

GLG Map Server Reference Manual

*GLG Toolkit
Version 4.6*

Generic Logic, Inc.

Generic Logic, Inc.

6 University Drive 206-125

Amherst, MA 01002

USA

Telephone: (413) 253-7491

FAX: (413) 241-6107

email: support@genlogic.com

web: www.genlogic.com

Copyrights and Trademarks

Copyright © 1994-2026 by Generic Logic, Inc.

All Rights Reserved. This manual is subject to copyright protection.

GLG Toolkit, GLG Widgets, GLG Graphics Builder, GLG HMI Configurator and GLG Map Server are trademarks of Generic Logic, Inc.

All other trademarks are acknowledged as the property of their respective owners.

April 22, 2026

Software Release Version 4.6

GLG Map Server Reference Manual

Table of Contents

Chapter 1 <i>Introduction</i>	21
Using the Map Server inside the GLG Drawing	21
Using the Stand-Alone Map Server	21
1.1 Overview	22
Map Server Executable	23
Tools and Utilities	23
Library Programming Interface	23
1.2 Using the Map Server: Logical Steps	24
Map Server Input: Managing Data	24
Requesting Maps	24
Map Server Output: Displaying a Map	25
GLG GIS Object	25
File Output	26
CGI Output	26
Map Server API	26
Chapter 2 <i>Structure of Data and File Layout</i>	27
2.1 Types of Data	27
Layers	27
Image Data	28
Vector Data	28
Attributes	29
Layer Attributes	29
Object Attributes	29
Fonts	30

2.2 Datasets, Layers, Tiles and Data..... 31

2.3 Configuration File Format..... 31

Datasets.....	32
GLM ROOT DIR=<dir>	33
LAYER DIR=<dir>	33
FONT DIR=<dir>	33
LAYER=<name> <lif>	33
ALIAS=<name> <value>	33
ZOOM FACTOR TYPE= <EXTENT or "EXTENT AND WIDTH">	34
BASE WIDTH= <value>	34
VECTOR FONT=<font_name> <font_file>	34
IMAGE FONT=<font_name> <font_file>	34
FONT ALIAS=<font_alias_name> "<font_name>,<font_name>,..."	34
DEFAULT FONT=<font_name>	35
ERROR FONT=<font_name>	35
ENCODING=<UTF8 or LATIN>	35
ALLOW OVERRIDE=<0, 1 or 2>	35
MAX IMAGE TILES.....	35
MAX VECTOR TILES.....	35
GLYPH CACHE TYPE=<type>	36
MAX ITERATIONS=<number>	36
A Simple Example of an SDF file	36
Layers and Tiles.....	36
The Zoom Factor.....	37
Tiling.....	37
Configuration Variables.....	38
TYPE=<type>	38
DEFAULT ON=<0 or 1>	38
RELATIVE ZOOM=<0 or 1>	38
TRANS TYPE=<type>	39
TRANS COLOR=<r,g,b>	39
TRANS COLOR PRECISION=<value>	39
ELEVATION MODE=<mode>	40
ALPHA=<value>	40
ATTR COND=<value>	40
FILENAME=<file>	41
VECTOR FORMAT=<type>	41
FILTER=<file>	41
IMAGE FORMAT=<format>	42
IMAGE ANTIALISING=<0 or 1>	42
FALLBACK LAYER=<layer name>	42
FALLBACK=<file> (deprecated: use FALLBACK LAYER).....	43
MAX TILES=<number>	43
MAX CACHE SIZE=<number>	43
NUM TILES X=<number>	43
NUM TILES Y=<number>	43

MIN PIXEL SIZE=<number>	43
CLIP=<0 or 1>	44
VOLATILE=<0 or 1>	44
LOCK FILE=<file>	44
LOCK TIMEOUT=<number>	44
REDIRECT FILE=<file>	44
REDIRECT PREFIX=<directory or prefix>	44
VOLATILE ERRORS=<0 or 1>	45
ALLOW MISSING TILES=<0 or 1>	45
TILES OVERLAP=<0 or 1>	45
BACKGROUND COLOR=<r,g,b>	45
BACKGROUND LAYER=<layer name>	46
WIDTH=<number>	46
HEIGHT=<number>	46
MIN LON=<number>	46
MIN LAT=<number>	46
MAX LON=<number>	46
MAX LAT=<number>	46
MIN ZOOM=<number>	46
MAX ZOOM=<number>	46
CLIP TYPE=<type>	47
CLIP MIN LON=<number>	47
CLIP MIN LAT=<number>	47
CLIP MAX LON=<number>	47
CLIP MAX LAT=<number>	47
GRID LAT INTERVAL=<number>	47
GRID LON INTERVAL=<number>	47
GRID MIN LON=<number>	48
GRID MIN LAT=<number>	48
GRID MAX LON=<number>	48
GRID MAX LAT=<number>	48
ADAPTIVE GRID=<number>	48
GRID LABELS=<0, 1, 2, 3 or 4>	48
DECIMAL GRID LABELS=<0 or 1>	48
GRID LABEL FORMAT=<format>	48
GRID LABEL FORMAT LON=<format>	48
GRID LABEL CHAR=<0, 1 or 2>	49
ROTATE LON LABELS=<0 or 1>	49
LON LABEL ANGLE=<value>	49
FONT=<name>	49
FONT SCALE=<number>	49
TEXT TYPE=<type>	49
TEXT ANGLE=<number>	50
TEXT ANCHOR=<type> <type>	50
LABEL FORMAT=<format>	50
LABEL FORMAT2=<format>	51
LABEL STYLE=<style>	51
LABEL OUTLINE COLOR=<r g b>	51
WRAP LABEL LENGTH=<number>	51
TEXT BOX OFFSET HORIZ=<number>	52

TEXT BOX OFFSET VERT=<number>	52
TBOX LINE WIDTH=<number>	52
LABEL MIN ZOOM=<value>	52
LABEL MAX ZOOM=<value>	52
LAYOUT TYPE=<value>	52
LABEL PRIORITY=<number>	52
LABEL REPEAT DISTANCE=<number>	53
LAYOUT MARGIN=<number>	53
LAYOUT MARKERS=<0 or 1>	53
MARKER ICON FILE=<file>	53
MARKER ICON TYPE=<type, type, ...>	53
MARKER ICON SIZE=<number>	54
TEXT OFFSET HORIZ=<number>	54
TEXT OFFSET VERT=<number>	54
FILL COLOR=<r,g,b>	54
EDGE COLOR=<r,g,b>	54
MARKER EDGE COLOR=<r,g,b>	54
MARKER FILL COLOR=<r,g,b>	54
LABEL COLOR=<r,g,b>	54
BOX EDGE COLOR=<r,g,b>	55
BOX FILL COLOR=<r,g,b>	55
GRADIENT COLOR=<r,g,b>	55
GRADIENT TYPE=<type>	55
GRADIENT LENGTH=<length>	55
FILL TYPE=<type>	55
LINE WIDTH=<number>	55
DRAW CENTER MARKER=<0 or 1>	56
POS RANGE=<0 or 1>	56
POLY LABEL FORMAT=<format>	56
POLY LABEL FORMAT2=<format>	56
POLY LABEL TYPE=<type>	56
POLY LABEL CENTER=<number>	57
LABEL SELECTION MODE=<mode>	57
ALLOW OVERRIDE=<0 or 1>	57
ATTR MAP=<number resource threshold_table REL/ABS>	57
CUSTOM ATTR=<number resource data_type>	59
PIXEL MAP=<number resource threshold_table REL/ABS>	60
INCLUDE=<file>	60
TILES SUBLIF=<top-level sublif file> [<bbox>]	
SUBLIF DIR=<directory> [<bbox>]	
SUBLIF FILE=<sublif file> [<bbox>]	60

Chapter 3 *Using the Map Server and Utilities* 63

3.1 Map Server Executable 63

Error handling	63
Synopsis	64
Description	64

Command Line Options	64
-help	64
-version.....	64
-arg-file <file>	64
-verbosity <number>	64
-progress <number>	65
Map Generation and Queries	65
Synopsis.....	65
Description	65
Options	65
-dataset <file>	65
-cgi	65
-fcgi	66
-oGISreq <string>	66
-output <file>	66
-max-iterations <N>	66
Map Query String.....	66
VERSION=<version>	67
REQUEST=<request type>	67
SRS=<value>	67
WIDTH=<number>	68
HEIGHT=<number>	68
BBOX=<min,min,max,max>.....	68
ANGLE=<number>	68
BGCOLOR=<color>	68
STYLES=<value>	68
FORMAT=<image format>	68
LAYERS=<list>	69
CENTER=<lon,lat>	70
EXTENT=<x,y>	70
STRETCH=<0 or 1>	70
IERRORS=<0 or 1>	70
IMAGE_ANTIALIASING=<0 or 1>	70
QUERY_LAYERS=<layer name>	70
INFO_TYPE=<info type>	70
INFO_FORMAT=<info format>	71
I=<number>	71
J=<number>	71
LON_LAT=<lon,lat>	71
SELECT_LABELS=<value>	71
PICK_RESOLUTION=<number>	71
Attribute Condition Syntax.....	71
Examples of Attribute Conditions.....	72
Examples of OpenGIS map query strings	72
Examples of coordinate conversion and elevation queries.....	74
Coordinate Conversion Query	74
Elevation Query for a Lat/Lon location	75

Elevation Query for an X/Y image point	76
GIS Selection Query Example	76
3.2 Tools and Utilities.....	77
Converting GVF ASCII and BINARY Files.....	78
Synopsis	78
Description.....	78
Options	78
-convert.....	78
-a.....	78
-b.....	78
-path <directory>.....	78
-pattern <file_name_pattern>	78
Examples	78
Merging Vector Data Files	79
Synopsis	79
Description.....	79
Options	79
-merge.....	79
-output <file>	79
-filter <file>	79
-a.....	79
-b.....	79
Examples	79
Bounding Box Extraction Utility.....	80
Synopsis	80
Description.....	80
Examples.....	80
Tiling Utility for Image and Vector Data	80
Synopsis	80
Description.....	80
Options	80
-tile	80
-image.....	81
-vector.....	81
-filename <file>.....	81
-template <string>	81
-num-x-tiles <number>	81
-num-y-tiles <number>	81
Image Tiling Options	81
-elevation.....	81

Vector Data Tiling Options	81
-filter <file>	81
-neg-range	82
-pos-range	82
-no-range	82
-custom-range	82
-min-wlon <number>	82
-max-wlon <number>	82
-gvf-extent	82
-world-extent	82
-min-lat <number>	83
-max-lat <number>	83
-min-lon <number>	83
-max-lon <number>	83
-a	83
-b	83
Examples	83
Splitting Vector Data	84
Synopsis	84
Description	84
Options	85
-split	85
-filename <file>	85
-output <file>	85
-filter <filter>	85
-attr <conditions>	85
-invert-attr <index>	86
-bbox	86
-min-lat <number>	86
-max-lat <number>	86
-min-lon <number>	86
-max-lon <number>	86
-num-x-tiles <number>	86
-num-y-tiles <number>	86
-num-points <number>	86
-neg-range	86
-pos-range	87
-no-range	87
-custom-range	87
-min-wlon <number>	87
-max-wlon <number>	87
Examples:	87
Slimming Utility for Vector Data	88
Synopsis	88
Description	88
Options	88

-slim.....	88
-filename <file>	88
-output <file>	88
-resolution <dir>	88
-a.....	88
-b.....	88
Examples	88
Shapefile Conversion Utility	89
Synopsis	89
Description	89
Options	89
-shp2gvf.....	89
-output <file>	89
-pattern <pattern>	89
-r	90
-a	90
-b.....	90
-show-attrs	90
-dump	90
-no-header.....	90
-m	90
-raw.....	90
-all-attrs	90
-no-attrs	90
-attr <index>	90
-marker-label-attr <index>	91
-close-polygons	91
Examples	91
Hierarchical Tile Parsing (Advanced)	92
Synopsis	92
Description	92
Options	92
-slf.....	92
-vector.....	93
-image.....	93
-single-layer.....	93
-multi-layer.....	93
-path <dir>	93
-suppress-bbox	93
-pattern <pattern>	93
-out-slf-name <filename>	94
Examples	95
Image and Elevation Data Import (Advanced).....	95
Synopsis	95

Description	95
Options	96
-verbosity <level>	96
-progress <number>	96
-info-only	96
-write-image	97
-elevation	97
-shadow	97
-data-shift	97
-data-offset	97
-slf	98
-out-format <format>	98
-out-ext <extension>	98
-postfix <extension>	98
-pattern <pattern>	98
-r	98
-u	98
-k	98
-f	98
-dont-emboss	99
Examples	99

Chapter 4 Map Server API 101

4.1 Overview..... 101

Resources	101
-----------------	-----

4.2 Basic Structure of a Mapping Application 102

DatasetName D	102
Projection D	102
CenterLon D.....	102
CenterLat D.....	102
VExtent D	103
HExtent D	103
AngleD	103
Width D.....	103
Height D.....	103
Stretch D	103
Background G	103
ImageAntiAliasing D	103
ImageErrors D.....	103
Verbosity D	103
Code Example.....	104

4.3 Linking with the Map Server Library..... 105

Linux/Unix.....	106
-----------------	-----

Windows	106
Using Static Libraries.....	106
Using DLLs.....	107
Include Files	107
4.4 Advanced features of the GLG Map Server API	107
Overview	107
Hierarchy of GlmObjects.....	107
Modifying Layers	108
Creating Datasets	108
Dataset Resources.....	108
GlmRoot S.....	109
LayerPath S	109
FontPath S	109
Encoding D.....	109
AllowOverride D.....	109
ZoomFactorType D.....	109
BaseWidth D	109
MaxImageTiles D.....	109
MaxVectorTiles D.....	109
MaxIterations D.....	109
GlyphCacheType D.....	110
DefaultFont S	110
ErrorFont S	110
Layer Resources	110
LayerType D	110
IsDefault D	110
TransType D.....	111
TransparentColor G.....	111
TransparentColorPrecision D.....	111
TilesOverlap D	111
BGColorRed D.....	111
BGColorGreen D.....	111
BGColorBlue D.....	111
Background Layer S.....	111
ElevationMode D	111
Alpha D	112
RelativeZoom D	112
ImageAntiAliasing D	112
MaxTiles D.....	112
MaxCacheSize D.....	112
MinPixelSize D	112
Clip D	112
LockTimeout D	112
Width D	112
Height D	112

MinLon D.....	112
MinLat D.....	112
MaxLon D.....	112
MaxLat D.....	113
MinZoom D.....	113
MaxZoom D.....	113
ClipType D.....	113
ClipMinLon D.....	113
ClipMinLat D.....	113
ClipMaxLon D.....	113
ClipMaxLat D.....	113
GridLatInterval D.....	113
GridLonInterval D.....	113
GridMinLon D.....	113
GridMinLat D.....	113
GridMaxLon D.....	113
GridMaxLat D.....	113
AdaptiveGrid D.....	114
GridLabels D.....	114
DecimalGridLabels D.....	114
GridLabelFormat S.....	114
GridLabelFormatLon S.....	114
GridLabelChar D.....	114
RotateLonLabels D.....	114
LonLabelAngle D.....	114
TextFont S.....	115
TextType D.....	115
FontScale D.....	115
TextAngle D.....	115
TextAnchor D.....	115
LabelFormat S.....	115
LabelFormat2 S.....	115
LabelStyle D.....	116
LabelOutlineColor G.....	116
TextBoxLineWidth D.....	116
TextBoxOffsetH D.....	116
TextBoxOffsetV D.....	116
WrapLabelLength.....	116
LabelMinZoom D.....	116
LabelMaxZoom D.....	116
LayoutType D.....	116
LabelPriority D.....	116
LabelRepeatDistance D.....	117
LayoutMargin D.....	117
LayoutMarkers D.....	117
MarkerIconFile S.....	117
MarkerType D.....	117
MarkerSize D.....	117
TextOffsetH D.....	117
TextOffsetV D.....	117

FillType D	117
LineWidth D.....	118
FillColor G	118
EdgeColor G.....	118
DrawCenterMarker D.....	118
MarkerFillColor G.....	118
MarkerEdgeColor G.....	118
LabelColor G.....	118
BoxEdgeColor G	118
BoxFillColor G.....	118
GradientColor G.....	118
GradientType D.....	118
GradientLength D.....	118
PolyLabelFormat S.....	119
PolyLabelFormat2 S.....	119
PolyLabelType D	119
PolyLabelCenter D.....	119
LabelSelectionMode D.....	119
Volatile D	119
VolatileErrors D	119
LockFile S	119
LockTimeout D	119
RedirectFile S.....	120
RedirectPrefix S	120
FallbackLayer S.....	120
AllowMissingTiles D.....	120
AllowOverride D.....	120

4.5 Function Descriptions..... 120

GlmInit.....	120
Parameters	120
GlmCreateDataset	120
Parameters	121
GlmAddLayer	121
Parameters	121
GlmCreateMap.....	121
GlmSetupMap	121
Parameters	122
GlmSetLayers.....	122
Parameters	122
GlmResetLayers.....	122
Parameters	122
GlmEnableLayer	123
Parameters	123
GlmDisableLayer	123
Parameters	123

GlmGetDResource	123
Parameters.....	124
GlmGetGResource	124
Parameters.....	124
GlmGetSResource	125
Parameters.....	125
GlmSetDResource	125
Parameters.....	125
GlmSetGResource	126
Parameters.....	126
GlmSetSResource.....	126
Parameters.....	126
GlmGetProjection.....	127
Parameters.....	127
GlmGetMapData	127
Parameters.....	127
GlmWriteImage.....	127
Parameters.....	128
GlmGetElevation.....	128
Parameters.....	128
GlmGetSelection	129
Parameters.....	129
GIS Selection Message	129
Programming Example	130
GlmGetLatLon	133
Parameters.....	133
GlmGetXYZ.....	133
Parameters.....	134
GlmConvert	134
Parameters.....	134
GlmLatLonToUtm.....	135
Parameters.....	135
GlmUtmToLatLon.....	136
Parameters.....	136
GlmUtmToMgrs.....	136
Parameters.....	136
GlmMgrsToUtm	137
Parameters.....	137
GlmUpdateMap	137
Parameters.....	137
GlmResetImageErrors	138

GlmRemoveLayer	138
Parameters	138
GlmDestroyMap.....	138
Parameters	138
GlmDestroyDataset.....	138
Parameters	138
GlmTerminate	139
Chapter 5 GVF Filters and Data Converters	141
5.1 GVF API Reference.....	141
Overview	141
GVF Functions	141
Headers.....	141
Writing a Filter or Data Convertor.....	142
Polygons.....	142
Markers	142
Attributes.....	143
Points.....	143
Examples	144
Function Descriptions.....	144
GvfWriteHeader	144
Parameters	144
GvfWritePolygon	145
Parameters	145
GvfWriteMarker.....	145
Parameters	146
GvfWriteAttribute.....	146
Parameters	147
GvfWritePoint.....	147
Parameters	147
5.2 GVF Format RFC.....	148
File Extension	148
Definitions	148
ASCII Format	149
Header.....	149
Body.....	149

Objects	150
Polygons	150
Ring	150
Number of Points	150
Number of Attributes	150
Markers	150
Marker Type	150
Scale	151
Angle	151
Anchoring	151
Text String	151
Number of Attributes	151
Attributes	151
Data Points	152
 Appendix A: Web Server Installation Notes.....	153
Introduction	153
Linux/Unix Notes	153
Windows Notes	153
Perl Notes	154
Map Server Setup on a Web Server	155
Map Server Run-Time Components	155
GlmScript setup	156
FastCGI setup	156
Testing Map Server Setup on a Web Server	157

GLG Map Server Reference Manual

This book provides information about using the GLG Map Server along with its utilities and using the Application Programming Interface (API).

The first four chapters of the book document the Map Server itself, while the last chapter contains a reference and RFC specification of the open vector data format used by the Map Server.

The book contains the following chapters:

Table of Contents

Introduction

A general overview of the GLG Map Server

Structure of Data and File Layout

A description of how data is organized in the Map Server and how to set it up.

Using the Map Server and Utilities

A description of how to use the Map Server executable and its options

Map Server API

How to use the Map Server from a program

GVF Filters and Data Converters

How to program custom filters and an RFC specification of the GVF format

Appendix A: Web Server Installation Notes

How to install the Map Server on a web server for use with the Java, C# and JavaScript programs.

Index

Chapter 1

Introduction

1

The GLG Map Server is a sophisticated GIS Map Server used to generate map images dynamically upon request. It is targeted towards applications which display either static or dynamic information on top of a map, providing functionality for zooming and panning the map at run time. As the user zooms or pans to a different region, the Map Server generates a new map image to be displayed. The map images serve as a background which provides contextual information. Either static or dynamic icons may be placed on top of the images to provide real-time application-specific and interactive functionality.

Using the Map Server inside the GLG Drawing

The GLG Map Server may be used in conjunction with the GLG Toolkit in both the C/C++, C#, Java and JavaScript versions via the integrated **GLG GIS Object** to incorporate maps into a GLG drawing and the GLG Graphics Builder.

The GLG Toolkit's resource-based interface and simple API provide a convenient way to interact with the GIS object in the drawing, shielding the user from the complexity of the Map Server and transparently handling zooming, panning, resizing, dynamic updates and user interaction. Dynamic GLG objects built with the GLG Graphics Builder may be used as dynamic or static icons displayed on top of the map, providing real-time data display and interactive capabilities.

When a drawing with the GIS Object is deployed in a **C/C++ application** (or an ActiveX control on Windows), the map server is deployed in the form of the map server library linked with the program executable.

When the drawing is deployed in a **Java or C# application**, as well as a **JavaScript application in a web browser**, the map server is deployed as a web-based map server. The map server is set up as a CGI-bin executable on a web server, and the application internally uses a URL to request maps from the map server.

In either deployment, the GIS Object transparently handles all details of the map server interactions, letting the application concentrate on the application logic.

Using the Stand-Alone Map Server

The Map Server may also be used outside of the GLG Toolkit to create independent mapping applications or to serve maps to other applications. The Map Server may be used either in the form of a library linked into an application, or in the form of a cgi-bin executable which serves maps from a web server.

When deployed as a library, the GLG Map Server allows the user to setup and run a custom in-house map server which serves maps to applications and does not require any insecure connections to the Internet or rely on a third party server to be running in order to request and receive maps. When used as a cgi-bin executable, the Map Server provides map generating capabilities to remote clients and applications deployed in a web browser.

The GLG Map Server provides numerous features to efficiently handle the ever-changing needs of a modern application. It includes Orthographic projection to render maps on a globe, layering, alpha-blending, tiling, caching and decluttering for optimized performance. It provides the functionality to quickly render millions of points and lightweight polygons and text, as well as detailed images, making it ideal for applications which require rendering a large number of polygons, such as a map of an airfield or a detailed map of a city.

The Map Server has its own high-performance rendering engine which does not depend on any windowing system, so it can operate on a headless server. Since it runs independently of the graphical environment, the maps it generates are guaranteed to be identical across all platforms.

1.1 Overview

The GLG Map Server consists of the following parts

- The Map Server executable used to generate maps for web-based or command-line deployment.
- A library of functions (the API), used to incorporate the GLG Map server into an application as a library without any network connections.
- A set of command line utilities used to aid the user in setup and optimization of data for use with the GLG Map Server.

Since the GLG Map Server does not require any database tools to handle the data it uses, the Map Server is simple to set up and very flexible. The data and the description of the data lie in separate places, which results in tremendous flexibility. For example, if a user has obtained a newer, more accurate version of old data, and wishes to use the new data instead of the old, the data itself does not need to be altered and no database updated. Instead, the user simply specifies the location of the new data and the Map Server will automatically use it.

All descriptions of data, for example the width of the line to render roads in, is entered in a simple ASCII format text file. It is therefore not necessary to master any new tools for editing these informative descriptions, so as much as time as possible is left for development of mapping applications, not learning new tools.

The GLG Map Server uses the same object oriented approach as the GLG Toolkit. Although each polygon defined in a data file does not have its own attributes, polygons in the same data file, organized as a layer, can possess common attributes that are applied to them when they are rendered. This methodology is used to conserve space since the number of polygons the Map Server aims to render quickly is orders of magnitude higher than that which the GLG Toolkit is geared for. For example, if a data file contains polygons that describe roads, and a user wants to display the roads in red, although each individual polygon does not have a color associated with it, the user can control the color of all roads in that specific data file. The common object attributes of a layer in the Map Server have intuitive names and are almost identical to those present in GLG drawings, so the transition can be made with ease.

The following sections describe the components of the GLG Map Server in greater detail.

Map Server Executable

The Map Server executable may be used in a stand-alone mode to generate a single map image. In this mode, the Map Server is invoked with command line parameters which define the map image request. The generated map image is saved into a file defined by the command line parameters as well.

The Map Server executable may also be invoked in cgi-bin mode, which is used in a web/cgi environment to serve map images via a headless web server. In the cgi-bin mode, the Map Server takes map request parameters from cgi-bin environment variables and outputs the image with a proper HTML header into its stdout stream.

Both of the stand-alone and cgi-bin functionalities are accessible from the same executable via a command line switch. Also, they both comply with the OpenGIS standard for requesting maps.

The Map Server is designed as a map service and does not possess the capability to display the maps it generates. In the web environment, the images are naturally displayed by the client side web browser. In an application, GLG Toolkit's GIS object integrates the Map Server functionality into the GLG drawings and handles all aspects of displaying the maps and user interaction. The application may also display the images in any application-specific way without the use of the GLG Toolkit and GLG GIS object.

Tools and Utilities

The GLG Map Server includes several useful command line utilities to aid in setting up the data to be used by the Map Server. These include:

- an image tiling utility, which can split JPEG and TIFF formatted images into an arbitrary number of rectangular tiles,
- a vector splitting utility which can split polygons by number of points, according to a specific attribute, or rectangular tiling based on coordinates, and
- a utility to traverse directory structures and generate descriptions of all data found in that hierarchy to later use with the Map Server.

All utilities reside in the same Map Server executable and are invoked using utilities' command line option switches.

Library Programming Interface

When a C/C++ application requires the use of an integrated map display functionality, it may use the Map Server in the form of a C/C++ library instead of invoking a separate executable to generate maps. Not only does this method provide easier integration but it is also much faster and more flexible, as it allows the user to utilize such features as caching of data between map requests and creation of multiple maps.

There are two options available to deploy the Map Server as a library:

- Using the GLG Toolkit's GIS object which integrates the Map Server's functionality into the GLG drawing. The GIS object uses the Map Server in the form of a library but frees the user from accessing the library directly. The user accesses the GIS object using GLG's high-level resource mechanism, and doesn't need to learn the Map Server Library's API. **C# / JAVA / JAVASCRIPT NOTE:** In these environment, the GLG GIS object always uses the Map Server in the form of a cgi-bin executable hosted on a web server.
- Using the Map Server Library API directly from a program, in which case the program is responsible for displaying the generated image in an application-specific way.

1.2 Using the Map Server: Logical Steps

Map Server Input: Managing Data

Regardless of the form the Map Server is used in (executable, library or GLG GIS object), the Map Server uses raster or vector GIS data to generate map images. The Map Server reads the data, processes them by applying the requested projection and generates a map image.

The Map Server is optimized to read only the data for the requested layers and process only the data for the requested area. Huge datasets may be optimized by splitting the data into tiles, so that only the tiles covering the requested area need to be read and processed. The data may also be split in a hierarchical way (for example states and counties), and the Map Server provides utilities for efficient setup and processing of such data.

The Server Dataset File (.sdf) serves as a top-level data descriptor for a GIS dataset. It contains a list of available layers and the path to the Layer Description Files (.lif) which contain attributes of the layer and location of the data. The SDF file also contains information about available fonts.

After the data has been setup, the Map Server needs to know where the data it will be utilizing resides. This is accomplished by specifying the SDF file: through a command line option for the Map Server executable, GIS object attribute in a GLG drawing or by using the Map Server API when used in a library form.

When used in the form of the Library API, functions of the API can also be used to specify locations of all data used by the mapping application. Additionally, these functions can be used to create multiple maps and manage multiple datasets at once, as well as dynamically change layer attributes.

Requesting Maps

In order to generate a map, the Map Server needs to know the region of which it is to generate a map and what features to display. This information is provided through the map request parameters of the Map Server executable, through the GIS object attributes or the Map Server API if the library is used.

The Map Server executable has the `-dataset` option to specify the SDF file and the `-oGISreq` option that specifies all parameters of the map request (requested layers, area, projection, etc.) as a request string.

When the GIS object is used, the `GISDataFile`, `GISCenter` and `GISExtent`, `GISLayers`, `GISProjection` and other attributes may be used to control the map.

When using the Map Server in the library form, the Map Server API provides an intuitive Get/Set Resource mechanism very similar to that used by the GLG Toolkit. A handful of functions can specify the parameters for a specific map request and generate the request image.

Map Server Output: Displaying a Map

Since the Map Server is designed as a map service and provides no direct way to display the maps it generates, there are several options for using and displaying map images to choose from.

GLG GIS Object

The GLG Toolkit provides a way to seamlessly integrate the GLG Map Server into graphical applications created with the GLG Toolkit. The GIS object can be added into a GLG drawing just like any other graphical object and, and an application can control its resources to display dynamic maps and handle user interaction.

The GIS object automatically handles damage repair, resizing, zooming and panning, making it easy to embed dynamic maps into the GLG drawings. When the user resizes the drawing or uses pan and zoom keys, the GIS object automatically generates a new map request and sends it to the Map Server, displaying the new map in the drawing. The GIS object also provides the benefit of not needing to regenerate the map image every time dynamic icons are updated on top of it.

The GIS object also provides coordinate conversion utilities to convert between latitude/longitude and GLG coordinates, providing a simple interface to position dynamic and interactive GLG icons on top of the map.

The GIS object is available in all versions of the GLG Toolkit: C/C++, C#, Java, JavaScript and ActiveX. When the GIS object is deployed in the C/C++ or ActiveX version of the Toolkit, the Map Server is used in the form of a C/C++ library. When the GIS object is deployed in the C#, Java or JavaScript version of the Toolkit, the Map Server is utilized in the form of a cgi-bin executable hosted on a web server.

The user and API interfaces for C/C++, C#, ActiveX, Java and JavaScript versions of the GIS object are identical, regardless of the different ways in which the Map Server is utilized within GLG.

For more information on the GLG GIS object, see the GLG User's Guide and Builder Reference.

File Output

When using the Map Server executable in the command line form, the standard way to display generated map images is to output them to a file. The output file is controlled via a command line option. The GLG Map Server API also provides an interface to writing generate maps into a file. The only file output option is the JPEG file format. The images may then be viewed using a web browser or any image viewer.

CGI Output

When using the executable in the CGI or FastCGI mode, the image is outputted as a JPEG file to the standard output with the proper HTML headers, so that the image can be displayed in a browser on the client side.

Map Server API

Finally, if the user desires neither to use the GIS object nor output generated map images to a file, or wants to invoke the Map Server from a program, he/she can use some of the advanced features of the API to access the raw bitmap information of the resulting map image and then display the image in an application-specific way.

Chapter 2

Structure of Data and File Layout

2

On the surface, any given map image generated by the GLG Map Server appears to be a collection of lines, areas, images and text. Each of these elements and its attributes must be defined somewhere. Due to the sheer number of objects that many map images require, it is not practical to describe each object separately, especially when many objects share common attributes and require similar operations to be performed on them.

The GLG Map Server provides a way to consistently and efficiently describe data being processed, so as to minimize the redundancy of setup. The following sections will explain the different types of data, how data is described, how it is organized into datasets, layers, tiles and objects, and where the data and descriptions are stored.

The following chapter will describe the three main types of files.

- Server Dataset File (SDF), the master file that lists all fonts and layers
- Layer Information File (LIF), layer files that describe data
- GVF Data Files, which contain the actual data that is used by the Map Server

2.1 Types of Data

The GLG Map Server can process two different kinds of data: raster image data and vector data. Raster image data can be used to view such features as land and ocean cover on a map as well as transparent weather maps or even custom detailed images of local terrain. Vector data can be used to view such features as roads, county areas or shorelines, as well as text labels and map grids.

Layers

All data are organized into layers. For image data, each layer control the use of one raster image (the image may actually be a collection of image tiles). For vector data, the layer may contain a collection of similar vector features, such as roads or state boundaries, defined in a single GVF file.

Each layer's properties are defined in its Layer Information File (LIF). For vector layers the LIF file also defines common rendering attributes for all objects in the layer.

Each layer may be enabled or disabled, allowing the user to control what features are shown on the generated map.

Image Data

Raster image data can itself come in two different file formats. Currently, the GLG Map Server supports both the JPEG, TIFF and PNG file formats. Baseline JPEG files are supported (except those with color-table information), as well as scan-line TIFF but not TIFF tiles. Instead, the Map Server has its own tiling mechanism. See the *Tiling* section on page 37 for details.

The raster data may be used as non-transparent background images, or as semi-transparent weather or elevation data to be combined with other image or vector layers.

The NASA Earth Image¹ supplied as part of the sample dataset provides an example of using image tiles and fallback layers for different zoom factors. A shaded relief layer of the sample dataset demonstrates the use of shaded relief images² to visualize terrain elevation.

World Satellite Images with 250m and 15m per pixel resolution are available from Generic Logic as preprocessed datasets ready to be used with the Map Server. They are tiled for optimum performance and include a complete set of setup and configuration files. US Aerial Images for individual US states with 0.5 per pixel resolution can also be provided.

Vector Data

Since vector data can come in many different specialized formats, the Map Server reads its own GVF vector data file format. Files in this format often have the *.gvf* extension. In order to interface with the myriad other file formats available, the Map Server provides a programmable filter interface by which a file can be loaded through a filter that reads the original data and converts it to the Map Server's format. The converted data can also be converted just once and saved into a GVF file for efficiency. The Map Server provides convenience functions for writing GVF format to assist application developers in writing filters and data converters.

-
1. The Earth Image is provided by NASA. The following describes the image license for the earth image:

The imagery is free for use. The only restrictions are:

- (1) NASA requires that they be given a credit, as owners of the imagery
- (2) Visible Earth requests that you provide a credit (with URL if possible) for them (<http://visibleearth.nasa.gov/>), since you found the imagery with us.
- (3) VE requests that you provide a credit for the Earth Observatory team (<http://earthobservatory.nasa.gov/>) for generating the imagery.

Of course, only (1) is required, (2) and (3) are entirely optional and have no effect on your permission to use the imagery.

2. The shaded relief image is derived from the Global Land One-kilometer Base Elevation (GLOBE) Digital Elevation Model, Version 1.0., by:

GLOBE Task Team and others (Hastings, David A., Paula K. Dunbar, Gerald M. Elphinstone, Mark Bootz, Hiroshi Murakami, Hiroshi Maruyama, Hiroshi Masaharu, Peter Holland, John Payne, Nevin A. Bryant, Thomas L. Logan, J.-P. Muller, Gunter Schreier, and John S. MacDonald), eds., 1999.

The Global Land One-kilometer Base Elevation (GLOBE) Digital Elevation Model, Version 1.0.

National Oceanic and Atmospheric Administration, National Geophysical Data Center, 325 Broadway, Boulder, Colorado 80303, U.S.A.

Digital data base on the World Wide Web (URL: <http://www.ngdc.noaa.gov/mgg/topo/globe.html>) and CD-ROMs.

Some commonly used GIS data, such as Open Street Map (OSM), Digital Chart of the World (DCW) and US Census Tiger data, are available in the converted GVF format from Generic Logic. The converted data are preprocessed into hierarchical tiled layout for optimum performance with the Map Server and include a complete set of setup files.

The Map Server's vector file format supports two different types of objects: polygons, and markers with text. All polygons are represented as a group of points. Each point is defined as a horizontal and a vertical coordinate. Each marker is represented as one point which defines the location of the marker and a text string, if any, to be rendered at that location. The text may be rendered using either the vector or TrueType font.

Attributes

Layer Attributes

All common properties used to render vector object, such as Line Width or Fill Color, are not defined in the data files for each individual object. They are instead provided in the descriptions of the layer (LIF file) for efficiency. These properties are referred to as layer attributes.

In the programming interface, a resource mechanism similar to that of the GLG Toolkit is used. Resources and layer attributes refer to the same property but differ in their method of access. Layer attributes are initially defined in the layer's configuration file (LIF file), and then can be accessed via resources.

Object Attributes

Each object in a GVF data file can contain what are known as attributes, which is data other than points or strings associated with the object. Each object can have an arbitrary number of attributes, which are numbered, and can be used in the description of the data to customize how each object is displayed.

For example, if a data file contains marker objects that represent cities and towns, the object is also likely to contain information about its population. In this case, an object's first attribute might be the city's name, while the second attribute might be its population. The population attribute may be used to vary the size of the marker used to represent the city, as well as the size of the font used to display its name. It may also be used to show only large cities at the low zoom levels, and display smaller cities and towns as the map is zoomed in.

Attributes associated with data objects in the data file function are almost identical to attribute objects in the GLG Toolkit. Just as in the GLG Toolkit, property is often used as a synonym.

There are three types of attributes that the Map Server supports. A Double attribute (otherwise known as a D attribute) can be used to express a floating point value. The Geometrical type (3 floating point values, otherwise known as a G attribute) can be used to express a 3 dimensional set of coordinates or a color in the RGB color space. A String (S) attribute can be used for names, as well as any other attributes that cannot be encoded as D or G attributes. Sometimes, it may seem more appropriate to use an integer for an attribute, such as the population of a city or the number of precincts in a state, but D attributes are used throughout for consistency with the GLG Toolkit and various API function calls.

Attributes are not named in the traditional sense. Instead, their names are 0-based integral numbers. These numbers are defined in the data file. For example, the city name may be defined as attribute 0, the city area defined as attribute 1 and the population defined as attribute 2. Some objects may have attributes that are not present in other objects. For example, some cities might not have information about their area. So, some objects will have the area (attribute 1) and the population (attribute 2), while other objects lacking the area information will just have population (attribute 2). To handle objects that do not have a complete set of attributes present, the integral number representing the name of the attribute is encoded as part of the attribute record.

Attribute conditions may be used to display only the objects that match the attribute conditions. For example, attribute conditions can be used to display only cities with population greater than 500K. Attribute conditions may be defined in the layer's LIF file or in the *Layers* string. Defining attribute conditions in the *Layers* string allows the application to dynamically control which objects will be displayed in the layer. Refer to the ATTR COND layer attribute on page 40 and the *Layers* string on page 69 for more information.

The attribute conditions defined in the layer string are logically ANDed with attribute conditions defined in the layer's LIF file using the ATTR COND directive, and only the objects that match both sets of conditions will be displayed.

Attributes can also be used to change appearance of objects depending on their attributes. For example, the size of a city marker can be changed to reflect a city's population, so that bigger markers will be used for larger cities. The ATTRIBUTE MAP feature described on page 57 is an example of using object attributes to change the object's appearance.

Fonts

The map server supports both TrueType and vector fonts that can be used for rendering text labels in the generated map image. A font directory of the sample map data provided with the GLG installation (*map_data/fonts*) contains the following fonts:

True Type fonts:

Roboto

MgOpenModerna

Vector fonts:

romans

scriptc

Additional fonts can be added by placing font files into the font directory. Google Fonts catalog provides a variety of free TrueType fonts that may be browsed and downloaded from the following link:

<https://fonts.google.com>

The open source fonts in the Google Fonts catalog are published under licenses that allow you to use them in both commercial and non-commercial applications.

The font aliases feature described on page 34 allows using multiple fonts with different charsets to render layers with labels that contain characters in different languages, such as English, Korean, Arabic, etc.

2.2 Datasets, Layers, Tiles and Data

At the most fundamental level, there exists data, either in the form of a raster image or vector data such as polygons, markers and text. In order for the Map Server to read and interpret this data, it needs to know where it resides and how to render it. This is done with Layer Information Files (LIF), which usually have the *.lif* extension for recognition. A LIF describes a data file or a group of data files, and defines common properties such as Line Width and Fill Color. The layer is somewhat analogous to a Group Object in the GLG Toolkit.

When requesting a map image from the Map Server, the user is in essence requesting which layers he/she wants to appear in the image. Therefore, the Map Server must know which layers can possibly be requested by the user. In order to provide this information, a Server Dataset File is used. It often has the *.sdf* extension. The Server Dataset File contains information about all layers, where to find their LIF files, as well as the location of the font files for vector and TrueType fonts.

2.3 Configuration File Format

All configuration files, such as the SDF and LIF files, have a similar configuration entry format. Comments can appear as lines starting with a hash (#) symbol. All lines not starting with a hash symbol will be assumed to be configuration entries. All entries appear as at least one word, usually the name of the configuration variable, written in capital letters followed by an equal sign and then its value.

For example, the following line may be used in the configuration file to define the line width of polygons:

```
LINE WIDTH=1
```

Three data types of configuration variables are supported: D (double), G (XYZ or RGB values), or S (string). All D variables must be entered as a single floating point number. All G variables must be entered as three floating point numbers, either with white space in between each number or separated by commas. S variables can be entered either with or without quotes, except for the case described below where quotes are required.

Commas or white spaces are also used as separators between individual strings when a list containing several strings is used. For example, the *TEXT ANCHOR* option requires two anchor type values, one for horizontal and another for vertical anchoring, separated by either a white space or a comma:

TEXT ANCHOR= HLEFT VCENTER

If the user wants to assign a string containing either a white space or a comma to an S variable, quotes must be used. Hence:

```
FILENAME=file.gvf
FILENAME= file.gvf
FILENAME="file.gvf"
```

are all valid. However,

```
FILENAME=" file.gvf"
```

is not valid if the data file is indeed called *file.gvf*, because use of quotes will pick up the white space within them. The following shows an example of setting a configuration variable value where the quotes are required due to spaces in the variable value:

```
FILL TYPE= "FILL AND EDGE"
```

An empty string may be defined by two quote characters: "".

The backslash has a special meaning and is used as an escape character. The following C-style escape characters are supported in strings:

```
\n - a new line character
\" - a literal quote character
\\ - a literal backslash character
```

The backslash may also be used to escape comma and white space separators. The following shows an alternative way of defining FILL AND EDGE fill type:

```
FILL TYPE= FILL\ AND\ EDGE
```

While it is a valid entry, using double quotation marks around the string with spaces and/or commas may still be a more readable approach.

All undefined variables assume reasonable default values. For example, the *LINE WIDTH* variable takes the default value of 1, i.e. one pixel thick lines will be rendered for polygons in that layer.

If variables are redefined, the last definition is the one which will be used, unless some intermediate variables depend on a previous definition.

Datasets

A Server Dataset File (SDF) enumerates all possible layers that the user can request as well as defines names for them, defines all fonts the Map Server has access to, and also specifies root directories where fonts and layers can be found.

What follows is a list of all possible configuration variables in an SDF file.

GLM ROOT DIR=<dir>

Defines where to look for all files; may also be relative to current directory. All files needed for the Map Server are relative to the GLM ROOT DIR unless an absolute path is used. If this variable is not specified in the SDF, it will default to the directory in which the SDF itself resides. The location of the SDF is specified as a command line option upon invocation of the Map Server executable, so the GLM ROOT DIR may be inherent to that invocation.

LAYER DIR=<dir>

Defines where the Map Server should look for layers. This entry is either an absolute filename or relative to the GLM ROOT DIR. If any relative path for a layer is later specified, it will be assumed to be relative to the LAYER DIR.

FONT DIR=<dir>

Defines where the Map Server should look for vector and TrueType font files. This entry is either an absolute filename or relative to the GLM ROOT DIR. If any relative path for a font is later specified, it will be assumed to be relative to the FONT DIR.

LAYER=<name> <lif>

Defines where to find a layer. Must specify the name by which this layer is later referred to in a query and where to find a LIF file for that layer. The path is relative the LAYER DIR unless an absolute path is used. This variable definition may be used multiple times in succession to define multiple layers.

ALIAS=<name> <value>

Defines an alias which maps a single alias name to a list of layer names specified by the alias value. Aliases may be used to hide details of the dataset layers organization, providing a convenient way to reference collections of layers. When an alias name is encountered in the layer string, it is replaced with its value, and the process is repeated recursively to handle any aliases which may be a part of the substituted alias value. The alias value must be quoted if it contains spaces, commas and any other special characters. The alias name should be different from the names of layers, fonts and reserved GLG resources.

The alias value may also contain attribute conditions described in the *Attribute Condition Syntax* section on page 71. For example, if the *city* layer has population as the second attribute (index=1), the following alias:

```
ALIAS=big_cities "earth,cities@1>500000"
```

may be used in the layer string to display cities with population greater than 500000 with the earth image in the background.

To avoid ambiguities, the “++” override prefix is not allowed in front of the alias name when it is used in the layer string but may be used as prefixes of layer names in the alias value. The “-” prefix may be used in front of the alias name in the layer string to disable the alias. However, it does not disable the individual layers in the alias if these layers are explicitly listed in the layer string. The “-” may also be used as a prefix of a layer name in the alias value to selectively disable the layer.

ZOOM FACTOR TYPE= <*EXTENT* or “*EXTENT AND WIDTH*”>

Defines how the zoom factor is calculated. If it is set to *EXTENT* (default), the zoom factor is calculated based on the map’s GIS extent and will not change when the size of the map image changes. If it is set to *EXTENT AND WIDTH*, the zoom factor will also increase or decrease proportionally to the new map size, with the exact ratio being controlled by the *BASE WIDTH* attribute described below. This may be used to display more detailed layers for larger maps. Refer to the *The Zoom Factor* section on page 37 for details.

BASE WIDTH= <*value*>

Defines the base size of the map (600 by default) for the *ZOOM FACTOR TYPE=EXTENT AND WIDTH*. When the size of the generated map image changes, the zoom factor is changed by the a ratio of the map pixel width to the value of the base width attribute. For example, if the map width increases to twice the value of the *BASE WIDTH* attribute, the zoom factor will increase by the factor of 2.

Since the zoom factor calculation depends on the value of *BASE WIDTH*, the value of the *BASE WIDTH* attribute may be used to adjust at what point the map layers are activated and deactivated. For example, if *BASE WIDTH* is increased by the factor of two, it will decrease the calculated Zoom Factor by the same factor, and all layers whose visibility is controlled by the *MIN ZOOM* attribute will be activated when the map is zoomed in twice more compared to the previous *BASE WIDTH* value.

VECTOR FONT=<*font_name*> <*font_file*>

Defines where to find a vector font. Must specify the name by which this font is later referred to and where to find the data for the vector font. The path is relative the FONT DIR unless an absolute path is used. The Map Server supports Hershey vector fonts. This variable definition may be used multiple times in succession to define multiple fonts.

IMAGE FONT=<*font_name*> <*font_file*>

Defines where to find a TrueType font. Must specify the name by which this font is later referred to and where to find the data for the font. The path is relative the FONT DIR, unless an absolute path is used. This variable definition may be used multiple times in succession to define multiple fonts. The unicode characters are supported with the TrueType fonts.

Any TrueType font installed on the system can be used with the Map Server by copying it into the Map Server’s font directory (or creating a symbolic link on Linux/Unix systems). If only TrueType fonts and no vector fonts are used, the FONT DIR may be set to point to the system’s font directory.

The TrueType font support is provided via the freetype library¹.

FONT ALIAS=<*font_alias_name*> “<*font_name*>, <*font_name*>, ...”

Font alias defines a comma-separated list of fonts that can be used to render layers with labels that contain characters in different languages, such as English, Korean, Arabic, etc. Such layers can use the font alias name instead of a font name as a value of the FONT variable in the layer’s LIF file.

1. Copyright 2006, 2007, 2008, 2009, 2010 by David Turner, Robert Wilhelm, and Werner Lemberg.

When a label containing characters in multiple charsets is rendered, for each of the label's characters, the renderer searches all fonts defined in the font alias and uses the first font that supports the character's charset. This makes it possible not only to use labels that use different charsets but also mix characters from different charsets in the same label.

Note: Vector fonts can't be used with font aliases.

DEFAULT FONT=<font_name>

Defines the dataset's default font. It is used for all text labels that do not specify the font explicitly. The default value is *roboto*.

ERROR FONT=<font_name>

Defines the dataset's error font used for rendering error messages. If omitted, the dataset's default font is used.

ENCODING=<UTF8 or LATIN>

Specifies the encoding used for the text strings. The default encoding is *UTF8*.

ALLOW OVERRIDE=<0, 1 or 2>

Defines the default setting for the use of the “++” override in the layer string. If set to 0, disallows the use of the override for all layers. Setting it to 1 (default) allows overrides for untiled layers, and suppresses overrides for tiled layers to avoid unintended consumption of significant memory and CPU resources. The setting of 2 allows overrides for all layers of the dataset.

MAX IMAGE TILES

Defines the maximum number of all image tiles that may be kept in the tile cache. While the *MAX CACHE SIZE* parameter of the layer controls the per-layer cache size, *MAX IMAGE TILES* may be used to control the total maximum size of the image tile cache for all layers, which is useful for datasets with a large number of layers.

The default value of -1 allows the unlimited total size of the image tile cache. If the parameter is set to a positive value, the image tile cache will be flushed if necessary after rendering each layer to prevent uncontrolled growth of the cache.

MAX VECTOR TILES

Defines the maximum number of all vector tiles that may be kept in the tile cache. While the *MAX CACHE SIZE* parameter of the layer controls the per-layer cache size, *MAX VECTOR TILES* may be used to control the total maximum size of the vector tile cache for all layers, which is useful for datasets with a large number of layers.

The default value of -1 allows the unlimited total size of the vector tile cache. If the parameter is set to a positive value, the vector tile cache will be flushed if necessary after rendering each layer to prevent uncontrolled growth of the cache.

GLYPH CACHE TYPE=<*type*>

Specifies the type of the glyph cache to use and may be set to *NONE* or *GLOBAL* (default). The cache is used to increase performance of TrueType font rendering.

MAX ITERATIONS=<*number*>

Defines the maximum number of internal iterations used to render filled polygons (default 10000), which prevents rendering delays if the map is zoomed in too much. This parameter may need to be increased if large filled polygons (such as OSM country or state areas) are rendered with very high zoom levels.

A Simple Example of an SDF file

```
GLM ROOT DIR=/usr/local/glm/map_server/
LAYER DIR=layers
FONT DIR=fonts
LAYER=earth earth.lif
LAYER=cities /home/user/cities.lif
VECTOR FONT=roman romans.data
IMAGE FONT=roboto_m Roboto-Medium.ttf
DEFAULT FONT=roboto_m
```

In this example, The Map Server will look for *earth.lif* in */usr/local/glm/map_server/layers/earth.lif*, but *cities.lif* will be found in */home/user/cities.lif*. It will look for *romans.data* and *Roboto-Medium.ttf* font files in the */usr/local/glm/map_server/fonts* directory.

Layers and Tiles

Layers are the best way an end user of a mapping application can interact with the Map Server. When a user requests a map image, he/she will be specifying which layers they want to be displayed. These layers are then superimposed in the final map image in the order they are specified in the query string. Each layer is described in a Layer Information File (LIF). This file contains information on what format the data is in, where to locate the data, and common attributes to assign to the data such as colors and line width.

There are a number of ways to control whether a layer is rendered into a map image or not. Primarily, it will be the end user that specifies which layers he/she wishes to be rendered into the final map image. However, there are a number of mechanisms that turn the layer on or off automatically, without user interaction.

When setting up the Map Server, any layer can be defined to be on by default. By specifying the *default* layer, all layers that have the default flag will be turned on. “*default*” is a reserved layer name which is associated with all layer with the flag turned on. For example, a query string that requests *default,clouds,cities* will render all default layers as well as the clouds and cities layer. A layer may also be “subtracted” from the default set of layers by using the “-” prefix. For example, the *default,-cities* query string will render all default layers but the cities.

Also, for those layers that utilize tiles, a maximum number of tiles may be defined. If the number of tiles required to render the map image exceeds a predefined number, the layer is not rendered, or a fallback layer is rendered instead. For example, if looking at the entire earth, a lower resolution earth image will be used, because the maximum number of tiles for the detailed earth image will be exceeded.

Finally, if the zoom factor exceeds a predefined number defined by the *MAX ZOOM* attribute or is less than a predefined number defined by *MIN ZOOM*, the layer will not be rendered. For example, a roads layer is only necessary when zoomed in sufficiently to distinguish unique roads.

The Zoom Factor

The Zoom Factor is a ratio that measures how “zoomed in” a map image is. A simple way to see what the Zoom Factor is for a specific map request is to increase the verbosity, which will cause the Zoom Factor to be printed to the error interface, *stderr*, as well as written in the image. By default, the Zoom Factor is the same for all map layers and is the reciprocal of the proportion of the entire world that is visible. For example, if the entire world is visible, the Zoom Factor will be 1. If one half of the world is visible, the Zoom Factor will be 2. If the proportion of the horizontal extent of the layer or world is different from the proportion of the vertical extent visible, the greater of the two Zoom Factors is used.

For layers with relative zoom enabled, the Zoom Factor is calculated for that specific layer as the reciprocal of the portion of the layer that is visible. Refer to the *RELATIVE ZOOM* layer attribute on page 38 for more information.

The dataset’s *ZOOM FACTOR TYPE* attribute may be used to change the Zoom Factor depending on the map image size. When the *ZOOM FACTOR TYPE* is set to *EXTENT* (default), the Zoom Factor is calculated based on the map’s GIS extent and will not change when the size of the map image changes. If it is set to *EXTENT AND WIDTH*, the Zoom Factor will also increase or decrease proportionally to the new map size. The *BASE WIDTH* attribute of the dataset defines the base map size used in zoom factor calculations. For example, if the width of the map increases to twice the value of the *BASE WIDTH* attribute, the Zoom Factor will increase by the factor of 2. Refer to the description of the *ZOOM FACTOR TYPE* and *BASE WIDTH* attributes on page 34 for details.

Tiling

In many cases, it is not feasible or necessary to load all data into the Map Server. For example, if looking at only one country, it is not necessary to load the entire image of the earth. Also, the entire earth image takes 400 megabytes of space, so using it would utilize a lot of memory and make the Map Server unnecessarily slow. Instead, the Map Server provides a tiling mechanism to alleviate this problem. Only the tiles needed to render a given map image are loaded, which cuts down on memory usage and enhances performance. Tiling capabilities exist for both raster image and vector data. There exist two types of tiling: Rectangular and Hierarchical.

Rectangular tiling is used for image layers and vector data split into rectangular tiles. However, vector data is often split and provided with non rectangular tiles. This happens if the data is grouped by political boundaries, for example. In this case, the Map Server’s Hierarchical Tiling can read a

directory structure with all the tiles in various locations and use only the ones necessary. For example, the directory hierarchy may contain directories with countries, which contain directories with provinces or states, which may contain directories with counties and/or towns.

LIF files for hierarchical tiling provide the extent of the layer, including all sublayers. This allows the Map Server to quickly read the LIF files to determine which tiles are needed to generate the requested image. The Map Server provides a utility to automatically generate hierarchical LIF files.

Configuration Variables

What follows is a list of all possible configuration variables in a LIF.

TYPE=<type>

Defines a layer's type and corresponds to the type of data used to render the layer. Can be either:

IMAGE (default)
VECTOR
GRID
OUTLINE
BACKGROUND

The first two layer types correspond to the type of their data: raster or vector. These are the main GIS layers that render either raster or vector data of the map image.

The last three types are special vector layers that have no data file associated with them and are defined completely within its respective LIF. The grid layer draws grid and grid labels, the outline layer draws a filled or unfilled outline. For example, an oval globe outline is drawn in the *ORTHOGRAPHIC* projection, and a rectangular outline of the whole world is drawn in the *RECTANGULAR* projection. The background layer is used to render a gradient around the globe in the *ORTHOGRAPHIC* projection.

Any variables which do not apply to a given layer type will be ignored. For example, the format of the image data clearly does not apply to a vector layer, so it will be ignored if defined for a vector layer.

DEFAULT ON=<0 or 1>

Defines whether or not a layer will be turned on by default, i.e. when a *default* layer name is included in the map request. The default value is 0, meaning it will not be included. Setting the value to 1 will include the layer into the "default" layer.

RELATIVE ZOOM=<0 or 1>

Specifies whether or not the relative zoom feature of the Map Server is used for this layer. This variable can take a value of either 0 (default) or 1. If set to 1, the layer's zoom factor is calculated using the extent of the layer. If set to 0, the zoom factor is calculated using the entire extent of the world.

MIN ZOOM and *MAX ZOOM* attributes can be used to turn layers on or off depending on the map zoom level. See the *Layers and Tiles* section on page 36 for details.

In most cases, it is preferable to disable *RELATIVE ZOOM*, because the user often does not know and does not need to know the extent of the layer. However, if a layer provides detailed imagery for a small area, relative zoom may be used to activate the layer when the map is zoomed on the area: the layer's zoom factor will be 1 when the map extent equals to the layer extent.

TRANS TYPE=<type>

Defines what blending function is to be used when rendering the layer. This variable is very similar to the use of blending functions in OpenGL. The following values are supported:

OPAQUE (default)
ALPHA WHITE OPAQUE
ALPHA BLACK OPAQUE
TRANSPARENT COLOR OPAQUE
XOR
BUMP MAP

The most common type is *OPAQUE*. This name is slightly misleading because it does not mean that the layer is necessarily fully opaque. What it does mean is that if its transparency is turned off (*ALPHA*=1), it will be fully opaque. The other two blending functions render layers similarly to how OpenGL renders light-maps. When set to *ALPHA WHITE OPAQUE*, the brighter the color, the more opaque it is. This option is commonly used when rendering transparent cloud layers, as a black pixel in the cloud image specifies no cloud cover, while a white pixels specifies full cloud cover. When set to *ALPHA BLACK OPAQUE*, the darker the color, the more opaque it is. This is identical to the default blending function for the second OpenGL texture unit (used for light maps). A white pixel will be transparent, while a black pixel will be opaque.

The *TRANSPARENT COLOR OPAQUE* type is similar to *OPAQUE*, but with the color defined by the *TRANS COLOR* LIF parameter rendered as transparent. If the type is set to *XOR*, the color will be XOR'ed with the pixel in the image. This causes lines to stand out, and is very useful for grid layers, where it is desirable to have grid lines that always contrast with the background.

The *BUMP MAP* option is used for rendering shaded relief images.

TRANS COLOR=<*r,g,b*>

Defines which color to render as transparent for the *TRANSPARENT COLOR OPAQUE* rendering mode. The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white.

TRANS COLOR PRECISION=<value>

Defines a maximum distance in RGB color space (in the range from 0 to 1) used to determine a color's transparency (default 0). While the TIFF and PNG formats are preferred for images that use *TRANSPARENT COLOR OPAQUE* rendering mode, *TRANS COLOR PRECISION* may be used with JPEG images to overcome the artifacts of JPEG's lossy compression, which slightly changes pixel colors. All colors within this distance from the transparent colors will be rendered as transparent. A typical parameter value is in the range from 0 to 0.5. If the value of 0 is used (default), only pixels with the exact transparent color value will be transparent.

ELEVATION MODE=<mode>

If set to a value other than *NONE*, defines the layer as an elevation layer. The following values are supported:

NONE (default)
DATA
COLOR

Elevation layers use elevation data in the form of single-channel TIFF files, where every pixel represents elevation data for the corresponding location. TIFF files with the *int16*, *uint16* and *float32* data formats are supported. DEM and DTED to TIFF converter is provided, refer to the *Image and Elevation Data Import Utility* section on page 95 of the *Tools and Utilities* chapter for more information.

If the mode is set to *DATA*, the layer will not be displayed but will be used for elevation queries, to return elevation values for points on the map.

If the mode is set to *COLOR*, the layer will be displayed as a color-coded image. This mode is also convenient for visual testing of elevation data before using them in the *DATA* mode. The *PIXEL MAP* parameter may be used to provide a custom color threshold table that will be used to convert elevation values to different colors.

If both color display and elevation queries are required, two separate layers may be set up using the same data file.

ALPHA=<value>

Defines the transparency of the layer, ranging from 0 to 1. An alpha value of 0 will make the layer invisible, while an alpha value of 1 (default) will make the layer fully opaque according to the blending function. If the user wishes to make a solid transparent layer, the *TRANS TYPE* should be set to *OPAQUE* and the alpha value changed. If any of the other blending functions are used, the transparency of any given pixel will depend on its color. For example, if the user wishes to see transparent county overlays over a state map, the values for the *TRANS TYPE* and *ALPHA* mentioned above should be used.

Note: Although setting the *ALPHA* to 0 is identical to turning the layer off in the map request, this should be avoided, as it will be slower, since every object will still be traversed.

ATTR COND=<value>

Defines an attribute condition. Only the object whose attribute values match the attribute condition will be displayed. For example, if the second attribute (index=1) of the cities layer contains city's population, the following attribute condition may be used to display only cities with population greater than or equal to 100K but less than 500K:

```
ATTR COND= @1 >= 100000 && @1 < 500000
```

While spaces in the value of the above attribute condition are optional, each part of the attribute condition value is a separate string, which means that the value cannot be surrounded with quotation marks. However, if a string with spaces is used in string attribute conditions, it may be quoted:

ATTR COND= @0 == “Las Vegas” || @0 == “New *”

Refer to the *Attribute Condition Syntax* section on page 71 for more information.

The attribute conditions defined in the ATTR COND directive are logically ANDed with attribute conditions defined in the *Layers* string, and only the objects that match both sets of conditions will be displayed.

FILENAME=<file>

Defines where to look for the data file(s). This path is relative to the path of the LIF itself, unless an absolute path is defined. If there are multiple data files, such as tiles, the filename is specified with ‘%’ characters which will be replaced by numbers when the file is loaded. For example, if tiles reside in files called:

```
image_0_0.jpg
image_0_1.jpg
image_0_2.jpg
image_1_0.jpg
image_1_1.jpg
...
image_2_2.jpg
```

then the *FILENAME* should be set to *image_%%.jpg* and the correct values will automatically be filled in:

FILENAME= image_%%.jpg

VECTOR FORMAT=<type>

An optional parameter that defines the format of the vector data used by the vector layer. The following values are supported:

```
VECTOR FORMAT=GVF
VECTOR FORMAT=SHP
```

GVF is the native format of the Map Server. The *SHP* format is used for shape files. If *VECTOR FORMAT* is omitted, the map server tries to determine the data format based on the vector file extension.

For maximum performance, shape files should be converted to the native GVF format. However, the *SHP* format may be used for quick testing of the shape files, when maximum performance is not important. To convert shape files to GVF format, use the shape file conversion utility described in the *Tools and Utilities* chapter on page 77.

FILTER=<file>

Defines the path of an executable file which contains a filter to convert vector data from a custom format to the Map Server’s format. The advantage of using a filter is to avoid creation of temporary files, as the output of the filter is read directly into the Map Server. However, this approach is slower

than running the filter beforehand and using an already converted file instead. The path defined is relative to the LIF itself unless an absolute path is defined. A guide to writing filters can be found in the *GVF Filters and Data Converters* chapter on page 141.

IMAGE FORMAT=<*format*>

An optional parameter that defines the format of the input image data used by the raster layer, or the format of the custom icon image for vector layers. The following values are supported:

JPEG
PNG
TIF

JPEG images provide more efficient compression and may be used to minimize the size of a dataset. TIF and PNG images provide lossless compression and are preferred in cases when preserving exact pixel values is important. For example, when TRANSPARENT COLOR OPAQUE blending function is used, JPEG's lossy compression would alter the exact pixel values used to determine the pixel's transparency, resulting in visible compression artifacts. If *IMAGE FORMAT* is omitted, the map server tries to determine the format of the image based on the file extension.

IMAGE ANTIALISING=<0 or 1>

Enables or disables image antialiasing for individual image layers. Image antialiasing is enabled by default.

FALLBACK LAYER=<*layer name*>

Defines the name of a fallback layer. If the layer exceeds the maximum memory allowance specified by *MAX TILES*, the fallback layer will be activated instead. For example, this parameter may be used to specify a lower resolution image of the earth, so that if the entire earth is being viewed, a lower resolution image is displayed, while higher resolution images will be used when viewing detailed maps. Without a fallback layer, all tiles of the detailed image would be loaded when the entire earth is displayed, consuming excessive CPU and memory resources.

The same effect might be achieved using the *MAX ZOOM* and *MIN ZOOM* parameters of several layers, so that the layers are switched depending on the zoom factor. The *FALLBACK LAYER* provides an alternative way to switch layers based on the maximum number of used tiles, which may be more convenient. It also provides a way to chain several layers, by specifying a *FALLBACK LAYER* parameter for the fallback layer as well.

For example, a high-resolution tiled layer may switch to a medium-resolution tiled layer when the number of used tiles exceeds 10. The medium-resolution layer may switch to a layer with a single image file when it's number of used tiles exceed 10 as well. All this may be accomplished by specifying medium-resolution layer as a fallback layer for the high-resolution one, and specifying a single image layer as a fallback layer for the medium-resolution layer. *MIN ZOOM* and *MAX ZOOM* limits of the layer may be used to further control the layer's visibility at different zoom levels.

FALLBACK=<file> (deprecated: use *FALLBACK LAYER*)

Defines where to find fallback data. If the file specified with the *FILENAME* variables exceeds the maximum memory allowance specified by *MAX TILES*, this file will be used instead. The file is relative to the path of the LIF itself, unless an absolute path is defined.

This parameter is now obsolete and supported only for backward compatibility. The *FALLBACK LAYER* parameter listed above provides the same functionality in a cleaner way and with more features.

MAX TILES=<number>

Defines the maximum number of tiles a layer can require before the fallback is activated. This variable should be set according to the amount of memory the user wishes to use for the Map Server. In addition, it should be set so that when a map consists of the area which requires *MAX TILES* tiles, the *FALLBACK* data should have sufficient detail when the area is viewed. The default value of -1 allows unlimited number of tiles.

MAX CACHE SIZE=<number>

Defines the maximum size of the layer's tile cache. This variable should be set according to the amount of memory the user wishes to use for the Map Server. It should be set to a value greater or equal to the *MAX TILES* variable to avoid tile swapping. The default value of -1 disables cache size limit.

NUM TILES X=<number>

NUM TILES Y=<number>

Define how many tiles exist in the horizontal and vertical directions, respectively. These numbers are used to calculate tile indices used to replace the '%' characters specified in the filename. If the '%' are missing, the indices will be appended to the filename.

MIN PIXEL SIZE=<number>

Specifies the minimal pixel size for map decluttering. All polygons whose pixel extent at the current zoom level is smaller than this parameter will not be drawn. The parameter may be set to a fractional value. The default value of the parameter is 0.1, which eliminates small polygons only when their extent becomes much smaller than one pixel. It may be set to a higher integer value, such as 2 or 3, to speed up rendering of a map that contains a lot of small polygons, such as an OSM water layer. Setting it to 0 disables decluttering and draws all small polygons.

An attribute map may be attached to this resource to control the minimum pixel size for each individual vector feature of a vector layer depending on a value of the feature's attribute.

Note: *MIN PIXEL SIZE*=1 may result in visible gaps for layers that contain lines represented by a large number of small segments, such as political boundaries. If the segment's size is less than or equal to 1, the segment will be eliminated. Set the parameter value to 0 to avoid it. The parameter works best for layers that contain bodies of water, with a large number of small lakes.

CLIP=<0 or 1>

Enables or disables clipping of the layer's marker and polygon features (enabled by default). The clipping is performed based on the lat/lon coordinates of the marker and polygon features, while their labels may extend outside their point boundaries. The clipping speeds up rendering but may be disabled to show partially visible marker and polygon labels which otherwise might be completely clipped out.

VOLATILE=<0 or 1>

Defines whether or not a layer is volatile. The data files for the volatile layers will be checked on every new map generation request, to find out if they changed and need to be re-read. The volatile setting may be used to update changing layers, such as current weather or radar maps. The default value is 0 (non-volatile).

LOCK FILE=<file>

Defines the lock file for volatile layers. If the variable is defined and the file does not exist, the Map Server waits for this file to be written before attempting to read layer's data files. This is used to synchronize the Map Server's data reads with volatile layer's data updates. The process that updates the volatile layer's data must remove the lock file, update the data and write the lock file back. The parameter is used with *REDIRECT FILE*, see below.

LOCK TIMEOUT=<number>

Defines the maximum time in seconds the Map Server waits for the lock file before proceeding to read data. The default value is 0.05 seconds.

REDIRECT FILE=<file>

Specifies the name of the file that contains *REDIRECT PREFIX* for volatile datasets. See *REDIRECT PREFIX* below.

REDIRECT PREFIX=<directory or prefix>

Used only in the redirect file to define a redirect prefix for volatile datasets. The prefix is appended in front of the dataset's *FILENAME* or *TILES_SUBLIF* before it is used to retrieve the data, and allows updating volatile datasets without compromising their data integrity while the data are being rewritten. The map server uses the old version of the data, while the new version is being written into a separate file or directory. When the new data are ready, the update process rewrites the content of the redirect file to provide a new data location. To avoid errors while the redirect file is being rewritten, the lock file is removed before rewriting the redirect file and is written back when finished, forcing the map server to wait. Since the redirect file is small, the lock wait time is very small as well and does not affect performance of the map server.

If the value redirect prefix ends with "/", it defines the name of a directory where the dataset is located. This is usually used for tiled datasets with a number of tiles. If the value of the redirect prefix does not contain "/", it specifies a file name prefix, which may be used for untiled files with just one data file.

Note: The redirect prefix is ignored if the path it is supposed to be appended to is defined using an absolute path name.

VOLATILE ERRORS=<0 or 1>

Defines whether or not correctable errors are reported for volatile errors. For example, if a volatile layer data file is in the process of being overwritten with new data, the map server will use the last valid data for this layer, and the error will not be reported if the variable is set to 0 (default). The default value is 1. (errors are reported). If *REDIRECT FILE* is used, the volatile errors must not happen.

ALLOW MISSING TILES=<0 or 1>

If set to 1, allows missing tiles in the square tiling mode. This is always required for vector layers with square tiles, as the vector tiling utility does not output tiles for regions that do not contain any vector features. The default value is 0.

TILES OVERLAP=<0 or 1>

May be set to 1 to indicate that tiles can overlap (default 0). This is used for satellite and areal imagery datasets that use UTM coordinates. Converting this tiles to the lat/lon coordinates results in non-rectangular tiles. The tiles use padding on the sides to make tile images rectangular. The unused area used for padding uses black color to indicate this area is not used and should be transparent. Satellite imagery datasets also use black color for masking out areas in the middle of an ocean, which have no data to minimize the size of an already huge dataset.

When tiles are used to compose a map image, the unused transparent padding areas of neighboring tiles may overlap, which requires additional processing when searching for a tile that should be used to render a particular pixel in the map image. Setting *TILES OVERLAP* to 1 activates this additional processing.

The *TRANS TYPE*, *TRANS COLOR* and *TRANS COLOR PRECISION* variables are used to make transparent the unused areas rendered using the black color, as shown in the following example:

```
TRANS TYPE="TRANSPARENT COLOR OPAQUE"  
TRANS COLOR=0 0 0  
TRANS COLOR PRECISION=0.025
```

The *TRANS COLOR PRECISION* is used to avoid JPEG compressing artifacts for images that were imported from JPEG.

The *BACKGROUND COLOR* and *BACKGROUND LAYER* variable may be used to provide a background for transparent areas, such as areas in the middle of an ocean.

BACKGROUND COLOR=<*r,g,b*>

Specifies a background color to use for rendering pixels for which no data is provided by the current layer, such as transparent areas in the middle of the ocean or the areas outside of the current layer data coverage. The default value (-1,-1,-1) disables background color.

BACKGROUND LAYER=<layer name>

Specifies a name of a layer to use for rendering pixels outside of the data coverage of the current layer. In the absence of a background color, the background layer is also used for rendering transparent pixels of the current layer, such as transparent areas in the middle of the ocean.

WIDTH=<number>

HEIGHT=<number>

Define the width and height of the image in pixels for image layers. For the tiled datasets with square tiles, these parameters are required to provide the extent of the whole image, which is used for tile math. If the width and height are not specified correctly, the image will not be tiled correctly and tiles might appear in wrong places on the map.

These parameters are optional for layers with a single untiled image file, as the width and height of the image are auto-detected. If the width and height parameters are supplied, they are compared with the ones detected in the image file, issuing a mismatch warning if necessary.

These parameters are not used and are ignored for hierarchical directory-based image layers with tree tiling. The layer width and height information is obtained from the TILES SUBLIF file, which is automatically generated by the tiling setup utility.

MIN LON=<number>

MIN LAT=<number>

MAX LON=<number>

MAX LAT=<number>

Defines the extent of the longitude and latitude in the layer and is mandatory for all image and vector layers that use square tiling. It is also necessary for untiled image layers where the raster image data itself includes no specification of what extent the image occupies.

For untiled vector layers these parameters are not required but are still useful for optimal performance: if they are specified, the layer may be efficiently clipped without reading its data file. The bounding box utility described in the *Bounding Box Extraction Utility* chapter on page 80 may be used to obtain the extent information for vector data files.

For the layers that use tree tiling, these parameters are not required, as they are provided by the SLF setup files of such layers.

MIN ZOOM=<number>

MAX ZOOM=<number>

Defines the minimum and maximum zoom factor at which the layer should be rendered. If the zoom factor is below or above these values, the layer will be omitted from rendering. These variables are useful for switching layers with various levels of detail depending on the zoom factor.

An attribute map may be attached to these attributes to control the minimum and maximum zoom limits for displaying each individual vector feature of a vector layer depending on a value of the feature's attribute. The original values of the *MIN ZOOM* and *MAX ZOOM* before applying the attribute map still control the zoom limits for the whole layer.

The default value is -1 which disables minimum and maximum zoom thresholds.

CLIP TYPE=<type>

Defines an optional clip type used to limit the layer output to the clip area. The following value are supported:

NONE (default)

No clipping is performed.

BOX (vector layers only)

The GIS features defined in the layer will be drawn if they are completely inside the clip box or partially intersect it. If a feature intersects the clip box and is drawn, it may extent outside of the box. This clip type has minimal performance penalty but may yield unexpected results for polygon features.

POLYGON PIXELS (vector layers only)

Performs exact clipping of polygon features to the clip box. Polygon feature labels may still extend outside the box.

ALL PIXELS

Performs exact clipping of all layer output to the defined clip box. All features defined in the layer as well as their labels will be clipped to the clip area.

For image layers and map selection requests, the *BOX* and *POLYGON PIXELS* clip types work the same way as *ALL PIXELS*.

CLIP MIN LON=<number>

CLIP MIN LAT=<number>

CLIP MAX LON=<number>

CLIP MAX LAT=<number>

Defines an optional longitude and latitude extent of the layer area to be drawn. While the MIN/MAX LAT/LON parameters define the extend of the data available for the layer, the CLIP parameters may be used to limit the drawn part of the layer to the defined clip box.

For image layers, the output is indeed clipped to the provided DRAW box. For the vector layers, all vector features that are either completely inside the box or partially intersect it will be drawn. As a result, the rendered vector features may extend outside the DRAW box.

GRID LAT INTERVAL=<number>

GRID LON INTERVAL=<number>

Defines the interval between neighboring horizontal (of equal latitude) and vertical (of equal longitude) lines. These variables are only useful if the layer type is set to *GRID* and *ADAPTIVE*=0. The default value is 10 for the latitude and 20 for longitude interval.

GRID MIN LON=<number>

GRID MIN LAT=<number>

GRID MAX LON=<number>

GRID MAX LAT=<number>

Defines the extent of grid lines in a grid layer. By default, these variables is set to encompass the entire globe. These variables are useful only if the layer type is set to *GRID* and *ADAPTIVE*=0.

ADAPTIVE GRID=<number>

Defines an adaptive grid if set to non-zero value (default 0). The Map Server will attempt to divide the extent of the map image into roughly the number of grid lines specified by the variable's value, while placing the grid lines on round numbers such as 10, 25, 50, etc. or 0.1, 0.2, 0.5, etc. if the image spans a small area.

GRID LABELS=<0, 1, 2, 3 or 4>

Defines whether or not to render labels next to the grid lines of the grid layer. A value of 0 (default) will not render grid labels, value of 1 will render labels on one side of the map, and the value of 2 will render them on both sides, except for the special cases of the ORTHOGRAPHIC projection (when one of the poles is visible, or the whole unzoomed globe is displayed), which will render labels at the center.

The values of 3 and 4 are used for a special case when a map in the ORTHOGRAPHIC projection is clipped to generate zoomed views of the globe with a visible horizon line. The value of 3 will display grid labels on three sides of the map (left, right and bottom), and the value of 4 will display labels on all four sides.

DECIMAL GRID LABELS=<0 or 1>

Defines the format for displaying the fractional part of the grid labels: if set to 1 (default), decimals are used; if set to 0, minutes and seconds are used.

GRID LABEL FORMAT=<format>

Specifies a custom format for grid labels. For the decimal grid labels, the format must contain a double C format specifier, otherwise it must contain up to four long integer C format specifiers for the degrees, minutes, seconds and deciseconds. For example, the following string may be used to display non-decimal grid labels with degrees, minus and seconds, with no deciseconds:

“%.3ld deg %.2ld’%.2ld”

If grid labels use a True Type font, a Unicode degree character may be embedded in the format string.

GRID LABEL FORMAT LON=<format>

Specifies a custom format for the longitudinal grid labels if they must have a different format than the latitudinal grid labels.

GRID LABEL CHAR=<0, 1 or 2>

Controls the use of the directional character (*N*, *S*, *E* or *W*) added to the label to indicate direction and may have the following values:

NONE
APPEND (*default*)
PREPEND

If set to *NONE*, no directional character will be added and a sign will be used instead. The *APPEND* or *PREPEND* values may be used to add the directional character at the beginning or at the end of the label string.

ROTATE LON LABELS=<0 or 1>

If set to 1 (default), the longitudinal grid labels will be rotated to be drawn along the vertical grid lines. If set to 0, the labels will be drawn horizontally.

LON LABEL ANGLE=<value>

Defines an optional rotation angle for the longitudinal grid labels, 90 degrees by default.

FONT=<name>

Defines the font to be used for text rendered from this layer. The name is a vector or TrueType font name specified in the SDF file. If omitted, a default font is used.

Font aliases described on page 34 may be used to define a list of fonts with different charsets for rendering labels containing characters in multiple languages. To use a font alias, use the font alias name instead of a font name.

FONT SCALE=<number>

For vector fonts, defines a multiplier for font units. A multiplier of 1 will render characters of the given font in their original size. A multiplier of 0.5 will render characters one half the size defined by the font.

For the image (TrueType) fonts, defines the font size in pixels.

The default value (12) is optimized for the TrueType fonts.

TEXT TYPE=<type>

Defines how the font will be rendered and how font units will be defined. This variable has three supported values:

FLAT
MAPPED FLAT (*default*)
TRANSFORMED

If set to *FLAT*, the location of the text is defined in terms of pixels from the top left corner. This also means that font units will also be pixels. Font units are used in vector text to specify the size of a

given character. For example, the definition of a character might specify that its height is 10 (font units). If *FLAT* is used, the character will be rendered 10 pixels high.

If set to *MAPPED FLAT*, the font units will still be pixels, but the location of the text is in lat/lon coordinates.

Finally, if set to *TRANSFORMED*, both the location and font units are lat/lon coordinates. This means that if the map has curvature, the text will curve around the globe. This also means that the text will appear different sizes depending on the zoom factor.

TEXT ANGLE=<number>

Defines the angle by which the rendered text should be rotated. It is defined in degrees counterclockwise from the right horizontal axis, i.e. an angle of 0 will render the text as it is seen on paper.

TEXT ANCHOR=<type> <type>

Defines how the text should be anchored. The value for this variable is the horizontal anchoring option followed by the vertical anchoring option, very similar to the text anchoring attribute in the GLG Toolkit. The anchoring options define where the marker should be relative to the text and where the text should be relative to its coordinates. The horizontal anchoring type can be one of:

HLEFT (default)
HRIGHT
HCENTER

Similarly, the vertical anchoring type can be one of:

VTOP (default)
VBOTTOM
VCENTER

The horizontal and vertical anchoring types have to be supplied as individual strings separated by either a white space or a comma. Hence:

TEXT ANCHOR= HLEFT VCENTER
 TEXT ANCHOR= HLEFT,VCENTER
 TEXT ANCHOR= "HLEFT" "VCENTER"

are all valid and will render the text centered on the right side of the marker. However,

TEXT ANCHOR= "HLEFT VCENTER"

is not valid, as it provides just one combined string.

LABEL FORMAT=<format>

Defines what text string is to be rendered alongside the marker. The parameter defines which marker attributes to display in a label, as well as the format string to use for formatting attribute values. The attributes are defined by their attribute numbers, which are specified inside the triangular brackets. The attribute number may be followed by a C *printf*-style format string. If the format string inside the brackets is omitted, a default format will be used for that attribute. Special

character sequences, such as “\n” are also allowed, as well as escaped character sequences. For example, if the first (index 0) attribute is the name of a city and the second (index 1) attribute is the population of a city, then:

```
LABEL FORMAT="<0>\npopulation: <1%.0lf>"
```

will render:

```
SomeCity population: 12345
```

for a city named “SomeCity” with a population of 12345. If some markers do not have attributes used in the label format, the missing attributes will be displayed as “***”. The first attribute listed in the label format is considered to be the main attribute. If the main attribute is missing, the label will not be displayed at all. If label format is not defined, marker labels will not be rendered.

LABEL FORMAT2=<format>

Defines an alternative label format to be used if the marker attribute used in *LABEL FORMAT* is missing for some objects.

LABEL STYLE=<style>

Specifies a list of optional rendering attributes for text labels. Supported options are:

```
BG_EDGE
BG_FILL
OUTLINE
```

The first two options are used for rendering a box around the text. *BG_EDGE* specifies that the box’s edge should be drawn and *BG_FILL* specifies that the box’s area should be drawn. The *OUTLINE* option draws an outline around the text label to improve label visibility when it is displayed on a background of a similar color. The *LABEL OUTLINE COLOR* parameter described below specifies the color of the outline.

More than one option may be listed in one string using a comma or space separator and surrounding the whole string with quotes, for example:

```
LABEL STYLE="OUTLINE,BG_EDGE"
```

LABEL OUTLINE COLOR=<*r g b*>

Defines the color of the outline drawn around text labels (default 0.7, 0.7, 0.7). The value must be a triplet of double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white.

WRAP LABEL LENGTH=<number>

Defines the maximum length of a label line in pixels. If this length is exceeded, the label text will be wrapped by splitting it into several lines. Setting it to 0 (default) disable wrapping. The parameter defines an approximate line length: since text lines are split at spaces, individual text lines may extend outside the box width defined by the parameter.

TEXT BOX OFFSET HORIZ=<number>

Defines the horizontal pixel margin between the label and the text box edge (default 3). Text box is enabled by setting *LABEL STYLE* to include *BG_EDGE* and/or *BG_FILL*.

TEXT BOX OFFSET VERT=<number>

Defines the vertical pixel margin between the label and the text box edge (default 1). Since many letters have a different height, the actual distance from a given character to the text box may vary with the font and character in question.

TBOX LINE WIDTH=<number>

Defines the line width in pixels of the text box drawn around text labels (default 1). Text box drawing is enabled by setting *LABEL STYLE* to include *BG_EDGE* and/or *BG_FILL*.

LABEL MIN ZOOM=<value>

LABEL MAX ZOOM=<value>

Defines the zoom factor range for which marker labels (such as city names) and polygon labels (such as street names) are displayed. The labels will not be displayed for zoom factors outside this range. This causes the Map Server to display labels only when a certain zoom level has been reached, and disable them again when the map is zoomed in too far, decluttering the map image. If these parameters are omitted or set to a default value of -1, labels will always be displayed (if enabled by label format attributes).

LAYOUT TYPE=<value>

Defines the label layout type and may have the following values:

NONE (default)

LAYER

GLOBAL

Specifying the layout type enables label layout negotiation for text labels. The label layout makes sure that text labels do not obscure each other and that labels with lower priority do not obscure labels with higher priority. When set to *LAYER*, the layout will be performed only for labels within the layer, which is faster but may result in label conflicts between labels in different layers. The *GLOBAL* value is used to perform the label layout negotiation across all layers.

Text layout negotiation is not supported for the *TRANSFORMED* text.

LABEL PRIORITY=<number>

Defines label priority for all labels in the layer (default 9). The labels with priority value of 0 have the highest priority. A threshold table may be attached to *LABEL PRIORITY* to vary priority of labels within one layer based on other object attributes, such as a road type or city population. While the priority values may be set to any number greater or equal to 0, using priority values with minimal gaps between them results in better performance.

LABEL REPEAT DISTANCE=<number>

Defines a minimum distance in pixels between labels with the same label string (default 0) when the label layout negotiation is enabled. It may be used to enhance rendering labels of named streets and roads that are split in multiple segments.

LAYOUT MARGIN=<number>

Defines an additional margin in pixels between text labels (default 1). Using a bigger value will result in more space between individual text labels when the label layout negotiation is enabled.

LAYOUT MARKERS=<0 or 1>

Specifies the way the layout negotiation handles markers. If set to 1 (default), the markers will be handled together with their labels, and will not be drawn if its label is not drawn. If set to 0, the markers will always be drawn, while their labels will be drawn only if not obscured by other labels with a higher priority.

MARKER ICON FILE=<file>

Specifies a JPEG, PNG or TIFF image file containing a custom icon for rendering markers. An attribute threshold may be attached to this attribute to use different icon types depending on a value of some other attribute. For example, different icons may be used for cities and towns depending on the value of their population attribute. Refer to the *ATTR_MAP* attribute on page 57 for more information on attribute threshold tables.

MARKER ICON TYPE=<type, type, ...>

Defines what type of marker to render at the location defined by the text object's control point. The value of this variable is a list of all desired marker types. All marker types in the GLG Toolkit are supported. The available marker types are:

NONE
CIRCLE_EDGE
CIRCLE_FILL
BOX_EDGE (default)
BOX_FILL
CROSS
DOT
CUSTOM ICON

The circle options draw a circular marker. The box options draw a square marker. The *CROSS* option draws a cross, and the *DOT* option draws a small dot in the center of the marker. The *CUSTOM ICON* option draws the custom icon specified by *MARKER ICON FILE*. If the marker type is omitted or set to *NONE*, no marker will be rendered.

More than one option may be listed in one string using a comma or space separator and surrounding the whole string with quotes, for example:

MARKER ICON TYPE="CIRCLE_FILL,CIRCLE_EDGE,DOT"

MARKER ICON SIZE=<number>

Defines the size in pixels at which the marker should be rendered (default 5). It is ignored if custom icons are used, in which case the size of the image in the icon file is used.

TEXT OFFSET HORIZ=<number>

For marker labels, defines the horizontal pixel distance between the marker and the label (default 1). For polygon labels, defines the horizontal label offset from the label's calculated position, factoring in a rotation angle for line labels.

TEXT OFFSET VERT=<number>

For marker labels, defines the vertical pixel distance between the marker and the label (default 0). For polygon line labels, defines distance between the label and the line, factoring in a rotation angle for line labels. For other types of polygon labels, defines the vertical label offset from the label's calculated position.

FILL COLOR=<r,g,b>

Defines the fill color of polygons that are rendered if there are any areas to fill, as well as the fill color of the outline and background layers (default 0.7, 0.7, 0.7). The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white. Image and grid layers are not affected by this variable.

EDGE COLOR=<r,g,b>

Defines the edge color of polygons, grid lines and outline (default 0, 0, 0). The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white.

MARKER EDGE COLOR=<r,g,b>

Defines the edge color of markers in a vector layer (default 0, 0, 0). This option only applies if the marker type includes *CIRCLE_EDGE*, *BOX_EDGE*, *CROSS* and *DOT*. The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white.

MARKER FILL COLOR=<r,g,b>

Defines the fill color of markers in a vector layer (default 0.7, 0.7, 0.7). This option only applies if the marker type includes *FILLED_CIRCLE* or *FILLED_BOX* options. Otherwise, the variable has no effect. The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white.

LABEL COLOR=<r,g,b>

Defines the color of the text labels in a vector layer (default 0, 0, 0). The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white.

BOX EDGE COLOR=<*r,g,b*>

Defines the edge color of the text box in a vector layer, if the text box is enabled (default 0, 0, 0). The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white. The text box is enabled by setting *LABEL STYLE* to include *BG_EDGE* and/or *BG_FILL*.

BOX FILL COLOR=<*r,g,b*>

Defines the fill color of the text box in a vector layer, if the text box is enabled (default 0.,7 0.7, 0.7). The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white. The text box is enabled by setting *LABEL STYLE* to include *BG_EDGE* and/or *BG_FILL*.

GRADIENT COLOR=<*r,g,b*>

Defines the gradient color of the background layer (default 0, 0, 0). The value must be three double values ranging from 0 to 1, where 0., 0., 0. is black and 1., 1., 1. is white.

GRADIENT TYPE=<*type*>

Defines the type of the gradient to be rendered for the background layer in the orthographic projection. Valid values are *NONE* (default) and *AROUND_GLOBE*. The *AROUND_GLOBE* gradient type renders a gradient that starts at the edge of the globe and gradually changes color from the *GRADIENT COLOR* to *FILL COLOR* as the distance from the center of the globe increases.

GRADIENT LENGTH=<*length*>

Defines the length of the gradient of the background layer (default 1). For the *AROUND_GLOBE* gradient type, the length is defined in relative units, with 0.3 corresponding to the 30 percent of the globe radius. The gradient starts the edge of the globe and extends outwards.

FILL TYPE=<*type*>

Defines the fill type of polygons to be rendered, analogous to the *FillType* option in the GLG Toolkit. Supported values are:

NONE
EDGE (*default*)
FILL
FILL AND EDGE

NONE disables rendering of the polygon's fill and edge, while still rendering a polygon's label (if defined).

LINE WIDTH=<*number*>

Defines the line width in pixels of polygons, grid lines and outline, analogous to the line width option in the GLG Toolkit. The default value is 1 pixel.

DRAW CENTER MARKER=<0 or 1>

If set to 1, a marker with or without a label will be drawn at the polygon center (default 0). This can be used to draw small (but important) polygons that otherwise would be invisible at low zoom levels due to their small size. The polygon's *FILL TYPE* may be set to *NONE* to disable rendering of the polygon when the center marker is used.

The marker and label attributes control rendering of the center marker and its label.

POS RANGE=<0 or 1>

While most vector data for the earth has longitude values ranging from 180 W to 180 E , some data, such as the common shoreline data is presented in the 0 E to 360 E range. This means that it is centered around 180 E as opposed to 0 E. A value of 1 specifies that the vector data is in such a format. The default value is 0.

POLY LABEL FORMAT=<format>

Defines which attributes of a polygon object in a vector data file to display in the polygon label. This parameter is commonly used to draw labels on streets and areas. The placement of the labels is determined by the Map Server. If this parameter is not specified, polygon labels will not be displayed. Refer to the *LABEL FORMAT*=<format> section on page 50 for information on the format string syntax.

POLY LABEL FORMAT2=<format>

Defines an alternative label format to be used if the polygon attribute used in *POLY LABEL FORMAT* is missing for some objects.

POLY LABEL TYPE=<type>

Defines where and how the polygon label is rendered. It can have one of the following types:

LINE LABEL (default)
MID LINE LABEL
AREA LABEL
BBOX LABEL

If the type is *LINE LABEL*, then labels are rendered at the beginning of polygons at an optimal angle. If the type is *MID LINE LABEL*, then labels are rendered in the middle of polygons at an optimal angle. If the type is *AREA LABEL*, then labels are rendered according to the *LABEL CENTER* attribute. If the type is *BBOX LABEL*, then labels are rendered in the center of the bounding box of the polygon.

If the type is either *AREA LABEL* or *BBOX LABEL*, then all attributes that define the anchoring and angle of the text apply.

POLY LABEL CENTER=<number>

Defines which attributed of a polygon object in a vector data file to use as the center of the label on the polygon. The number must be the name (attribute index) of the data attribute to use. This attribute is used when the *LABEL TYPE* attribute is set to *AREA LABEL*. The attribute must be of G type. The default value of -1 indicates an unset value.

LABEL SELECTION MODE=<mode>

Specifies the label selection mode for the GIS selection and can use one of the following modes:

NONE
YES
INTILE
UNSET (default)

NONE suppresses label selection for the layer. *YES* enables GIS label selection with maximum precision, *INTILE* forces the in-tile label selection described in the *GlmGetSelection* section on page 129 and *UNSET* allows the label selection mode to be controlled by either the *SELECT_LABELS* parameter of the map request or the *select_labels* parameter of the *GlmGetSelection* method.

ALLOW OVERRIDE=<0 or 1>

If set to 0, disallows the use of the “++” override in the layer string for this layer. By default, the overrides are allowed for untiled layers and suppressed for tiled layers, to avoid unintended consumption of significant memory and CPU resources. The default setting for all layers of the dataset may be specified by the *ALLOW OVERRIDE* parameter of the dataset’s SDF file.

ATTR MAP=<number resource threshold_table REL/ABS>

The Map Server provides a mechanism for mapping attributes of vector data to layer resources. For example, it is possible to vary the color and size of the city marker depending on the city’s population. The *ATTR MAP* parameter defines the threshold table and the way it is applied.

number

the name of an attribute (0-based attribute index) associated with any given data object. This attribute will be used as an input of the threshold table. Numbers bigger than the maximum attribute index may be used to reference custom attributes described below.

Negative numbers may be used to define special attributes provided by the map server:

- 3 Random attribute that generates a random number ranging from zero to the maximum (last) value in the threshold table
- 4 Zoom factor

resource

defines the name of the layer’s resource to vary. This resource will be used as the output of the threshold table. All resources names are concatenations of words, each of which start with a capital letter. This is the same resource naming convention used in the GLG Toolkit. For example, the *ALPHA* attribute becomes the resource named *Alpha* and the

MARKER FILL COLOR attribute becomes the *MarkerFillColor* resource. While most of the attributes previously described can be used as the value of the resource parameter, some cannot. See the *Map Server API* chapter on page 101 for details and a full list of resources.

threshold_table

the path of a threshold table file, which often has a *.tt* extension. The format of a threshold table file is as follows:

```
INPUT DATA TYPE=<type>
OUTPUT DATA TYPE=<type>
```

The input data type defines the type of the resource that is used as an input to the threshold table, and the output data type defines the type of resource that is being set. Supported options are:

```
GLM_D
GLM_G
GLM_S
```

What follows is list of values, one line at a time in the following format:

```
<input value> <output value>
<input value> <output value>
<input value> <output value>
...
```

Usually attributes of D type are used as input, but they may control an output resource of any type. The input values of the D type must be defined in the table in ascending order. All input values that are greater than or equal to the last output value will be mapped to the last output value. All input values between the second to last and last input value will be mapped to the second to last output value, and so on. The first input value is ignored: all input values less than the second input value will be mapped to the first output value. See example below.

If input values of G type are used, they may be listed in any order. If the exact match is found, the corresponding output value is returned. If no match is found, the last output value in the list is returned (the last input value is ignored).

If input values of G type are used, they may be listed in any order as well. The input values in the list may contain wild cards: '*' matches any number of any characters and '?' matches one character. If a match is found, the corresponding output value is returned. If no match is found, the last output value in the list is returned (the last input value is ignored).

REL/ABS

either *REL* or *ABS*. If set to *ABS*, the resource value will be set to that defined in the threshold table. If set to *REL*, the value will be relative, i.e. multiplied by the value of resource.

Naturally, *REL* cannot be used for S resources. For G resources, each component is multiplied by the respective component in the threshold table's resource value.

For example, if the second (index 1) attribute of markers in a data file is the population of a city:

```
ATTR MAP=1 EdgeColor popul_to_color.tt REL
```

and the threshold file (*popul_to_color.tt* in the sample dataset):

```
INPUT DATA TYPE= GLM_D
OUTPUT DATA TYPE= GLM_G
```

```
0      0.4 0.4 0.4
10000  0. 1. 1.
50000  1. 1. 0.
100000 0. 1. 0.
500000 1. 0. 0.
```

will make all large cities (>500k) red, all small cities (100k - 500k) green, all large towns (50k - 100k) yellow, all medium towns (10k - 50k) purple and the rest of the small towns grey.

CUSTOM ATTR=<number resource data_type>

Defines a custom attribute that can be used in an attribute map. This is useful for defining custom descriptions of data to be used in the label format.

number

defines the number of the custom attribute. This number must be greater than the largest attribute index in the file.

resource

the name of a custom resource to create.

data_type

the type of that resource.

A custom attribute definition creates a custom attribute which also becomes the layer's resource.

For example,

```
CUSTOM ATTR=5 CityDescriptor GLM_S
ATTR MAP=1 CityDescriptor popul_to_city.tt ABS
LABEL FORMAT="<5>"
```

in the LIF file as well as the following lines in a threshold table (*popul_to_city.tt*):

```
DATA TYPE= GLM_S
0      "small town"
10000  "town"
50000  "big town"
100000 "small city"
500000 "big city"
```

will make the marker label either "small town", "town", "big town", "small city" or "big city" depending on the population, which is the second attribute of a marker object.

The following example adds *MinPopulation* custom attribute that is used to display smaller cities at higher zoom levels:

```
# Controls what cities are displayed at the current zoom level based on the MinPopulation custom
# attribute that depends on the zoom factor.
ATTR COND= @8 > @14

# A custom attribute that controls a minimum city population to be displayed depending on the zoom.
CUSTOM ATTR=14 MinPopulation GLM_D

# Maps zoom factor (attribute index=-4) to the value of the MinPopulation custom attribute using a
# threshold table.
ATTR MAP= -4 MinPopulation zoom_to_min_popul.tt ABS
```

PIXEL MAP=<number resource threshold_table REL/ABS>

Defines a custom color threshold table for mapping elevation values to different colors. The *number* and *resource* parameters are ignored (use zeros), and the rest is the same as for the *ATTR MAP* parameter above. The threshold table must have the *INPUT_DATA_TYPE*=GLM_D and *OUTPUT_DATA_TYPE*=GLM_G, mapping elevation value (D type) to the RGB color values (G type). A sample color threshold table called *elev_to_color.tt* is provided with the sample dataset.

INCLUDE=<file>

Includes all variable definitions in the specified file into the current LIF. This variable is useful to either split information into multiple files or when hierarchical tiling LIFs are generated and then included into a more general layer description.

TILES SUBLIF=<top-level sublif file> [<bbox>]

SUBLIF DIR=<directory> [<bbox>]

SUBLIF FILE=<sublif file> [<bbox>]

If hierarchical tiling is used, defines a sub-LIF file (SLF) with tiling information. The sub-LIF files are automatically generated by hierarchical tiling setup utility and contains information about the tile hierarchy for directory-based tiled datasets. The utility traverses the directory that contains a hierarchical dataset and all its subdirectories, generating a hierarchy of linked SLF files that describe the data in a way that allows to quickly find tiles for the requested area on the map. The top-level SLF file generated by the utility is specified as the *TILES SUBLIF* parameter of the LIF. The *TILES SUBLIF* file is relative to the directory of the LIF itself, unless an absolute path is specified.

All SLF files include the *MIN LAT*, *MAX LAT*, *MIN LON* and *MAX LON* entries that define a bounding box covered by the SLF file and all subordinate SLF files it may recursively include. An SLF file may also contain a list of *SUBLIF DIR* entries that specify subdirectories that contain more tiles. Each listed subdirectory will contain an SLF file with the same name as the top level SLF file specified with *TILES SUBLIF*, and that SLF file may recursively include more *SUBLIF DIR* entries, to any level of depth.

An SLF file may also contain a list of *SUBLIF FILE* entries, which are leaf nodes of the hierarchy tree describing data for one tile. These bottom-level SLF files reference their tile data using the *FILENAME* parameter and must not include any other *SUBLIF DIR* or *SUBLIF FILE* entries.

For image tiles, the bottom-level SLF file may contain *USED WIDTH* and *USED HEIGHT* entries, which are automatically generated by the Map Server's tiling utility in cases when the image's width or height could not be evenly divided into the requested number of rows or columns of tiles, and the last tile in the row or column is just partially filled with data. In this case, the entries provide the actual width and height of the last tile, different from the width and/or height of the regular tiles defined in the LIF.

Any of these SUBLIF configuration variables may include an optional BBOX directive in the following format:

BBOX= lon1 lat1 lon2 lat2

While the BBOX directive is optional, it is included by default by the tiling setup utility to increase performance of large hierarchical datasets.

Using the Map Server and Utilities

3.1 Map Server Executable

The Map Server executable generates maps from GIS data according to input parameters. It is used for providing a map service from a web server using GCI or FastCGI interface, and it may also be invoked from a command line to generate images without a web server setup.

When used in the GCI or FastCGI environment on a web server, the map server receives an HTTP GET request in the OpenGIS WMS format and returns an image which contains the map image requested by the user. The map server is usually invoked from a simple wrapper script, which hides details of the GIS data setup and map server invocation options, presenting the web application with a logical view of the map generation service. A sample script called *GlmScript* on Linux/Unix platforms and *GlmScript.pl* on Windows is provided. Refer to the *Appendix A: Web Server Installation Notes* chapter on page 153 for information more information on the setup script.

The map server supports several OpenGIS WMS request types: *GetMap* for generating map images, *GetFeatureInfo* and *GetLocationInfo* for coordinate conversion and elevation queries, *GetSelection* for getting information about objects inside the map at the mouse location, and *GetCapabilities* for querying capabilities and available datasets.

The map server executable also provides command line options for generating map images and executing other queries without a web server setup, as well as miscellaneous setup and debugging options.

What follows is a description of the command line options of the Map Server executable. It is called *GlmMap* and is located in the `<glg_dir>/bin` directory.

Error handling

All errors produced by the Map Server executable (usually data-related errors) are printed to the standard error (which can be redirected to a file if desired) and also logged into the *glm_error.log* file in the directory defined by the GLM_ERROR_LOG, GLG_ERROR_LOG or GLG_DIR environment variables. The environment variables are searched in the order above, and if the log file directory is not defined, the current directory is used.

The Map Server uses GLG library, and if there are any GLG errors (such as setting a non-existing resource by an application using the Map Server API), they are logged in the *glg_error.log* file, whose location is controlled by the GLG_ERROR_LOG or GLG_DIR environment variables.

By default, the Map Server errors are also written into the generated image, so the user can see what went wrong. This behavior can be toggled.

Synopsis

GlmMap [options]

Description

The Map Server executable. Depending on the command line options, it functions both as an executable to generate map images and perform map queries, as well as the executable for the various data file manipulation utilities.

Command Line Options

While most of the command line options differ depending on the desired function, the following options transcend all uses:

-help

Prints all available command-line options in the terminal window.

-version

Prints the map server version.

-arg-file <file>

Specifies a file from which to read command line arguments. This option is useful on Windows, where an invocation of the Map Server executable often exceeds the maximum command length.

-verbosity <number>

Defines how verbose the Map Server should be. The verbosity level values are handled differently depending on the mode the map server executable was started in: to generate a map or perform a map query in the map server mode, or to perform data massaging in the utility mode. The following lists the use of the verbosity values in both modes.

In the Map Server mode:

If verbosity level is set to positive integer value from 1 to 6, diagnostic information will be generated. If verbosity level is set to a negative integer value from -1 to -6, the performance and timing output will be produced. If verbosity level is set to 1001 or 1002, tile extent information is displayed as tile outlines in the generated images for the *GetMap* requests. The 1003 verbosity level displays vector tiles coverage as filled areas without displaying the tiles' content. The verbosity level of 0 disables all diagnostic output.

The verbosity output is printed to the standard error, the higher the number, the more output is generated. Errors and information are also displayed in the generated image (unless the *IERRORS* parameter of the map generation request is set to 0) and logged into the *glm_error.log* file. By default, the verbosity is set to 0.

In the Utility mode:

Setting verbosity level to an integer value from 1 to 6 prints information about the data being processed according to the specified verbosity level. The verbosity level of 0 disables all diagnostic output. By default, the verbosity is set to 0.

-progress <number>

In the Utility mode, prints progress information for every N processed files, where N is the option value. The default value is 100. Use 0 to suppress progress reporting.

Map Generation and Queries

Synopsis

```
GlmMap [-arg-file] -generate [-verbosity N] [options]
```

Description

The *-generate* option is used to generate map images and execute map queries in the stand-alone mode, either in the CGI environment or from the command line.

Options

The following command line options can be defined after *-generate* to specify map query parameters:

-dataset <file>

Defines the location of the Server Dataset File (SDF) to be used by the Map Server. If no root directory is specified within the SDF, the path of the SDF itself will be used as the root.

-cgi

Specifies whether or not to run in the CGI mode. This options is utilized when run with a CGI (Common Gateway Interface) capable web server. Only the GET method for http requests is supported. In this mode, all map request information (besides the dataset file) will be obtained from the QUERY_STRING environment variable. In order to request a map image when running in this mode, all request information must be appended to the URL, for example:

```
http://my_web_server.com/cgi-bin/GlmScript?option1=value&  
option2=value...
```

Here, *GlmScript* is a script which invokes the Map Server with the proper dataset file. The options appended to the URL are passed to the Map Server via the QUERY_STRING environment variable. The options are described in the *Map Query String* section on page 66. See the *Examples of OpenGIS map query strings* section on page 72 for examples.

Windows Note: On Windows, the script is called *GlmScript.pl*. If the IIS web server is used instead of apache, the *cgi-bin* directory will be called *Scripts*.

-fcgi

Same as the *-cgi* option, except that it enables the FastCGI mode for web servers that support it. In the FastCGI mode, the map server executable does not exit after processing each map request. Instead, it is reused to process other map requests, which makes more efficient use of system resources and map server's tile cache. This results in significant performance increase for map requests that use the same map data tiles.

This mode is transparent to the end user, and the map server is invoked using the same URL as in the CGI mode. If this option is used with a set up that does not support FastCGI, the map server will automatically use the standard CGI mode as a fallback.

The web servers that support FastCGI mode provide a set of parameters to fine-tune the use of the map server in this mode. These parameters define how many of the map server processes may be used and how often they are restarted to avoid memory fragmentation and other potential problems. The FastCGI mode can also be used for distributing the load, off-loading the map server to a remote machine to free the web server. Refer to the FastCGI documentation for more information about potential usage and web server configurations.

-oGISreq <string>

This option supplies the OpenGIS map query string when the Map Server executable is used in a stand-alone mode to generate image from a command line, without a web server which supplies the query string in CGI or FastCGI mode. The query string is defined with a series of tokens. Each token is separated with an ampersand ('&' symbol) and those tokens which have values associated with them have their values defined with an equality (=). For example, *option1=value1&option2=value2* defines two options each of which has a value associated with it. Refer to the *Map Query String* section on page 66 for the query string format information. See the *Examples of OpenGIS map query strings* section on page 72 for examples.

-output <file>

Defines an output file for the generated image for the stand-alone usage. This option is only used when the Map Server is not run in the CGI or FastCGI mode, because in these modes the image is outputted to the standard output with the appropriate HTML header. If this option is set to "-", the standard output is used.

-max-iterations <N>

Defines the maximum number of internal iterations used to render filled polygons, which prevents rendering delays if the map is zoomed in too much. This parameter may need to be increased if large filled polygons (such as OSM country or state areas) are rendered with very high zoom levels. This value may be overridden by the MAX ITERATIONS parameter in the dataset's SDF file.

Map Query String

The map query string contains all information (other than the dataset file) for generating a map image. The format of the map query string follows the OpenGIS WMS map query standard. The *-oGISreq* command line parameter provides a query string for the stand-alone use of the map server. When the map server is used in the CGI or FastCGI mode, the query string's tokens are provided

via the parameters of the map server URL and passed to the map server executable using the standard CGI or FastCGI mechanisms. What follows is a description of valid tokens and their possible values.

VERSION=<version>

Defines the version of the OpenGIS protocol to use. The following versions are supported:

- 1.1.0
- 1.2.0
- 1.3.0

REQUEST=<request type>

Defines the type of request. The OpenGIS VMS standard version 1.3.0 requires a conforming map server to support at least the *GetMap* and *GetCapabilities* requests. The following request types are supported by the GLG Map Server:

- *GetMap* - generates a map image.
- *GetFeatureInfo* - queries information at the X/Y image location defined by the *I* and *J* parameters of the request. The type of the returned information (i.e. elevation, lat/lon coordinates, etc.) is defined by the *INFO_TYPE* parameter.
- *GetLocationInfo* - queries information at a lat/lon location defined by the *LON_LAT* parameter of the request. The type of returned information (i.e. elevation, lat/lon coordinates, etc.) is defined by the *INFO_TYPE* parameter.
- *GetSelection* - returns a list of objects (including object attributes) selected by the supplied point in pixel coordinates. It may be used to query objects in the map selected by the mouse.
- *GetCapabilities* - queries information about the capabilities of the Map Server, including a list of layers available in the map server's data directory.

SRS=<value>

Defines the Spatial Reference System, which specifies the projection to use. Projections are the way the earth is seen. To view the earth as a three dimensional sphere, the Orthographic projection is used. To view the earth as a flat rectangle, the Rectangular projection is used. The SRS codes are defined by the OpenGIS standard. The following SRS projection codes are supported:

- *EPSG:4326* - rectangular projection, lat/lon
- *CRS:84* - rectangular projection, lon/lat coordinate order
- *AUTO:42003,9001,<lon>,<lat>* - orthographic projection with center at lon/lat. The specified lon/lat center will be displayed at the center of the image.
- *AUTO2:42003,<scale_factor>,<lon>,<lat>* - orthographic projection with center at lon/lat and scale factor defining the ratio of the BBOX or EXTENT units to the coordinate system units.

For example, the SRS projection code *AUTO2:42003,1,-97,38* defines an orthographic projection where the point with longitude=-97 degrees and latitude=38 degrees is displayed at the center of the image.

The SRS projection code *SRS=AUTO2:42003,0.3048006096012192,-97,38* defines a similar orthographic projection, but with bounding box units defined in US feet (scale factor = 0.3048006096012192).

WIDTH=<number>

Defines, in pixels, the width of the image to be generated.

HEIGHT=<number>

Defines, in pixels, the height of the image to be generated.

BBOX=<min,min,max,max>

Defines the extent of the image. This token is provided to conform with the OpenGIS standard. The *EXTENT* token is also provided and is simpler to use.

If the rectangular projection is being used, the first two values are the minimum longitude and latitude, respectively, and the last two values are the maximum longitude and latitude, respectively.

If the orthographic projection is being used, the first two values are the minimum horizontal and vertical extent in meters (e.g. the equatorial radius is 6378136 and the polar radius is 6356752) and the last two values are the maximum horizontal and vertical extent in meters. The maximum values must be opposites of the minimum values. If *AUTO2* SRS projection is used, the bounding box units are multiplied by the scale factor to convert them to different units.

ANGLE=<number>

Defines the counter-clock-wise map rotation angle in degrees for rectangular projections, ignored for orthographic projections.

BGCOLOR=<color>

Defines the background color to be rendered in the map image. The color is a hexadecimal value, such as 0xffffffff (white), where the most significant 16 bits (the first two characters) define the red component of the color, the middle 16 bits (middle two characters) defines the green component and the least significant 16 bits (last two chars) define the blue component.

STYLES=<value>

Defines the styles to use. This token exists to conform with the OpenGIS standard and must be set to *default*. *STYLES* is reserved for future extensions.

FORMAT=<image format>

Defines the file format of the output image for the *GetMap*, as well as the output format for the *GetSelection* request. Must be set to either *image/jpeg* or *image/jpg* for *GetMap*, and to *text/plain* or *text/xml* for *GetSelection*.

LAYERS=<list>

Defines the layers to turn on in the generated image for the *GetMap* request, as well as vector layers to be queried for the *GetSelection* request. The list must be a comma separated list of layer names or aliases defined in the Server Dataset (SDF) file. For example, the string “*earth,cities*” specified the earth and cities layers to be rendered.

If *default* is used as a layer name, all layers that are on by default (DEFAULT ON=1 in the LIF file) will be enabled. If “*” is used as a layer name, all layers will be enabled.

If a layer name starts with ‘-’, the layer is disabled. This may be used to disable some layers enabled by the *default* or “*” layer string values. For example, *LAYERS=default,-cities* will display all default layers except the *cities* layer, while *LAYERS=*, -cities* will show all layers defined in the SDF file except the *cities* layer.

If a layer name starts with “++” override sign and the overrides are not suppressed by the ALLOW OVERRIDE dataset attribute, the layer will be displayed regardless on the MIN ZOOM, MAX ZOOM and MAX TILES parameters of the layer. In this case any layer’s attribute conditions defined by the ATTR COND directive of the layer’s LIF file will also be ignored, and only the attribute conditions defined for the layer in the layer string will be followed. By default, the overrides are allowed only for untiled layers.

Note: The ‘++’ prefix is not supported for the *default* layer and aliases to avoid ambiguities. The “-” prefix can be used with both the *default* layer and aliases. However, it does not disable an individual layer in the *default* layer or an alias if this layer is explicitly listed in the *LAYERS* string.

If individual layers are listed, the layers will be displayed in the order of their appearance in the layer string. For layers enabled by the *default* or “*” values of the layer string, their display order is defined by the order in which the layers are listed in the SDF file.

A layer in the *LAYERS* list may have a list of attribute conditions appended to it, with each condition containing an attribute index starting with ‘@’ character, comparison operation and condition value. For example, if the second attribute (index=1) of the *cities* layer contains city’s population, *LAYERS=cities @1>=100000 && @1<500000* may be used to display only cities with population greater than or equal to 100K but less than 500K. Refer to the *Attribute Condition Syntax* section on page 71 for a complete description of the attribute condition syntax.

If the layer has ‘-’ prefix, the condition is reversed and the features matching the attribute condition will not be listed. For example, *LAYERS=default,-cities@1<100000* might be used to eliminate cities with population under 100K from the map display that includes the *cities* layer.

The layer attribute conditions defined in the layer string are logically ANDed with attribute conditions defined in the layer’s LIF file using the ATTR COND directive. Only the features that match both sets of conditions will be displayed.

The attribute conditions are processed at run time, and therefore are slower than switching the whole layer on or off. They should be used to filter layer features only when the filter conditions are not known at the design time. If the subsets of a layer which will need to be turned on and off are known at the application design time, it is more efficient to split the layer into a few layers with

desired ranges of the attribute value and then switch on and off these new layers. For example, if an application needs to have toggles to control the display of small, medium and large towns and cities, it would be more efficient to split the cities layer into the desired categories based on the value of the population attribute using the Map Server's split utility. However, using attribute conditions is more flexible and convenient when the number of categories is large, or when the thresholds may change frequently.

CENTER=<lon,lat>

Defines the longitude and latitude of the center of the map image. This token overrides the center lon/lat specified in the SRS definition. It is used with the *EXTENT* token and provides an alternative way to specify center of the map as a separate token independent of the SRS definition, for both the rectangular and orthographic projections.

EXTENT=<x,y>

Defines the extent of the map image and provides a more convenient alternative to the *BBOX* parameter. If the rectangular projection is being used, the two values are the longitudinal and latitudinal extents of the image, respectively. If the orthographic projection is being used, the values are the extent of the image in meters. If *AUTO2* SRS projection is used, the extent units are multiplied by the scale factor to convert them to different units.

STRETCH=<0 or 1>

Defines whether or not to maintain the correct lon/lat aspect ratio. If set to 1, the image will be stretched according to the *WIDTH* and *HEIGHT* tokens. If set to 0, the image will maintain its lon/lat aspect ratio. For example, in the orthographic projection, if the entire earth is being viewed, if *STRETCH* is set to 0, the earth will always appear as a perfect sphere, instead of an oblong one.

IERRORS=<0 or 1>

Defines whether or not to render errors into the map image. If set to 0, no errors will be rendered. If set to 1, errors will be rendered into the map image in the top left corner. Default value is 1.

IMAGE_ANTIALIASING=<0 or 1>

Defines whether or not to use anti-aliasing for scaling raster layers. This setting is overwritten by the corresponding setting in the LIF file. The default value is 1.

QUERY_LAYERS=<layer name>

Defines query layers for *GetLocationInfo* and *GetFeatureInfo* requests. For coordinate conversion queries, *default* may be used. For elevation queries, it is set to the name of the layer that contains elevation data.

INFO_TYPE=<info type>

Defines the type of the information to be returned for *GetLocationInfo* and *GetFeatureInfo* requests. May have the following values:

- *lat_lon* - returns lat/lon coordinates of the requested point
- *elevation* - returns elevation of the requested point.

INFO_FORMAT=<info format>

Defines the output format for the *GetFeatureInfo*, *GetLocationInfo* and *GetCapabilities* requests, as well as the error format for the *GetMap* request. May be set to *text/xml* or *text/plain*. Default value is *text/xml* for the CGI use and *text/xml* when invoked from the command line.

I=<number>

Defines X coordinate of the image point for the *GetFeatureInfo* and *GetSelection* requests.

J=<number>

Defines Y coordinate of the image point for the *GetFeatureInfo* and *GetSelection* requests.

LON_LAT=<lon,lat>

Defines lat/lon coordinates of the point for the *GetLocationInfo* request.

SELECT_LABELS=<value>

Specifies a global label selection mode for the *GetSelection* request:

- *NONE* - disables label selection
- *YES* - performs label selection with the maximum precision
- *INTILE* - performs label selection with the in-tile precision.

The global label selection mode may be overridden by a layer's *LABEL SELECTION MODE* setting in the layer's LIF file.

PICK_RESOLUTION=<number>

Specifies pick resolution in pixels for the *GetSelection* request (default 2). All objects within this distance from the selection point will be reported in the GIS selection.

Attribute Condition Syntax

An attribute condition list contains one or more attribute conditions combined using "&&" (local AND) or "||" (logical OR) operations.

Each attribute condition contains an attribute index, comparison operation and condition value. A special attribute index value of -4 may be used to reference the current zoom factor.

The following comparison operations are supported:

- | | |
|----|---|
| == | - equal |
| != | - not equal |
| > | - greater than (numerical attributes only) |
| >= | - greater than or equal (numerical attributes only) |
| < | -less than (numerical attributes only) |
| <= | -less than or equal (numerical attributes only) |

The condition value is interpreted as a numerical value for numerical attributes, or as a string value for string attributes. For string attributes, the string value may contain ‘?’ (one character match) and ‘*’ (multiple characters match) wildcards, in which case it is treated as a regular expression. Spaces are allowed between each element of the attribute condition, and string values containing spaces must be either quoted or the spaces in the values have to be escaped with the backslash to treat them literally. The backslash cannot be used with wildcards, which are always handled as special wildcard characters. An empty string can be defined by using two quotes: “”.

Note: When an attribute condition is specified as a value of the command-line attribute in the Linux/Unix environment, it must be quoted, or its special characters must be escaped with ‘\’ to avoid expansion by the shell.

Examples of Attribute Conditions

For example, if the second attribute (index=1) of the *cities* layer contains the city’s population, the following attribute condition

```
@1>=100000 && @1<500000
```

may be used to display only cities with population greater than or equal to 100K but less than 500K.

If the first attribute of the *cities* layer contains the name of each city, the following attribute condition

```
@0 == "New-"
```

will display only the cities whose names start with “New-”.

Examples of OpenGIS map query strings

What follows are examples of query strings for generating map images in CGI environment, and corresponding command-line equivalents for testing the query string without a web server setup. These examples use the sample dataset supplied with the Map Server.

Windows Note for all examples: On Windows, the script is called *GlmScript.pl*, and if using IIS instead of apache, the *cgi-bin* directory is called *Scripts*.

The following query will display the map of the United States with state outlines in the orthographic projection using the map server’s CGI or FastCGI mode, with grid and states layers:

```
http://www.myserver.com/cgi-bin/GlmScript?
  VERSION=1.3.0&REQUEST=GetMap&
  SRS=AUTO2:42003,1.,-97.,38.&
  WIDTH=700&HEIGHT=700&
  BBOX=-2500000,-2500000,2500000,2500000&
  BGCOLOR=0x0&
  STYLES=default&
  FORMAT=image/jpeg&
  LAYERS=earth,grid,states&
  STRETCH=0
```


The same on Windows with IIS:

```
http://www.myserver.com/Scripts/GlmScript.pl?
VERSION=1.3.0&REQUEST=GetMap&
SRS=AUTO2:42003,1.,-97.,38.&
WIDTH=700&HEIGHT=700&
BBOX=-2500000,-2500000,2500000,2500000&
BGCOLOR=0x0&
STYLES=default&
FORMAT=image/jpeg&
LAYERS=earth,grid,states&
STRETCH=0
```

or from the command line (a proper path to the *GlmMap* executable and *sample.sdf* file may be required):

```
GlmMap -generate -dataset sample.sdf -output earth.jpg -oGISreq
"VERSION=1.3.0&REQUEST=GetMap&
SRS=AUTO2:42003,1.,-97,38&
WIDTH=700&HEIGHT=700&
BBOX=-2500000,-2500000,2500000,2500000&
BGCOLOR=0x0&
STYLES=default&
FORMAT=image/jpeg&
LAYERS=earth,grid,states&
STRETCH=0"
```

The following query will display the view of the whole world in the orthographic projection with political boundaries:

```
http://www.myserver.com/cgi-bin/GlmScript?
VERSION=1.3.0&REQUEST=GetMap&
SRS=AUTO2:42003,1,10.,40.&
WIDTH=700&HEIGHT=600&
BBOX=-6500000.,-6500000.,6500000.,6500000.&
BGCOLOR=0x0&
STYLES=default&
FORMAT=image/jpeg&
LAYERS=earth,political&
STRETCH=0
```

or on the command line:

```
GlmMap -generate -dataset sample.sdf -output earth.jpg -oGISreq
"VERSION=1.3.0&REQUEST=GetMap&
SRS=AUTO2:42003,1,10.,40.&
WIDTH=700&HEIGHT=600&
BBOX=-6500000.,-6500000.,6500000.,6500000.&
BGCOLOR=0x0&
STYLES=default&
FORMAT=image/jpeg&
LAYERS=earth,political&
STRETCH=0"
```

The following query will display the world in the rectangular projection with political boundaries:

```
http://www.myserver.com/cgi-bin/GlmScript?
  VERSION=1.1.0&REQUEST=GetMap&
  SRS=EPSG:4326&
  WIDTH=800&
  HEIGHT=600&
  BBOX=-180,-90,180,90&
  BGCOLOR=0x0&
  STYLES=default&
  FORMAT=image/jpeg&
  LAYERS=earth,political&
  STRETCH=1
```

or on the command line:

```
GlmMap -generate -dataset sample.sdf -output earth.jpg -oGISreq
  "VERSION=1.1.0&REQUEST=GetMap&
  SRS=EPSG:4326&
  WIDTH=800&
  HEIGHT=600&
  BBOX=-180,-90,180,90&
  BGCOLOR=0x0&
  STYLES=default&
  FORMAT=image/jpeg&
  LAYERS=earth,political&
  STRETCH=1"
```

Examples of coordinate conversion and elevation queries

Note: The query request is for use with CGI/FastCGI version of the map server. Refer to the GLG Map Server documentation for the Map Server Library API.

Coordinate Conversion Query

To convert screen coordinates of a point to lat/lon, the *GetFeatureInfo* request type is used with *INFO_TYPE=lat_lon*. The *I* and *J* parameters of the request supply the X and Y image coordinates of the point of interest, and an optional *INFO_FORMAT* parameter may be set to either *text/xml* or *text/plain* to specify a desired output format. The map image parameters (i.e. width and height, projection, extent, etc.) need to be supplied as well in order to convert the point's X/Y coordinates to a lat/lon location. The map server will produce an output containing the lat/lon coordinates of the point.

For example, the following coordinate conversion request:

```
http://www.myserver.com/cgi-bin/GlmScript?
  VERSION=1.3.0&REQUEST=GetFeatureInfo&
  SRS=AUTO:42003,9001,-97.,38.&
  WIDTH=400&HEIGHT=400&
  BBOX=-2500000,-2500000,2500000,2500000&
  STRETCH=0&
  QUERY_LAYERS=default&
  INFO_TYPE=lat_lon&
  INFO_FORMAT=text/xml&
  I=200&J=200
```

will produce the following output:

```
Content-Type: text/xml

<GlmLatLonData>
  <QueryException>None</QueryException>
  <Lat>38.000000</Lat>
  <Lon>-97.000000</Lon>
</GlmLatLonData>
```

The following command will produce the same output from the command line:

```
GlmMap -generate -dataset sample.sdf -oGISreq
      "VERSION=1.3.0&REQUEST=GetFeatureInfo&
      SRS=AUTO:42003,9001,-97.,38.&
      WIDTH=400&HEIGHT=400&
      BBOX=-2500000,-2500000,2500000,2500000&
      STRETCH=0&
      QUERY_LAYERS=default&
      INFO_TYPE=lat_lon&
      INFO_FORMAT=text/xml&
      I=200&J=200"
```

Elevation Query for a Lat/Lon location

To obtain elevation of a point defined by the lat/lon coordinates, the *GetLocationInfo* request is used with *INFO_TYPE=elevation*. The *LON_LAT* parameter provides the lat/lon coordinates of the point. An optional *INFO_FORMAT* parameter may be set to either *text/xml* or *text/plain* to specify a desired output format.

The following elevation request for a point at a sea level defined by its lat/lon coordinates:

```
http://www.myserver.com/cgi-bin/GlmScript?
      VERSION=1.3.0&REQUEST=GetLocationInfo&
      QUERY_LAYERS=elevation&
      INFO_TYPE=elevation&
      INFO_FORMAT=text/xml&
      LON_LAT=10.,-30.
```

will produce the following output:

```
Content-Type: text/xml

<GlmElevationData>
  <QueryException>None</QueryException>
  <Elevation>0.000000</Elevation>
</GlmElevationData>
```

Note: This example uses *elevation* layer which is not defined in the sample dataset.

The following command will produce the same output from the command line:

```
GlmMap -generate -dataset sample.sdf -oGISreq
      "VERSION=1.3.0&
      REQUEST=GetLocationInfo&
      QUERY_LAYERS=elevation&
      INFO_TYPE=elevation&
      INFO_FORMAT=text/xml&
      LON_LAT=10.,-30."
```

Elevation Query for an X/Y image point

To obtain elevation of a point defined by a pair of X/Y image coordinates, the *GetFeatureInfo* request is used instead of *GetLocationInfo*. The point is defined by its X/Y image coordinates (*I* and *J* request parameters). The map image parameters (i.e. with and height, projection, extent, etc.) need to be supplied as well in order to convert the point's X/Y coordinates to the lat/lon location:

```
http://www.myserver.com/cgi-bin/GlmScript?
      VERSION=1.3.0&
      REQUEST=GetFeatureInfo&
      SRS=AUTO:42003,9001,-97.,38.&
      WIDTH=400&HEIGHT=400&
      BBOX=-2500000,-2500000,2500000,2500000&
      STRETCH=0&
      QUERY_LAYERS=default&
      INFO_TYPE=elevation&
      INFO_FORMAT=text/xml&
      I=200&J=200
```

or from the command line:

```
GlmMap -generate -dataset sample.sdf -oGISreq "VERSION=1.3.0&
      REQUEST=GetFeatureInfo&
      SRS=AUTO:42003,9001,-97.,38.&
      WIDTH=400&HEIGHT=400&
      BBOX=-2500000,-2500000,2500000,2500000&
      STRETCH=0&
      QUERY_LAYERS=default&
      INFO_TYPE=elevation&
      INFO_FORMAT=text/xml&
      I=200&J=200"
```

Note: This example uses *elevation* layer which is not defined in the sample dataset.

GIS Selection Query Example

To query a list of objects selected by the mouse click in the map generated by the *GetMap* request, the *GetSelection* request must provide all parameters that were used to generate the map, such as projection, center, extent, etc. The *LAYERS* parameter must be provided as well, but its value is different for this request and defines a vector layer (or list of layers) to be queried. Additionally, the *GetSelection* request must provide the *I* and *J* parameters to specify selection point location in pixels relatively to the map image origin, as well as the *SELECT_LABELS* parameter that controls label selection. The *FORMAT* parameter defines the output format and may be set to *text/plain* or *text/xml*.

For example, the following selection request:

```
http://www.myserver.com/cgi-bin/GlmScript?
  VERSION=1.3.0&REQUEST=GetSelection&
  SRS=EPSG:4326&STRETCH=1&
  WIDTH=800&HEIGHT=600&BBOX=-180,-90,180,90&
  LAYERS=us_cities&I=241&J=384&
  SELECT_LABELS=INTILE&FORMAT=text/plain
```

or a command from the command line:

```
GlmMap -generate -dataset sample.sdf -oGISreq "VERSION=1.3.0&
  REQUEST=GetSelection&
  SRS=EPSG:4326&STRETCH=1&
  WIDTH=800&HEIGHT=600&BBOX=-180,-90,180,90&
  LAYERS=us_cities&I=241&J=384&
  SELECT_LABELS=INTILE&FORMAT=text/plain"
```

may produce the following text output:

```
GlmMap: Info: Selection in layer: us_cities
GlmMap: Info: Selection type: object
GlmMap: Info: Feature type: marker
GlmMap: Info: Feature id: 7dc97c70
GlmMap: Info: Feature lon: -86.158056
GlmMap: Info: Feature lat: 39.768333
GlmMap: Info: Num attrs: 3
GlmMap: Info: Attribute: 0
GlmMap: Info: Attribute value: INDIANAPOLIS
GlmMap: Info: Attribute: 1
GlmMap: Info: Attribute value: 700807
GlmMap: Info: Attribute: 2
GlmMap: Info: Attribute value: IN
```

The output includes attributes of the selected city defined in the GVF file: a city name, its population and the state code.

The FORMAT parameter may be changed to text/xml to generate XML output.

3.2 Tools and Utilities

The GLG Map Server provides the following utilities:

- GVF ASCII/BINARY conversion utility
- Image and vector data tiling utility
- Vector data splitting utility
- Shapefile conversion utility
- Hierarchical tile parser and setup files generator

What follows are the command line options required to operate each of these utilities. The same executable is used, and the first option defines the mode.

Converting GVF ASCII and BINARY Files

Synopsis

```
GlmMap [-arg-file] -convert [-verbosity N] [-progress N] [options]
```

Description

The *-convert* option is used to convert GVF files between ASCII and BINARY formats. The ASCII format is cross-platform, but it is slower since the numerical data has to be converted from the string to internal native representation. The BINARY format is faster but will not allow sharing data between machines with different architectures.

Options

The following command line options are available for the GVF conversion utility.

-convert

If used as the first command line argument, will invoke the GVF ASCII/BINARY conversion utility. What follows is a description of subsequent command line options.

-a

Saves the converted GVF file in the ASCII format.

-b

Saves the converted GVF file in the BINARY format.

-path <directory>

Specifies the top-level directory with GVF files to be converted. All subdirectories of this directory will be traversed, making it easy to convert directory-based hierarchical datasets containing large number of files, as well as all GVF files in the map data directory.

-pattern <file_name_pattern>

Specifies GVF filenames to be converted, may use the “*” and “?” wild cards. If the pattern does not use wildcards, the directory and all its subdirectories are traversed and all files with a specified file name are converted. If the pattern contains the wild cards, all files matching the pattern in the directory and all its subdirectories are converted.

The files are converted in-place and the converted file is saved back into the original file, replacing it. It is recommended to backup the directory before running the conversion utility.

Examples

The following command converts all GVF files in the directory and all its subdirectories to BINARY GVF format:

```
GlmMap -convert -b -path /usr/local/map_data -pattern \*.gvf
```

The ‘*’ wildcard is escaped to pass it unchanged and prevent the shell from expanding it. It is not needed on Windows.

Merging Vector Data Files

Synopsis

```
GlmMap [-arg-file] -merge [-verbosity N] [-progress N] [options] files
```

Description

The *-merge* option is used to merge several vector data files into one GVF file that contains vector features from all merged files.

Options

The following command line options are available for the merge utility.

-merge

If used as the first command line argument, will invoke the GVF ASCII/BINARY conversion utility. What follows is a description of subsequent command line options.

-output <file>

Specified the output file for the merged data. The output data are written in the Map Server’s GVF format.

-filter <file>

Defines the path of an executable file which contains an optional filter to convert vector data from a custom format to the Map Server’s GVF format. Examples of filters and a guide to writing them can be found in the *GVF Filters and Data Converters* chapter on page 141.

-a

Saves the merged GVF file in the ASCII format.

-b

Saves the merged GVF file in the BINARY format.

Examples

The following merges all GVF files in the current directory and writes the result into the *merged.gvf* file:

```
GlmMap -merge -output merged.gvf *.gvf
```

Bounding Box Extraction Utility

Synopsis

```
GlmMap [-arg-file] -gvf-info [-verbosity N] [-progress N] files
```

Description

The *-gvf-info* option is used to display the bounding box of all vector features in each of the specified GVF files. The bounding box information may be used for specifying the *MIN LAT*, *MAX LAT*, *MIN LON* and *MAX LON* LIF parameters for layers that use square tiles.

Examples

The following displays bounding boxes of all GVF files in the current directory:

```
GlmMap -gvf-info *.gvf
```

Tiling Utility for Image and Vector Data

Synopsis

```
GlmMap [-arg-file] -tile [-verbosity N] [-progress N] [options]
```

Description

The *-tile* option is used for splitting a large high resolution image or vector data file into a number of smaller rectangular tiles for more efficient operation. When the layer is rendered, only the tiles visible in the current map request will be loaded, instead of loading the whole multi-megabyte image or vector file.

In the image mode, *-elevation* option may be used to tile elevation data. In the vector mode, one of the *-pos-range*, *-neg-range* or *-no-range* options must be specified to define the mode for handling the wrap-around of longitude values. It is used for handling lines that cross the world boundaries, such as the line extending from lon=359 degrees to lon=361 degrees.

Options

The following command line options are available for the tiling utility.

-tile

If used as the first command line argument, will invoke the image tiling utility.

-image

Specifies the image tiling mode.

-vector

Specifies the vector data tiling mode.

-filename <file>

Specifies the location of the image or vector data file to tile. The image must be either in the JPEG or TIF file formats. The vector data must be in the GVF or shapefile format, unless the *-filter* option is used to convert the data.

-template <string>

Defines the template to use for outputted tiles. The template must contain two “%” signs to be replaced by numbers representing the horizontal and vertical index of the tile. For example, the first tile, given the template *newimage_%_%.jpg* will become *newimage_0_0.jpg*. In the image tiling mode, the format of the generated tiles will be determined by the template’s extension or, if it’s missing, the extension of the *filename* parameter. In the vector data tiling mode, the output data files will be written in the GVF format.

-num-x-tiles <number>

Defines the number of horizontal tiles to tile into.

-num-y-tiles <number>

Defines the number of vertical tiles to tile into.

Image Tiling Options

The following tiling option is used only with *-image*:

-elevation

Specifies the elevation mode for tiling single-channel TIFF files containing *int16*, *uint16* or *float32* elevation data.

Vector Data Tiling Options

The following tiling options are used only with *-vector*:

-filter <file>

Defines the path of an executable file which contains an optional filter to convert vector data from a custom format to the Map Server’s GVF format. If a filter string contains the “%” character, it will be replaced with the file specified by the *-filename* option, otherwise the file will be appended at the end of the filter string. Examples of filters and a guide to writing them can be found in the *GVF Filters and Data Converters* chapter on page 141.

-neg-range

Specifies the most common -180 to 180 coordinate range for handling the wrap-around of longitude values.

-pos-range

Specifies a 0 to 360 coordinate range for handling the wrap-around of longitude values. This range is used by some datasets, for example, the shorelines data.

-no-range

Suppresses handling of the wrap-around of longitude values.

-custom-range

Specifies a custom coordinate range for handling the wrap-around of longitude values. The extent of the latitude range are provided by the *-min-wlon* and *-max-wlon* options.

-min-wlon <number>***-max-wlon <number>***

These options define the minimum and maximum longitude that appear in a data file for the custom coordinate range. These two options should be used with care as they define the central meridian and where coordinates should be wrapped.

Note: One of the *-neg-range*, *-pos-range*, *-custom-range*, or *-no-range* options must be specified for tiling vector datasets.

-gvf-extent

Directs the tiling utility to use the actual extent of the data in the GVF file for calculating extents of the generated tiles.

If the tiles are used in the square tiling mode, the *MIN_LON*, *MAX_LON*, *MIN_LAT*, *MAX_LAT* parameters of the layer's LIF file must be set to match the GVF extent used to generate the tiles. The *-gvf-info* command line option may be used to obtain the bounding box of the vector data contained in a GVF file.

If the tiles are used in the tree tiling mode, the Hierarchical Tile Parsing utility automatically handles tile extents.

-world-extent

Directs the tiling utility to use the extent of the whole world for calculating extents of the generated tiles. This option may be used for generating square tiles for a dataset that covers most of the world. The *MIN_LON*, *MAX_LON*, *MIN_LAT*, *MAX_LAT* parameters of the layer's LIF file must be set to match the whole world extent used to generate the tiles.

-min-lat <number>

-max-lat <number>

-min-lon <number>

-max-lon <number>

Region extent options, define the rectangular region to be used for calculating extents of the generated tiles. These options may be used for square-tiled datasets that cover only a small local area of the map. Tiling such datasets with the *-world-extent* option would result in the whole dataset being placed in either one tile or in just a few tiles. Defining a custom region using these extent options will generate tiles with boundaries that are better suited to such local-area datasets. The *MIN_LON*, *MAX_LON*, *MIN_LAT*, *MAX_LAT* parameters of layer's LIF file must be set to match the extent options used to generate the tiles.

Note: One of the *-gvf-extent*, *-world-extent*, or *-min/max-lat/lon* options must be specified for tiling vector datasets.

-a

Saves the converted GVF file in the ASCII format (default).

-b

Saves the converted GVF file in the BINARY format.

Examples

The following invocation of the image tiling utility will split an image into 100 (10x10) tiles:

```
GlmMap -tile -image -filename my_image.tif
       -num-x-tiles 10 -num-y-tiles 10
       -template my_image_tiles_%.%.jpg
```

The same for vector data:

```
GlmMap -tile -vector -neg-range -world-extent -filename my_vector.gvf
       -num-x-tiles 10 -num-y-tiles 10
       -template my_vector_tiles_%.%.gvf
```

Splitting Vector Data

Synopsis

```
GlmMap [-arg-file] -split [-verbosity N] [-progress N] [options]
```

Description

The *-split* option is used for splitting vector data file into several files based on the value of some attribute for more efficient operation, extracting data for some region of interest, or splitting large polygon features for faster rendering.

A vector data file may be split in a variety of ways by using different splitting options:

- *-bbox* option extracts and saves vector features that intersect the lat/lon region specified with the bounding box. If the *-num-x-tiles* and *-num-y-tiles* options are used, the file is also split into the requested number of rectangular tiles, as described in the *Tiling Utility for Image and Vector Data* section on page 80.
- *-attr* option extracts and saves vector features that have the value of the specified attribute matching specified attribute condition. For example, it may be used to split a data file containing all roads into several files each containing the roads of a certain type: primary roads, secondary roads, local roads, etc., based on the value of their CFCC code attribute. Each of the split data file may be set up as a separate layer, making it easier to switch each road type display on or off, and loading data only for the type of roads that needs to be displayed.
- *-num-points* option splits all polygons into smaller segments, up to the specified maximum number of points. Consider an example when a vector data file contains a shoreline of the whole continent as one polygon, with tens of thousands of points. Even if the map is zoomed on a small region, the whole shoreline has to be traversed to figure out what segments of it have to be rendered. Splitting the shoreline into smaller segments with, for example, five hundred points in each segment, will significantly increase the layer's rendering speed.

After the shoreline has been split into smaller segments with *-num-points*, it also becomes possible to split the data file into tiles using the tiling utility, which wouldn't work when the shoreline was defined as one large object (see below).

Note: Splitting into smaller segments should not be used for filled polygons, as it will distort the filled area. For more efficient rendering of large filled polygons, consider using smaller areas, for example, use state boundaries instead of the US boundaries to fill the background.

Only one of *-bbox*, *-attr* or *-num-points* splitting options can be used at once. To split based on several conditions or to split into several attribute ranges, the splitting utility should be invoked several times, once for each condition.

When tiling is performed, the utility does not split vector features that intersect several tiles. Instead, it replicates these features in each of the tiles. Therefore, tiling a dataset that contains a shoreline of America in the form of one polygon with thousands and thousands of points into 10x10 tiles would not do any good, as the shoreline would be replicated in each of the tiles. Since the shorelines are usually rendered as unfilled edge polygons, the right way of doing the tiling would be using the

-num-points option first to split the shoreline into smaller segments, and only then split the data file into tiles. The number of points in the segments should be chosen depending on the size of one tile, so that each tile contains at least a few segments.

For *-bbox* and *-num-points*, one of the *-pos_range*, *-neg_range*, *-custom-range* or *-no-range* options must be specified to define the mode for handling the wrap-around of longitude values. It is used for handling lines that cross the world boundaries, such as the line extending from lon=359 degrees to lon=361 degrees.

Options

The following command line options are available for the vector data splitting utility.

-split

If used as the first command line argument, will invoke the vector data tiling and splitting utility. What follows is a description of subsequent command line options.

-filename <file>

Specifies the location of the vector data file to tile. The data must be either in the Map Server's GVF or shapefile format, unless a filter is used.

-output <file>

Specified the output file for the processed data. The output data are written in the Map Server's GVF format.

If the vector data is also being split into multiple tiles using the *-num-x-tiles* and *-num-y-tiles* options, the *-output* option must specify a tiling template. The template must contain two “%” signs to be replaced by numbers representing the horizontal and vertical index of the tile. For example, given the template *newvector_%_.gvf*, the first tile will become *newvector_0_0.gvf*.

-filter <filter>

Defines the path of an executable file which contains a filter to convert vector data from a custom format to the Map Server's GVF format. If a filter string contains the “%” character, it will be replaced with the file specified by the *-filename* option, otherwise the file will be appended at the end of the filter string. Examples of filters and a guide to writing them can be found in the *GVF Filters and Data Converters* chapter on page 141.

-attr <conditions>

Defines an attribute condition or list of attribute conditions used to split the vector data. Only the vector features that match the attribute condition will be written. If *-attr* is used multiple times to specify several sets of attribute conditions, a vector feature will be written if it matches any of the specified conditions. The *-invert-attr* option may be used to invert attribute conditions.

For example, the following command-line option will select all vector features that have the value of the second attribute (index=1) within the specified range:

```
-attr "@1>=100000 && @1<500000"
```

Refer to the *Attribute Condition Syntax* section on page 71 for a complete description of the attribute condition syntax.

Note: On Linux/Unix systems, the value of the *-attr* option must be quoted to avoid expansion by the shell. Alternatively, all special characters in the attribute condition, including the quotes that are part of the string value must be escaped with `\`.

-invert-attr <index>

Inverts attribute conditions. Outputs only vector features that do not match any of the conditions specified with the *-attr* option.

-bbox

Specifies that only vector features fitting into a certain rectangular region should be outputted.

-min-lat <number>

-max-lat <number>

-min-lon <number>

-max-lon <number>

To be used with the *-bbox* option, defines the minimum and maximum longitude and latitude of a rectangular region: all vector features intersecting this region will be outputted.

-num-x-tiles <number>

-num-y-tiles <number>

Defines an optional number of horizontal and vertical tiles to tile the output into for the *-bbox* option (default is 1). If values different from 1 are used, the *-output* option must specify a tiling template.

-num-points <number>

Defines the maximum number of points an outputted polygon may have. This is useful for data that contains polygons with large numbers of points. A common value is 10. For example, if there exists a 36 point polygon in the file, the utility will split the polygon into 4 polygons with at most 10 points each. **Note:** This option cannot be used with filled polygons.

-neg-range

Specifies the most common -180 to 180 coordinate range for handling the wrap-around of longitude values.

-pos-range

Specifies a 0 to 360 coordinate range for handling the wrap-around of longitude values. This range is used by some datasets, for example, the shorelines data.

-no-range

Suppresses handling of the wrap-around of longitude values.

-custom-range

Specifies a custom coordinate range for handling the wrap-around of longitude values. The extent of the latitude range are provided by the *-min-wlon* and *-max-wlon* options.

-min-wlon <number>*-max-wlon* <number>

The options define the minimum and maximum longitude that appear in a data file for the custom coordinate range. These two options should be used with care as they define the central meridian and where coordinates should be wrapped.

Note: One of the *-neg-range*, *-pos-range*, *-custom-range*, or *-no-range* options must be specified for the *-bbox* and *-num-points* split modes.

Examples:

The following invocations of the vector splitting utility will split a dataset file. First, the data will be split into polygons of no more than 10 points:

```
GlmMap -split -neg-range -filename data_orig.gvf -num-points 10
      -output data_10.gvf
```

Then, only the data in the boundaries of the U.S. will be extracted:

```
GlmMap -split -neg-range -filename data_10.gvf
      -bbox -min-lat 25 -max-lat 50 -min-lon -120 -max-lon -70
      -output data_US.gvf
```

Finally, the data will be tiled into 100 tiles (10x10):

```
GlmMap -tile -vector -neg-range -filename data_US.gvf
      -num-x-tiles 10 -num-y-tiles 10 -output data_tiled_%.%.gvf
```

The last two operations could be combined and performed in one step:

```
GlmMap -split -neg-range -filename data_10.gvf
      -bbox -min-lat 25 -max-lat 50 -min-lon -120 -max-lon -70
      -num-x-tiles 10 -num-y-tiles 10 -output data_tiled_%.%.gvf
```

Slimming Utility for Vector Data

Synopsis

```
GlmMap [-arg-file] -slim [-verbosity N] [-progress N] [options]
```

Description

The *-slim* option is used for slimming high-resolution datasets to generate lower-resolution overview layers.

Options

The following command-line options are supported for the slimming utility:

-slim

If used as the first command line argument, will invoke the vector data slimming utility.

-filename <file>

Specifies the location of the vector data file to tile. The data must be either in the Map Server's GVF or shapefile format, unless a filter is used.

-output <file>

Specified the filename to output the processed data to. The output data files are written in the Map Server's GVF format.

-resolution <dir>

Defines the slimming resolution in lat/lon degrees. All polygon features with extents smaller than the resolution value will be eliminated, and all polygon points with distances between them less than the resolution value will be merged.

-a

Saves the converted GVF file in the ASCII format (default).

-b

Saves the converted GVF file in the BINARY format.

Examples

The following command will slim the GVF file to the resolution of 0.1 degree:


```
GlmMap -slim -resolution -filename shorelines.gvf  
      -output shorelines_0.1.gvf
```

Shapefile Conversion Utility

Synopsis

```
GlmMap [-arg-file] -shp2gvf [-verbosity N] [-progress N] [options]  
      [list of shapefiles or directories to convert]
```

Description

The *-shp2gvf* option is used for converting shapefiles to the Map Server's GVF format. While the shapefiles may be used without conversion, it is more efficient to preprocess and convert them to GVF format at the setup time. The conversion parameters also allow to eliminate the shapefile attributes that are not used for rendering the data.

The *-r* option may be used to convert all shapefiles in a directory structure. By default, the output will be written into a file whose name is formed by adding the ".gvf" extension to the name of the shapefile.

The *-show-attrs* options may be used to display shapefile's attributes. The *-verbosity* option may be used to display the content of the shapefile being converted.

The *-gvf-info* command line option may be used to obtain the bounding box of the generated GVF file to use it in a LIF file. See the *Bounding Box Extraction Utility* section on page 80 for details.

Options

The following command-line options are supported for the shapefile conversion utility:

-shp2gvf

If used as the first command line argument, will invoke the shapefile conversion utility.

-output <file>

Specified an optional output file for the processed data when a single shapefile is being processed. If not specified, the output will be written into a file whose name is formed by adding the ".gvf" extension to the name of the corresponding shapefile.

-pattern <pattern>

Defines a shapefile name pattern for the recursive option. When a directory is traversed, all shapefiles whose name matches the pattern will be converted. Wild cards are supported and the default value is "*".

-r

Recursive mode. If a directory is specified as an input file, all its subdirectories will be traversed as well, and all shapefiles matching the pattern will be converted.

-a

Saves the converted GVF file in the ASCII format (default).

-b

Saves the converted GVF file in the BINARY format.

-show-attrs

Prints out the content of the shapefile's attribute table without converting the data.

-dump

When used with *-show-attrs*, dumps attribute values.

-no-header

Suppresses the output of an attribute table header for the *-dump* option.

-m

Specifies a multi-line mode for the *-dump* option.

-raw

When used with *-dump*, dumps raw attribute values with no formatting.

-all-attrs

Causes all shapefile attributes to be written into the GVF file in the order they are defined in the shapefile. This is the default when no attributes are defined with the *-attr* option.

Writing all attributes is excessive: only a small number of attributes defined in a shapefile is used at rendering time. Use *-attr* option to define the attributes to be written into the GVF file.

-no-attrs

Causes no shapefile attributes to be written into the GVF file.

-attr <index>

Specifies an index of the shapefile attribute to be included into the output. More than one attribute option may be specified. The attributes will be converted in the order they are defined: the attribute specified by the first *-attr* option will be written into the GVF file as an attribute with index=0, the second - as attribute index=1, and so on. Shapefile attributes of the types *FTInteger* and *FTDouble* are written into the GVF file as double attributes, and shapefile attributes of *FTString* type are written as string attributes.

If the *-attr* option is used, only the specified shapefile attributes will be written into the GVF file, and the rest of the attributes will be discarded. If no attributes are specified, all attributes will be written.

-marker-label-attr <index>

Specifies an optional index of the shapefile attribute to be used for the label string of point features. The index must point to an attribute of *FTString* type.

If *-marker-label-attr* is defined, point features are written into the GVF file as markers of the *GVF_M_LABEL* type with the label string. If *-marker-label-attr* is not defined, point features from the shapefile are written as marker objects of the *GVF_M_MARKER* type, and the *LABEL FORMAT* LIF option may be used to display one of the marker attributes as its label.

-close-polygons

Coverts polylines to polygons by adding a closing segment.

Examples

The following command will convert the *shorelines.shp* shape file and write the GVF output into the *shorelines.gvf* file, discarding all shapefile attributes:

```
GlmMap -shp2gvf -no-attrs -output shorelines_0.1.gvf shorelines.shp
```

The same, but preserving all shapefile attributes in the GVF file:

```
GlmMap -shp2gvf -all-attrs -output shorelines_0.1.gvf shorelines.shp
```

The following command will preserve just two shapefile attributes with indices 2 and 4, saving them as the GVF attributes with index 0 and 1:

```
GlmMap -shp2gvf -attr 2 -attr 4 -output shorelines_0.1.gvf  
shorelines.shp
```

The following command will display all attributes of a shapefile without converting the data:

```
GlmMap -shp2gvf -show-attrs shorelines.shp
```

The following command will convert all shapefiles in the *my_shapefile_data* directory, writing the converted data into the corresponding files with the “.gvf” extension:

```
GlmMap -shp2gvf -r -all-attrs -pattern *.shp my_shapefile_data
```

Hierarchical Tile Parsing (Advanced)

Synopsis

```
GlmMap [-arg-file] -slf [-verbosity N] [-progress N] [options]
```

Description

The *-slf* option is used for automatic generation of setup files for directory-based vector or raster datasets containing multiple data files in hierarchically organized subdirectories. Each data file will be handled as a tile.

Vector tiles may be non-rectangular and grouped in geographical or administrative regions. The subdirectories contain tiles for one region, for example a state. Each of the state subdirectories, in turn, may contain another level of subdirectories containing data files for counties, and so on, with no limit on the depth of the hierarchy. Such organization of data is called hierarchical tree tiling, and it is used in datasets such as the US Census Tiger/Line data, Digital Chart of the World (DCW/VMAP) dataset, and many others.

Image tiles are always rectangular but may cover non-rectangular area with some tiles missing. The image tiles may be grouped in geographical or administrative non-rectangular regions as well.

The utility traverses all subdirectories of the dataset, extracting the extent information for each data file. For each traversed directory, the utility writes the directory's Sub-Layer File (SLF), that includes the list of all data files and subdirectories in the directory, as well as the combined extent information for all directory entries. The extent information in the SLF file enables the Map Server to efficiently determine if any given subdirectory of the dataset is needed for rendering the current map request, and to do so without traversing each of the data files and subdirectories it contains.

The top-level SLF file generated by the utility for the top-level directory of the dataset is included in the layer's LIF file using TILES SUBLIF parameter. The LIF file will contain user defined attributes, while the included SLF file is generated by the utility and contains data description. This way, if the SLF file is regenerated, user defined layer data, such as *LINE WIDTH*, *EDGE COLOR* and other visual attributes, will not be lost.

The utility needs to be run for the initial setup of a hierarchical tree-tiled datasets, or when data files are added or changed.

Options

What follows is a description of command line options for both vector and image mode.

-slf

If used as the first command line argument, will invoke the hierarchical tile parsing utility.

-vector

Specifies the vector mode for processing vector data tiles. Tile extent information will be extracted from the GVF data files.

-image

Specifies the image mode for processing image tiles. Tile extent information is extracted from the SLF files which are associated with the image files and were produced by the image import and conversion utility.

-single-layer

Specifies the single layer mode. In this mode, all tiles specified by the *pattern* parameter are combined into a single layer. This is the default for the image mode.

-multi-layer

Specifies the multi-layer mode for processing vector tiles. In this mode, the tiles specified by the *pattern* parameter are combined into multiple layers based on their names. For example, all *roads.gvf* files located in any of the tile subdirectories will be aggregated into the road layer with the *roads.gvf.slf* name used for its SLF file, all *lakes.gvf* files will be aggregated into a lakes layer with *lakes.gvf.slf* SLF file, and so on. This mode is available only for the vector data and is used for processing all layers of the DCW and TIGER datasets at once.

-path <dir>

Defines the top-level directory of the dataset to start parsing from, either as an absolute path or relative to the current directory.

-suppress-bbox

Suppresses generation of the inlined BBOX for included SLFs. By default, the extent of the bounding box is added to each included SLF line using the BBOX descriptor to increase performance of large hierarchical data sets.

-pattern <pattern>

Defines a filename pattern that describes what files to look for in the specified path. Wild cards are supported.

In the **multi-layer mode** for vector data, the pattern should match the names of the vector data files. The dataset may contain data files for all layers combined in the same directories, and all of them may be processed at once by using “*.gvf” as a pattern. For example, a US dataset may contain a number of state subdirectories, each containing data for roads, land and water features. Each state directory may contain subdirectories containing data for its counties, and so on. The data files for the road features will be named the same way in all state and county subdirectories, and the same is true for the land and water data files.

By using “*.gvf” as a pattern, all datasets will be processed at once, generating SLF files for roads, land and water features in each of the dataset’s subdirectories. The top-level directory of the dataset will contain the top-level SLF files generated for the road, land and water layers.

These top-level SLF files will contain cumulative information for each layer's data and should be included in the corresponding layer LIF files using the *TILES SUBLIF* directive.

The parsing utility traverses the directory structure looking for data files that match the pattern. When a matching data file is found, a Sub-Layer File (SLF) is generated for it. It will contain the extent of the vector data in that file and the name of the file. The name of the SLF file is formed by adding the *.slf* extension at the end of the file name.

For each directory containing data files matching the pattern, the utility writes a directory SLF file that contains cumulative extent of all data files in the directory and all its subdirectories. Since the utility may be invoked to process directory structure that may contain data for more than one layer, one subdirectory SLF file is written for each layer. The name of each directory SLF file is the same as the name of the corresponding SLF written for the data file. If a directory contains both the data files and subdirectories with data files, the data file SLF is using *.sub.slf* extension to avoid name conflicts. This process is repeated recursively for all directories in the hierarchy to an unlimited depth.

In the **single layer mode** for vector data, the tile parsing utility has to be run once for each vector layer. For each layer, the provided *pattern* parameter has to match the names of the *.gvf* files for that layer, and the *-out-slf-name* parameter should provide the name of the generated directory SLF files for the layer. For each directory that contains matching data files, the utility generates a directory SLF file containing cumulative extent of all data files in the directory and all its subdirectories.

The parsing utility traverses the directory structure looking for the GVF files that match the pattern. When a matching data file is found, a Sub-Layer File (SLF) is generated for it. It will contain the extent of the vector data in that file and the name of the file. The name of the SLF file is formed by adding the *.slf* extension at the end of the file name.

For each directory containing GVF files matching the pattern, the utility writes a directory SLF file that contains cumulative extent of all GVF files in the directory and all its subdirectories. The *-out-slf-name* parameter defines the name of the generated directory SLF files. This process is repeated recursively for all directories in the hierarchy to an unlimited depth.

In the **single layer mode** for image, the image import and conversion utility has to be run first to produce image SLF files containing georeferencing information for each image file. Names of these files are formed by adding the *".slf"* extension to the corresponding image filename.

After that, the tile parsing utility has to be run once for each image layer, with the pattern set to match the names of the image SLF files for the layer and the *-out-slf-name* providing the name of the generated directory SLF files for that layer. For each directory that contains matching data files, the utility generates a directory SLF file containing cumulative extent of all data files in the directory and all its subdirectories.

The directory SLF files that are generated for the top level directory are included in the corresponding layer LIF files using the *TILES SUBLIF* directive.

The parsing utility traverses the directory structure looking for the image SLF files that match the pattern. These files were generated by the image import and conversion utility and contain the names and extent of a corresponding image file.

For each directory containing SLF files matching the pattern, the utility writes a directory SLF file that contains cumulative extent of all image files in the directory and all its subdirectories. The *-out-slf-name* parameter defines the name of the generated directory SLF files. This process is repeated recursively for all directories in the hierarchy to an unlimited depth.

-out-slf-name <filename>

Specifies the name for the generated directory SLF files for the single-layer mode and is ignored in the multi-layer mode. In the single layer mode, all tiles are combined in one layer and the *filename* is used for the generated directory SLF files in each of the subdirectories of the dataset. The

directory SLF file generated in the top-level directory is included in the layer's LIF file using the *TILES SUBLIF* directive. Unique filenames must be used for generating SLF files for different layers.

Examples

The following command will generate SLF files for all layers of the hierarchically tiled vector dataset in the *my_vector_data* directory:

```
GlmMap -slf -vector -multi-layer -path my_vector_data -pattern *.gvf
```

The top-level SLF files generated in the *my_vector_data* directory are then included into the corresponding layers' LIF files using the *TILES SUBLIF* directive.

The following command will generate directory SLF files for one layer of the hierarchically tiled image dataset in the *my_image_data* directory:

```
GlmMap -slf -image -path my_image_data -pattern \*.GN\?.tif.slf  
-out-slf-name GN.slf
```

It will generate directory SLF files by combining image information from all files ending with *“.GN?.tif.slf”*, which were generated by the image import utility, and writing out combined *GN.slf* files for the layer. The ‘?’ wildcard in the pattern was used to match a single character, to select files ending with *“.GN1.tif.slf”*, *“.GN2.tif.slf”*, and so on. The top-level *GN.slf* file in the *my_image_data* directory is then included into the layer's LIF file using the *TILES SUBLIF* directive.

Image and Elevation Data Import (Advanced)

Synopsis

```
gis_image_import [options] <list of input image files or directories>
```

Description

The GIS image import utility is used for importing raster and elevation data in various image interchange formats and converting them to the more common TIFF or JPEG formats. This eliminates dependencies on various image format libraries at run time and greatly reduces the complexity and size of the map server executable. The utility is based on the GDAL¹ library and is provided for the Linux platform by default.

The utility may be used with several processing mode options to perform various raster data processing:

- *-write-image* option is used to convert images to an 8-bit per channel TIFF, PNG or JPEG formats supported by the map server.
- *-elevation* option is used to convert elevation data to a single-channel TIF file containing ele-

vation data.

- *-shadow* option is used to generate a shadow relief image in the JPEG format from the input elevation data and may be used in conjunction with the *-elevation* option or stand-alone.
- *-slf* option scans the input files and generates SLF setup files with extent information for them. It may be used in conjunction with the *-write-image*, *-elevation* and *-shadow* options or stand-alone. The hierarchical data parsing utility may then be used to combine these SLF files into a directory SLF file for a layer, which is included into the layer's LIF file.

The output file name is derived by adding an extension specified with the *-out-ext* option to the original file name.

For hierarchical directory-based datasets, the utility may be invoked in the recursive mode, traversing all subdirectories of the dataset and processing all files that match a specified pattern.

Options

What follows is a description of command line options.

-verbosity <level>

Verbose mode. When set to value greater than 0, produces verbose output according to the specified verbosity level in the range from 1 to 10. By default, the verbosity is set to 1 to print only the basic information.

-progress <number>

Prints progress information for every N processed files, where N is the option value. The default value is 100. Use 0 to suppress progress reporting.

-info-only

Used with other options to perform a "dry run", printing out information but not writing out any output files.

1. Copyright (c) 2000, Frank Warmerdam.

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 or substantial portions of the 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.

-write-image

Converts input images to the image format defined by the *-out-format* option. Each converted image is written to a file whose name is formed by adding the suffix defined by the *-out-ext* option to the name of the corresponding input image file. Using this option requires the *gdal_translate* utility to be present on the system and in the search path.

-elevation

Handles input files as elevation data and writes the output as elevation data in the single-channel TIFF format. The output data format (*int16*, *uint16* or *float32*) is chosen as a closest match to the format of the input data. The name of the output file is formed by adding the “.tif” suffix to the name of the corresponding input file.

-shadow

Handles input files as elevation data and generates shadow relief images in JPEG format for each of them. The name of each generated shadow relief image file is formed by adding the “.jpg” suffix to the name of the corresponding input file.

For large data files, verbosity values greater than 8 may be used to indicate the progress of the shadow relief image generation.

-data-shift

Specifies an optional data shift value used to fit sample data into a single byte used for generating shadow relief image. The default shift value is 6, which discards 6 lower bits of the elevation data in the *int16* and *uint16* formats to accommodate heights up to 16383 ($2^{14} - 1$). For data files with low elevation, discarding the lower bits will result in the zero elevation values. For such data files, a lower data shift may be used. For example, for data files where with maximum elevation values do not exceed 255 ($2^8 - 1$), a data shift value of 0 may be used, effectively scaling the range to have 255 as the range high. The resulting shadow relief image will have exaggerated shadow compared with images that used the default data shift value.

The *-verbosity* option with a value greater than 3 may be used to output information about the range of values in the input data file, which may be used to select a proper data shift value.

-data-offset

Specifies an optional data offset value to be added to each sample's data when generating shadow relief image from data in the *int32* format. The shadow relief image generation process discards negative elevation data, since negative values are often used to represent areas with missing data. If negative data have to be included into the generated shadow relief image, a positive offset may be specified to shift all data upwards to make a desired range of data positive. The default offset value is 0.

The *-verbosity* option with a value greater than 3 may be used to output information about the range of values in the input data file, which may be helpful for selecting a proper data offset value.

-slf

Generates SLF setup files containing image extent for each of the input raster files. The name of each generated SLF file is formed by adding the suffix defined by the *-out-ext* option and then the “.slf” suffix to the name of the corresponding input image file.

-out-format <format>

Specifies a GDAL-supported raster format for the converted image generated using the *-write-image* option. Use *GTiff* to generate TIFF files, or *JPEG* for JPEG files. Refer to the documentation of the *gdal_translate* utility for a list of available GDAL output raster formats.

-out-ext <extension>

Specifies a suffix added to the name of the original input file to produce the name of the output file. This option is mandatory.

-postfix <extension>

Specifies an optional suffix added to the name of the output file prior to adding the suffix specified by the *-out-ext* option.

-pattern <pattern>

Specifies an optional filename pattern for recursive mode. Only the input files matching the pattern will be processed. The pattern may contain the ‘*’ and ‘?’ wild cards, which must be escaped with ‘\’ in Linux/Unix environment. The default pattern is “*” and all files in the directories are processed.

-r

Recursive mode. If a directory is specified as an input file, all its subdirectories will be traversed as well, and all files matching the pattern will be converted.

-u

Disables compression for the files generated with the *-write-image* option. By default, LZW compression is used for the converted image files.

-k

In case of an error, skip the error file and continue. By default, the utility stops on the first encountered error. This option may be used for processing large image datasets which may contain a few files with errors.

-f

In case of an error, still try to process the file if possible. This option may be used to force the utility to save the converted file in spite of some errors, in order to inspect the content of the file for the clues on the causes of the error.

-dont-emboss

Modifies the *-shadow* option behavior to write the output JPEG file as is, without embossing it. This option may be used to debug problems with the input elevation data files.

Examples

The following command converts all raster files in the current directory with the “.nitf” extension to TIFF and generates SLF setup files for them:

```
gis_image_import -write-image -slf -out-ext ".tif"
                 -out-format GTiff *.nitf
```

The following command processes the directory-based hierarchical dataset in the *NITF_dir* directory, converting all “.nitf” files to TIFF and generating SLF setup files:

```
gis_image_import -r -k -verbosity 1 -write-image -slf -out-ext ".tif"
                 -out-format GTiff -pattern \*.nitf NITF_dir
```

The following command converts DTED elevation tiles of a directory-based hierarchical dataset in the *DTED_dir* directory to a single-channel TIFF elevation tiles, generating SLF setup files for them:

```
gis_image_import -r -elevation -slf -out-ext ".tif"
                 -pattern \*.dted DTED_dir
```

The following command regenerates SLF files for the elevation tiles generated in the previous example without generating the TIFF files:

```
gis_image_import -r -slf -out-ext ".tif" -pattern \*.dted DTED_dir
```

Note the use of the “.tif” extension, which should match the extension used for the generated elevation tiles.

The following command generates shadow relief images for the DTED elevation dataset used above and generates SLF setup files for the shadow relief tiles:

```
gis_image_import -r -shadow -slf -out-ext ".jpeg"
                 -pattern \*.dted DTED_dir
```


Chapter 4

Map Server API**4****4.1 Overview**

The GLG Map Server Application Programming Interface (API), provided as a set of C functions, provides a cross-platform way to incorporate the Map Server into an application. The API provides all functionality available in the Map Server executable as well as some advanced functionality to optimize the Map Server's performance. The API allows a developer to create multiple Map Objects and multiple Datasets, as well as cache vector and image data for enhanced performance.

The GLG Map Server is intended as a mapping service and therefore provides no way of displaying generated map images. However, the GLG GIS Object can be used to integrate the generated map images into a graphical GLG application.

If the GLG GIS Object is used, it will handle all invocations of the Map Server API transparently, so that the user does not need to know the Map Server API, and this chapter may be skipped.

The Map Server API provides functions to read data, create and manipulate Map Objects and write the generated images to a file, and is used in applications that do not use the GLG GIS Object.

In order to run an application utilizing the Map Server API on various platforms, the code simply needs to be recompiled. Although the Map Server API itself is completely cross-platform, it is up to the developer to make the rest of the application platform-independent.

Resources

The Map Server API utilizes the same resource mechanism that is used in the GLG Toolkit. Maps and Datasets are both objects whose resources can be Get and Set. Similarly, there are three types of resources: D, S and G which represent double, string and geometrical data. Both Maps and Datasets are GlmObjects. The basic resource access mechanism has the same format as in the GLG Toolkit. For example, to set the Height of a map object, a D resource:

```
GlmSetDResource( map_object, "Height", 600. ); // set the height
GlmGetDResource( map_object, "Height", &height ); // get
```

Geometrical resources are passed three values or pointers:

```
GlmSetGResource( map_object, "FillColor", 1., 0., 0. );
// set the color
GlmGetGResource( map_object, "FillColor", &r, &g, &b ); // get
```

String resources are passed a string or a pointer to a string:

```
GlmSetSResource( map_object, "Dataset", "earth.sdf" );
// set the string
GlmSetSResource( map_object, "Dataset", &dataset ); // get
```

4.2 Basic Structure of a Mapping Application

The basic structure of any mapping application requires:

- Initialization of the Map Server
- Creation of a Map Object
- Specifying the desired attributes of the map image
- Generation of the image
- Writing the image to a file

Any mapping application must first include the file *GlmApi.h* via the *#include* directive. The initialization of the Map Server is accomplished via a call to the *GlmInit()* function. The function has one parameter, a string, which defines the path to a Server Dataset File (SDF). All layer and font information will be loaded from this file.

A Map Object is created via the *GlmCreateMap()* function. Its resources are set via the *GlmSetResource* functions. What follows is a detailed description of available resources. These are the same resources that are specified in the *-oGISreq* command line option.

The type of the resource (D,G or S) is specified after the resource name. When a default value is not listed, it is 0 for D resources, (0,0,0) for G resources and NULL for S resources.

DatasetName D

Specifies the name of the dataset to use. The default is "DefaultDataset", which is always created by *GlmInit* function. If the application uses multiple datasets, the *DatasetName* resource is used to set the name of the dataset to be used for generating the map. If the *DatasetName* resource was changed, the *GlmSetupMap* function must be called before selecting layers.

Projection D

Specifies which projection is to be used to render the map image. Valid values are *GLM_ORTHOGRAPHIC* (default) and *GLM_RECTANGULAR*.

CenterLon D

Defines the longitude at the center of the map image. A value of 0 is at the Greenwich meridian. Negative values are west of the meridian and positive values are east of the meridian.

CenterLat D

Defines the latitude at the center of the map image. A value of 0 is at the equator. Negative values are below the equator and positive values are above the equator.

VExtent D

Defines the vertical extent of the map image. If using the rectangular projection, the extent is in latitudinal coordinates. If using the orthographic projection, the extent is in meters.

HExtent D

Defines the horizontal extent of the map image. If using the rectangular projection, the extent is in longitudinal coordinates. If using the orthographic projection, the extent is in meters.

AngleD

For the rectangular projection, defines the map rotation angle, in degrees counter-clockwise.

Width D

Defines the width of the generated map image in pixels.

Height D

Defines the height of the generated map image in pixels.

Stretch D

Defines whether or not to maintain the correct lon/lat aspect ratio. If set to 1, the image will be stretched according to the Width and Height resources. If set to 0 (default), the image will maintain its lon/lat aspect ratio. For example, in the orthographic projection, if the entire earth is being viewed, if Stretch is set to 0, the earth will always appear as a perfect sphere, instead of an oblong one.

Background G

Defines the background color to be rendered in the map image. The three values are in the range 0 to 1 and define the red, green and blue components of the color, respectively.

ImageAntiAliasing D

Enables or disables antialiasing while scaling images of raster layers. Will be turned on if set to 1 (default 0).

ImageErrors D

Defined whether or not to render errors into the image. If set to 1, errors will be rendered. If set to 0 (default), they will not.

Verbosity D

Defines the verbosity value (default 0). If verbosity level is set to positive integer value, from 1 to 6, diagnostic information will be generated. If verbosity level is set to a negative integer value, from -1 to -6, the performance and timing output will be produced. If verbosity level is set to the values 1001 or 1002, tile extent information is displayed in the generated images for the *GetMap* requests. The verbosity output is printed to the standard error, the higher the number, the more output is generated. Errors and info are also displayed in the generated image (if IERRORS is set to 1) and logged into the *glm_error.log* file. By default, the verbosity is set to 0.

Finally, the layers to be rendered into the map image need to be set. The most commonly used function is the *GlmSetLayers* function. It takes a comma separated list of layers as its parameter. It also takes *default* and "*" as valid layer names. If *default* is used as a layer name, all layers with the *DEFAULT ON* flag specified will be rendered, and if "*" is used as a layer name, all layers in the SDF file will be rendered. For example:

```
GlmSetLayers( map_object, "default,clouds,shores" );
```

will turn on all default layers as well as the clouds and shores layer. Two other common functions are: *GlmEnableLayer*, which enables one layer, and *GlmDisableLayer*, which disables one layer in the map object. If the map object is used repeatedly, the *GlmResetLayers* function may be called to clear the previous selection prior to selecting new layers.

Once the Map Object is created and the resources set, the actual map generation needs to occur. This is accomplished with a call to *GlmGetProjection()*. Its one parameter is the map object with all its parameters specified. It is important to note that after the image is generated, the map object does not need to be discarded. It may be used to generate other images after altering some or all of the map object's resources.

After a call to *GlmGetProjection*, the *GlmWriteImage* function can be used to write the resulting map image into a file. For example:

```
GlmGetProjection( map_object );
GlmWriteImage( NULL, "output.jpg", map_object );
```

generates a map image and writes it to a file called *output.jpg*. The first parameter may specify an open file handle. For example, to write them to the standard output, the following can be used:

```
GlmGetProjection( map_object );
GlmWriteImage( stdout, NULL, map_object );
```

The *GlmGetMapData* function can be used to get access to the raw data of the generated image.

The *GlmGetElevation*, *GlmGetLatLon*, *GlmGetXYZ* and *GlmConvert* functions may be used to query elevation and perform coordinate conversion.

Code Example

What follows is an example of a simple mapping application which creates a map object, generates a map image, and writes it to a file. It is available as a *map_generation.c* file in the *glg/map_server/src/* directory.


```
#include "GlgApi.h"
#include "GlmApi.h"
main()
{
    GlmObject map; // The map object to be used

    // Loads a dataset from SDF file in the current directory or GLG_PATH
    GlmInit( "earth.sdf" );

    // Creates the map object using default dataset created by GlgInit.
    map = GlmCreateMap();

    // Sets the image width and height
    GlmSetDResource( map, "Width", 800. );
    GlmSetDResource( map, "Height", 600. );

    // Sets the map projection
    GlmSetDResource( map, "Projection", (double)GLM_ORTHOGRAPHIC );

    // Sets the center lat/lon of the image
    GlmSetDResource( map, "CenterLat", 30. );
    GlmSetDResource( map, "CenterLon", 0. );

    // Sets the vertical and horizontal extents, in meters for the
    // ORTHOGRAPHIC projection.
    GlmSetDResource( map, "VExtent", 14000000. );
    GlmSetDResource( map, "HExtent", 14000000. );

    GlmSetDResource( map, "Stretch", 0. ); // Turns off X/Y stretching

    // Sets up the map object. It is optional here but must be used
    // before selecting layers if the DatasetName resource was changed.
    GlmSetupMap( map );

    // Clears any previous layer selections.
    GlmResetLayers( map );

    // Enables the earth and clouds layers in the generated map.
    GlmSetLayers( map, "earth,clouds" );

    GlmGetProjection( map ); // Generates the map image

    // Writes the image to a file
    GlmWriteImage( NULL, "output.jpg", map );
}
```

4.3 Linking with the Map Server Library

The details of the process of linking an application program with the GLM Map Server library depends on the operating system and windowing environment where the application will run. Since the Map Server depends on the GLG Toolkit, GLG libraries will also need to be linked with.

Linux/Unix

The *libglg_map.a* library supplied with the GLG Map Server contains all functions in this guide. The Map Server uses the GLG Standard API and depends on the GLG library.

The Map Server also needs the following additional libraries: *libfreetype*¹, *libtiff*², *libjpeg*³ and *libpng*⁴. The exact linking command depends on the version of the GLG library (GTK or legacy X11) being used, as some of the additional libraries may have already been pulled in by the GTK libraries.

The makefiles in the *src* directory of the GLG installation may be used for linking. Separate makefiles are provided for linking with either version of the GLG library. To link with the Map Server, uncomment the *GLG_MAP_LIBS* line that used the map server library in the selected makefile. Refer to the *Makefiles for Linux/Unix* section on page 45 of the *GLG Programming Reference Manual* for more information.

If an application uses the Map Server but does not need the TrueType font support in the Map Server, the *libfreetype* library may be replaced with *libfreetype_stub*.

Windows

On Windows, the Map Server library is included into the GLG library. To use Map Server, simply link with one of the GLG API libraries: Standard, Intermediate or Extended.

Using Static Libraries

On Windows, you can choose whether to use a static or dynamic library. Two versions of static libraries are provided: one compiled with the *multi-threaded DLL (/MD)* option, and another compiled with the *multi-threaded (/MT)* option. The libraries compiled with the */MT* option use the MT suffix to differentiate them from the libraries compiled with */MD*, for example: *GlgLibMT.lib* vs. *GlgLib.lib*. A version of the library matching the compilation options of your code should be used for linking with the GLG static libraries.

The static versions of the GLG Standard Library are called *GlgLib.lib* and *GlgLibMT.lib*.

The static versions of the GLG Intermediate Library are called *GlgInLib.lib* and *GlgInLibMT.lib*. They also require linking with the corresponding version of the GLG Standard Library: *GlgLib.lib* or *GlgLibMT.lib*.

The static versions of the GLG Extended Library are called *GlgExLib.lib* and *GlgExLibMT.lib*. They also require linking with the corresponding version of the GLG Standard Library: *GlgLib.lib* or *GlgLibMT.lib*.

1. Portions of this software are copyright © 2022 The FreeType Project (www.freetype.org). All rights reserved.

2. Copyright © 1988-1997 Sam Leffler. Copyright © 1991-1997 Silicon Graphics, Inc.

3. Copyright © 1991-2016, Thomas G. Lane, Guido Vollbeding.

4. Copyright © 2000-2002, 2004, 2006-2018 Glenn Randers-Pehrson.

The project files supplied with the Toolkit link with the GLG DLLs by default. In order to use the static version of the GLG libraries, define `GLG_STATIC` in your code:

```
#define GLG_STATIC
```

This must appear before the *GlgApi.h* and *GlmApi.h* files, or it may be specified as a preprocessor definition in the project's settings.

Using DLLs

The DLL version of the GLG Standard API Library is called *Glg.dll*, and the DLL version of the Extended and Intermediate APIs are called *GlgIn.dll* and *GlgEx.dll*. To use them, link your program with *Glg.lib*, *GlgIn.lib* or *GlgEx.lib*.

The GLG Intermediate and Extended API DLLs include the GLG Standard API. This means you don't have to link with the *Glg.dll* if using *GlgEx.dll* or *GlgIn.dll*.

Include Files

In order to use the Map Server library, both *GlgApi.h* and *GlmApi.h* need to be included. They reside in the *glg/include* directory. See page 104 for an example.

4.4 Advanced features of the GLG Map Server API

Overview

Occasionally, simply creating and manipulating a map object is not sufficient. For example, a developer might want to change the color of roads rendered into a map image. In order to accomplish this, resources of layers and datasets need to be set. In the Map Server, datasets and layers are also *GlmObjects*. By manipulating these objects and setting their resources many custom effects can be accomplished.

Hierarchy of *GlmObjects*

All datasets are held in an array called *GlmDatasetArray*. Each dataset holds layer objects. Map Objects are entirely separate. Map Objects require that a dataset be associated with them. This is accomplished by setting the "Dataset" resource, an S resource, of the map object. By default, this is not necessary, but if multiple datasets are to be created, each Map Object needs to know which dataset it will be using.

For example, if there are three datasets, and three different map objects:

```
GlmSetSResource( map1, "Dataset", "Earth1" );
GlmSetSResource( map2, "Dataset", "Earth2" );
GlmSetSResource( map3, "Dataset", "Earth3" );
```

Modifying Layers

If only one dataset is being used, using NULL as the object in `GlmGetResource` and `GlmSetResource` calls will set resources of layers. Layers are accessed by name, so for example:

```
GlmSetDResource( NULL, "clouds/Alpha", 0.5 );
```

will change the transparency of the clouds layer in the default dataset. If multiple datasets are used, the dataset object returned by `GlmCreateDataSet` can be used. For example:

```
data_obj = GlmCreateDataset( "mydata", "mydata.sdf" );
GlmSetDResource( data_obj, "clouds/Alpha", 0.5 );
```

See the *Creating Datasets* section on page 108 for details.

If the GLG Extended API is used, the dataset and layer objects can be modified directly. For example, the following code will retrieve a dataset named *mydata*, retrieve from it a layer called *clouds*, and set its transparency.

```
GlmObject layer;
layer = GlgGetResourceObject( GlmDatasetArray, "mydata/clouds" );
GlmSetDResource( layer, "Alpha", 0.5 );
```

Creating Datasets

If more than one dataset is desired, the `GlmCreateDataset` function can be used. The function returns a *GlmObject* which has already been added to the dataset array. Its parameters are the name of the dataset and the SDF to load it from. For example, the following code creates a dataset called *mydata* and sets it as the dataset to use for a given map object:

```
GlmObject map;
GlmCreateDataset( "mydata", "mydata.sdf" );
map = GlmCreateMap();
GlmSetSResource( map, "Dataset", "mydata" );
```

Dataset Resources

The dataset resources are almost identical to the attributes available in SDF files but instead may be accessed through a programmatic interface. Dataset resources must be set prior to generation of a map image in which they will be used. Refer to the *Datasets* chapter on page 32 for more information about each resource.

The resource type is listed after each resource name and may be *D* (double), *S* (string) or *G* (RGB color triplet or XYZ coordinate).

The following lists dataset resources that can be accessed programmatically.

GlmRoot S

Defines the root path of the dataset for the *GlmAllLayer* function.

LayerPath S

Defines the layer path used to locate LIF files for the *GlmAllLayer* function.

FontPath S

Defines the font path used to locate font files. This resource is read-only.

Encoding D

Specifies the encoding used for the text strings, may be either *GLM_LATIN_ENCODING* or *GLM_UNICODE_ENCODING* (default).

AllowOverride D

Defines the default setting that allows or disallows the use of the “++” override in the layer string. This default setting is inherited by layers created using the *GlmAllLayer* function and may have one of the following values:

GLM_DISABLE_OVERRIDE
GLM_OVERRIDE_UNTILED (default)
GLM_OVERRIDE_ALL

The *AllowOverride* resource of a layer may be used to override this global setting on a per-layer basis.

ZoomFactorType D

Defines how zoom factor is calculated, may be either *GLM_EXTENT_ONLY* (default) or *GLM_EXTENT_AND_WIDTH*.

BaseWidth D

Defines the base width for the *GLM_EXTENT_AND_WIDTH* zoom factor type (default 600).

MaxImageTiles D

Specifies the maximum number of cached image tiles for all layers (default -1).

MaxVectorTiles D

Specifies the maximum number of cached vector tiles for all layers (default -1).

MaxIterations D

Defines the maximum number of internal iterations used to render filled polygons. The greater of this value and the value of the *GlmMaxIterations* global configuration resource (default 10000) will be used for rendering the map.

GlyphCacheType D

Specifies the type of the glyph cache to use and may be set to *GLM_NO_GLYPH_CACHE* or *GLM_GLOBAL_GLYPH_CACHE* (default). The cache is used to increase performance of TrueType font rendering.

DefaultFont S

Specifies the default font for the dataset (default *roboto*).

ErrorFont S

Specifies the font for rendering error messages (default *roboto*).

Layer Resources

Layers, just like datasets and maps, are also *GlmObjects*. What follows is a description of all resources that can be set in a layer. The resources available are almost identical to the attributes available in LIF files but instead may be accessed through a programmatic interface. Layer resources must be set prior to generation of a map image in which they will be used. Refer to the *Configuration Variables* chapter on page 38 for more information about each resource.

The resource type is listed after each resource name and may be *D* (double), *S* (string) or *G* (RGB color triplet or XYZ coordinate). *D* resources that turn a layer feature ON or OFF, such as *IsDefault*, may be set to 0 to disable or to 1 to enable the corresponding functionality. When a default value is not listed, it is 0 for *D* resources, (0,0,0) for *G* resources and NULL for *S* resources. For *D* resources that define a non-negative number, such as *MaxTiles* or *MinZoom*, the value of -1 may be used to indicate an unset state.

The following lists layer resources that can be accessed programmatically.

LayerType D

Specifies the layer type, may be one of the following:

GLM_IMAGE (default)
GLM_VECTOR
GLM_GRID
GLM_OUTLINE
GLM_BACKGROUND

This resource is set when the layer is created and should not be changed afterwards.

IsDefault D

Defines whether or not a layer will be turned on by default (default 0).

TransType D

Defines what blending function is to be used when rendering the layer. This variable is very similar to the use of blending functions in OpenGL. It has four supported values:

GLM_OPAQUE (default)
GLM_ALPHA_WHITE_OPAQUE
GLM_ALPHA_BLACK
GLM_TRANSPARENT_COLOR_OPAQUE
GLM_XOR
GLM_BUMP_MAP

See the *Structure of Data and File Layout* chapter on page 27 for details.

TransColor G

Defines the transparent color value for the *GLM_TRANSPARENT_COLOR_OPAQUE* rendering mode.

TransColorPrecision D

Defines the transparent color matching precision in the range from 0 to 1 for the *GLM_TRANSPARENT_COLOR_OPAQUE* rendering mode.

TilesOverlap D

May be set to 1 to indicate that tiles can overlap. Refer to the *TILES OVERLAP* layer parameter on page 45 for more information.

BGColorRed D

BGColorGreen D

BGColorBlue D

Specifies R, G and B components (in 0-1 range) of the background color to use for rendering pixels for which no data is provided by the current layer, such as transparent areas in the middle of the ocean or the areas outside of the current layer data coverage.

Background Layer S

Specifies a name of a layer that will be used for rendering pixels outside of the data coverage of the current layer in the absence of a background color.

ElevationMode D

Specifies the elevation mode for the layer:

GLM_ELEVATION_NONE
GLM_ELEVATION_DATA
GLM_ELEVATION_COLOR

The *GLM_ELEVATION_COLOR* mode is used to display elevation data as a color-coded image. The *GLM_ELEVATION_DATA* mode is used for elevation queries.

Alpha D

Defines the transparency of the layer, ranging from 0 to 1. An alpha value of 0 will make the layer invisible, while an alpha value of 1 will make the layer fully opaque according to the blending function.

RelativeZoom D

Specifies whether or not to use the relative zoom for this layer.

ImageAntiAliasing D

For raster layers, enables or disables antialiasing when scaling images (default 1).

MaxTiles D

Defines the maximum number of tiles a layer can require before the fallback is activated.

MaxCacheSize D

Defines the maximum number of cached tiles in the layer tile cache (default 100).

MinPixelSize D

Specifies the minimal pixel size for map decluttering. All polygons whose pixel extent at the current zoom level is smaller than this parameter will not be drawn. The default value of the parameter is 1.

Clip D

Enables or disables clipping of layer's features (default 1).

LockTimeout D

Defines the maximum time (in seconds) the Map Server waits for the lock file before proceeding. The default value is 0.05 seconds.

Width D

Defines the width of the image data in pixels if the layer is one which uses raster image data.

Height D

Defines the height of the image data in pixels if the layer is one which uses raster image data.

MinLon D

Defines the minimum value of the longitude in the layer.

MinLat D

Defines the minimum value of the latitude in the layer.

MaxLon D

Defines the maximum value of the longitude in the layer.

MaxLat D

Defines the maximum value of the latitude in the layer.

MinZoom D

Defines the minimum zoom factor at which the layer should be rendered.

MaxZoom D

Defines the maximum zoom factor at which the layer should be rendered.

ClipType D

Defines the clip type to use for clipping the layer output and may be set to one of the following values:

GLM_CLIP_NONE, (default)
GLM_CLIP_BOX,
GLM_CLIP_POLYGON_PIXELS,
GLM_CLIP_ALL_PIXELS

*ClipMinLon D**ClipMinLat D**ClipMaxLon D**ClipMaxLat D*

Defines clip box for limiting the layer output.

GridLatInterval D

Defines the interval between neighboring horizontal (of equal latitude) lines (default 10). This variable is only useful if the type of the layer is set to *GLM_GRID*.

GridLonInterval D

Defines the interval between neighboring vertical (of equal longitude) lines (default 20).

GridMinLon D

Defines the minimum longitudinal value of grid lines in a grid layer.

GridMinLat D

Defines the minimum latitudinal value of grid lines in a grid layer.

GridMaxLon D

Defines the maximum longitudinal value of grid lines in a grid layer.

GridMaxLat D

Defines the maximum latitudinal value of grid lines in a grid layer.

AdaptiveGrid D

Defines an adaptive grid number. The Map Server will attempt to divide the extent of the map image into roughly the number of grid lines specified by the variable's value, while placing the grid lines on round numbers. Default is 0 (adaptive grid disabled).

GridLabels D

Defines whether or not to render labels next to the grid lines of the grid layer. A value of 0 will not render grid labels, value of 1 (default) will render labels on one side of the map, and the value of 2 will render them on both sides, except for the special cases of the *ORTHOGRAPHIC* projection (when one of the poles is visible, or the whole unzoomed globe is displayed), which will render labels at the center.

The values of 3 and 4 are used for a special case when a map in the *ORTHOGRAPHIC* projection is clipped to generate zoomed views of the globe with a visible horizon line. The value of 3 will display grid labels on three sides of the map (left, right and bottom), and the value of 4 will display labels on all four sides.

DecimalGridLabels D

Defines the format for displaying the fractional part of the grid labels: decimals are used if set to 1 (default), minutes and seconds are used if set to 0.

GridLabelFormat S

Specifies a custom format for the grid labels. Refer to the *GRID LABEL FORMAT=<format>* section on page 48 for more information.

GridLabelFormatLon S

Specifies a custom format for the longitudinal grid labels if they should have a different format than the latitudinal grid labels.

GridLabelChar D

Controls the use of the directional character (*N*, *S*, *E* or *W*) added to the label to indicate direction, may have the following values:

GLM_NO_CHAR
GLM_APPEND_CHAR (default)
GLM_PREPEND_CHAR

RotateLonLabels D

If set to 1 (default), the longitudinal grid labels will be rotated and drawn along the vertical grid lines. If set to 0, the labels will be drawn horizontally.

LonLabelAngle D

Defines the rotation angle for the longitudinal grid labels, 90 degrees by default.

TextFont S

Defines the name of the font to be used for text rendered from this layer.

TextType D

Defines how the font will be rendered and how font units will be defined. This variable has three supported values:

GLM_FLAT
GLM_MAPPED_FLAT (default)
GLM_TRANSFORMED

See the *TEXT TYPE=<type>* section on page 49 for details.

FontScale D

Defines a multiplier for font units, which defines the size of the text.

TextAngle D

Defines the angle by which the rendered text should be rotated.

TextAnchor D

A bitwise OR of the horizontal and vertical anchoring types for rendering text labels.

Horizontal types:

GLM_HLEFT (default)
GLM_HRIGHT
GLM_HCENTER

Vertical types:

GLM_VTOP (default)
GLM_VBOTTOM
GLM_VCENTER

LabelFormat S

Defines what text string is to be rendered alongside the marker.

LabelFormat2 S

Defines an alternative label format to be used if the marker attribute used in *LabelFormat* is missing for some objects.

LabelStyle D

Specifies optional rendering attributes for text labels. Must be a bitwise OR of all desired options. Supported options are:

GLM_TEXT_BG_EDGE
GLM_TEXT_BG_FILL
GLM_TEXT_OUTLINE

LabelOutlineColor G

Defines the color of the outline drawn around text labels (default (0.7,0.7,0.7)).

TextBoxLineWidth D

Defines the line width in pixels of the text box drawn around text labels (default 1). Text box drawing is enabled by setting *LABEL STYLE* to include *BG_EDGE* and/or *BG_FILL*.

TextBoxOffsetH D

Defines the horizontal pixel margin between the label and its text box, if enabled (default 3).

TextBoxOffsetV D

Defines the vertical pixel margin between the label and its text box, if enabled (default 1).

WrapLabelLength

Defines the maximum length of a label line in pixels. If this length is exceeded, the label text will be wrapped by splitting it into several lines.

*LabelMinZoom D**LabelMaxZoom D*

Defines the range of zoom factor for which marker labels (such as city names) and polygon labels (such as street names) are displayed.

LayoutType D

Defines the type of the label layout negotiation and may have the following values:

GLM_LAYOUT_NONE
GLM_LAYOUT_LAYER
GLM_LAYOUT_GLOBAL

LabelPriority D

Defines label priority for all labels in the layer (default 9). The labels with priority value of 0 have the highest priority. A threshold table may be attached to label priority to vary priority of labels within one layer based on other object attributes, such as a road type or city population. While the priority values may be set to any number greater or equal to 0, using priority values with minimal gaps between them results in better performance.

LabelRepeatDistance D

Defines a minimum distance in pixels between labels with the same label string (default 0). It may be used to enhance label rendering for named streets and roads that are split in multiple segments.

LayoutMargin D

Defines an additional margin in pixels between text labels (default 1). Using a bigger value will result in more space between individual text labels when the label layout negotiation is enabled.

LayoutMarkers D

Specifies the way the layout negotiation handles markers. If set to 1 (default), the markers will be handled together with their labels, and will not be drawn if its label is not drawn. If set to 0, the markers will always be drawn, while their labels will be drawn only if not obscured by other labels with a higher priority.

MarkerIconFile S

Specifies an image file containing a custom icon for rendering markers.

MarkerType D

Defines what type of marker to render at the location defined by the text object's control point. Must be a bitwise OR of all desired types. Possible types:

GLM_CROSS
GLM_BOX (default)
GLM_FILLED_BOX
GLM_CIRCLE
GLM_FILLED_CIRCLE
GLM_DOT
GLM_CUSTOM_ICON

MarkerSize D

Defines the marker size in pixels (default 5).

TextOffsetH D

Defines the horizontal pixel distance between the marker and its label (default 1).

TextOffsetV D

Defines the vertical pixel distance between the marker and its label (default 1).

FillType D

Defines the fill type of polygons for vector layers. Valid values are:

GLM_EDGE (default)
GLM_FILL
GLM_FILL_AND_EDGE

LineWidth D

Defines the line width in pixels of polygons, grid lines and outline, analogous to the line width option in the GLG Toolkit. The default value is 1 pixel.

FillColor G

Defines the fill color of polygons that are rendered if there are any areas to fill, as well as the fill color of the outline and background layers (default (0.7,0.7,0.7)).

EdgeColor G

Defines the edge color of polygons, grid lines and outline.

DrawCenterMarker D

If set to 1, a marker with or without a label will be drawn at the polygon center (default 0).

MarkerFillColor G

Defines the fill color of markers in a vector layer (default (0.7,0.7,0.7)).

MarkerEdgeColor G

Defines the edge color of markers in a vector layer (default (0.7,0.7,0.7)).

LabelColor G

Defines the color of text labels.

BoxEdgeColor G

Defines the edge color of the text box.

BoxFillColor G

Defines the fill color of the text box (default (0.7,0.7,0.7)).

GradientColor G

Defines the gradient color of the background layer.

GradientType D

Defines the type of the gradient to be rendered for the background layer. Valid values are *GLM_NO_GRADIENT* and *GLM_AROUND_GLOBE_GRADIENT*.

GradientLength D

Defines the length of the gradient of the background layer (default 1). For the *GLM_AROUND_GLOBE_GRADIENT* gradient type, the length is defined in relative units, with 0.3 corresponding to the 30 percent of the globe radius. The gradient starts the edge of the globe and extends outwards.

PolyLabelFormat S

Defines the format and attributes to be displayed in a polygon label.

PolyLabelFormat2 S

Defines an alternative label format to be used if the polygon attribute used in *PolyLabelFormat* is missing for some objects.

PolyLabelType D

Defines the type of polygon label to render. Valid values are:

GLM_LINE_LABEL (default)
GLM_MID_LINE_LABEL
GLM_AREA_LABEL
GLM_BBOX_LABEL

PolyLabelCenter D

Defines which attribute to use as the center of the label on a polygon (default -1).

LabelSelectionMode D

Specifies the layer's label selection mode for the GIS selection. Valid values are:

GLM_LBL_SEL_NONE
GLM_LBL_SEL_IN_TILE_PRECISION,
GLM_LBL_SEL_MAX_PRECISION,
GLM_LBL_SEL_UNSET (default)

If set to *GLM_LBL_SEL_UNSET*, the label selection mode is controlled by either the *SELECT_LABELS* parameter of the map request or the *select_labels* parameter of the *GlmGetSelection* method.

Volatile D

Defines layer's data as volatile if set to 1 (default value is 0). Volatile data are checked before they are used and re-read if the data have changed since the last time they were used. This resource is set when the layer is created and must not be changed afterwards.

VolatileErrors D

Suppresses correctable errors for volatile data if set to 0 (default value is 1).

LockFile S

Specifies the name of a lock file for volatile layers.

LockTimeout D

Specifies the lock timeout in seconds. Default value is 0.05.

RedirectFile S

Specifies the name of the redirect file for volatile layers. This resource is set when the layer is created and must not be changed afterwards.

RedirectPrefix S

Specifies the redirect prefix for volatile layers. This resource is read from the redirect file and must not be changed.

FallbackLayer S

Specifies the name of a fallback layer to switch to when the maximum number of tiles is exceeded. This resource is set when the layer is created and must not be changed afterwards.

AllowMissingTiles D

Suppresses errors for missing tiles, which may not be an error in case of square tiling of vector datasets. When tiling such datasets, the tiles that do not contain any data are not written out. It may also be used with image layers that use square tiling, but some of the tiles are not provided.

AllowOverride D

May be used for a per-layer override of the global *AllowOverride* value defined in the dataset by setting it to 0 or 1. The default value is inherited from the dataset.

4.5 Function Descriptions

GlmInit

```
void GlmInit( dataset )
    char * dataset;
```

*Parameters***dataset**

specifies the name of the Server Dataset File.

Initializes the Map Server. Must be called before any other GLM function calls

GlmCreateDataset

Loads an SDF file and creates a dataset.

```
GlmObject GlmCreateDataset( dataset_name, filename )
char * dataset_name;
char * filename;
```


Parameters

dataset_name

The name for the created dataset.

filename

The SDF file that contains the dataset's information. If filename is NULL, an empty dataset with no layers is created.

This function creates a new dataset object and adds to it layers defined in the SDF file. The function returns the dataset object it created. The dataset may later be accessed using either its name or object ID.

GlmAddLayer

Loads a LIF file and creates a layer.

```
GlmObject GlmAddLayer( dataset_name, layer_name, filename )
char * dataset_name;
char * layer_name;
char * filename;
```

Parameters

dataset_name

The name of the dataset to add the new layer to.

layer_name

The name for the created layer.

filename

The LIF file that contains the layer's information, must not be NULL.

This function reads a LIF file and creates a new layer object for it, adding the layer object to the specified dataset. The function returns the layer object that was created. The layer may later be accessed using either its name or object ID.

GlmCreateMap

Returns a newly created map object.

```
GlmObject GlmCreateMap( void )
```

This function creates a new map object which may be used to generate map images. It also holds attributes of map requests.

GlmSetupMap

Sets up the map object after changing its *DatasetName* resource.

```
GlmBoolean GlmSetupMap( map )
    GlmObject map;
```

Parameters

map

The map object to set up.

This function must be called before selecting layers if the map's *DatasetName* resource was changed. The function returns FALSE if errors were encountered.

GlmSetLayers

Enables the specified layers in the map object

```
void GlmSetLayers( map, layers )
    GlmObject map;
    char * layers;
```

Parameters

map

The map object in which to enable layers

layers

A comma separated string of layers to enable.

This function enables all the layers and aliases specified in the layers string. All other layers are disabled. If “default” is used as a layer name, all layers that are ON by default will be enabled. If “*” is used as a layer name, all layers will be enabled. If a layer name starts with ‘-’, the layer is disabled. This may be used to disable some layers enabled by the “default” or “*” layer string values. If individual layers are listed, their names should match those in the SDF file. The layers will be displayed in the order of their appearance in the layer string. For layers enabled by the “default” or “*” values of the layer string, their display order is defined by the order in which the layers are listed in the SDF file. The “++” prefix may be used to override zoom and number of tiles limits defined in the layer file. The attribute conditions may be appended to layer names. Refer to the description of the LAYERS string on page 69 for more information.

GlmResetLayers

Resets any previously selected layers in the map object.

```
void GlmResetLayers( map )
    GlmObject map;
```

Parameters

map

The map object in which to reset layer selection.

When the map object is used repeatedly, this function may be used to reset any previous layer selections before *GlmSetLayers* function is invoked.

GlmEnableLayer

Enables one layer.

```
void GlmEnableLayer( map, layer, override )
    GlmObjet map;
    char * layer;
    GlgBoolean override;
```

Parameters

map

Specifies the map object.

layer

Specifies the name of the layer to enable.

override

If set to *True*, forces the layer to be visible regardless of its MIN ZOOM, MAX ZOOM, MAX TILES and other visibility thresholds.

This function enables one layer. the layer name can be *default* or “*”. If the layer does not exist, an error is generated. If *default* is used a layer name, all layers which have the *DEFAULT ON* flag defined will be turned on. if “*” is used as a layer name, all layers in the SDF file will be turned on.

GlmDisableLayer

Disables one layer.

```
void GlmDisableLayer( map, layer )
    GlmObjet map;
    char * layer;
```

Parameters

map

Specifies the map object.

layer

Specifies the name of the layer to disable.

This function disables one layer. the layer name can be “default” or “*”. If the layer does not exist, an error is generated. If “default” is used a layer name, all layers which have the *DEFAULT ON* flag defined will be turned off. if “*” is used as a layer name, all layers in the SDF file will be turned off.

GlmGetDResource

Returns the current value of a scalar resource.

```
GlgBoolean GlmGetDResource( object, resource_name, d_value_ptr )
    GlmObject object;
    char * resource_name;
    double * d_value_ptr;
```

Parameters

object

Specifies a GLM Object

resource_name

Specifies the name of the double resource to query

d_value_ptr

Specifies a pointer for returning the resource value

If a scalar resource the input name exists, this function copies its current value into the address specified by *d_value_ptr* and returns *GlgTrue*, otherwise it generates an error message and return *GlgFalse*.

GlmGetGResource

Returns the set of three values making up a geometrical resource:

```
GlgBoolean GlmGetGResource( object, resource_name,
    g_value1, g_value2, g_value3 )

    GlmObject object;
    char * resource_name;
    double * g_value1_ptr, * g_value2_ptr, * gvalue3_ptr;
```

Parameters

object

Specifies a GLM object.

resource_name

Specifies the name of the geometrical resource to query.

g_value0_ptr, g_value1_ptr, g_value2_ptr

Specify pointers to the locations to which the XYZ or RGB values of the resource are returned.

If a geometrical resource with the input name exists, this function copies its current three values to the addresses specified with the *g_value* pointers and returns *GlgTrue*. Otherwise it generates an error message and returns *GlgFalse*.

For a geometrical point, *g_value0_ptr* , *g_value1_ptr* and *g_value2_ptr* are set to the X, Y and Z coordinates of the point, respectively. For a color resource, they will be set to the R, G and B values of the color, respectively.

GlmGetSResource

Returns the value of a string resource.

```
GlgBoolean GlmGetSResource( object, resource_name, s_value_ptr )
    GlmObject object;
    char * resource_name;
    char ** s_value_ptr;
```

Parameters

object

Specifies a GLM object.

resource_name

Specifies the name of the string resource to query.

s_value_ptr

A pointer for returning the resource value.

If a string resource with the given name exists, this function sets the `s_value_ptr` to point to the internal string value and returns *GlgTrue*. Otherwise it returns *GlgFalse*.

Warning: The returned pointer points to GLG internal data structures and should not be modified. The pointer is valid only immediately after a call to the *GlmGetSResource* function. To store the returned string, create a copy of it using the *GlgStrClone* function.

GlmSetDResource

Sets a new value for a resource of type D.

```
GlgBoolean GlmSetDResource( object, resource_name, d_value )
    GlmObject object;
    char * resource_name;
    double d_value;
```

Parameters

object

Specifies a GLM object.

resource_name

Specifies the name of the scalar resource to be set.

d_value

Specifies a new value for the resource.

If a scalar resource with the given name exists, this function sets it to a new value and returns *GlgTrue*, otherwise it prints an error message and returns *GlgFalse*.

GlmSetGResource

Sets the values of a geometrical resource.

```
GlgBoolean GlmSetGResource( object, resource_name,
                           g_value1, g_value2, g_value3 )

GlgObject object;
char * resource_name;
double g_value1, g_value2, g_value3;
```

Parameters

object

Specifies a GLM object.

resource_name

Specifies the name of the geometrical resource to be set.

g_value0, g_value1, g_value2

Specify the new XYZ or RGB values for the resource.

If a geometrical resource with the given name exists, this function sets it to the input values and returns *GlgTrue*. Otherwise, the function prints an error message and returns *GlgFalse*.

GlmSetSResource

Replaces the string in a string resource.

```
GlgBoolean GlmSetSResource( object, resource_name, s_value )
GlmObject object;
char * resource_name;
char * s_value;
```

Parameters

object

Specifies a GLM object.

resource_name

Specifies the name of the string resource to be set.

s_value

Specifies the new string for the resource.

If a string resource with the given name exists, this function creates a copy of the input string, and sets that copy as the new value of the resource. The function returns *GlgTrue* if the resource has been successfully changed, otherwise it generates an error message and returns *GlgFalse*. The memory associated with the old string is automatically freed.

GlmGetProjection

Generates a map image and stores it in the map object.

```
GlgLong GlmGetProjection( map )
GlmObject map;
```

Parameters

map

Specifies the map object from which to generate a map image.

This function generates a map image with all information present in the map object. The function returns -1 if the map generation failed, 0 if the map was generated with no errors, and 1 if the map was generated with errors. Setting any resources after the map images is generated will not affect the image.

GlmGetMapData

Returns a pointer to the raw data of the generated image.

```
unsigned char * GlmGetMapData( map, size, reset_data )
GlmObject map;
GlgLong * size;
GlgBoolean reset_data;
```

Parameters

map

Specifies the map object containing the image.

size

Returns the byte size of the image data.

reset_data

Set to *True* if the returned data will be freed by the code using *GlgFree*. If set to *False*, the map server will free the data automatically.

This function returns a pointer to the map image generated by the last *GlmGetProjection* call. The data represent tightly packed RGB triplets that use 24 bits per pixel; the first byte of each triplet representing red, the second green and the third blue component. If *reset_data* is set to *True*, the returned pointer must be freed with *GlgFree* when finished.

GlmWriteImage

Writes the image in the map object to a file.

```
char * GlmWriteImage( file, filename, map )
FILE * file;
char * filename;
GlmObject map;
```

*Parameters***file**

Specifies a file descriptor to write to.

filename

Specifies a filename to write to.

map

Specifies the map object that contains the generated map image

This function writes the generated map image to either the file descriptor, or, if file is NULL, opens a file with the name specified by filename, and writes the generated image. The written file will be in the JPEG file format.

GlmGetElevation

Returns elevation of the specified lat/lon point.

```
GlgBoolean GlmGetElevation( map, layer_name, lat, lon, elevation )
    GlmObject map;
    char * layer_name;
    double lat, lon
    double * elevation;
```

*Parameters***map**

Specifies the map object used to generate a map image. The map object has to be in an updated state. If the map object has been just created or changed since the last call to the *GlmGetProjection* or *GlmUpdateMap* functions, it has to be updated using the *GlmUpdateMap* function prior to invoking *GlmGetLatLon*.

layer_name

Specifies the name of the elevation layer to query. The layer must have *ELEVATION MODE=DATA* setting in the LIF file.

lat, lon

Specifies the lat/lon coordinates of a point on the map. Use *GlmGetLatLon* function to convert screen X/Y coordinates to lat/lon if necessary.

elevation

Specifies a pointer to double variable for returning elevation value.

This function returns elevation data in the units used in the data file. The function returns *GlgFalse* if the layer does not provide elevation coverage for the requested lat/lon location.

GlmGetSelection

Returns a message object containing information about the GIS features located at the specified position on the map.

```
GlgObject GlmGetSelection( map, x, y, select_labels )
    GlmObject map;
    double x, y
    GlgLong select_labels;
```

Parameters

map

Specifies the map object to generate a selection for. The map object has to be in an updated state. If the map object has been just created or changed since the last call to the *GlmGetProjection* or *GlmUpdateMap* function, it has to be updated using the *GlmUpdateMap* function prior to invoking *GlmGetSelection*.

x, y

Specifies the X and Y screen coordinates of a point on the map.

select_labels

Specifies the label selection mode, can have the following values (defined in the *GlmLabelSelectionMode* enum in the *GlmApi.h* file):

- GLM_LBL_SEL_NONE - GIS features' labels are not considered for the GIS selection.
- GLM_LBL_SEL_IN_TILE_PRECISION - enables labels to be considered for the GIS selection. For a faster selection, this option does not check labels that belong to GIS features in a different tile but extend to the current tile.
- GLM_LBL_SEL_MAX_PRECISION - enables labels to be considered for the GIS selection and performs selection with the maximum label precision.

The layer's settings have precedence over the *select_labels* parameter if *LABEL SELECTION MODE* is specified in the layer's LIF file and is set to a value different than *UNSET*.

The *NONE* setting disables label selection for the layer. The *INTILE* setting performs label selection using the in-tile precision, and the *YES* setting will always perform label selection with the maximum precision.

The *GlgGISPickResolution* global configuration resource is set to 2 by default and controls selection precision by defining a pixel radius of an area around the selection point. All objects intersecting that area will be reported in the GIS selection.

The function returns a message object containing information about the selected GIS features as described below. The message object has to be dereferenced using the *GlgDropObject* function when finished.

If the verbosity level is set to 2000 or 2001, extended selection information is written into the *glg_error.log* file and (on Linux/Unix) is also printed to the terminal for debugging purposes.

GIS Selection Message

The GIS selection message has the following structure:

GIS selection message object : GLG array containing all selected layers

Layer : GLG array containing all selected features of one layer

Name : S resource (stored as the name of the Layer object), contains the layer name

Selection : GLG array named "*Selection*", contains information about one selected GIS feature

SelectionType : S resource, contains a selection type:
 "*object*" - when the feature was selected
 "*label*" - when only the feature's label was selected

Feature : GLG array named "*Feature*", contains information about the selected GIS feature (polygon or marker)

FeatureType : S resource, contains the type of the selected GIS feature:
 "*polygon*" or "*marker*"

FeatureID : S resource, contains volatile internal map server feature ID inside the tile. The ID may change if the tile is reloaded.

Location : G resource, contains the lat/lon of the marker feature or the lat/lon of the center of the bounding box of the polygon feature
 (x=lon, y=lat)

Attributes : GLG array named "*Attributes*", contains all attributes of the selected GIS feature. May not be present if the feature does not have any attributes.

Attribute : D, S or G resource matching the type of the corresponding attribute. It is named using the attribute index ("*0*", "*1*", etc.) and contains the value of the attribute.

Attribute : May repeat if the feature has several attributes

...

Selection : May repeat if several features are selected in one layer

...

Layer : May repeat if several layers are selected

...

All arrays in the message object have *HasResources*=YES, which makes it possible to query resources contained inside them via the *GlgGetResourceObject* method. For example, the following code uses the layer's name to check if the layer was selected:

```
if( GlgGetResourceObject( selection_message, layer_name ) )
    printf( "Layer %s was selected.", layer_name );
```

Programming Example

The following example prints the content of the returned GIS selection message.

```
void PrintSelection( GlgObject selection_message )
{
    int
        i, j, k,
        num_layers,
        num_selections,
        num_attributes;
    GlgObject
        layer_obj,
        selection_obj,
        feature_obj,
        attributes,
```

```
    attribute_obj;
char
    * layer_name,
    * selection_type,
    * feature_type,
    * attr_name;
double
    location_x, location_y, location_z,
    data_type;

if( selection_message )
    num_layers = GlgGetSize( selection_message );
else
    num_layers = 0;

if( !num_layers )
{
    printf( "No selection\n" );
    return;
}

printf( "Number of selected layers: %s\n", num_layers );
for( i=0; i<num_layers; ++i )
{
    layer_obj = GlgGetElement( selection_message, i );
    if( !layer_obj )
    {
        Error( "Null layer." );
        return;
    }

    num_selections = GlgGetSize( layer_obj );
    GlgGetSResource( layer_obj, "Name", &layer_name );

    printf( "Layer: %s\n", layer_name );
    printf( "Number of selected objects: %d\n", num_selections );

    for( j=0; j<num_selections; ++j )
    {
        selection_obj = GlgGetElement( layer_obj, j );
        if( !selection_obj )
        {
            Error( "Null selection." );
            return;
        }

        /* Query selection type: object or label */
        if( !GlgGetSResource( selection_obj, "SelectionType",
                               &selection_type ) )
            selection_type = "undefined";

        printf( "    SelectionType: %s\n", selection_type );

        feature_obj =
            GlgGetResourceObject( selection_obj, "Feature" );
```

```

if( feature_obj )
{
    printf( "    Feature\n" );

    /* Query feature type: polygon or marker */
    if( !GlgGetSResource( feature_obj, "FeatureType",
                        &feature_type ) )
        feature_type = "undefined";
    if( !GlgGetGResource( feature_obj, "Location",
                        &location_x, &location_y,
                        &location_z ) )
    {
        location_x = 0.;
        location_y = 0.;
        location_z = 0.;
    }

    printf( "        FeatureType: %s\n", feature_type );
    printf( "        Location: %.6lf %.6lf ",
            location_x, location_y );
    attributes =
        GlgGetResourceObject( feature_obj, "Attributes" );
    if( attributes )
        num_attributes = GlgGetSize( attributes );
    else
        num_attributes = 0;

    for( k=0; k<num_attributes; ++k )
    {
        attribute_obj = GlgGetElement( attributes, k );
        if( !attribute_obj )
        {
            Error( "Null attribute." );
            return;
        }

        GlgGetSResource( attribute_obj, "Name", attr_name );
        GlgGetDResource( attribute_obj, "DataType",
                        data_type );
        switch( (int)data_type )
        {
            case GLG_D:
                GlgGetDResource( attribute_obj, NULL, &dvalue );
                printf( "        D attribute, name: %s, value: %lf\n",
                        attr_name, dvalue );
                break;

            case GLG_S:
                GlgGetSResource( attribute_obj, NULL, &svalue );
                printf( "        S attribute, name: %s, value: %s\n",
                        attr_name, svalue );
                break;

            case GLG_G:
                GlgGetGResource( attribute_obj, NULL,

```

GlmGetLatLon

```
GlgBoolean GlmGetLatLon( map, x, y, lat_lon )
    GlmObject map;
    double x, y;
    GlgPoint * lat_lon;
```

map

 $\mathbf{x}, \mathbf{y}$

lat_lon

This function converts the X/Y image coordinates to the lat/lon coordinates using information in the map object. The image coordinate system has its origin in the upper left corner. The longitude is returned as the *x* field of the *GlgPoint* structure, and the latitude is returned as the *y* field. The function returns *GlgFalse* if the X/Y point is outside of the map image area, or outside of the globe area if the orthographic projection is used.

Converts lat/lon coordinates of a point to the X/Y screen coordinates of the map image.

```
void GlmGetLatLon( map, lat, lon, xyz )
    GlmObject map;
    double lat, lon;
    GlqPoint * xyz;
```

*Parameters***map**

Specifies the map object used to generate a map image. The map object has to be in an updated state. If the map object has been just created or changed since the last call to the *GlmGetProjection* or *GlmUpdateMap* functions, it has to be updated using the *GlmUpdateMap* function prior to invoking *GlmGetLatLon*.

lat, lon

Specifies the lat/lon coordinates of a point.

xyz

Specifies a pointer to the *GlgPoint* structure for returning X/Y coordinates.

This function converts the lat/lon coordinates to X/Y image coordinates using information in the map object. The image coordinate system has origin in the upper left corner. The z coordinate of the returned structure is set to a negative value if the point is outside of the visible area of the map when the rectangular projection is used. When the orthographic projection is used, the z coordinate is set to a negative value if the point is on the invisible part of the globe, and to zero when the point is on the edge of the globe. In the orthographic projection, the returned X and Y values may be outside of the image area and must be checked before they are used.

GlmConvert

A low level function that performs coordinate conversion between the GIS coordinates of a map and the X/Y coordinates of the drawing without the use of a map object.

```
void GlmConvert( projection, stretch, coord_type, coord_to_lat_lon,
                center, extent, angle,
                min_x, max_x, min_y, max_y, in_point, out_point )
GlgProjectionType projection;
GlgBoolean stretch;
GlgCoordType coord_type;
GlgBoolean coord_to_lat_lon;
GlgPoint * center, * extent;
double angle;
double min_x, max_x, min_y, max_y;
GlgPoint * in_point, * out_point;
```

*Parameters***projection**

A GIS map projection, may have values of GLG_RECTANGULAR or GLG_ORTHOGRAPHIC.

stretch

Specifies if the map preserves the X/Y ratio (GlgFalse) or not (GlgTrue).

coordinate_type

The type of the coordinate system to convert to or from: *GLG_SCREEN_COORD* for screen coordinates or *GLG_OBJECT_COORD* for the GLG world coordinates.

coord_to_lat_lon

GlgTrue to convert from X/Y coordinates to GIS longitude and latitude, *GlgFalse* to convert from lat/lon to X/Y coordinates.

center

A pointer to the *GlgPoint* structure containing lat/lon coordinates of the map center. The Z coordinate must be set to 0.

extent

A pointer to the *GlgPoint* structure containing X and Y extent of the map in the selected projection: in degrees for the rectangular projection or in meters for the orthographic projection. The Z extent must be set to 0.

angle

A map rotation angle in degrees.

min_x, max_x, min_y, max_y

A map image extent in the selected coordinate system. To get X/Y coordinates relative to the map image origin, use *min_x=0*, *min_y=0*, *max_x=image_width*, *max_y=image_height*.

in_point

A pointer to the *GlgPoint* structure containing coordinate values to be converted.

out_point

A pointer to the *GlgPoint* structure that receives converted coordinate values.

When converting from X/Y to lat/lon coordinates in the orthographic projection, the Z coordinate is set to a negative GLG_GIS_OUTSIDE_VALUE value for points on the invisible part of the globe or outside the globe area, and to zero for points on the edge of the globe. The coordinates of the returned point may be outside of the visible portion of the map for all projections.

GlmLatLonToUtm

Converts lat/lon coordinates to the UTM coordinates.

```
void GlmGetLatLon( lat, lon, utm_point)
    double lat;
    double lon;
    GlmUtmPoint * utm_point;
```

*Parameters***lat, lon**

Specifies the lat/lon coordinates.

utm_point

Specifies a pointer to the *GlmUtmPoint* structure for returning converted UTM coordinates:

```

struct
{
    GlgLong northing;
    GlgLong easting;
    GlgLong zone_number;
    char zone_letter;
} GlmUtmPoint;

```

GlmUtmToLatLon

Converts UTM coordinates the lat/lon coordinates.

```

void GlmUtmToLatLon( utm_point, lat, lon )
    GlmUtmPoint * utm_point;
    double * lat;
    double * lon;

```

Parameters

utm_point

Specifies a pointer to the *GlmUtmPoint* structure (described above) containing the UTM coordinates to be converted.

lat, lon

Specifies pointers to the variables that will receive converted lat/lon coordinates.

GlmUtmToMgrs

Converts UTM coordinates to a MGRS string.

```

char * GlmUtmToMgrs( utm_point, accuracy, mgrs_buf)
    GlmUtmPoint * utm_point;
    char * accuracy;
    char * mgrs_buf;

```

Parameters

utm_point

Specifies a pointer to the *GlmUtmPoint* structure (described above) containing the UTM coordinates to be converted.

accuracy

Specifies MGRS accuracy using values of the *GlgMGRSAccuracy* enum:

```

GLM_10000_METER
GLM_1000_METER
GLM_100_METER
GLM_10_METER
GLM_1_METER

```

mgrs_buf

Specifies a pointer to a character array the MGRS string will be written to. The array must be big enough to hold the converter MGRS string.

The function returns a pointer to the character array passed to it by the *mgrs_buf* parameter.

GlmMgrsToUtm

Converts a MGRS string to UTM coordinates.

```
void GlmMgrsToUtm( mgrs_buf, utm_point )
    char * mgrs_buf;
    GlmUtmPoint * utm_point;
```

Parameters

utm_point

Specifies a pointer to the *GlmUtmPoint* structure (described above) for returning converted UTM coordinates.

mgrs_buf

Specifies a pointer to a MGRS string to be converted.

On success, the function returns MGRS accuracy using values of the *GlmMGRSAccuracy* enum:

```
GLM_10000_METER
GLM_1000_METER
GLM_100_METER
GLM_10_METER
GLM_1_METER
```

On failure, the GLM_FAILED_ACCURACY enum value is returned.

GlmUpdateMap

Updates the map object in preparation for invoking *GlmGetElevation*, *GlmGetLatLon* and *GlmGetXYZ* functions.

```
GlgBoolean GlmUpdateMap( map )
    GlmObject map;
```

Parameters

map

Specifies the map object to update after it has been created or its parameters have been changed since the last call to the *GlmGetProjection* or *GlmUpdateMap* functions.

This function sets up internals of the map objects to prepare for a more efficient handling of multiple calls to elevation query and coordinate conversion functions. By encapsulating setup into a separate function, it may be performed just once per a series of queries, instead of doing it for each query. The function returns *GlgFalse* in case of errors in data setup.

GlmResetImageErrors

Resets any accumulated error messages that are about to be displayed in the generated map image:

```
void GlmResetImageErrors( void )
```

GlmRemoveLayer

Removes a layer from the dataset.

```
void GlmRemoveLayer( dataset_name, layer_name )
char * dataset_name;
char * layer_name;
```

Parameters

dataset_name

The name of the dataset that contains the layer.

layer_name

The name of the layer to be removed.

The function finds a named layer of the dataset, removes it from the dataset and destroys the layer object.

GlmDestroyMap

Destroys a map object

```
void GlmDestroyMap( map )
GlmObject map;
```

Parameters

map

Map object to destroy.

This function destroys the given map object. After it is destroyed, it may no longer be used.

GlmDestroyDataset

Destroys and frees the data in the given dataset.

```
void GlmDestroyDataset( dataset )
char * dataset;
```

Parameters

dataset

Name of the dataset to destroy.

This function destroys the given dataset and frees all data present in the fonts and layers therein. After destruction, the dataset may no longer be used.

GlmTerminate

Terminates usage of the Map Server.

```
void GlmTerminate( void )
```

This functions is called when the Map Server is no longer needed.

Chapter 5

GVF Filters and Data Converters

5

5.1 GVF API Reference

Overview

The GLM Vector Format (GVF) is an open format for describing vector and text data, and is used by the GLG Map Server. The ASCII version of the GVF format is portable across architectures of varying endianness as well as 32-bit and 64-bit architectures, while the binary version provides faster loading time.

If data used in the GLG Map Server is not in the GLM Vector Format, it must be converted. It can either be converted on the fly or converted and saved to a GVF file, which can then be used by the Map Server directly. When the data are converted on the fly, the program that performs the conversion is used as a filter by the Map Server. Converting just once and saving converted data into a file is more efficient, but filters may be used as an interface to external databases and data sources.

Some commonly used GIS data, such as Digital Chart of the World (DCW) and US Census Tiger Data, are available in the converted ASCII GVF format from Generic Logic. However, there exist many other custom vector data formats requiring conversion, and the Map Server provides convenience functions for writing a data file in GVF format. The convenience filter functions are provided in a form of the *Gvf.c* file (in the *map_server/src* directory) containing the source code of these functions. In order to use it, it must be included in any project that uses the functions. The *map_server/src* directory contains source code examples of a few commonly used converters.

GVF Functions

The first two parameters of GVF functions are common to all of them. The first parameter of all GVF API functions is a file descriptor of an already open file to use for writing GVF data. When the functions are used to write a filter used with the Map Server to convert data files on the fly, *stdout* must be used as the file parameter to write GVF data to the standard output. Furthermore, no other data may be written to the standard output by the filter. Most of the examples in this chapter use *stdout* as the first parameter for simplicity.

The second parameter to all GVF functions defines the GVF file format: *GVF_ASCII* or *GVF_BINARY*. The ASCII data are cross-platform, but binary data provide much faster load time. It is recommended to write all data in the ASCII mode, and then convert them to the binary mode using the Map Server's GVF converter utility on each of the target platforms.

Headers

The provided header file *Gvf.h* must be included in all filters:

```
#include "Gvf.h"
```

Writing a Filter or Data Convertor

Before any data can be written, the *GvfWriteHeader* function must be called to write the appropriate header to the output file. If this header is not written, the Map Server will not recognize the file as a GVF file and will not read it correctly:

```
GvfWriteHeader( stdout, GVF_ASCII );
```

After the header is written, objects can be written to the file. There are two types of objects, polygons and markers (text is a marker too). Before the data for each polygon or marker is written, it's header must be written. After the object's header is written, all attributes associated with that object must be written, if any exist. Finally, the actual point data can be written to the file.

Polygons

The polygon header is written via a call to the function *GvfWritePolygon*. The function must be passed whether the polygon has rings or not, how many points are in the polygon, and how many attributes it contains. The rings parameter is used for composite polygons. These are polygons in which adjacent points are not necessarily connected. For example, a lake with islands might be a composite polygon. If the polygon is composite, each section of the polygon is defined as a polygon itself. All but the last section will have the rings parameter set to 1, but the last section will be set to 0. Any non-composite polygon should have the rings parameter set to 0. For example:

```
GvfWritePolygon( stdout, GVF_ASCII, 0, 100, 3 );
```

will define a polygon that is non-composite, has 100 points and has 3 attributes. The attributes are written into the data file separately using the *GlmWriteAttribute* function, and the points are written last using the *GvfWritePoint* function.

Note: Attributes, as referred to here, are custom attributes associated with polygons. Color, linewidth and other graphical properties are defined in the layer.

Markers

The marker headers is written via a call to the function *GvfWriteMarker*. The function must passed the marker type, the scale, the angle, the anchor, the string and the number of attributes in the object. The type can have the following values:

GVF_M_MARKER

The simplest type, with all marker attributes specified in the LIF file. The marker's string label may be defined in the data file as a marker's attribute and rendered using the *LABEL FORMAT* LIF option.

GVF_M_LABEL

All attributes except the string (scale, angle and anchor) are specified in the LIF file, and the string specified with the marker object in the data file, as part of the marker.

GVF_M_TEXT

All attributes (scale, angle, anchor and string) are specified with the object in the data file.

Regardless of type, all markers can have a graphical marker and a label. The various marker types specify where different properties of the marker are defined. The position is always defined in the data file. In almost all cases, *GVF_M_MARKER* is desired.

For example:

```
GvfWriteMarker( stdout, GVF_ASCII, GVF_M_MARKER, 0., 0., 0, NULL, 2 );
```

defines a simple marker with two attributes (which may be population and city name). The attributes are written into the data file separately using the *GlmWriteAttribute* function and the marker's point must be written last using the *GlmWritePoint* function.

Attributes

After an object is defined, its attributes, if any exist, must be written. Each attribute is written via a call to the *GvfWriteAttribute* function. It is passed the number of the attribute, the type, the double value, if a D type, a string, if an S type, and three doubles if a G type attribute. Valid values for the type are:

GLM_D

double

GLM_G

geometrical or color, 3 doubles

GLM_S

string

For example:

```
GvfWriteAttribute( stdout, GVF_ASCII, 1, GLM_D, 50000, NULL,  
                  0., 0., 0. );
```

defines the second attribute to be of type D, and have a value of 50000. The second parameter supplies a numerical attribute name. The third parameter is used for passing a string value for string attributes, and the last three parameters are used to pass values of geometrical or color attributes.

Points

After the header and attributes are written, points data can be written to the output file. Data points are a pair of values, usually longitude and latitude. Polygons have an arbitrary number of data points, defined by the *num_points* parameter of the *GlmWritePolygon* function. Markers must have exactly one data point defined. Each data point is written into the file with a call to the *GvfWritePoint* function. It is passed the two point values and a printf style format used in the ASCII mode. For use with the Map Server, the format should contain two space-separated floating point format specifications, for example:

```
"%lf %lf "
```

Since six digit precision is usually enough for most of the GIS application, the format may parameter may specify precision for more compact data:

```
"%.6lf %.6lf "
```

The format parameter is ignored in the binary mode.

The following example:

```
GvfWritePoint( stdout, GVF_ASCII, 10., 30., "%.6lf %.6lf " );
```

defines a point with longitude 10 and latitude 30.

Examples

The following example defines a polygon with 1 attribute and 5 points

```
GvfWritePolygon( stdout, GVF_ASCII, 0, 5, 1 );
GvfWriteAttribute( stdout, GVF_ASCII, 1, GLM_D, 50000, NULL,
                  0., 0., 0. );
GvfWritePoint( stdout, GVF_ASCII, 10., 30., "%lf %lf " );
GvfWritePoint( stdout, GVF_ASCII, 20., 31., "%lf %lf " );
GvfWritePoint( stdout, GVF_ASCII, 30., 32., "%lf %lf " );
GvfWritePoint( stdout, GVF_ASCII, 40., 33., "%lf %lf " );
GvfWritePoint( stdout, GVF_ASCII, 50., 34., "%lf %lf " );
```

The following example writes a simple marker with one attribute

```
GvfWriteMarker( stdout, GVF_ASCII, GVF_M_MARKER, 0., 0., 0, NULL, 1 );
GvfWriteAttribute( stdout, GVF_ASCII, 1, GLM_D, 50000, NULL,
                  0., 0., 0. );
GvfWritePoint( stdout, GVF_ASCII, 10., 30., "%lf %lf " );
```

Function Descriptions

GvfWriteHeader

Writes the header.

```
void GvfWriteHeader( file, mode )
    FILE * file;
    GvfSaveFormat mode;
```

Parameters

file

Specifies a file descriptor.

mode

Specifies GVF_ASCII or GVF_BINARY file writing mode.

This function writes the Glm Vector Format header to the specified file descriptor. Must be called before any other GVF function calls.

GvfWritePolygon

Writes a polygon.

```
void GvfWritePolygon( file, mode, ring, num_points, num_attr )
    FILE * file;
    GvfSaveFormat mode;
    long ring;
    long num_points;
    long num_attr;
```

Parameters

file

Specifies a file descriptor.

mode

Specifies GVF_ASCII or GVF_BINARY file writing mode.

ring

Specifies whether or not the polygon has sub-sections.

num_points

Defines the number of data points.

num_attr

Defines the number of attributes.

This function writes a polygon definition to the specified file descriptor with the given number of data points and the given number of attributes.

GvfWriteMarker

Writes a marker.

```
void GvfWriteMarker( file, mode, type, scale, angle, anchor, string,
                    num_attr )
    FILE * file;
    GvfSaveFormat mode;
    GvfMarkerType type;
    double scale;
    double angle;
    long anchor;
    char * string;
    long num_attr;
```

*Parameters***file**

Specifies a file descriptor.

mode

Specifies GVF_ASCII or GVF_BINARY file writing mode.

type

Specifies the marker type, must be one of:

```
GVF_M_MARKER
GVF_M_LABEL
GVF_M_TEXT
```

scale

Defines the scale of the marker's text.

angle

Defines the angle of the marker's text.

anchor

Defines the anchoring of the marker's text.

string

Defines the text-string of the marker's text.

num_attr

Defines the number of attributes.

This function writes a marker definition to the specified file descriptor with the given properties and the given number of attributes. If the type is *GVF_M_MARKER*, then none of the properties needs to be defined. If the type is *GVF_M_LABEL*, only the string needs to be defined. If the type is *GVF_M_TEXT*, the scale, angle, anchoring and string need to be defined.

GvfWriteAttribute

Writes an attribute.

```
void GvfWriteAttribute( file, mode, name, data_type, d_val, s_val,
                       g_val_x, g_val_y, g_val_z )
FILE * file;
GvfSaveFormat mode;
long name;
GlmDataType data_type;
double d_val;
char * s_val;
double g_val_x, g_val_y, g_val_z;
```

Parameters

file

Specifies a file descriptor.

mode

Specifies GVF_ASCII or GVF_BINARY file writing mode.

name

Specifies the attribute number. Attribute numbers are used as attribute names for efficiency.

data_type

Specifies the type of the attribute, must be one of:

GLM_D
GLM_S
GLM_G

d_val

Defines the value of the attribute if the type is *GLM_D*.

s_val

Defines the value of the attribute if the type is *GLM_S*.

g_val_x, g_val_y, g_val_z

Define the value of the attribute if the type is *GLM_G*.

This function writes an attribute definition to the specified file descriptor with the given name, type and value.

GvfWritePoint

Writes a data point.

```
void GvfWritePoint( file, mode, x, y, format )  
    FILE * file;  
    GvfSaveFormat mode;  
    double x;  
    double y;  
    char * format;
```

Parameters

file

Specifies a file descriptor.

mode

Specifies GVF_ASCII or GVF_BINARY file writing mode.

x

Defines the x value of the point.

y

Defines the y value of the point.

format

Specifies a printf style format for the point to be written in.

This function writes a data point to the specified file descriptor with the given location and uses the given format. For use with the Map Server, the format should contain two space-separated floating point format specifications, for example:

```
"%lf %lf "
```

Since six digit precision is usually enough for most of the GIS application, the format may parameter may specify precision for more compact data:

```
"%.6lf %.6lf "
```

The number of points must match the *num_points* parameter passed to *GvfWritePolygon*.

5.2 GVF Format RFC

File Extension

The GLM Vector Format, as described in this RFC uses, throughout, the extension and abbreviation “GVF”. Although GVF files must not necessarily have this extension, it is used for the purpose of this RFC.

Definitions

Wherever constants are mentioned with the GLM or GVF prefix, their enumerated values should be substituted. What follows is the enumeration of constants to be used throughout the RFC.

```
GVF_MAGIC 0x69c
GVF_OBJ_HEADER 0x7
GVF_ASCII 0x0
GVF_BINARY 0x1

GVF_POLYGON 0x0
GVF_MARKER 0x1

GVF_M_MARKER 0x0
GVF_M_LABEL 0x1
GVF_M_TEXT 0x2
```

```
GLM_S 0x1
GLM_D 0x2
GLM_G 0x3

GLM_HCENTER 0x0
GLM_HRIGHT 0x1
GLM_HLEFT 0x2

GLM_VCENTER 0x0
GLM_VTOP 0x10
GLM_VBOTTOM 0x20
```

ASCII Format

This format specification only describes the plain-text ASCII version of the GLM Vector format. The binary version uses the native data representation for each platform and is produced by writing data into the file in the same sequence, but using the native raw data format. The first line of the GVF file contains a GVF file header and is always written in ASCII format. It is recommended that all saved data are first saved in the platform-independent ASCII format and then converted to the binary format using the Map Server's GVF data converter.

In the ASCII version of the GVF format, all integral values are written in hexadecimal form, but without the preceding 0x. All floating point values, such as those present in data points, will be written in their standard decimal form as written by the standard C library function *printf*. All strings are written by first specifying the length of the string as an integral value and then the characters of the string as ASCII characters, including the terminating null character. In an ASCII version of the GVF format, all entries must be separated by at least one whitespace character.

Header

The first three entries in a GVF file must be the magic number, *GVF_MAGIC*, the version number, and the format, *GVF_ASCII* or *GVF_BINARY*.

Body

- After the header is defined, objects may be defined one after another in the following format.
- The object header, *GVF_OBJ_HEADER*, must be written.
- The type of object must be written
- The object must be defined with the required properties, including:
- The number of attributes associated with that object.
- If the number of attributes is non-zero, the correct number of attributes must be defined after the object itself is defined.
- The correct number of data points must be written.

Objects

After the object header is written, the type of object may be one of:

GVF_POLYGON
GVF_MARKER

If the object type is *GVF_POLYGON*, a polygon definition must follow. If the object type is *GVF_MARKER*, a marker definition must follow.

Polygons

Polygons are defined to have 3 integral properties following the header. These properties are:

Ring

Specifies whether or not the polygon has sub-sections, i.e. whether or not the polygon is composite. If set to 0, the polygon is assumed to be non-composite. If set to 1, the polygon is assumed to be composite and that subsequent polygon definitions define other sections of the polygon. The last section in the definition of a composite polygon must have a *Ring* parameter of 0.

Number of Points

Specifies the number of points in the polygon as a positive integral value. After the polygon definition, exactly this number of points must be defined. Defining a different number of points after the polygon is invalid.

Number of Attributes

Specifies the number of attributes associated with the given polygon. After the polygon definition, exactly this number of attributes must be defined. Defining a different number of attributes after the polygon is invalid.

Markers

Markers are defined to have up to 6 properties following the header. These properties are:

Marker Type

Specifies the type of marker to be written, must be one of:

GVF_M_MARKER

The simplest type, with all marker attributes specified in the LIF file. The marker's string label may be defined in the data file as a marker's attribute and rendered using the *LABEL FORMAT LIF* option.

GVF_M_LABEL

All attributes except the string (scale, angle and anchor) are specified in the LIF file, and the string specified with the marker object in the data file, as part of the marker.

GVF_M_TEXT

All attributes (scale, angle, anchor and string) are specified with the object in the data file.

Scale

Defines the size of the marker's text as a floating point value.

Angle

Defines the angle of the marker's text as a floating point value. Must be in degrees, counter-clockwise from the right horizontal axis.

Anchoring

Defines the anchoring of the marker's text as an integral value. Must be a bitwise OR of the horizontal and vertical anchoring types. The horizontal anchoring must be one of:

```
GLM_HCENTER  
GLM_HRIGHT  
GLM_HLEFT
```

and the vertical anchoring must be one of:

```
GLM_VCENTER  
GLM_VTOP  
GLM_VBOTTOM
```

Text String

Defines the text-string of the marker's text

Number of Attributes

Specifies the number of attributes associated with the given marker object. After the marker definition, exactly this number of attributes must be defined. Defining a different number of attributes after the marker is invalid.

All marker types require the *Marker Type* and *Number of Attributes* properties, but the rest of the properties depends on the marker type. If the type is *GVF_M_MARKER*, none of the properties are necessary. If the type is *GVF_M_LABEL*, only the *Text String* is required. If the type is *GVF_M_TEXT*, then the *Scale*, *Angle*, *Anchoring* and *Text String* are required.

Attributes

Attributes are defined by first writing their name and type. The name is an integral value and the type must be one of:

```
GLM_D  
GLM_S  
GLM_G
```

After the name and type, the value must be written. If the type is *GLM_D*, the value is a floating point number. If the type is *GLM_S*, the value must be a string. If the type is *GLM_G*, the value must be exactly three floating point numbers.

Data Points

Data Points are written as two consecutive floating point numbers, usually the X and Y values of a point.

Appendix A: Web Server Installation Notes

A

Introduction

In order to use the Map Server in a web/cgi-bin environment, a web server must be installed. Common web servers include Apache (on both Linux/Unix and Windows) and Internet Information Services - IIS (Windows only). To install a web server, follow its manufacturer's instructions.

The Map Server uses the Common Gateway Interface (**CGI**), which is supported by most of the web servers. It may also use the **FastCGI** version of the interface, which allows the map server executable to stay resident in memory and serve multiple map requests, instead of starting and exiting after each request.

In Linux/Unix environment, a bourne shell is used as a scripting language. In Windows environment, a Perl scripting language also needs to be installed, so that the Map Server executable can be invoked by the web server. On Linux/Unix systems, a *GlmScript* bourne shell script that invokes the Map Server is provided. On Windows, a *GlmScript.pl* Perl script is provided.

The *GlmScript* file (*GlmScript.pl* on Windows) must be placed into the web server's cgi-bin directory. The map server executable and the map data set must be also installed in the location pointed to by the script file.

Linux/Unix Notes

The *GlmScript* shell script provided with the Map Server in the `<glg_dir>/map_server/cgi-bin` directory should be placed in the *cgi-bin* directory of the web server. The script source contains step-by-step installation instructions.

If using Apache, cgi-bin scripts usually reside in the `/usr/lib/cgi-bin` directory. If using a web server other than Apache, see the manufacturers instructions on where cgi-bin scripts are located.

The script must have execute permission for the user that the web server uses to invoke cgi-bin scripts. If a log file is used, it must also have write permissions for that user.

An alternative *GlmScript.pl* perl script can be used with perl.

Refer to the comments in the scripts for detailed installation instructions.

Windows Notes

The *GlmScript.pl* perl script provided with the Map Server in the `<glg_dir>/map_server/cgi-bin` directory should be placed in the scripts directory of the web server. The source of the script contains step-by-step installation instructions.

If using Internet Information Services (IIS), scripts reside in the *Scripts* directory, which is created by IIS and is usually located at `C:\inetpub\wwwroot\Scripts`. If using Apache, cgi-bin scripts usually reside in the *cgi-bin* directory of the Apache installation directory.

The Perl scripting language must also be installed as described in the following note.

Perl Notes

On Windows, it is not possible to use a batch script to invoke the Map Server, because it is not secure, so a scripting language such as Perl must be installed. A free implementation of Perl can be downloaded at www.activeperl.com. Perl must be installed after the web server so that application bindings are properly installed.

To test if Perl is working properly, create a file *test.pl* with the following contents:

```
print "If you are seeing this message, the script executed correctly";
```

Then, from a command prompt on Windows or console session on Linux/Unix, execute the following command from the directory where *test.pl* resides:

```
perl test.pl
```

If the message in the file is printed, then the script executed correctly.

In order to test if cgi-bin scripts work with the installed web server, create a file called *test.pl* with the following contents:

```
print "Content-Type: text/html\n\n";  
print "If you are seeing this message,<br> the script executed  
correctly";
```

Additionally, if using the Apache web server, the first line of the file must contain *#!* and the location of the Perl executable.

Place the file into the cgi-bin scripts directory of the web server, change its permissions accordingly, and enter the following URL in a web browser (where *mywebserver.com* is the location of the webserver):

If running Apache:

```
http://mywebserver.com/cgi-bin/test.pl
```

If running IIS:

```
http://mywebserver.com/Scripts/test.pl
```

If the message in the file is printed, then the script executed correctly.

Map Server Setup on a Web Server

Map Server Run-Time Components

The following components are required at run time and need to be copied into the web server's **cgi-bin** scripts directory:

- The Map Server executable: *GlmMap* on Linux/Unix or *GlmMap.exe* on Windows. The executable is located in the `<glg_dir>/bin` directory.
- The cgi-script: *GlmScript* on Linux/Unix, *GlmScript.pl* on Windows or if Perl setup is used. The script is located in the `<glg_dir>/map_server/cgi-bin` directory.
- The GIS data directory `<glg_dir>/map_data`. The data directory may also be located outside of the cgi-bin scripts directory and pointed to by the *GlmScript setup script*.

Make sure *GlmMap* and *GlmScript* (or *GlmMap.exe* and *GlmScript.pl*) have execute permission.

To debug Map Server setup, create *glm_err.log* file in the cgi-bin scripts directory (or copy it from `<glg_dir>/map_server/cgi-bin` directory) and make it writable by the web server's cgi-bin scripts (www-data user and/or group in the default linux/apache configuration). This file will contain the Map Server's error messages. To debug Map Server setup, uncomment lines for extra logging and verbosity in the *GlmScript* script as described in the next chapter.

Note: The log file debugging output can be used only while debugging map server setup. It should be disabled in the production setup, as it will cause problems when multiple web clients cause simultaneous writing into the log file, resulting in CGI-bin HTTP errors.

GlmScript setup

The *GlmScript* is an easy-to-edit setup script that connects the web server with the Map Server executable. The script defines the location of the map server executable and its command line options, as well as the location of the GIS data. In addition to serving as a “glue” script, it also hides the details of the specific web setup from the web clients, presenting them with a logical interface to the map generation service.

The setup script starts with comments containing detailed setup instructions as well as sample HTTP requests for testing a map server setup using the sample GIS data. The rest of the setup script contains configuration variables that need to be changed according to the specific web server setup.

While a bourne shell called *GlmScript* is used in the Linux/Unix environment and a Perl script called *GlmScript.pl* is used on Windows, the editing instructions are the same for both scripts:

- Place the script into the *cgi-bin* directory on the web server (the *cgi-bin* directory is named *Scripts* for IIS on Windows).
- Edit the `MAP_SERVER_EXE` variable in the script to point to the location of the Map Server executable.
- Edit the `DATASET` variable in the script below to point to the location of the dataset's SDF file.
- If an evaluation version of the Map Server is used, uncomment the `GLG_EVAL_STRING` variable setting and set the variable to the up-to-date evaluation code obtained from Generic Logic (www.genlogic.com). For the production or free non-commercial version of the Map Server, comment out `GLG_EVAL_STRING` setting.
- To debug the Map Server setup, uncomment the command line near the end of the file for testing the map server installation and comment out the command line for the production setup, as instructed by comment next to the lines.

For the production setup, reverse it, commenting out the test line and enabling the production line.

FastCGI setup

FastCGI mode allows the Map Server executable to stay resident in memory and server multiple map requests compared with invoking a separate Map Server executable for each request in CGI mode and exiting when the request is processed. The FastCGI mode not only eliminates excessive executable start-up time, but also speeds up map generation requests by reusing data tiles in Map Server's cache.

In CGI mode, the content of the tile cache is lost when the Map Server executable exits. In the FastCGI mode, consecutive map requests reuse data tiles which have been already read by the previous map requests, saving on the data read time and speeding up map generation. The *MAX CACHE SIZE* parameter of the *LIG* file controls the cache size on per-layer basis.

To use the Map Server in the FastCGI environment, edit *GlmScript* (or *GlmScript.pl* on Windows) and replace *-cgi* option with *-fcgi*, then follow the FastCGI setup instructions for your web server.

Testing Map Server Setup on a Web Server

To test the Map Server's setup, use the sample HTTP map requests listed in the comments section of the *GlgScript* file, for example on Linux/Unix:

```
http://my_web_server.com/cgi-bin/GlmScript?VERSION=1.3.0&
REQUEST=GetMap&SRS=EPSG:4326&WIDTH=800&HEIGHT=700&
BBOX=-180,-90,180,90&BGCOLOR=0x0&STYLES=default&
FORMAT=image/jpeg&LAYERS=earth&STRETCH=0
```

or for the Windows-based setup:

```
http://my_web_server.com/Scripts/GlmScript.pl?VERSION=1.3.0&
REQUEST=GetMap&SRS=EPSG:4326&WIDTH=800&HEIGHT=700&
BBOX=-180,-90,180,90&BGCOLOR=0x0&STYLES=default&
FORMAT=image/jpeg&LAYERS=earth&STRETCH=0
```

Cut and paste one of the map requests into the address field of a web browser, change *my_web_server.com* to the real domain of your web server and press *<Enter>*. If everything was setup correctly, the requested map image will appear. Otherwise, an error message will indicate the origin of a problem. If logging is enabled, check the *glm_error.log* file for more information.

Try changing parameters of the map requests to see how they affect the generated map image.

A

ALPHA BLACK OPAQUE, 39
 ALPHA WHITE OPAQUE, 39
 Anchoring, 151
 Angle, 151
 antialiasing, 42
 ASCII converter, 78
 ASCII format, 141
 Attribute Condition Syntax, 71
 attribute name, 147
 attribute number, 147
 Attributes, 143, 151

B

BACKGROUND, 38
 BINARY converter, 78
 binary format, 141
 Bounding Box extraction utility, 80
 BUMP MAP, 39

C

CGI mode, 65
 Command Line Options
 -a, 78, 79, 83, 88, 90
 -all-attrs, 90
 -arg-file, 64
 -attr, 85, 90
 -b, 78, 79, 83, 88, 90
 -bbox, 86
 -cgi, 65
 -close-polygons, 91
 -convert, 78
 -custom-range, 82, 87
 -data-offset, 97
 -dataset, 65
 -data-shift, 97
 -dont-emboss, 99
 -elevation, 81, 97
 -f, 98
 -fcgi, 66
 -filename, 81, 85, 88
 -filter, 79, 81, 85
 -gvf-extent, 82
 -help, 64
 -image, 81, 88, 93
 -info-only, 96

 -invert-attr, 86
 -k, 98
 -lif, 92
 map query string, 66
 -marker-label-attr, 91
 -max-iterations, 66
 -max-lat, 83, 86
 -max-lon, 83, 86
 -max-wlon, 82, 87
 -merge, 79
 -min, 86
 -min-lat, 83, 86
 -min-lon, 83, 86
 -min-wlon, 82, 87
 -multi-layer, 93
 -neg-range, 82, 86
 -no-attrs, 90
 -no-range, 82, 87
 -num-points, 86
 -num-x-tiles, 81, 86
 -num-y-tiles, 81, 86
 -oGISreq, 66
 -out-ext, 98
 -out-format, 98
 -output, 66, 79, 85, 88, 89
 -out-slf-name, 94
 -path, 78, 88, 93
 -pattern, 78, 89, 93
 -pos-range, 82, 87
 -postfix, 98
 -progress, 96
 -r, 90, 98
 -resolution, 88
 -shadow, 97
 -show-attrs, 90
 -shp2gvf, 89
 -single-layer, 93
 -slf, 98
 -slim, 88
 -split, 85
 -suppress-bbox, 93
 -template, 81
 -tile, 80
 -u, 98
 -vector, 81, 88, 93
 -verbosity, 64, 96

- version, 64
- world-extent, 82
- write-image, 97
- Comments, 31
- Configuration File Format, 31
- Configuration Variables, 38
 - ADAPTIVE GRID, 48
 - ALIAS, 33
 - ALLOW MISSING TILES, 45
 - ALLOW OVERRIDE, 35, 57
 - ALPHA, 40
 - ATTR COND, 40
 - ATTR MAP, 57
 - BACKGROUND COLOR, 45, 46
 - BACKGROUND LAYER, 46
 - BASE WIDTH, 34
 - BOX EDGE COLOR, 55
 - BOX FILL COLOR, 55
 - CLIP, 44
 - CLIP MAX LAT, 47
 - CLIP MAX LON, 47
 - CLIP MIN LAT, 47
 - CLIP MIN LON, 47
 - CLIP TYPE, 47
 - CUSTOM ATTR, 59
 - DECIMAL GRID LABELS, 48
 - DEFAULT FONT, 35
 - DEFAULT ON, 38
 - DRAW CENTER MARKER, 56
 - EDGE COLOR, 54
 - ELEVATION MODE, 40
 - ENCODING, 35
 - ERROR FONT, 35
 - FALLBACK, 43
 - FALLBACK LAYER, 42
 - FILENAME, 41, 60
 - FILL COLOR, 54
 - FILL TYPE, 55
 - FILTER, 41
 - FONT, 49
 - FONT DIR, 33
 - FONT SCALE, 49
 - GLM ROOT DIR, 33
 - GLYPH CACHE TYPE, 36, 57
 - GRADIENT COLOR, 55
 - GRADIENT LENGTH, 55
 - GRADIENT TYPE, 55
 - GRID LABELS, 48
 - GRID LAT INTERVAL, 47
 - GRID LON INTERVAL, 47
 - GRID MAX LAT, 48
 - GRID MAX LON, 48
 - GRID MIN LAT, 48
 - GRID MIN LON, 48
 - HEIGHT, 46
 - IMAGE ANTIALIASING, 42
 - IMAGE ANTIALISING, 42
 - IMAGE FONT, 34
 - IMAGE FORMAT, 42
 - INCLUDE, 60
 - LABEL COLOR, 54
 - LABEL FORMAT, 50
 - LABEL FORMAT2, 51
 - LABEL MAX ZOOM, 52
 - LABEL MIN ZOOM, 52
 - LABEL OUTLINE COLOR, 51
 - LABEL PRIORITY, 52
 - LABEL REPEAT DISTANCE, 53
 - LABEL SELECTION MODE, 57
 - LABEL STYLE, 51
 - LAYER, 33
 - LAYER DIR, 33
 - LAYOUT MARGIN, 53
 - LAYOUT MARKERS, 53
 - LAYOUT TYPE, 52
 - LINE WIDTH, 55
 - LOCK TILES, 44
 - LOCK TIMEOUT, 44
 - MARKER EDGE COLOR, 54
 - MARKER FILL COLOR, 54
 - MARKER ICON FILE, 53
 - MARKER ICON SIZE, 54
 - MARKER ICON TYPE, 53
 - MAX CACHE SIZE, 43
 - MAX IMAGE TILES, 35
 - MAX ITERATIONS, 36
 - MAX LAT, 46, 60
 - MAX LON, 46, 60
 - MAX TILES, 43
 - MAX VECTOR TILES, 35
 - MAX ZOOM, 46
 - MIN LAT, 46, 60
 - MIN LON, 46, 60
 - MIN PIXEL SIZE, 43
 - MIN ZOOM, 46
 - NUM TILES X, 43
 - NUM TILES Y, 43
 - PIXEL MAP, 60
 - POLY LABEL CENTER, 57
 - POLY LABEL FORMAT, 56

POLY LABEL FORMAT2, 56
 POLY LABEL TYPE, 56
 POS RANGE, 56
 REDIRECT FILE, 44
 REDIRECT PREFIX, 44
 REL ZOOM, 35
 SUBLIF DIR, 60
 SUBLIF FILE, 60
 TBOX LINE WIDTH, 52
 TEXT ANCHOR, 50
 TEXT ANGLE, 50
 TEXT BOX OFFSET HORIZ, 52, 54
 TEXT BOX OFFSET VERT, 52
 TEXT OFFSET HORIZ, 54
 TEXT OFFSET VERT, 54
 TEXT TYPE, 49
 TILES OVERLAP, 45
 TILES SUBLIF, 60
 TRANS TYPE, 39
 TYPE, 38
 USED HEIGHT, 61
 USED WIDTH, 61
 VECTOR FONT, 34
 VECTOR FORMAT, 41
 VOLATILE, 44
 VOLATILE ERRORS, 44, 45
 WIDTH, 46
 WRAP LABEL LENGTH, 51
 ZOOM FACTOR TYPE, 34

Creating Datasets, 108

D

Data Points, 152
 Data Types, 31
 Dataset resources
 AllowOverride, 109
 BaseWidth, 109
 DefaultFont, 110
 Encoding, 109
 ErrorFont, 110
 FontPath, 109
 GlmRoot, 109
 GlyphCacheType, 110
 LayerPath, 109
 MaxImageTiles, 109
 MaxIterations, 109
 MaxVectorTiles, 109
 ZoomFactorType, 109
 Datasets, 32
 decluttering, 43

default, 36

E

Elevation color threshold, 60
 error log file, 63
 Examples
 API Usage, 104
 Attribute Map, 58
 Attributes, 143
 Custom Attribute, 59
 Elevation color threshold, 60
 Enabling layers, 104
 Filters and Data Converters, 144
 Font Anchor, 50
 Generating an image, 104
 Label Format, 51
 Map Requests
 Orthographic, 72
 Political boundaries, 73
 Rectangular projection, 74
 Windows, 73
 Markers, 143
 Modifying layers, 108
 Naming Tiles, 41
 Polygons, 142
 Server Dataset File, 36
 Threshold table file, 58
 Vector splitting, 87

F

FastCGI mode, 66
 filter header, 142
 Fonts, 30

G

GIS Selection Message, 129
 GLG_DIR environment variable, 63
 GLG_ERROR_LOG environment variable, 63
 GlgGISPickResolution, 129
 GLM_ERROR_LOG environment variable, 63
 GlmAddLayer, 121
 GlmConvert, 134
 GlmCreateDataset, 120
 GlmCreateMap, 121
 GlmDestroyDataset, 138
 GlmDestroyMap, 138
 GlmDisableLayer, 123
 GlmEnableLayer, 123
 GlmGetDResource, 123
 GlmGetElevation, 128
 GlmGetGResource, 124

- GlmGetLatLon, 133
- GlmGetMapData, 127
- GlmGetProjection, 127
- GlmGetSelection, 129
- GlmGetSResource, 125
- GlmGetXYZ, 133
- GlmInit, 120
- GlmLatLonToUtm, 135
- GlmMgrsToUtm, 137
- GlmRemoveLayer, 138
- GlmResetImageErrors, 138
- GlmResetLayers, 122
- GlmSetDResource, 125
- GlmSetGResource, 126
- GlmSetLayers, 122
- GlmSetSResource, 126
- GlmSetupMap, 102, 121
- GlmTerminate, 139
- GlmUpdateMap, 137
- GlmUtmToLatLon, 136
- GlmUtmToMgrs, 136
- GlmWriteImage, 127
- GRID, 38
- GRID LABEL CHAR, 49
- GRID LABEL FORMAT, 48
- GRID LABEL FORMAT LON, 48
- GVF, 148
- GVF ASCII/BINARY converter, 78
- GVF data file, 27
- GVF file, 28
- GVF_ASCII, 141
- GVF_BINARY, 141
- GvfWriteAttribute, 146
- GvfWriteHeader, 144
- GvfWriteMarker, 145
- GvfWritePoint, 147
- GvfWritePolygon, 145

H

- Hierarchical Tile Parsing, 92
- Hierarchical Tiling, 37
- hierarchical tiling, 92
- Hierarchy of GlmObjects, 107

I

- IMAGE, 38
- Image and Elevation Data Import, 95
- Image Data, 28

L

- Layer Information File (LIF), 36
- Layer Resources, 110
 - AdaptiveGrid, 114
 - AllowMissingTiles, 120
 - AllowOverride, 120
 - Alpha, 112
 - BackgroundLayer, 111
 - BGColorBlue, 111
 - BGColorGreen, 111
 - BGColorRed, 111
 - BoxEdgeColor, 118
 - BoxFillColor, 118
 - Clip, 112
 - ClipMaxLat, 113
 - ClipMaxLon, 113
 - ClipMinLat, 113
 - ClipMinLon, 113
 - ClipType, 113
 - DecimalGridLabels, 114
 - DrawCenterMarker, 118
 - EdgeColor, 118
 - ElevationMode, 111
 - FallbackLayer, 120
 - FillColor, 118
 - FillType, 117
 - FontScale, 115
 - GlyphCacheType, 120

GradientColor, 118
 GradientLength, 118
 GridLabelChar, 114
 GridLabelFormat, 114
 GridLabelFormatLon, 114
 GridLabels, 114
 GridLatInterval, 113
 GridLonInterval, 113
 GridMaxLat, 113
 GridMaxLon, 113
 GridMinLat, 113
 GridMinLon, 113
 Height, 112
 ImageAntiAliasing, 112
 IsDefault, 110
 LabelEdgeColor, 118
 LabelFormat, 115
 LabelFormat2, 115
 LabelMaxZoon, 116
 LabelMinZoom, 116
 LabelOutlineColor, 116
 LabelPriority, 116
 LabelRepeatDistance, 117
 LabelSelectionMode, 119
 LabelStyle, 116
 LayerType, 110
 LayoutMargin, 117
 LayoutMarkers, 117
 LayoutType, 116
 LineWidth, 118
 LockFile, 119
 LockTimeout, 112, 119
 LonLabelAngle, 114
 MarkerEdgeColor, 118
 MarkerFillColor, 118
 MarkerIconFile, 117
 MarkerSize, 117
 MarkerType, 117
 MaxCacheSize, 112
 MaxLat, 113
 MaxLon, 112
 MaxTiles, 112
 MaxZoom, 113
 MinLat, 112
 MinLon, 112
 MinPixelSize, 112
 MinZoom, 113
 PolyLabel, 119
 PolyLabelCenter, 119
 PolyLabelFormat, 119

PolyLabelFormat2, 119
 PolyLabelType, 119
 RedirectFile, 120
 RedirectPrefix, 120
 RotateLonLabels, 114
 TextAnchor, 115
 TextAngle, 115
 TextBoxLineWidth, 116
 TextBoxOffsetH, 116, 117
 TextFont, 115
 TextOffsetH, 117
 TextOffsetV, 117
 TextType, 115
 TilesOverlap, 111
 TransColor, 111
 TransColorPrecision, 111
 TransType, 111
 Volatile, 119
 VolatileErrors, 119
 Width, 112
 WrapLabelLength, 116
 Layers, 27
 Layers and Tiles, 36
 LIF file, 27, 31
 linking
 MS Windows, 107
 logging errors, 63
 LON LABEL ANGLE, 49

M

Map Resources
 Angle, 103
 Background, 103
 CenterLat, 102
 CenterLon, 102
 DatasetName, 102
 Height, 103
 HExtent, 103
 ImageAntiAliasing, 103
 ImageErrors, 103
 Projection, 102
 Stretch, 103
 Verbosity, 103
 VExtent, 103
 Width, 103
 Marker Type, 150
 Markers, 142, 150
 Merging utility, 79
 Modifying Layers, 108

N

Number of Attributes, 150, 151

Number of Points, 150

O

OPAQUE, 39

Open Gis Tokens

BBOX, 68

BGCOLOR, 68

CENTER, 70

EXTENT, 70

FORMAT, 68

HEIGHT, 68

I, 71

IERRORS, 70

IMAGE_ANTIALIASING, 70

INFO_FORMAT, 71

INFO_TYPE, 70

J, 71

LAYERS, 69

LON_LAT, 71

PICK_RESOLUTION, 71

QUERY_LAYERS, 70

REQUEST, 67

SELECT_LABELS, 71

SRS, 67

STRETCH, 70

STYLES, 68

VERSION, 67

WIDTH, 68

OUTLINE, 38

P

-pattern, 98

Perl Notes, 154

Points Data, 143

Polygons, 142, 150

Q

QUERY_STRING, 65

R

Resources, 101

Ring, 150

rings, 142

ROTATE LON LABELS, 49

S

Scale, 151

SDF file, 27

Shapefile Conversion Utility, 89

SLF file, 60

Slimming Utility, 88

Splitting Vector Data, 84

Street Labels, 56

sub-LIF file, 60

T

Text Box, 52

Text String, 151

The Zoom Factor, 37

Threshold table, 58

Tiling, 37

Tiling image and vector data, 80

Tiling Images, 80

TRANS COLOR, 39

TRANS COLOR PRECISION, 39

TRANSPARENT COLOR OPAQUE, 39

tree tiling, 92

Types of Data, 27

U

Unix Notes, 153

V

VECTOR, 38

Vector Data, 28

Attributes, 29

verbosity, 37, 64, 103, 155

W

Web Server Installation Notes, 153

Windows Notes, 153

Writing a Filter, 142

X

XOR, 39