_contactvoicetelephone
/tinyows/contact@fax	/map/metadata@ows_contactfacsimiletelephone
/tinyows/contact@online_resource	
/tinyows/contact@address	/map/metadata@ows_address
/tinyows/contact@city	/map/metadata@ows_city
/tinyows/contact@administrative_area	
/tinyows/contact@country	/map/metadata@ows_country
/tinyows/contact@hours_of_service	
/tinyows/contact@contact_instructions	

10.1.4 Sample: WFS-T with TinyOWS and OpenLayers

0. Install PostGIS and TinyOWS (*TinyOWS Installation*)

1. Within PostGIS, create a spatial database called ‘tinyows’

```
createdb -U postgres tinyows
psql -U postgres -d tinyows < `pg_config --sharedir`/contrib/postgis-2.1/postgis.sql
psql -U postgres -d tinyows < `pg_config --sharedir`/contrib/postgis-2.1/spatial_ref_sys.sql
psql -U postgres -d tinyows < `pg_config --sharedir`/contrib/postgis-2.1/rtpostgis.sql
psql -U postgres -d tinyows < `pg_config --sharedir`/contrib/postgis-2.1/topology.sql
```

2. Import Frida data (we will use the parks layer) into your PostGIS database

```
wget ftp://ftp.intevation.de/freegis/frida/frida-1.0.1-shp.tar.gz
tar xvzf frida-1.0.1-shp.tar.gz
cd frida-1.0.1-shp
shp2pgsql -g geom -s 31467 -W LATIN1 -I gruenflaechen.shp frida | psql -U postgres -d tinyows
```

3. Configure TinyOWS by editing /usr/local/tinyows/config.xml

```
<tinyows online_resource="http://127.0.0.1/cgi-bin/tinyows"
  schema_dir="/usr/local/share/tinyows/schema/">

<pg host="127.0.0.1" user="postgres" password="postgres" dbname="tinyows" port="5432"/>

<metadata name="TinyOWS Server"
  title="TinyOWS Server - WFS-T Frida Service" />

<contact name="TinyOWS Server"
  site="http://www.tinyows.org/"
  email="tinyows-users@lists.maptools.org" />

<layer retrievable="1"
  writable="1"
  ns_prefix="tows"
  ns_uri="http://www.tinyows.org/"
  name="frida"
  title="Frida Parks" />

</tinyows>
```

4. Test your installations of TinyOWS and PostGIS

```
./YOUR_CGI-BIN_PATH/tinyows --check
TinyOWS version: 1.1.0
FCGI support: Yes
Config File Path: /etc/tinyows.xml (TinyOWS XML)
PostGIS Version: 2.0.1
PostGIS dsn: host=127.0.0.1 user=postgres password=postgres dbname=tinyows port=5432
Output Encoding: UTF-8
Database Encoding: UTF8
Schema dir: /usr/local/share/tinyows/schema/
Display bbox: Yes
Estimated extent: No
Check schema: Yes
Check valid geoms: Yes
Available layers:
- public.frida (31467) -> tows:frida [RW]
```

5. Install OpenLayers

```
wget http://openlayers.org/download/OpenLayers-2.12.tar.gz
tar xvzf OpenLayers-2.12.tar.gz
sudo mv OpenLayers-2.12 /YOUR/SERVER/HTDOCS/
```

6. Install the OpenLayers proxy (you need the Python interpreter to make it work)

```
sudo cp OpenLayers-2.12/examples/proxy.cgi /YOUR/SERVER/CGI-BIN/
```

7. Add your server IP to proxy allowedHosts in (/YOUR/SERVER/CGI-BIN/proxy.cgi)

```
allowedHosts = ['127.0.0.1', 'www.openlayers.org', 'openlayers.org', ... ]
```

8. Create a new file at OpenLayers-2.12/examples/tinyows.html

```
<html>
<head>
  <link rel="stylesheet" href="../theme/default/style.css" type="text/css" />
  <link rel="stylesheet" href="style.css" type="text/css" />
  <script src="../lib/OpenLayers.js"></script>
  <style>
    #map {
      width: 800px;
      height: 500px;
      float: left;
      border: 1px solid #ccc;
    }
    #message {
      position: relative;
      left: 5px;
    }
    #docs {
      float: left;
    }
    .customEditingToolbar {
      float: right;
      right: 0px;
      height: 30px;
      width: 200px;
    }
    .customEditingToolbar div {
```

```

    float: right;
    margin: 5px;
    width: 24px;
    height: 24px;
}
.olControlNavigationItemActive {
    background-image: url("../theme/default/img/editing_tool_bar.png");
    background-repeat: no-repeat;
    background-position: -103px -23px;
}
.olControlNavigationItemInactive {
    background-image: url("../theme/default/img/editing_tool_bar.png");
    background-repeat: no-repeat;
    background-position: -103px -0px;
}
.olControlDrawFeaturePolygonItemInactive {
    background-image: url("../theme/default/img/editing_tool_bar.png");
    background-repeat: no-repeat;
    background-position: -26px 0px;
}
.olControlDrawFeaturePolygonItemActive {
    background-image: url("../theme/default/img/editing_tool_bar.png");
    background-repeat: no-repeat;
    background-position: -26px -23px ;
}
.olControlModifyFeatureItemActive {
    background-image: url("../theme/default/img/move_feature_on.png");
    background-repeat: no-repeat;
    background-position: 0px 1px;
}
.olControlModifyFeatureItemInactive {
    background-image: url("../theme/default/img/move_feature_off.png");
    background-repeat: no-repeat;
    background-position: 0px 1px;
}
.olControlDeleteFeatureItemActive {
    background-image: url("../theme/default/img/remove_point_on.png");
    background-repeat: no-repeat;
    background-position: 0px 1px;
}
.olControlDeleteFeatureItemInactive {
    background-image: url("../theme/default/img/remove_point_off.png");
    background-repeat: no-repeat;
    background-position: 0px 1px;
}
}
</style>
<script src="tinyows.js"></script>
</head>
<body onload="init()" >
  <h1 id="title">WFS Transaction Example, (TinyOWS ans OpenLayers)</h1>
  <div id="tags"></div>
  <p id="shortdesc">
    Shows the use of the WFS Transactions (WFS-T).
    Parks of Osnabruck (Frida).
  <br />
    Base layers is OpenStreetMap from Omniscale WMS Server.
  </p>
  <div id="map"></div>

```

```

<div id="message"></div>
<div id="docs">
  <p>
    The WFS protocol allows for creation of new features and
    reading, updating, or deleting of existing features.
  </p>
  <p>
    Use the tools to create, modify, and delete (in order from left
    to right) features. Use the save tool (picture of a disk) to
    save your changes. Use the navigation tool (hand) to stop
    editing and use the mouse for map navigation.
  </p>
  <p>
    See the <a href="tinyows.js" target="_blank">
    wfs-protocol-transactions.js source</a> to see how this is done.
  </p>
</div>
</body>
</html>

```

9. Create a new file at OpenLayers-2.12/examples/tinyows.js (and replace all 127.0.0.1 addresses by your IP server if necessary)

```

var map, wfs;
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
var DeleteFeature = OpenLayers.Class(OpenLayers.Control, {
  initialize: function(layer, options) {
    OpenLayers.Control.prototype.initialize.apply(this, [options]);
    this.layer = layer;
    this.handler = new OpenLayers.Handler.Feature(
      this, layer, {click: this.clickFeature}
    );
  },
  clickFeature: function(feature) {
    // if feature doesn't have a fid, destroy it
    if(feature.fid == undefined) {
      this.layer.destroyFeatures([feature]);
    } else {
      feature.state = OpenLayers.State.DELETE;
      this.layer.events.triggerEvent("afterfeaturemodified",
        {feature: feature});
      feature.renderIntent = "select";
      this.layer.drawFeature(feature);
    }
  },
  setMap: function(map) {
    this.handler.setMap(map);
    OpenLayers.Control.prototype.setMap.apply(this, arguments);
  },
  CLASS_NAME: "OpenLayers.Control.DeleteFeature"
});
function showMsg(szMessage) {
  document.getElementById("message").innerHTML = szMessage;
  setTimeout(
    "document.getElementById('message').innerHTML = ''",2000);
}
function showSuccessMsg(){
  showMsg("Transaction successfully completed");
}

```

```
};
function showFailureMsg() {
    showMsg("An error occurred while operating the transaction");
};
function init() {
    map = new OpenLayers.Map('map', {
        projection: new OpenLayers.Projection("EPSG:31467"),
        units: "m",
        maxResolution: "auto",
        maxExtent: new OpenLayers.Bounds(3427000, 5788000, 3444000, 5800000),
        controls: [
            new OpenLayers.Control.PanZoom()
        ]
    });
    var osm = new OpenLayers.Layer.WMS(
        "OSM by Omniscale WMS",
        "http://osm.omniscale.net/proxy/service",
        {layers: 'osm', format: 'image/jpeg'},
        {projection:"EPSG:31467", units:"m", maxResolution: "auto", maxExtent: new OpenLayers.Bounds
    });
    var saveStrategy = new OpenLayers.Strategy.Save();
    saveStrategy.events.register("success", '', showSuccessMsg);
    saveStrategy.events.register("fail", '', showFailureMsg);
    wfs = new OpenLayers.Layer.Vector("Editable Features", {
        strategies: [new OpenLayers.Strategy.BBOX(), saveStrategy],
        projection: new OpenLayers.Projection("EPSG:31467"),
        protocol: new OpenLayers.Protocol.WFS({
            version: "1.1.0",
            srsName: "EPSG:31467",
            url: "http://127.0.0.1/cgi-bin/tinyows",
            featureNS : "http://www.tinyows.org/",
            featureType: "frida",
            geometryName: "geom",
            schema: "http://127.0.0.1/cgi-bin/tinyows?service=wfs&request=DescribeFeatureType&version="
        })
    });
    map.addLayers([osm, wfs]);
    var panel = new OpenLayers.Control.Panel(
        {'displayClass': 'customEditingToolbar'}
    );
    var navigate = new OpenLayers.Control.Navigation({
        title: "Pan Map"
    });
    var draw = new OpenLayers.Control.DrawFeature(
        wfs, OpenLayers.Handler.Polygon,
        {
            title: "Draw Feature",
            displayClass: "olControlDrawFeaturePolygon",
            multi: true
        }
    );
    var edit = new OpenLayers.Control.ModifyFeature(wfs, {
        title: "Modify Feature",
        displayClass: "olControlModifyFeature"
    });
    var del = new DeleteFeature(wfs, {title: "Delete Feature"});
    var save = new OpenLayers.Control.Button({
        title: "Save Changes",
```



```

trigger: function() {
    if(edit.feature) {
        edit.selectControl.unselectAll();
    }
    saveStrategy.save();
},
displayClass: "olControlSaveFeatures"
});
panel.addControls([navigate, save, del, edit, draw]);
panel.defaultControl = navigate;
map.addControl(panel);
map.zoomToMaxExtent();
}

```

10.1.5 Server Tuning: How to speed up your TinyOWS server

Tips and Tricks for PostgreSQL / PostGIS databases

- Use Spatial Indexes on your geometry/geography columns [PostGIS Spatial Indexes](#).
- Index any column that could be used frequently as a filter
- [General PostGIS Performance tips](#)
- [General PostgreSQL Performance tips](#)
- [Even more tips on Tuning PostgreSQL](#)

Tips and Tricks for Apache

Using Fast-CGI

- Check that your TinyOWS is compiled with FastCGI support:

```

[user@host mapserver]$ tinyows --check
TinyOWS version:  1.1.0
FCGI support:    Yes
...

```

Fast-CGI in Apache

- In Apache, activate `mod_fastcgi`

```

$ sudo apt-get install -y libapache2-mod-fastcgi
$ sudo a2enmod fastcgi

```

- Apache fast-cgi configuration:

```

#in your cgi-bin directive, add the following to run all cgi-bin using FastCGI
SetHandler fastcgi-script

#in your FastCGI config file (typically something like /etc/apache2/mods-enabled/fastcgi.conf)
FastCgiServer /usr/lib/cgi-bin/tinyows.fcgi -processes 10

```

Fast-CGI in MS4W

- Please refer to the [fastcgi doc in ms4w](#)
- Add the following 2 lines:

```
DefaultInitEnv TINYOWS_CONFIG_FILE "/ms4w/apps/tinyows/config.xml"  
DefaultInitEnv TINYOWS_SCHEMA_DIR "/ms4w/apps/tinyows/schema/"
```

HTTP GZip compression

- In Apache, activate `mod_deflate`
- Deflate basic configuration, (note we're including xml so gml and json):

```
AddOutputFilterByType DEFLATE text/html text/plain text/xml application/xml application/json
```

10.1.6 Working Around the LibXML2 XSD Schema GML Bug

Issue

TinyOWS makes use of GML, an XML-based language that encodes geometry. Frequently the input and the output of TinyOWS are in GML.

Even if LibXML2 is a great lib, older versions (i.e previous to 2.8) didn't handled correctly GML 3.1.1 XSD Schema (see https://bugzilla.gnome.org/show_bug.cgi?id=630130).

Workaround and options

For TinyOWS users, you have several options

- Take a recent libxml2 version (i.e 2.8 or later)
- Patch your oldest copy of LibXML2 release and link TinyOWS against your local copy:

```
--- xmlschemas.c.orig 2011-04-24 14:58:16.000000000 +0000  
+++ xmlschemas.c      2011-04-24 15:47:50.000000000 +0000  
@@ -15158,7 +15158,11 @@  
 }  
+ if ( (WXS_IS_LIST(type) || WXS_IS_UNION(type)) &&  
+      (WXS_IS_RESTRICTION(type) == 0) &&  
-      (! WXS_IS_ANY_SIMPLE_TYPE(baseType))) {  
+      (! WXS_IS_ANY_SIMPLE_TYPE(baseType))  
+      && (baseType->type != XML_SCHEMA_TYPE_SIMPLE)  
+      )  
+      ) {  
+      xmlSchemaPCustomErr(ctxt,  
+        XML_SCHEMAP_ST_PROPS_CORRECT_1,  
+        WXS_BASIC_CAST type, NULL,  
+      )  
+    }  
+  }  
+ }  
+ }
```

- Modify the XSD GML Schema itself (but you violate the OGC License if you do this!):

```

Index: schema/gml/3.1.1/base/valueObjects.xsd
=====
--- schema/gml/3.1.1/base/valueObjects.xsd (revision 550)
+++ schema/gml/3.1.1/base/valueObjects.xsd (revision 561)
@@ -200,11 +200,13 @@
     <group name="ValueExtent">
         <choice>
+<!--
             <element ref="gml:CategoryExtent"/>
             <element ref="gml:QuantityExtent"/>
+-->
             <element ref="gml:CountExtent"/>
         </choice>
     </group>
- <!-- =====
- <element name="QuantityExtent" type="gml:QuantityExtentType" substitutionGroup="gml:_Value">
+ <!-- =====
+ <element name="QuantityExtent" type="gml:QuantityExtentType" substitutionGroup="gml:_Value">
     <element name="QuantityExtent" type="gml:QuantityExtentType">
         <annotation>
@@ -212,5 +214,4 @@
         </annotation>
     </element>
- <!-- -->
     <complexType name="QuantityExtentType">
         <annotation>
@@ -223,6 +224,7 @@
         </simpleContent>
     </complexType>
+-->
     <!-- =====
- <element name="CategoryExtent" type="gml:CategoryExtentType" substitutionGroup="gml:_Value">
+ <element name="CategoryExtent" type="gml:CategoryExtentType" substitutionGroup="gml:_Value">
     <element name="CategoryExtent" type="gml:CategoryExtentType">
         <annotation>
@@ -230,5 +232,4 @@
         </annotation>
     </element>
- <!-- -->
     <complexType name="CategoryExtentType">
         <annotation>
@@ -241,4 +242,5 @@
         </simpleContent>
     </complexType>
+-->
     <!-- =====
     <element name="CountExtent" type="gml:CountExtentType" substitutionGroup="gml:_Value"> -->

```

- Or choose to use only GML 2.1.2 in the meantime.

See also:

[Developer documentation](#)

(French) [Utiliser TinyOWS comme serveur WFS-T](#)

11.1 Optimization

11.1.1 Debugging MapServer

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Last Updated 2013-07-02

Table of Contents

- *Debugging MapServer*
 - *Introduction*
 - * *Links to Related Information*
 - *Steps to Enable MapServer Debugging*
 - * *Step 1: Set the MS_ERRORFILE Variable*
 - * *Step 2: Set the DEBUG Level*
 - * *Step 3: Turn on CPL_DEBUG (optional)*
 - * *Step 4: Turn on PROJ_DEBUG (optional)*
 - * *Step 5: Test your Mapfile*
 - * *Step 6: Check your Web Server Logs*
 - * *Step 7: Verify your Application Settings*
 - *Debugging MapServer using Compiler Debugging Tools*
 - * *Running MapServer in GDB (Linux/Unix)*
 - *Debugging Older Versions of MapServer (before 5.0)*

Introduction

When developing an application for the Internet, you will inevitably across problems many problems in your environment. The goal of this guide is to assist you with locating the problem with your MapServer application.

Links to Related Information

- RFC 28: Redesign of LOG/DEBUG output mechanisms
- *MapServer Errors*

Steps to Enable MapServer Debugging

Starting with MapServer 5.0, you are able to control the levels of debugging/logging information returned to you by MapServer, and also control the location of the output log file.

In technical terms, there are `msDebug()` calls in various areas of the MapServer code that generate information that may be useful in tuning and troubleshooting applications.

Step 1: Set the `MS_ERRORFILE` Variable

The `MS_ERRORFILE` variable is used to specify the output of debug messages from MapServer. You can pass the following values to `MS_ERRORFILE`:

[filename] Full path and filename of a log file, to contain MapServer's debug messages. Any file extension can be used, but `.log` or `.txt` is recommended. The file will be created, if it does not already exist.

Starting with MapServer 6.0, a filename with relative path can be passed via the `CONFIG MS_ERRORFILE` directive, in which case the filename is relative to the mapfile location. Note that setting `MS_ERRORFILE` via an environment variable always requires an absolute path since there would be no mapfile to make the path relative to.

stderr Use this to send MapServer's debug messages to the Web server's log file (i.e. "standard error"). If you are using Apache, your debug messages will be placed in the Apache `error_log` file. If you are using Microsoft IIS, your debug messages will be sent to `stdout` (i.e. the browser), so its use is discouraged. With IIS it is recommended to direct output to a file instead.

stdout Use this to send MapServer's debug messages to the standard output (i.e. the browser), combined with the rest of MapServer's output.

windowsdebug Use this to send MapServer's debug messages to the Windows `OutputDebugString` API, allowing the use of external programs like SysInternals `debugview` to display the debug output.

Through the Mapfile The recommended way to set the `MS_ERRORFILE` variable is in your mapfile, within the `MAP` object, such as:

```
MAP
...
CONFIG "MS_ERRORFILE" "/ms4w/tmp/ms_error.txt"
...
LAYER
...
END
END
```

Through an Environment Variable You can also set the `MS_ERRORFILE` variable as an environment variable on your system. Apache users can set the environment variable in Apache's `httpd.conf` file, such as:

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

Windows users can alternatively set the environment variable through the Windows System Properties; but make sure to set a `SYSTEM` environment variable.

Note: If both the `MS_ERRORFILE` environment variable is set and a `CONFIG MS_ERRORFILE` is also set, then the `CONFIG` directive takes precedence.

Step 2: Set the DEBUG Level

You can retrieve varying types of debug messages by setting the *DEBUG* parameter in the *Mapfile*. You can place the *DEBUG* parameter in any *LAYER* in the mapfile for layer-specific debug information, or instead, set it once in the *MAP* object to get general debug information. Use the value of the *DEBUG* parameter to set the type of information returned, as follows:

DEBUG Levels

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. *this is the recommended minimal debugging level*

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 developers than to users.

Mapfile Example: Map-Level Debug The following example is the recommended method to set the *DEBUG* parameter for the map-level:

```
MAP
...
CONFIG "MS_ERRORFILE" "/ms4w/tmp/ms_error.txt"
DEBUG 5
...
LAYER
...
END
END
```

Mapfile Example: Layer-Level Debug The following example is the recommended method to set the *DEBUG* parameter for a layer:

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

```
    DEBUG 5
    ...
  END
END
```

The MS_DEBUGLEVEL Environment Variable Instead of setting the *DEBUG* Debug level in each of your mapfiles, you can also be set the level globally by using the *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.

Apache users can set the environment variable in Apache's *httpd.conf* file, such as:

```
SetEnv MS_DEBUGLEVEL 5
```

Windows users can alternatively set the environment variable through the Windows System Properties; but make sure to set a *SYSTEM* environment variable.

Step 3: Turn on CPL_DEBUG (optional)

MapServer relies on the *GDAL* library to access most data layers, so you may wish to turn on *GDAL* debugging, to hopefully get more information on how *GDAL* is accessing your data file. This could be very helpful for problems with accessing raster files and *PostGIS* tables. You can trigger this *GDAL* output by setting the *CPL_DEBUG* variable in your mapfile, within the *MAP* object, such as:

```
MAP
  ...
  CONFIG "CPL_DEBUG" "ON"
  ...
  LAYER
    ...
  END
END
```

Step 4: Turn on PROJ_DEBUG (optional)

MapServer relies on the *PROJ.4* library to handle data projections, so you may wish to turn on *PROJ* debugging, to hopefully get more information back from the *PROJ* library. You can trigger this *PROJ* output by setting the *PROJ_DEBUG* variable in your mapfile, within the *MAP* object, such as:

```
MAP
  ...
  CONFIG "PROJ_DEBUG" "ON"
  ...
  LAYER
    ...
  END
END
```


Step 5: Test your Mapfile

Once you have set the `MS_ERRORFILE` and `DEBUG` level in your mapfile, you should now test your mapfile and read your generated log file.

Using `shp2img` The recommended way to test your mapfile is to use the MapServer commandline utility `shp2img`, to verify that your mapfile creates a valid map image. `shp2img` should be included in your MapServer installation (MS4W users need to execute `setenv.bat` before using the utility).

You can set the `DEBUG` level by passing the `shp2img` following parameters to your commandline call:

Note: If you have already set `MS_ERRORFILE` in your mapfile, you must comment this out in order to use these `shp2img` options

Note: When using `shp2img` to debug, your layer's STATUS should be set to ON or DEFAULT. If the layer's STATUS is set to OFF, you must additionally pass the layer name to `shp2img` by using the `"-l layername"` syntax

-all_debug Use this setting to set the debug level for the MAP object and all layers. *this is the recommended switch to use*

```
shp2img -m spain.map -o test.png -all_debug 5

msLoadMap(): 0.002s
msDrawMap(): Layer 0 (spain provinces), 0.012s
msDrawRasterLayerLow(orthophoto): entering.
msDrawGDAL(): src=0,0,3540,2430, dst=188,48,1,1
source raster PL (-793.394,-1712.627) for dst PL (188,48).
msDrawGDAL(): red,green,blue,alpha bands = 1,2,3,0
msDrawMap(): Layer 1 (orthophoto), 0.150s
msDrawMap(): Layer 2 (urban areas), 0.004s
msDrawMap(): Layer 3 (species at risk), 0.008s
msDrawMap(): Layer 4 (populated places), 1.319s
msDrawMap(): Drawing Label Cache, 0.014s
msDrawMap() total time: 1.513s
msSaveImage() total time: 0.039s
msFreeMap(): freeing map at 0218C1A8.
freeLayer(): freeing layer at 0218F5E0.
freeLayer(): freeing layer at 030C33A0.
freeLayer(): freeing layer at 030C3BC8.
freeLayer(): freeing layer at 030C4948.
freeLayer(): freeing layer at 030C7678.
shp2img total time: 1.567s
```

-map_debug Use this setting to set the debug level for the MAP object only.

```
shp2img -m spain.map -o test.png -map_debug 5

msDrawMap(): Layer 0 (spain provinces), 0.012s
msDrawRasterLayerLow(orthophoto): entering.
msDrawMap(): Layer 1 (orthophoto), 0.144s
msDrawMap(): Layer 2 (urban areas), 0.004s
msDrawMap(): Layer 3 (species at risk), 0.008s
msDrawMap(): Layer 4 (populated places), 1.323s
msDrawMap(): Drawing Label Cache, 0.013s
```

```
msDrawMap() total time: 1.511s
msSaveImage() total time: 0.039s
msFreeMap(): freeing map at 0205C1A8.
```

-layer_debug Use this setting to set the debug level for one layer object only.

```
shp2img -m spain.map -o test.png -layer_debug orthophoto 5

msDrawRasterLayerLow(orthophoto): entering.
msDrawGDAL(): src=0,0,3540,2430, dst=188,48,1,1
source raster PL (-793.394,-1712.627) for dst PL (188,48).
msDrawGDAL(): red,green,blue,alpha bands = 1,2,3,0
msDrawMap(): Layer 1 (orthophoto), 0.151s
freeLayer(): freeing layer at 02F23390.
```

Set CPL_DEBUG At the commandline execute the following:

```
set CPL_DEBUG=ON

shp2img -m spain.map -o test.png -layer_debug orthophoto 5

msDrawRasterLayerLow(orthophoto): entering.
GDAL: GDALOpen(D:\ms4w\apps\spain\map\..\data\ov172068_200904_c100u50x75c24n.jpg, this=0
4059840) succeeds as JPEG.
msDrawGDAL(): src=0,0,3540,2430, dst=188,48,1,1
source raster PL (-793.394,-1712.627) for dst PL (188,48).
msDrawGDAL(): red,green,blue,alpha bands = 1,2,3,0
GDAL: GDALDefaultOverviews::OverviewScan()
msDrawMap(): Layer 1 (orthophoto), 0.155s
freeLayer(): freeing layer at 03113390.
GDAL: GDALDeregister_GTIff() called.
```

Reading Errors Returned by shp2img If there is a problem with your mapfile, *shp2img* should output the line number in your mapfile that is causing the trouble. The following tells us that there is a problem on line 85 of my mapfile:

```
getSymbol(): Symbol definition error. Parsing error near (truetype2):(line 85)
```

If you are using mapfile *INCLUDEs*, it may be tricky to track down this line number, but most of the time the line number is useful.

Using mapserv CGI Another handy way to test your mapfile is to call the mapserv CGI executable at the *commandline*, such as the following:

```
mapserv -nh "QUERY_STRING=map=/ms4w/apps/spain/map/spain.map&mode=map"
```

ON_MISSING_DATA If you are using tile indexes to access your data, you should also be aware of the configuration settings added in MapServer 5.4 that allow you to tell MapServer how to handle missing data in tile indexes. Please see the *CONFIG* parameter's *ON_MISSING_DATA* setting in the *MAP* object for more information.

Hint: You can check the attributes in the tileindex by executing “*ogrinfo -al*” on your data file

Step 6: Check your Web Server Logs

Once you have verified that there are no problems with you mapfile, next you should check your Web server log files, for any related information that may help you narrow down your problem.

Apache Unix users will usually find Apache's *error_log* file in a path similar to:

```
/var/log/apache2/
```

Windows users will usually find Apache's log files in a path similar to:

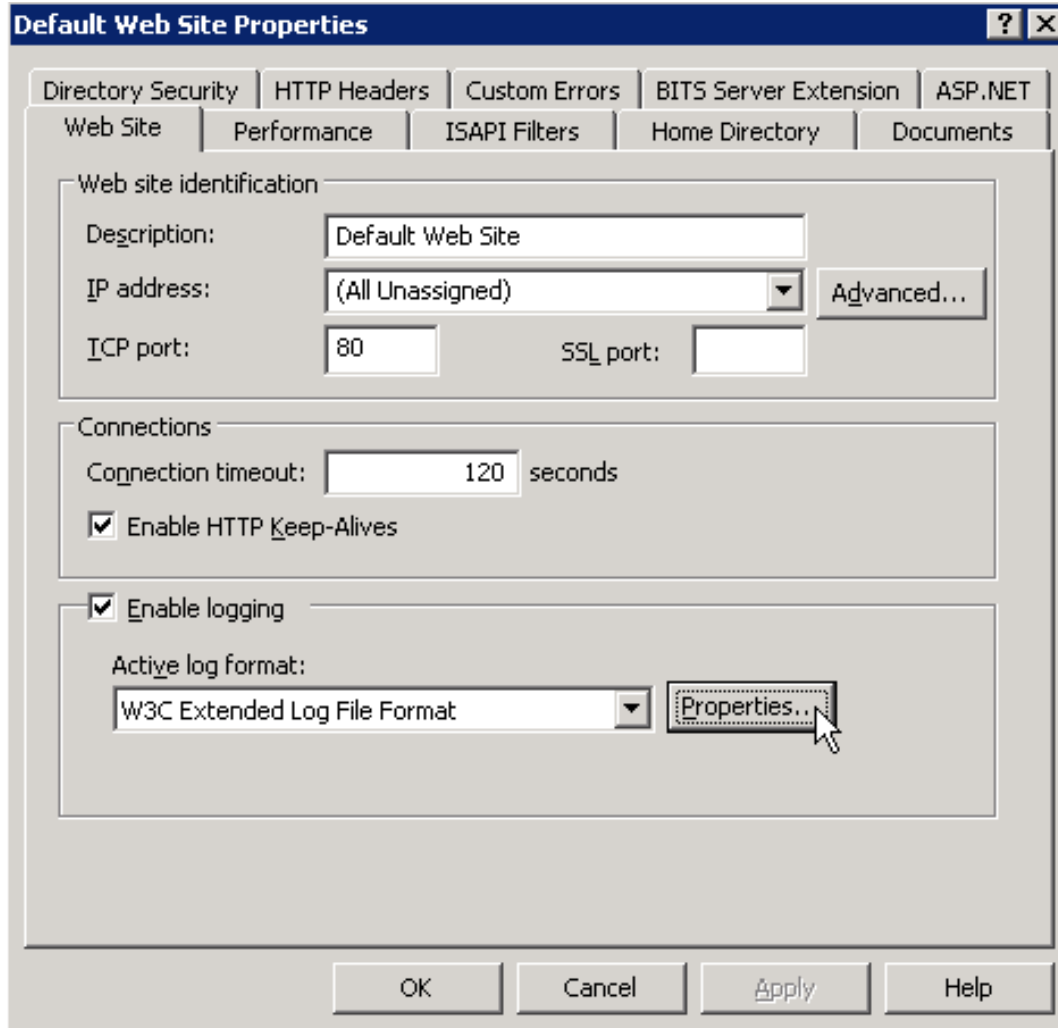
```
C:\Program Files\Apache Group\Apache2\logs
```

MapServer for Windows (**MS4W**) users will find Apache's log files at:

```
\ms4w\Apache\logs
```

Microsoft IIS IIS log files can be located by:

1. Go to Start -> Control Panel -> Administrative Tools
2. Open the Internet Information Services (IIS) Manager.
3. Find your Web site under the tree on the left.
4. Right-click on it and choose Properties.
5. On the Web site tab, you will see an option near the bottom that says "Active Log Format." Click on the Properties button.



6. At the bottom of the General Properties tab, you will see a box that contains the log file directory and the log file name. The full log path is comprised of the log file directory plus the first part of the log file name, for example:

```
C:\WINDOWS\system32\LogFiles\W3SVC1\ex100507.log
```

You may also want to check the Windows Event Viewer logs, which is located at:

1. Go to Start -> Control Panel -> Administrative Tools
2. Computer Management
3. Event Viewer

Warning: As mentioned previously, in IIS the MapServer *stderr* debug output is returned to the client instead of routed to the Web Server logs, so be sure to log the output to a file, using:

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

CGI Error - The specified CGI application misbehaved by not returning a complete set of HTTP headers This error is often caused by missing DLL files. You should try to execute *mapserv -v* at the commandline, to make sure

that MapServer loads properly.

Step 7: Verify your Application Settings

If you have verified that MapServer creates a valid map image through *shp2img*, you've checked your MapServer log files, and there are no problems noted in your Web server logs, then you should focus your attention on possible application configuration problems. "Application" here means how you are displaying your map images on the Web page, such as with *OpenLayers*.

PHP MapScript If you are using PHP MapScript in your application, here are some important notes for debugging:

1. Make sure your *php.ini* file is configured to show all errors, by setting:

```
display_errors = On
```

2. To enable debugging in PHP MapScript, if you are using MapServer 5.6.0 or more recent, make sure to define *ZEND_DEBUG* in the PHP source.

If you are using MapServer < 5.6.0, then:

- open the file */mapscript/php3/php_mapscript.c*
- change the following:

```
#define ZEND_DEBUG 0

to

#define ZEND_DEBUG 1
```

Debugging MapServer using Compiler Debugging Tools

Running MapServer in GDB (Linux/Unix)

Section author: Frank Warmerdam

Building with Symbolic Debug Info It is not strictly necessary to build MapServer with debugging enabled in order to use *GDB* on linux, but it does ensure that more meaningful information is reported within *GDB*. To enable full symbolic information use the *-enable-debug* configure switch. Note that use of this switch disables optimization and so it should not normally be used for production builds where performance is important.

```
./configure --enable-debug <other switches>
make clean
make
```

Running in the Debugger To run either *mapserv* or *shp2img*, give the name of the executable as an argument to the "gdb" command. If it is not in the path, you will need to provide the full path to the executable.

```
gdb shp2img
GNU gdb (GDB) 7.0-ubuntu
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
```

```
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /wrk/home/warmerda/mapserver/shp2img...done.
(gdb)
```

Once you are at the “(gdb)” prompt you can use the run command with the arguments you would normally have passed to the mapserv or shp2img executable.

```
(gdb) run -m test.map -o out.png
Starting program: /wrk/home/warmerda/mapserver/shp2img -m test.map -o out.png
[Thread debugging using libthread_db enabled]

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff67594a2 in JP2KAKDataset::Identify (poOpenInfo=0x0)
    at jp2kakdataset.cpp:962
    962             if( poOpenInfo->nHeaderBytes < (int) sizeof(jp2_header) )
Current language: auto
The current source language is "auto; currently c++".
(gdb)
```

If the program is crashing, you will generally get a report like the above indicating the function the crash occurred in, and some minimal information on why. It is often useful to request a traceback to see what functions led to the function that crashed. For this use the “where” command.

```
(gdb) where
#0  0x00007ffff67594a2 in JP2KAKDataset::Identify (poOpenInfo=0x0)
    at jp2kakdataset.cpp:962
#1  0x00007ffff67596d2 in JP2KAKDataset::Open (poOpenInfo=0x7ffffffffffb6f0)
    at jp2kakdataset.cpp:1025
#2  0x00007ffff6913339 in GDALOpen (
    pszFilename=0x83aa60 "/home/warmerda/data/jpeg2000/spaceimaging_16bit_rgb.jp
    2", eAccess=GA_ReadOnly) at gdaldataset.cpp:2170
#3  0x00007ffff69136bf in GDALOpenShared (
    pszFilename=0x83aa60 "/home/warmerda/data/jpeg2000/spaceimaging_16bit_rgb.jp
    2", eAccess=GA_ReadOnly) at gdaldataset.cpp:2282
#4  0x0000000000563c2d in msDrawRasterLayerLow (map=0x81e450, layer=0x839140,
    image=0x83af90, rb=0x0) at mapraster.c:566
#5  0x000000000048928f in msDrawRasterLayer (map=0x81e450, layer=0x839140,
    image=0x83af90) at mapdraw.c:1390
#6  0x0000000000486a48 in msDrawLayer (map=0x81e450, layer=0x839140,
    image=0x83af90) at mapdraw.c:806
#7  0x00000000004858fd in msDrawMap (map=0x81e450, querymap=0) at mapdraw.c:459
#8  0x0000000000446410 in main (argc=5, argv=0x7ffffffffffd918) at shp2img.c:300
(gdb)
```

It may also be helpful to examine variables used in the line where the crash occurred. Use the print command for this.

```
(gdb) print poOpenInfo
$1 = (GDALOpenInfo *) 0x0
```

In this case we see that the program crashed because poOpenInfo was NULL (zero). Including a traceback like the above in bug report can help the developers narrow down a problem more quickly, especially if it is one that is difficult for the developers to reproduce themselves.

Debugging Older Versions of MapServer (before 5.0)

1. Make sure that MapServer is compiled in debug mode (on unix this is enabled through `./configure --enable-debug`).

You can verify that your build was compiled in debug mode, by executing the following at the commandline (look for “DEBUG=MSDEBUG”):

```
./mapserv -v

MapServer version 4.10.2 OUTPUT=GIF OUTPUT=PNG OUTPUT=WBMP
OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER
SUPPORTS=WMS_CLIENT SUPPORTS=WCS_SERVER SUPPORTS=THREADS SUPPORTS=GEOS
INPUT=EPPL7 INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
DEBUG=MSDEBUG
```

2. Set the `MS_ERRORFILE` variable in your mapfile, within the `MAP` object, such as:

```
MAP
...
CONFIG "MS_ERRORFILE" "/ms4w/tmp/ms_error.txt"
...
LAYER
...
END
END
```

3. If you don't use the `MS_ERRORFILE` variable, you can use the `LOG` parameter in your `WEB` object of the mapfile, such as:

```
MAP
...
WEB
  LOG "mapserver.log"
END
...
LAYER
...
END
END
```

4. Specify `DEBUG ON` in your `MAP` object, or in your `LAYER` objects, such as:

```
MAP
...
WEB
  LOG "mapserver.log"
END
DEBUG ON
...
LAYER
...
END
END
```

5. Note that only errors will be written to the log file; all `DEBUG` output goes to `stderr`, in the case of Apache that is Apache's `error_log` file. If you are using Microsoft IIS, debug output is routed to `stdout` (i.e. the browser), so be sure to remove `DEBUG ON` statements if using IIS on a production server.

11.1.2 FastCGI

Author Frank Warmerdam

Contact warmerdam at pobox.com

Author Howard Butler

Contact hobu.inc at gmail.com

Revision \$Revision\$

Date \$Date\$

Last Updated 2008/07/15

Table of Contents

- *FastCGI*
 - *Introduction*
 - *Obtaining the necessary software*
 - *mod_fcgid Configuration*
 - *Deprecated mod_fcgi Configuration*
 - *Common mod_fcgid/mod_fcgi Configuration*
 - *Common Problems*
 - *FastCGI on Win32*

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.

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 the `fastcgi` module. There are two `fastcgi` implementations for `apache`:
 - *mod_fcgid*: `mod_fcgid` is the newer and recommended way to run `fastcgi` programs under recent `apache` versions. It can be downloaded from the [Apache fcgid homepage](#)
 - **deprecated** *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/local
```

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-6.2.0]# ./mapserv -v
MapServer version 6.2.0 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
```

mod_fcgid Configuration

1. Modify http.conf to load the FastCGI module.

```
LoadModule fcgid_module modules/mod_fcgid.so
```

2. Add an Apache handler for FastCGI applications.

```
AddHandler fcgid-script fcgi
```

3. Set FastCGI processing information

```
<IfModule mod_fcgid.c>
  FcgidMaxProcessesPerClass 30
  FcgidInitialEnv PROJ_LIB /usr/local/share/proj
  FcgidInitialEnv LD_LIBRARY_PATH "/usr/local/lib:/usr/local/pgsql/lib:/usr3/pkg3/oracle9/lib"
</IfModule>
```

Deprecated mod_fcgi Configuration

This section is left for reference. The recommended way to run fastcgi programs under apache is to use mod_fcgid, as detailed above.

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
```

Common mod_fcgid/mod_fcgi Configuration

1. Install a copy of the mapserv executable (originally **mapserv** or **mapserv.exe**) into the cgi-bin directory with the extension **.fcgi** (ie. **mapserv.fcgi**). Use this executable when you want to utilize fastcgi support.

For some platforms, the MapServer link would then have to be changed from:

```
http://your.domain.name/cgi-bin/mapserv?MAP=/path/to/mapfile.map
```

To:

```
http://your.domain.name/cgi-bin/mapserv.fcgi?MAP=/path/to/mapfile.map
```

For other platforms, the MapServer link would then have to be changed from:

```
http://your.domain.name/cgi-bin/mapserv.exe?MAP=/path/to/mapfile.map
```

To:

```
http://your.domain.name/cgi-bin/mapserv.fcgi?MAP=/path/to/mapfile.map
```

2. 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"
```

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.

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 the 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.

11.1.3 Mapfile

Author David Fawcett

Contact david.fawcett at gmail.com

Revision \$Revision\$

Date \$Date\$

Last Updated 2007/08/01

Table of Contents

- *Mapfile*
 - *Introduction*

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.

11.1.4 Raster

Author HostGIS

Revision \$Revision\$

Date \$Date\$

Last Updated 2008/08/08

Table of Contents

- *Raster*
 - *Overviews*
 - *Tileindexes and Internal Tiling*
 - *Image formats*
 - *Remote WMS*

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.

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
```

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.

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

11.1.5 Tile Indexes

Author HostGIS

Revision \$Revision\$

Date \$Date\$

Last Updated 2013/07/04

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*
 - *Tileindexes with tiles in different projections*

Introduction

An introduction to tileindexes, MapServer's method for doing on-the-fly mosaicing.

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 `ogrtindex` 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:

```
ogrtindex streets.shp tiger/CA/*.shp tiger/NV/*.shp
```

Note: `ogrtindex` and `gdaltindex` **add** the specified files to the index. Sometimes you'll have to delete the index file to avoid creating duplicate entries.

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 *ogrtindex*.

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 *ogrtindex* 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 *ogrtindex/gdaltindex* from there; this ensures that I specify the correct pathnames.

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.

Tileindexes with tiles in different projections

Starting with MapServer 6.4, a raster tileindex can contain rasters in different projections. Such tileindex can be generated with *gdaltindex* (GDAL 1.11 or later), with the *-t_srs* and *-src_srs_name* options. The *-t_srs* instructs *gdaltindex* to write the envelope of each raster tile into a common target projection, so that the geometries written in the tile index are consistent. This common projection must be the projection of the raster layer.

```
gdaltindex -t_srs EPSG:4326 -src_srs_name src_srs imagery.shp imagery/*.tif
```

The corresponding *LAYER* definition will need to specify the *TILESRS* keyword with the value of the *-src_srs_name* option.

e.g:

```
LAYER
  NAME "My Imagery"
  STATUS ON
  TYPE RASTER
  TILEINDEX "imagery.shp"
  TILEITEM "LOCATION"
  TILESRS "src_srs"
  PROJECTION
    AUTO
    # or :
    # "+init=EPSG:4326"
  END
END
```

MapServer will then be able to proceed to on-the-fly mosaicing and reprojection.

For layers that must be exposed as WCS layers, a few metadata fields (“wcs_extent”, “wcs_size”, “wcs_resolution”) must be specified in the LAYER definition, so as to define a “virtual dataset” coverage (see *WCS Server*). The GDAL `wcs_virtlds_params.py` sample script can help generating those metadata fields.

Note: this support of tileindex with mixed projections is only available for raster layers for now.

11.1.6 Vector

Author HostGIS

Revision \$Revision\$

Date \$Date\$

Last Updated 2008/08/08

Table of Contents

- *Vector*
 - *Splitting your data*
 - *Shapefiles*
 - *PostGIS*
 - *Databases in General (PostGIS, Oracle, MySQL)*

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.

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.

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](#).

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.

12.1 Utilities

12.1.1 legend

Purpose

Creates a legend from a mapfile. Output format depends on the graphics library used for rendering.

Syntax

```
legend [mapfile] [output image]
```

12.1.2 msencrypt

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
```

Syntax

To create a new encryption key:

```
msencrypt -keygen [key_filename]
```

To encrypt a string:

```
msencrypt -key [key_filename] [string_to_encrypt]
```

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
```

2. 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
```

3. 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
```

4. 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"
```

5. Done! Give your new encrypted mapfile a try with the *shp2img* utility!

12.1.3 scalebar

Purpose

Creates a scalebar from a mapfile. Output is either PNG or GIF depending on what version of the GD library used.

Syntax

```
scalebar [mapfile] [output image]
```

12.1.4 shp2img

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.

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
```

12.1.5 shptree

Purpose

Creates a quadtree-based spatial index for a Shape data set. 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.

Description

This utility is a must for any MapServer application that uses Shape data sets. Shptree creates a spatial index of your Shape data set, 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 Shape data set, 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.

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)

Mapfile Notes

Shape data sets 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

12.1.6 shptreetst

Purpose

Executes a spatial query on an existing spatial index (.qix), that was created by the *shptree* utility. This utility is useful to understand how a search of a Shape data set and its *qix* index works.

Syntax

```
shptreetst shapefile {minx miny maxx maxy}
```

Example

```
shptreetst esp 879827.480246 4317203.699447 884286.289767 4321662.508967

This new LSB index supports a shapefile with 48 shapes, 4 depth
shapes 6, node 4, -13702.315770,3973784.599548,1127752.921471,4859616.714055
shapes 5, node 3, -13702.315770,3973784.599548,614098.064712,4460992.262527
shapes 1, node 1, -13702.315770,3973784.599548,331587.893495,4241748.814186
shapes 1, node 0, 141678.278400,3973784.599548,331587.893495,4121164.917599
shapes 1, node 0, 268807.855447,4193028.047889,614098.064712,4460992.262527
shapes 1, node 0, 268807.855447,3973784.599548,614098.064712,4241748.814186
shapes 7, node 4, -13702.315770,4372409.051076,614098.064712,4859616.714055
shapes 1, node 0, -13702.315770,4372409.051076,331587.893495,4640373.265714
shapes 3, node 1, -13702.315770,4591652.499417,331587.893495,4859616.714055
shapes 1, node 0, -13702.315770,4712236.396004,176207.299326,4859616.714055
shapes 2, node 0, 268807.855447,4372409.051076,614098.064712,4640373.265714
shapes 3, node 2, 268807.855447,4591652.499417,614098.064712,4859616.714055
shapes 2, node 0, 424188.449617,4712236.396004,614098.064712,4859616.714055
shapes 1, node 0, 424188.449617,4591652.499417,614098.064712,4739032.817468
shapes 2, node 1, 499952.540988,3973784.599548,1127752.921471,4460992.262527
shapes 2, node 0, 499952.540988,4193028.047889,845242.750254,4460992.262527
shapes 5, node 3, 499952.540988,4372409.051076,1127752.921471,4859616.714055
shapes 1, node 1, 499952.540988,4372409.051076,845242.750254,4640373.265714
shapes 1, node 0, 655333.135158,4372409.051076,845242.750254,4519789.369127
shapes 1, node 0, 499952.540988,4591652.499417,845242.750254,4859616.714055
read entire file now at quad box rec 20 file pos 1084
result of rectangle search was
8, 10, 36, 37, 38, 39, 42, 46,
```

Result The above output from the *shptreetst* command tells us that the existing *.qix* index is for a Shape data set that contains 48 shapes; indeed the Shape data set used in this example, *esp.shp*, contains 48 polygons of Spain. The command also tells us that *qix* file has a quadtree depth of 4.

Most importantly, the resulting shape IDs (or feature IDs) that were contained in the bounding box that we passed in our example were returned at the bottom of the output: “8, 10, 36, 37, 38, 39, 42, 46”. You can use a tool such as QGIS to view those feature IDs and check what shapes MapServer is querying when a user clicks within that bounding box.

.index:: pair: Utility; shptreevis

12.1.7 shptreevis

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).

Syntax

```
shptreevis shapefile new_shapefile
```

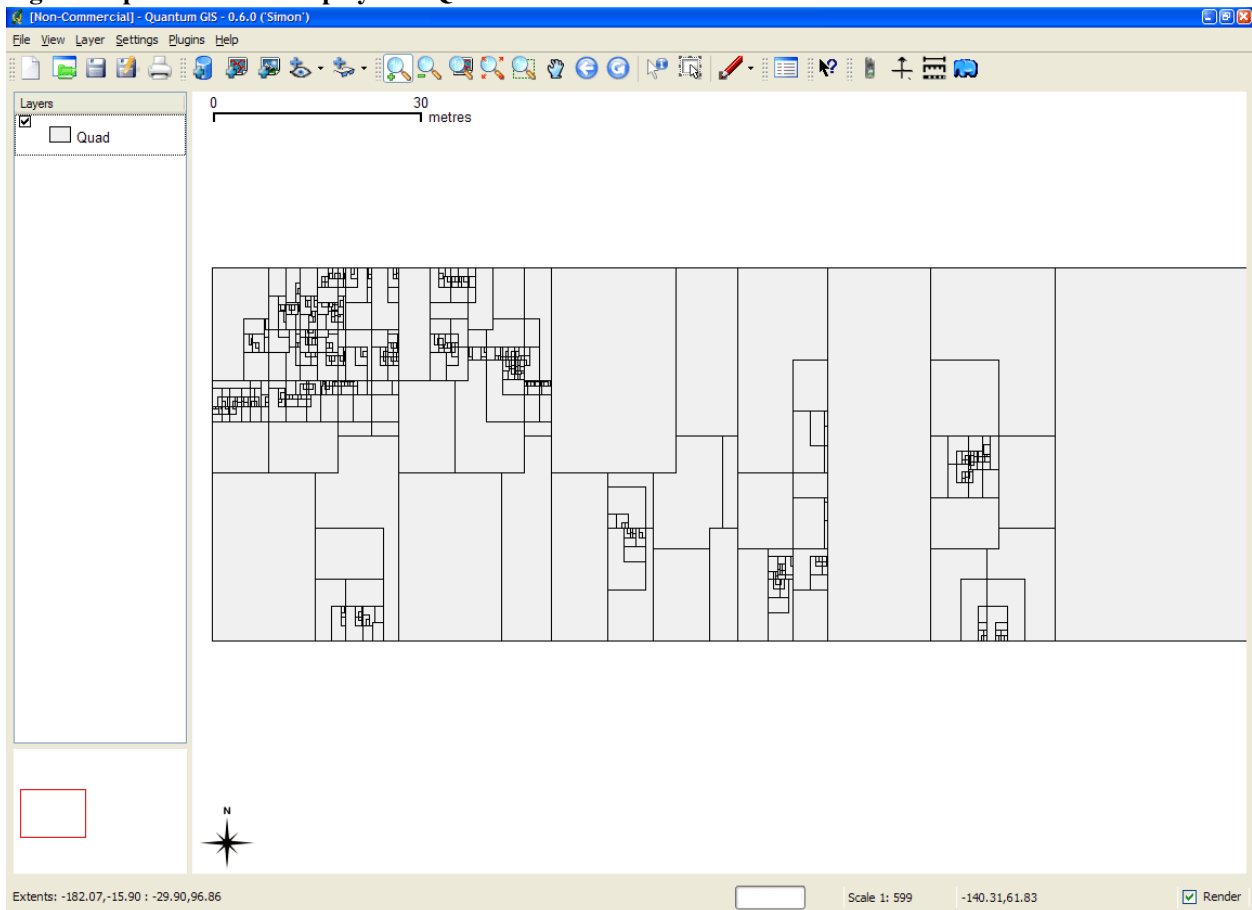
Example

```
shptreevis us_states.shp quad.shp
```

This new LSB index supports a shapefile with 2895 shapes, 10 depth

Result A Shape data set named 'quad.shp' is created. You can now view this Shape data set 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



12.1.8 sortshp

Purpose Sorts a Shape data set 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 Shape data set 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 Interstate		496
2		2 Urban Interstate		496
3		2 Urban Interstate		94
4		2 Urban Interstate		94
5		2 Urban Interstate		475
6		1 Rural Interstate		96
7		1 Rural Interstate		96
8		2 Urban Interstate		475
9		1 Urban Interstate		475
10		1 Urban Interstate		696
11		2 Urban Interstate		196
12		2 Urban Interstate		196
13		2 Urban Interstate		1 35E
14		2 Urban Interstate		1 35E

12.1.9 sym2img

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](#))

Syntax

```
sym2img [symbolfile] [outfile]
```

12.1.10 tile4ms

Purpose

Creates a tile index Shape data set for use with MapServer's TILEINDEX feature. The program creates a Shape data set of rectangles from extents of all the Shape data sets listed in [metafile] (one Shape data set name per line) and the associated DBF with the filename for each shape tile in a column called LOCATION as required by mapserv.

Note: Similar functionality can be found in the GDAL commandline utilities `ogrindex` (for vectors) and `gdalindex` (for rasters).

Description

This utility creates a Shape data set 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).

Syntax

```
tile4ms <meta-file> <tile-file> [-tile-path-only]
<meta-file>      INPUT   file containing list of Shape data set 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.
```

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
```

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 Shape 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
...
```

2. 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
```

3. 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

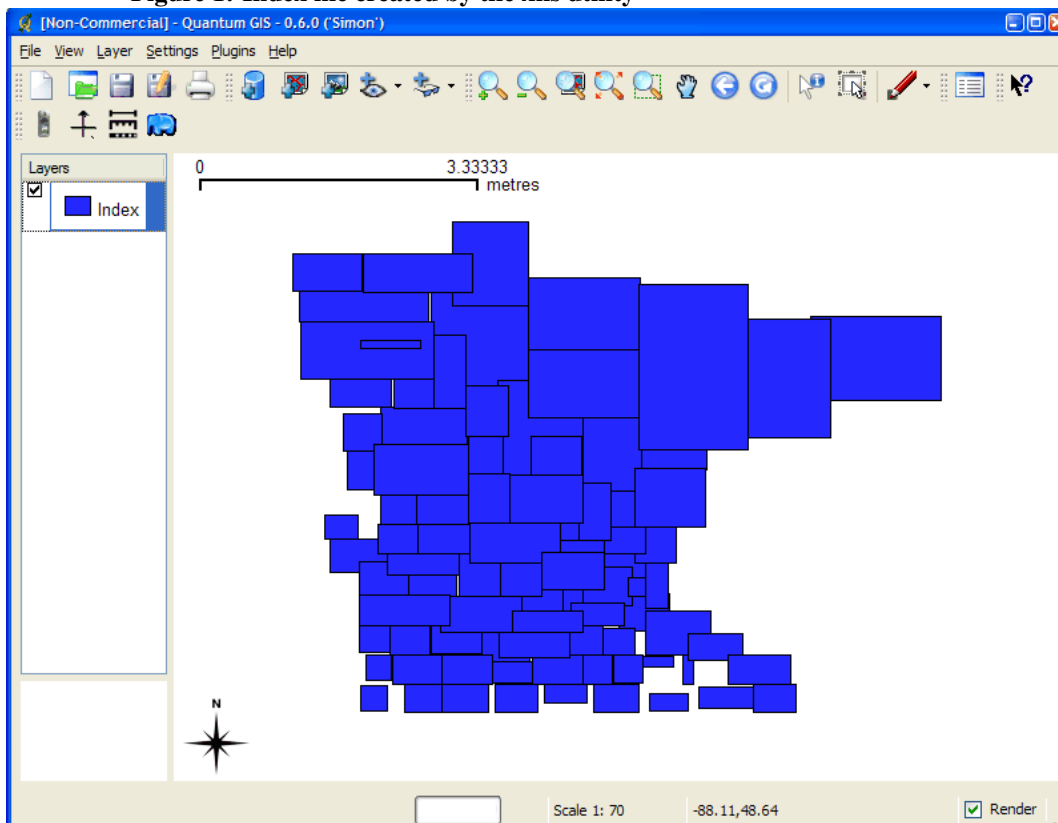


Figure 2: Attributes of index file created by tile4ms utility

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

4. 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.

12.1.11 Batch Scripting

If you need to run the utilities on multiple files/folders, here are some commands that will help you:

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
```

Linux

```
find /path/to/data -name "*.shp" -exec shptree {} \;
```

12.1.12 File Management

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.

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 *.*
```


13.1 CGI

Date 2008/09/09

Author Daniel Morissette

Contact dmorissette at mapgears.com

Author Jeff McKenna

Contact jmckenna at gatewaygeomatics.com

Author Frank Koormann

13.1.1 MapServer CGI Introduction

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.

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 a query item) to the query function. To do this two 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.

13.1.2 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
```

To save the output of an OGC WMS GetMap request into a valid image file, remove first three lines using the pipe- and e.g. tail commands, such as:

```
mapserv -nh "QUERY_STRING=EXCEPTIONS=application/vnd.ogc.se_inimage&REQUEST=GetMap&SERVICE=WMS&VERSION=1.0.0" > test.png
```

13.1.3 Map Context Files

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.

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"
...

```

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

```

13.1.4 MapServer CGI Controls

Variables

Variable names are not case sensitive.

BUFFER [distance] A distance, in the same coordinate system as the map file, used in conjunction with MAPXY to create a new map extent.

CLASSGROUP [name] The name of a *LAYER CLASSGROUP*. Set the *LAYER CLASSGROUP* to *name* for all layers that have at least one *CLASS* that is using the given CLASSGROUP *name*.

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 ...**] | [**WKT**] An arbitrary polygon shape (specified using **image coordinates**) to be used for query purposes.

The polygon is specified by listing its coordinates (multiple instances simply add parts to the shape so it is possible to construct a shape with holes) or by specifying the WKT (Well Known Text) representation.

Used with the NQUERY mode.

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.

MAPSHAPE [**x1 y1 x2 y2 x3 y3 ...**] | [**WKT**] An arbitrary polygon shape (specified using **map coordinates**) to be used for query purposes.

The polygon is specified by listing its coordinates (multiple instances simply add parts to the shape so it is possible to construct a shape with holes) or by specifying a WKT (Well Known Text) representation of the polygon.

Used with the NQUERY mode.

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.

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:

BROWSE Fully interactive interface where maps (and interactive pages) are created. This is the default mode.

FEATURENQUERY A spatial search that uses multiple features from SLAYER to query other layers.

FEATUREQUERY A spatial search that uses one feature from SLAYER to query other layers.

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.

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).

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).

ITEMNQUERY A text search of attribute data is triggered using a QSTRING. Returns all matches.

ITEMQUERY A text search of attribute data is triggered using a layer QSTRING. Returns 1st match.

LEGEND The created legend is returned. Used within an tag.

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=/mapfiles/gmap75.map&MODE=legendicon&ICON=popplace,0
```

MAP The created map is returned. Used within an tag.

NQUERY A spatial search (finds all) is triggered by a click in a map or by user-define selection box.

QUERY A spatial search (finds closest) is triggered by a click in a map.

REFERENCE The created reference map is returned. Used within an tag.

SCALEBAR The created scalebar is returned. Used within an tag.

ZOOMIN Switch to mode BROWSE with ZOOMDIR=1

ZOOMOUT Switch to mode BROWSE with ZOOMDIR=-1

COORDINATE To be clarified.

Note: The previously available map-only query modes, e.g. ITEMQUERYMAP, are no longer supported. The *QFORMAT* parameter is now used instead.

QFORMAT [outputformat | mime/type] (optional) The *OUTPUTFORMAT* name or mime/type to be used to process a set of query results (corresponds to the *WEB* object's *QUERYFORMAT* parameter). The parameter is optional and used in conjunction with query *MODEs*. The default is text/html.

Example (map output):

```
...&mode=nquery&qformat=png24&...
```

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.

QLAYER [name] Query layer. The name of the layer to be queried as it appears in the map file.

QSTRING [expression] Attribute queries: Query string passed to the query function. Since 5.0, qstring will have to be specified in the *VALIDATION* parameter of the *LAYER* for qstring queries to work (*qstring_validation_pattern* *LAYER*-level *METADATA* for MapServer versions prior to 5.4).

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).

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 (see *Run-time Substitution*). The syntax for this is fairly straightforward, 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. *VALIDATION* is used to control run-time substitution. Try it out.

<p>Warning: This functionality is only available via the <i>mapserv</i> CGI application. Within MapScript this is easy to do by yourself since the developer has complete control over how input is handled.</p>

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 had 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_text=My+House!&...
```

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.

Specifying the location of mapfiles using an Apache variable

Apache variables can be used to specify the location of map files (instead of exposing full mapfile paths to the outside world).

1. Set the variable (in this example *MY_MAPFILE*) in Apache's httpd.conf:

```
SetEnv MY_MAPFILE "/opt/mapserver/map1/mymapfile.map"
```

2. Refer to the variable in the mapserver CGI URL:

```
http://localhost/cgi-bin/mapserv?map=MY_MAPFILE&mode=...
```

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).

13.1.5 Run-time Substitution

Author Steve Lime

Contact steve.lime at DNR.STATE.MN.US

Revision \$Revision\$

Last Updated \$Date\$

Table of Contents

- *Run-time Substitution*
 - *Introduction*
 - *Basic Example*
 - *Parameters Supported*
 - *Default values if not provided in the URL*
 - *VALIDATION*
 - *Magic values*

Introduction

Run-time substitution for the MapServer CGI has been around since version 4.0 and its use has continued to expand. In short, it allows you to alter portions of a mapfile based on data passed via a CGI request.

Warning: This functionality is only available via the *mapserv* CGI application. Within MapScript this is easy to do by yourself since the developer has complete control over how input is handled.

Case sensitivity

Since version 6.4, CGI parameter names are not case sensitive. CGI parameter values are case sensitive.

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.

A request should look like (assuming "marsh" is a valid land cover class):

```
http://...mapserv?map=...&ctype=marsh
```

You could then set up a layer like so:

```
LAYER
  NAME 'covertypes'
  ...
  VALIDATION
    "ctype" "[a-z]+"
  END
  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.

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. All parameters *must* be validated. Remember, you also can do run-time configuration using the *map_object_property* type syntax detailed in *Changing map file parameters via a form or a URL*. Below is a list of properties that do allow run-time substitution:

- CLASS: EXPRESSION
- CLASS: TEXT
- LAYER: CONNECTION
- LAYER: DATA
- LAYER: FILTER
- LAYER: TILEINDEX

- OUTPUTFORMAT: FORMATOPTION: FILENAME

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 postgis (since version 6.0, validation is mandatory). Within the layer you'd do the following:

```
VALIDATION
  'multimedia' '^yes|no$'
  'sound' '^yes|no$'
  'nseats' '^([0-9]){1,2}$'
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.

Default values if not provided in the URL

The runtime substitution mechanism will usually create syntactically incorrect, and almost always semantically incorrect mapfiles if the substitution parameter was not provided in the calling URL.

Since version 5.6, you can provide a default value for any substitution parameter, that will be applied if the parameter was not found in the url. You do this by providing special entries inside *CLASS*, *LAYER* or *WEB* validation blocks:

```
VALIDATION
  'default_sound' 'yes'
  'default_nseats' '5'
  'default_multimedia' 'yes'
END
```

In this example, the mapfile will be created as if the url contained “&sound=yes&nseats=5&multimedia=yes”.

If identical default keys appear in more than one validation block then keys in more specialised blocks override those in more generalised blocks. i.e. *CLASS* overrides *LAYER* which overrides *WEB*. The same functionality is available using *METADATA* blocks instead of *VALIDATION* but this is deprecated as of MapServer 5.4.0. This behavior is also accessible in the shp2img utility, allowing you to test runtime substitution mapfiles without using a webserver.

VALIDATION

Because runtime substitution affects potentially sensitive areas of your mapfile, such as database columns and file-names, it is mandatory that you use pattern validation (since version 6.0)

Pattern validation uses regular expressions, which are strings that describe how to compare strings to patterns. The exact functionality of your systems' regular expressions may vary, but you can find a lot of general information by a Google search for “regular expression tutorial”.

As of MapServer 5.4.0 the preferred mechanism is a *VALIDATION* block in the *LAYER* definition. This is only slightly different from the older *METADATA* mechanism. *VALIDATION* blocks can be used with *CLASS*, *LAYER* and *WEB*.

VALIDATION

```
# %firstname% substitutions can only have letters and hyphens
'firstname'      '^[a-zA-Z\-\-]+$'

# %parcelid% must be numeric and between 5 and 8 characters
'parcelid'      '^[0-9]{5,8}$'

# %taxid% must be two capital letters and six digits
'taxid'         '^[A-Z]{2}[0-9]{6}$'
```

END

If identical keys appear in more than one validation block then keys in more specialised blocks override those in more generalised blocks. i.e. *CLASS* overrides *LAYER* which overrides *WEB*.

Magic values

Some runtime substitutions have special caveats.

ID In addition to any defined *METADATA* or *VALIDATION*, the 'id' parameter will be subjected to a special check. It must be alphanumeric and cannot be longer than 63 characters.

13.1.6 A Simple CGI Wrapper Script

Author Steven Monai

Revision \$Revision\$

Date \$Date\$

Last Updated 2006/01/26

Table of Contents

- *A Simple CGI Wrapper Script*
 - *Introduction*
 - *Script Information*

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.

Warning: Using a wrapper script is inefficient as it implies spawning two processes (shell+mapserv) instead of only one (mapserv), and should not be used on production installations. Refer to *Changing the Online Resource URL* for examples of how to setup similar functionality directly by using webserver configuration options.

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 scripts are wrappers for CGI GET requests that should be generic enough to run on any OS with `/bin/sh`.

Alternative 1

```
#!/bin/sh
MAPSERV="/path/to/my/mapserv"
MS_MAPFILE="/path/to/my/mapfile.map" exec ${MAPSERV}
```

You should set the `MAPSERV` and `MS_MAPFILE` variables as appropriate for your configuration. `MAPSERV` points to your MapServer executable, and `MS_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. Put the script in your web server's `cgi-bin` directory, and make it executable.

This solution should support both GET and POST requests.

Alternative 2

```
#!/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.

13.1.7 MapServer OpenLayers Viewer

MapServer provides a simple, built-in method for testing a mapfile using OpenLayers. This feature is for testing/development purposes only, and not for production or deploying full-featured sites. You can preview, test, and navigate a mapfile by accessing a special url which will return a built-in OpenLayers template.

Note: This feature was discussed in [rfc 63](#) and in the ticket <http://trac.osgeo.org/mapserver/ticket/3549>

Using the OpenLayers viewer

Opening the OpenLayers viewer in your browser

Assuming you are running mapserver on your local machine, and you have the Itasca demo setup, a basic url would be (split into two lines for readability):

```
http://localhost/cgi-bin/mapserv?mode=browse&template=openlayers
&layer=lakespy2&layer=dlgstln2&map=/var/www/workshop/itasca.map
```

Here is a quick breakdown of that url:

- Basic Parameters for activating the OpenLayers browser:

```
template=openlayers
mode=browse
```

- Basic Map / Layer Parameters:

```
map=/var/www/workshop/itasca.map
layer=lakespy2
layer=dlgstln2
```

That's it!

Opening the OpenLayers viewer in the form of a WMS request

This feature is useful when debugging WMS requests. You can write one of these by hand, or copy the URL for a WMS tile. Running the following should give you a simple OpenLayers demo around the BBOX (split into several lines for readability):

```
http://localhost/cgi-bin/mapserv?map=/var/www/workshop/itasca.map
&LAYERS=lakespy2&VERSION=1.1.1&SERVICE=WMS&REQUEST=GetMap
&FORMAT=application/openlayers&WIDTH=512&HEIGHT=512&SRS=EPSG:26915
&BBOX=429956.19803725,5231780.0814818,444078.32296225,5245902.2064068
```

Here is a quick breakdown of the interesting parts of that URL:

- Special Parameter for activating the OpenLayers viewer:

```
FORMAT=application/openlayers
```

- Basic MapServer Parameters:

```
map=/var/www/workshop/itasca.map
```

- Basic WMS parameters:

```
#Layers, our bounding box and projection
LAYERS=lakespy2
BBOX=429956.19803725,5231780.0814818,444078.32296225,5245902.2064068
SRS=EPSG:26915

#Version and other WMS request params
SERVICE=WMS
VERSION=1.1.1
REQUEST=GetMap
WIDTH=512
```

```
HEIGHT=512
TRANSPARENT=true
```

Customizing settings

- environment variables:

```
MS_OPENLAYERS_JS_URL - The URL to the OpenLayers javascript library. Per default the library loa
```

Environment Variables

14.1 Environment Variables

A number of environment variables can be used to control MapServer's behavior or specify the location of some resources.

CURL_CA_BUNDLE

Used to specify the location of the Certificate Authority (CA) bundle file to be used by Curl when using HTTPS connections in WMS/WFS client layers. Curl comes bundled with its own CA bundle by default, so this variable is not required unless you have an unusual installation:

```
export CURL_CA_BUNDLE=/path/to/ca-bundle.crt
```

New in version 5.4.1.

MS_DEBUGLEVEL A default *DEBUG* level value can be set using the *MS_DEBUGLEVEL* environment variable in combination with the *MS_ERRORFILE* 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.

New in version 5.0.

See also:

`rfc28`

MS_ENCRYPTION_KEY

See also:

`msencrypt`

New in version 4.10.

MS_ERRORFILE The *MS_ERRORFILE* environment variable specifies the location of the logging/debug 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 do 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 is 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 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.

See also:

rfc28

MS_MAP_NO_PATH The *MS_MAP_NO_PATH* environment variable can be set to any value 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 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=...
```

New in version 5.4.

See also:

rfc56

MS_MAPFILE The mapfile to use if the map=... URL parameter is not provided.

It is also possible to use an environment variable name as the value of the map=... URL parameter. The value of this environment variable will be used as the mapfile path:

```
map=ENV_VAR
```

MS_MAPFILE_PATTERN *MS_MAPFILE_PATTERN* can be used to override the default regular expression which is used to validate mapfile filename extensions.

The default value for this variable is:

```
MS_MAPFILE_PATTERN='\\.map$'
```


MS_MAP_PATTERN The *MS_MAP_PATTERN* environment variable can be used to specify a Regular Expression that must be matched by all mapfile paths passed to the mapserv CGI in the map=... URL parameter.

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/"
```

New in version 5.4.

See also:

rfc56

MS_MODE Default value for the mode=... URL parameter. Setting mode=... in the URL takes precedence over the environment variable.

MS_OPENLAYERS_JS_URL The URL to the OpenLayers javascript library (can be used when testing WMS services using imagetype application/openlayers), for instance:

```
http://openlayers.org/api/OpenLayers.js
```

MS_TEMPPATH Set the *WEB TEMPPATH*.

New in version 6.0.

MS_XMLMAPFILE_XSLT Used to enable XML Mapfile support. Points to the location of the XSLT to use for the XML->text mapfile conversion.

See also:

XML Mapfile support

PROJ_LIB The *PROJ_LIB* environment variable or *CONFIG* directive can be used to specify the directory where the PROJ.4 data files (including the "epsg" file) are located, if they are not in the default directory where PROJ.4 expects them.

See also:

Setting the location of the epsg file in Errors.

15.1 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*

16.1 Errors

16.1.1 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.
```

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.

16.1.2 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.

16.1.3 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
```

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*.

16.1.4 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

16.1.5 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

16.1.6 msLoadMap(): Failed to open map file

Error displayed when trying to display map image.

The message tells you that MapServer cannot find map file or has problems with the map file. 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. Verify that the map file using shp2img to make sure that the syntax is correct.

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
```

16.1.7 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>

16.1.8 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.

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/
```

16.1.9 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.

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>

16.1.10 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.

16.1.11 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

16.1.12 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

16.1.13 msWMSLoadGetMapParams(): WMS server error. Image Size out of range, WIDTH and HEIGHT must be between 1 and 2048 pixels

Error that is returned / displayed when a user has requested a map image (via WMS) that exceeds the maximum width or height that the service allows.

To increase the maximum map width and height for the service, use the MAXSIZE parameter of the *MAP* object. Producing larger map images requires more processing power and more memory, so take care.

16.1.14 Unable to load dll (MapScript)

One of the dll-s could not be loaded that mapscript.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 mapscript.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 mapscript sample applications properly your compilation might be correct and all of the dlls are available.

C#-specific information

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.

17.1 FAQ

17.1.1 Where is the MapServer log file?

See rfc28

17.1.2 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.

17.1.3 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.

17.1.4 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.

17.1.5 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.

17.1.6 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.

17.1.7 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.

17.1.8 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.

17.1.9 What is MapScript?

MapScript is the scripting interface to MapServer, usually generated by *SWIG* (except in the case of *PHP MapScript API*). MapScript allows you to program with MapServer's objects directly instead of interacting with MapServer

through its *CGI* and *Mapfile*.

17.1.10 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.

17.1.11 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 your application up to use this service to provide lat/lon pairs for addresses, and then use MapServer to display those points.

17.1.12 How do I set line width in my maps?

In the current MapServer version, line width is set using the STYLE parameter WIDTH. For a LINE layer, lines can be made red and 3 pixels wide by using the following style in a CLASS.

```
STYLE
  COLOR 255 0 0
  WIDTH 3
END
```

In earlier versions of MapServer , you could set the symbol for the LAYER to '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
```

17.1.13 Why do my JPEG input images look crappy via MapServer?

You must be using an old version of MapServer (where GD was the default library for rendering).

Newer versions of MapServer use AGG for rendering, and the default output formats is 24 bit colour, so there should not be a problem.

The default output format for MapServer with GD was 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).

With GD output, MapServer used quite a simple method to do the transformation, converting pixels to the nearest color in a 175 color colorcube. This would usually result in blotchy color in a fairly smoothly varying image.

For GD, solutions used to be:

- 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*.

17.1.14 Which image format should I use?

Although MapScript can generate the map in any desired image format it is 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 17.1: 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 be 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.

17.1.15 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
```

17.1.16 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.

You must be using an old version of MapServer . This should not be problem with newer versions. The following explains the old (GD) behaviour.

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.

17.1.17 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.

Example Layer

```
LAYER
  NAME "copyright"
  STATUS on
  TYPE point
  TRANSFORM ll # set the image origin to be lower left
  UNITS PIXELS # sets the units for the feature object
  FEATURE
    POINTS
      60 -10 # the offset (from lower left) in pixels
    END # Points
    TEXT "© xyz company 2006" # this is your displaying text
  END # Feature
  CLASS
    STYLE # has to have a style
    END # style
    LABEL # defines the font, colors etc. of the text
      FONT "times"
      TYPE truetype
      SIZE 8
      BUFFER 1
      COLOR 0 0 0
      FORCE true
      STYLE
        GEOMTRANSFORM 'labelpoly'
        COLOR 255 255 255 # white
      END # Style
    END # Label
  END # Class
END # Layer
```

Result



17.1.18 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
  OUTLINECOLOR 0 0 0
  WIDTH 3
END
```

17.1.19 How can I create simple antialiased line features?

With AGG (used in recent MapServer version), antialiased lines is the default, and can't be turned off.

With GD, 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 for GD:

```
...
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.

17.1.20 Which OGC Specifications does MapServer support?

See: *MapServer OGC Specification support*.

17.1.21 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
```

17.1.22 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.).

17.1.23 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=-10
```

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](#)

17.1.24 Using MapServer 6.0, why don't my layers show up in GetCapabilities responses or are not found anymore?

MapServer 6.0 introduces the option of hiding layers against OGC Web Service requests. OGC Web Services can provide powerful access to your geospatial data. It was decided to disable layer level request access as a default. See rfc67 provides a full explanation of the changes and implications.

To enable pre-6.0 behaviour, you can add the following to the *WEB* object's METADATA section in your mapfile:

```
"ows_enable_request" "*"

```

This will enable access of all layers to all OGC Web Service requests.

17.1.25 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>

17.1.26 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 +ellps=GRS80" output.shp input.shp

```

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!

17.1.27 How can I help improve the documentation on this site?

New documentation material and corrections to existing documentation are definitely very welcome. These contributions are handled through the same issue tracker used to track software bugs and enhancements.

Follow the directions for submitting bugs at: <http://www.mapserver.org/development/bugs.html>. When creating a ticket, in the Component field, select *MapServer Documentation*. If our ticket pertains to a specific web page, please include the URL to that page.

If you have tips or examples that don't really fit the definition of documentation, a good place to put them is the MapServer wiki at: <https://github.com/mapserver/mapserver/wiki>

17.1.28 What's with MapServer's logo?

The MapServer logo illustrates the confluence of the Minnesota and Mississippi rivers, quite near to the home of the St. Paul Campus of the University of Minnesota, which was the birthplace of MapServer.

18.1 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.

18.2 Credits

Major funding for the 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. We would like to acknowledge other major contributions as well:

- 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.
- Ongoing maintenance and support frequently provided by the US Army Corps of Engineers
- Several version 6.2 features funded through Météo-France