GLG Map Server Tutorial

GLG Toolkit

Version 4.1

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 © 1994-2021 by Generic Logic, Inc.
All Rights Reserved. This manual is subject to copyright protection.

GLG Toolkit, GLG Widgets, and GLG Graphics Builder are trademarks of Generic Logic, Inc.
All other trademarks are acknowledged as the property of their respective owners.

May 3, 2021
Software Release Version 4.1

GLG Map Server Tutorial

Table of Contents

1. Introduction 5

Map Server Overview 5
- Integrated Map Server Use 5
- Stand-Alone Map Server Use 6
- GIS Data and Setup Files 6

Key Features 6

Demos and examples 7
- C/C++ Demos and Examples 7
- Java Demos and Examples 7
- JavaScript Demos and Examples 8
- C# Demos and Examples 8
- VB.NET Demos and Examples 8

Compilation and Map Server setup 8
- Compilation notes for C/C++ 8
- Web Server Setup 8

2. GIS Object usage in the GLG Builder 9

Adding GIS Object to the drawing 9

GIS Object Properties 9
- GISProjection 10
- GISCenter 10
- GISExtent 10
- GISStretch 11
- GISDataFile 11
- GISMapServerURL 12
- GISLayers 13
- GISVerbosity 13

Positioning of the GIS Object Inside the Viewport 14

Zooming and Prototyping in the Builder 15

Zooming and Panning in the Edit Mode 15
- GIS Editing Mode 15
- GIS Zoom Mode 16

Zooming and Panning in the Run Mode 16

GIS Object Resources 18

Adding dynamic objects to the map 19
- GIS Rendering Mode 19

GIS Editing Mode 20
Programmatic Access at Run-Time....21

3. Map Data Setup 21

Server Data File (SDF) 21

Layer Information File (LIF) 23

Using Zoom Factor Limits for Automatic Layer Switching 25

GIS Data Formats for Vector Layers 26

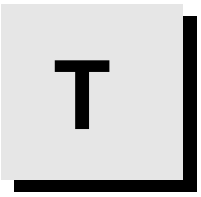
Image Layers and Tiling 26

Transparency and Blending Types 29
Simple Transparent Layers....29
Layers with Transparent Background 29
Special Blending Types....30

Markers and Labels 30

Dynamic Attribute Thresholds 32

GLG Map Server Tutorial



1. Introduction

This tutorial presents an overview of the GLG Map Server and explains the basics of the map server's GIS data setup. While the map server may be used as an integrated component in the GLG Toolkit or as a stand-alone product, the materials presented in this tutorial are relevant to both of these usage scenarios.

The GIS Object of the GLG Builder described in the first part of this tutorial is used as a convenient map server prototyping tool, for both the integrated and standalone use. It allows you to quickly test the map server setup, as well as experiment with various map server settings without issuing map server requests manually.

The second part of the tutorial will walk you through the basics of the map server data setup. The map server data setup is the same for either the integrated or standalone use of the map server.

The Demo of the GLG Toolkit available for download on the Generic Logic's website contains all files used in this tutorial, including the GLG Builder and the GIS data used by the map server. The evaluation and production versions of the Toolkit may also be used.

The Map Server FAQs located on the website provide examples of command-line options and map query requests for the stand-alone use of the map server.

For programming examples, refer to the demos and examples listed in the *Demos and examples* section of the tutorial. For the complete documentation of the GLG Map Server, refer to the *GLG Map Server Reference Manual.*

Map Server Overview

GLG Map Server is a powerful and compact GIS Map Server designed to add mapping functionality to the real-time and web-based applications. It provides a high-performance rendering engine for vector and raster GIS data, generating detailed real-time map images for requested areas from the supplied GIS data. The Map Server is self-contained, does not depend on any third-party software or windowing system at run-time and can operate on a head-less server on a variety of hardware and software platforms.

The Map Server also provides capabilities for querying the lat/lon and elevation values of a selected point on the map.

Integrated Map Server Use

When the Map Server is used in conjunction with the Toolkit, the GIS Object is created in the GLG Builder as a part of the GLG drawing to provide an interface to the Map Server. The GIS Object displays map images and transparently handles all low-level details of the map generation, automatically issuing map generation requests to the Map Server when the map is zoomed or panned.

Since the GIS Object is a part of the GLG drawing, other dynamic or static GLG objects may be placed on top of the map to visualize positions of moving objects, such as airplanes, delivery vehicles, patrol cars and other points of interest. Dynamic icons may be animated and positioned using real-time data and provide interfaces for handling object selection and mouse over events.

When the drawing containing the GIS Object is deployed in a **C/C++ application**, it utilizes the Map Server in the form of a library, accessing GIS data from a local drive without a need for an Internet connection. For cases when it is desirable to keep all GIS data on a central server, the GIS Object provides an option for getting map images from a web-based map server via a URL, to avoid replicating GIS data on each local drive. Both options are also available for the **ActiveX control** on Windows as well.

When the drawing is deployed in a **Java, C# or JavaScript application**, the GIS Object uses the web access option, obtaining map images from a web server-based Map Server via a URL.

Stand-Alone Map Server Use

If an application needs to generate map images, but does not need other functionality provided by the Toolkit, the Map Server can be deployed as a stand-alone product. Two deployment options are provided: the Map Server library and Map Server executable.

The Map Server library provides an API for generating map images. The library is self-contained and is available on a variety of platforms. The *GLG Map Server Reference Manual* provides source code examples for the stand-alone library use.

The Map Server executable supports the *CGI* and *FastCGI* mode and is used for web-based setups. It is available for a variety of Unix/Linux and Windows platforms. The Map Server executable may also be used to generate map images using command-line parameters. Refer to the *GLG Map Server Reference Manual* for the command-line parameters and web server setup instructions.

When the stand-alone version of the map server is used, the GIS Object in the GLG Builder may be used for fast prototyping, providing a unique and convenient way for rapid application development and testing.

GIS Data and Setup Files

Regardless of the usage mode, the Map Server accesses GIS data and setup files in order to generate map images. All setup files are usually kept in one directory, and the setup is the same for any of the usage modes. The setup files describe available map layers and define layers' rendering styles, while the GIS data provide lat/lon coordinates of the GIS features as well as optional attributes. For example, a city may have a population and name attribute.

Key Features

The following is a brief list of the GLG Map Server features:

- Unlimited zooming and panning, map rotation support for the "pilot view" displays.
- Dynamic activation/deactivation of map layers, switching to more detailed layers as the map is zoomed in.

- Dynamic attribute thresholds to vary the feature's appearance depending on its attributes.
- Elevation display and query capabilities, support for shadow relief images.
- On-the-fly data projection, rectangular and orthographic.
- Support for volatile data and semi-transparent weather overlays.
- Support for rectangular and hierarchical directory-based tiling, with tiling and setup utilities for handling huge datasets.
- Tile cache and automatic switching to fallback layers when allowed number of tiles is exceeded.
- Adaptive grid with support for custom grid labels.
- Fast prototyping in the GLG Builder, via the use of the GIS Object.
- Easy data setup via the use of ASCII files.
- Real-time performance optimizations.
- OpenGIS compliance and cross-platform availability.

Demos and examples

Several GLG demos and examples demonstrate how the Map Server may be used in a GLG application that requires mapping functionality. The GIS and Airtraffic demos available in the GLG's *DEMOS* directory and on-line, on the Generic Logic's website, provide examples of air traffic monitoring applications. Smaller GIS examples in the examples directories may be used as a starting point to familiarize yourself with a structure of a GLG mapping application.

In all demos and examples, an interactive map is displayed in the background using the GIS Object, which transparently handles map zooming and panning, as well as activation or deactivation of map layers depending on the zoom factor.

The map has dynamic icons moving on top of it, which are rendered using GLG graphical objects. The icons are positioned using lat/lon coordinates and updated in real-time, displaying a tooltip on the mouse event and allowing the user to select an icon with the mouse.

The following GIS demos and examples for various programming environments are provided:

C/C++ Demos and Examples

```
<glg_dir>/DEMOS/gis_demo
<glg_dir>/DEMOS/airtraffic
<glg_dir>/examples_c_cpp/misc/GIS
<glg_dir>/examples_c_cpp/misc/GIS2
```

Java Demos and Examples

```
<glg_dir>/DEMOS_JAVA/GlgGISDemo.java
<glg_dir>/DEMOS_JAVA/GlgAirTrafficDemo.java
<glg_dir>/examples_java/GIS
<glg_dir>/examples_java/GIS_PilotView
```

JavaScript Demos and Examples

```
<glg_dir>/DEMOS_HTML5/gis_demo.html
<glg_dir>/DEMOS_HTML5/airtraffic_demo.html
```

C# Demos and Examples

```
<glg_dir>/DEMOS_C#/GIS
<glg_dir>/DEMOS_C#/AirTraffic
<glg_dir>/DEMOS_C#/AirTrafficExt
<glg_dir>/examples_csharp.NET/GIS
```

VB.NET Demos and Examples

```
<glg_dir>/examples_vbnet_ocx/gis
```

A programming example of using the GLG Map Server library as a standalone product may be found in the *GLG Map Server Reference Manual.*

Compilation and Map Server setup

GLG Map Server may be used either in the form of a library linked into an application, or in the form of a *GCI-bin* executable which serves maps from a web server. The library version of the Map Server is used for C/C++ applications, as well as C# and VB.NET applications that use ActiveX control.

The *CGI-bin* Map Server executable is used for Java, JavaScript and C# applications, as well as C/C++ applications which request map images from a local or remote web server via a URL.

Compilation notes for C/C++

When linking a GLG C/C++ mapping application on the Unix/Linux platforms, the Map Server library has to be included in addition to the GLG library. A sample makefile is provided in the *src* directory. Refer to the *Linking with the Map Server Library* chapter of the *GLG Map Server Reference Manual* for more information.

On Windows, no extra action is required, as the Map Server library is included in both the GLG library and GLG ActiveX control.

Web Server Setup

When the Map Server is deployed as a library, no web server setup is required.

When the Map Server is used as a *CGI-bin* executable to serve map images over the web, the Map Server should be set up on a local or remote web server. Refer to the *Appendix A: Web Server Installation Notes* chapter of the *GLG Map Server Reference Manual* for detailed setup instructions.

2. GIS Object usage in the GLG Builder

Adding GIS Object to the drawing

Let's create a GLG drawing and add a GIS Object to it to display a map.

Start the GLG Builder and use the *File, New Widget* option from the main menu to start a new drawing. Use the *OK* button to close the information dialog.

Click on the *GIS Object* button in the object palette to create a GIS Object. Following the prompt at the bottom of the Builder, select two points to define a rectangular area for the object.

A *File Dialog* comes up, asking you to select a GIS Object's datafile. For now, accept the default, i.e. the *sample.sdf* file from the *<glg_dir>/map_data* directory. A satellite image of the world with a vector layer of the United States boundaries will be displayed in the GIS Object using rectangular projection.

The *sample.sdf* file is a server data file (SDF), which describes layout and setup of the GIS data used to render the map image. We will examine the content of the SDF file in the *Map Data Setup* chapter.

Likewise other GLG objects, such as a polygon or arc, a GIS Object is a GLG graphical object with a set of properties that control its appearance. The object's type, **GIS**, is displayed in the status area at the bottom of the drawing area. Same as other GLG objects, it is created as a child of the drawing area (viewport object) and positioned inside the viewport using the object's control points. The GIS Object's properties and resources are used to configure the object, as well as control it at run-time.

GIS Object Properties

Make sure that the GIS Object is selected, otherwise select it by clicking on it with the mouse.

Bring the object's Properties dialog, using the *Object, Properties* option from the main menu.

The *Properties* dialog for the GIS Object is similar to the *Properties* dialog of other GLG objects. The properties that are common for all GLG objects, such as the object's name, type, visibility, are displayed at the top of the dialog. The *Type* field displays the object's type: *GIS*.

Enter *"MapObject"* in the *Name* field to name the object.

The *FillColor* property controls the background color of the map. Right now the background is not visible, because it is completely covered by the raster *earth* layer. The background will be visible when raster layers are used, or when the map image doesn't fill the whole area of the GIS Object, for example when the whole globe is displayed in orthographic projection.

The rest of the properties are specific to the GIS Object.

GISProjection

This property defines the projection of the rendered map; possible settings are RECTANGULAR or ORTHOGRAPHIC.

Change *GISProjection* to ORTHOGRAPHIC and view the map being displayed in the form of a globe.

Set the *FillColor* property to black, by clicking on the ellipses button and selecting the black color in the color palette. Notice that the background surrounding the globe changes its color to black.

Change *GISProjection* back to RECTANGULAR.

GISCenter

This property defines the center of the displayed map in the *lat/lon* coordinates, it is also used to pan the map. This property contains a triplet of the X, Y and Z coordinates: the X coordinate specifying the longitude, and Y specifying the latitude of the map center. The Z coordinate is reserved and must be 0. The property's data type is *G* (geometrical).

The default value of *"0 0 0"* requests the map to be centered around the Greenwich Meridian in longitudinal direction (lon=0), and around equator in latitudinal direction (lat=0).

Change the value of the *GISCenter* property to *"-100 40 0"* and press *Enter*. Notice that the map has moved and is now centered at the middle of North America (lon=-100, lat=40).

GISExtent

While the control points of the GIS Object define the size of the map in screen coordinates, the *GISExtent* property defines the extent of the longitude and latitude shown in the map area. This is also a *G* (geometrical) property containing a triplet of the X, Y and Z coordinates, specifying the longitudinal and latitudinal extent of the map, respectively. The Z coordinate is reserved and must be 0.

For the RECTANGULAR projection, *GISExtent* is specified in degrees. The default value of *"360 180 0"* requests the map of the whole world (360 degrees of longitude and 180 degrees of latitude) to be displayed.

Change *GISExtent* to the value *"100 70 0"*. Notice that only North America is being displayed now, instead of the whole world.

The combination of the *GISCenter* and *GISExtent* properties define the exact mapping region being displayed in the GIS Object. In the above example, the area of (100,70) lon/lat degrees, with the lon/lat center (-100,40) covers the region of North America.

The units of *GISExtent* depend on the *GISProjection*. In accordance with the OpenGIS standard, the *GISExtent* is specified in degrees for the rectangular projection and in meters for the orthographic projection.

Let's try using this property in the orthographic projection.

Change the *GISProjection* property to ORTHOGRAPHIC.

Notice that the value of *GISExtent* has changed to *"1.4e+07 1.4e+07 0"*. Since the value of 1.4e+07 is slightly bigger than the length of the Earth's diameter in meters, the whole globe will be displayed.

Set the value of *GISExtent* to *"7000000 7000000 0"*. It will display only the North America area, in the orthographic projection.

Note: For the orthographic projection, the X and Y values of the *GISExtent* property must be the same.

GISStretch

This property determines whether or not the map is stretched when the X/Y ratio of the GIS Object does not match the X/Y ratio of the requested *GISExtent*.

Currently, *GISStretch*=ON, and the displayed map image may be stretched.

Resize the GIS Object by dragging its control points with the mouse a few times and notice that the map gets stretched as the object's size changes.

In the *Properties* dialog, set *GISStretch* to OFF.

Resize the GIS Object again and notice that the aspect ratio of the map gets preserved. As a side effect, the map image may include an area which is slightly bigger than the area defined by the *GISExtent* attribute.

GISDataFile

This property points to an SDF file that describes the GIS data setup, including layers and fonts used to render the map. Currently it points to *<glg_dir>/map_data/sample.sdf*, a sample SDF file included in the GLG release.

Let's take a quick look at the content of a typical SDF file:

```
LAYER DIR=layers
FONT DIR=fonts

LAYER=earth earth.lif
LAYER=political polit_bound.lif
LAYER=states states.lif
LAYER=states_fill states_fill.lif
LAYER=grid grid.lif
LAYER=us_cities us_cities.lif

VECTOR FONT=roman romans.data
VECTOR FONT=script scriptc.data
IMAGE FONT=moderna MgOpenModernaRegular.ttf
IMAGE FONT=moderna_b MgOpenModernaBold.ttf
```

You can see that the format of the SDF file is very simple. It lists the layers and fonts that may be used to render the map. The following layers are defined in the SDF file and may be used for rendering the map: *earth, political, states, states_fill, grid* and *us_cities.*

Each layer line specifies a layer name used to reference the layer, such as "*earth*" or "*states*", and the layer information file (LIF) containing layer information, *earth.lif* and *states.lif* respectively.

An absolute or relative path may be used for LIF files; if a relative path is used, the LIF file is found in the directory specified by the *LAYER DIR* attribute. If the *LAYER DIR* is not defined, the filename is relative to the location of the SDF file.

In our example, all LIF files are using relative paths and are located in the *layers* directory of the dataset. The *layers* directory is located in the same directory as the SDF file.

The layer information file (LIF) specifies the GIS data used to render the layer; it also specifies the values of layer attributes used to render the layer. The format and parameters of the SDF and LIF files are described in greater detail in the *Map Data Setup* chapter.

The SDF and LIF files are ASCII text files, which provides great flexibility and makes the map server setup easy to customize.

GISMapServerURL

While the *GISDataFile* property points to an SDF file on a local filesystem and assumes that GIS data files are also located on a local filesystem, *GISMapServerURL* is used for mapping applications where map images are supplied over the web via a URL. The property points to a URL used for map query requests for the web-based Map Server. In this case, all Map Server data files, including the *SDF* and GIS data files are located on a web server.

If the web access option is used, the GIS Object transparently handles all interaction with the remote map server, issuing map server requests as needed. The local or remote web access mode is determined by the environment in which the GIS Object is used, and may be controlled by an application.

GISLayers

This is a string property that specifies a comma separated list of layers to be included in the map. While the *sample.sdf* file lists all layers, *GISLayers* string defines which layers should actually be rendered on the map. For example, the following GISLayers string will display the *earth*, *states* and *grid* layers:

earth,states,grid

The default setting of the *GISLayers* property is "*default*", which means that only the layers that are marked as being '*default*' are included. The layer is marked as default by setting the *DEFAULT* attribute of the LIF file to *1*.

The *earth* layer's LIF file, *earth.lif*, contains the following attribute setting:

```
DEFAULT=1
```

This makes the *earth* layer a default layer, automatically displayed when *GISLayers* contains '*default*' as a layer name.

As you can see on the map, only two layers, *earth* and *states,* are included by default. Other layers need to be added explicitly.

Try changing the *GISLayers* property to

default,grid

and notice that the *grid* layer has been added to the map in addition to *earth* and *states*.

Try changing *GISLayers* string to:

```
default,-states,grid
```

It would exclude the *states* layer from the default layer list, so that you will see only *earth* and *grid* layers on the map.

Note: Comma is the only separator that may be used in GISLayers; space is not allowed between the layer names. For example, the following setting would cause an error due to the space before the grid layer: *"earth,states, grid"*.

GISVerbosity

This property is used for testing purposes, to display detailed information about the current map request. It may have a value in the range [-3,10]. When it is not equal zero, the information about the generated map image is displayed on top of the map.

The negative values of the property activate display of timing information, which may be used for performance testing and benchmarks. The positive values activate display of information about layers, zoom factor, extent, used tiles and so on.

Set the *GISVerbosity* property to 1. The map needs to be regenerated to display the verbose output.

Click on the *Reset* toolbar button to reset the drawing and regenerate the map image.

The information about current zoom factor and total map rendering time will be displayed on top of the map.

Note: If verbose output is merging with the map colors making it hard to read, try changing the GIS Object's *FillColor* to black or white colors. This will cause the output to be displayed in a different color as well, making it easier to read. On Unix/Linux systems, the output is also printed to *stderr* and will appear on the terminal from which the Builder was started.

Set the *GISVerbosity* property to 4 and click on the *Reset* toolbar button again to reset the drawing.

More detailed information will be displayed, including the center and extent of the map, zoom factor, layer and tiling information.

Set the *GISVerbosity* property to -3 and click on the *Reset* toolbar button again.

The timing information will be displayed, including rendering time for each layer.

Set *GISVerbosity* property back to 0 and click on the *Reset* toolbar button.

Positioning of the GIS Object Inside the Viewport

When the GIS Object is used in a mapping application, it usually covers the whole area of the viewport that contains the map. This is achieved by positioning the GIS Object's control points in the opposite corners of the viewport. If the viewport has *Stretch=YES* (default setting) and there is no zooming, the extent of the visible area of the viewport is [-1000;+1000] coordinates in both X and Y directions.

The GIS Object's control points can be positioned at these exact coordinates by either resizing the object with the mouse using the *Snap To Grid* feature, or entering the control point coordinates directly, as shown below.

If not yet selected, click on the GIS Object with the mouse to select it. Notice the two control points shown in the opposite corners of the object. Dragging the control points with the mouse changes the object's size.

Press the *Hierarchy Up* button to go to the viewport level to see how the map image will look in an application. Notice that the map image covers only a portion of the viewport window, according to the GIS Object size that we previously defined.

Click on the *Hierarchy Down* button to go down into the viewport again. Click on the GIS Object to select it.

Drag one of the object's control points with the mouse and position it in a vicinity of the lower left corner of the drawing area. Drag the second control point and position it near the upper right corner of the drawing area.

Shift-click on the lower left control point with the left mouse button to bring its *Control Point* dialog. and set its coordinates to *"-1000 -1000 0"* in the *Value* text box.

Shift-click on the upper right control point and set its the coordinate values to *"1000 1000 0"*.

Close the *ControlPoint* dialog. The map now covers the entire drawing area.

Press *Hierarchy Up* button to go to the top viewport level. Notice that the map image now covers the entire viewport window.

Try resizing the *$Widget* viewport by dragging its control points with the mouse, and notice that the map inside it is resized with the viewport, always covering the whole area of the viewport.

Zooming and Prototyping in the Builder

A typical mapping application needs to support map zooming and panning functionality, including zooming in and out, as well as zooming to a region selected by the user with the mouse.

This functionality is transparently integrated in the GIS Object, which automatically issues a map request to display a new map image when the map is zoomed, panned or resized. In an application, a single API call is used to zoom and pan the map when a user requests it.

Zooming and panning operations allow to prototype the behavior of the GIS Object, including the automatic activation of layers and fallback layers based on the zoom level, as well as testing the setup of tiled data sets. They may also be used to define the initial map region to be displayed at the application start-up; when the drawing is loaded in the application, it will appear in the exact map zooming state that it had been saved.

The map may be zoomed and panned either in the *Edit* mode or *Run* mode.

Zooming and Panning in the Edit Mode

GIS Editing Mode

The map may be zoomed or panned by traversing down into the GIS Object to activate the *GIS Editing Mode*. In this mode, the zoom

and pan controls of the GLG Builder can be used to zoom and pan the map.

In our drawing, select *$Widget* viewport with the mouse.

Click on the *Hierarchy Down* button to go down into the viewport. Click on the GIS Object to select it.

Click on the *Hierarchy Down* button again to go down into the GIS Object.

Click on the *ZoomTo* button and define a rectangular area on the map with the mouse to zoom to. The map is automatically zoomed to the selected region.

Likewise, you may try clicking on the *ZoomIn* or *ZoomOut* button, as well as use the Drawing Area's scrollbars to pan the map.

The map may be also dragged with the mouse. Press and hold the *Ctrl* key, then click on the map with the mouse and move the mouse to drag the map.

For maps in the RECTANGULAR projection, the slider at the bottom of the Control Panel (*Rotate Z* slider) may be used to rotate the map.

On each zoom and pan request, a new map image is automatically generated by the GIS Object, and its *GISCenter* and *GISExtent* attributes get updated accordingly. These settings are persistent and will be saved with the drawing.

In the GIS Editing Mode you can also define objects to be drawn on the map in GIS coordinates, as described in the *Adding dynamic objects to the map* chapter on page 19.

Click on the *Hierarchy Up* button to go up to the viewport level.

GIS Zoom Mode

If the viewport's *GIS Zoom Mode* is activated, the map can also be zoomed and panned on the viewport level without traversing down into the GIS Object. In the *GIS Zoom Mode*, the zoom and pan controls of the Builder will zoom and pan the map inside the GIS Object instead of zooming and panning the viewport's drawing. This setting is persistent and needs to be done only once.

To set the *GIS Zoom Mode*, make sure the GIS Object is selected and use the *Set as Parent's Viewport GIS Object* option of the *Arrange* menu at the top of the Builder. The currently selected GIS Object (*MapObject* in our case) will be assigned as the GIS Object of its parent viewport, automatically setting the viewport's zoom mode to *GIS Zoom Mode*.

Try to use the zoom controls

to zoom in and out. Since the zoom mode of the viewport is set to the *GIS Zoom Mode*, these controls will zoom the map inside the GIS Object. Try scrolling the map using the Drawing Area's scrollbars, as well as dragging the map with the mouse using the *Ctrl-click-drag* sequence described in the previous section. Maps in the RECTANGULAR projection may also be rotated using the *Rotate Z* slider at the bottom of the Control Panel.

The *GIS Zoom Mode* setting is persistent, so it needs to be done only once. Once the viewport's *GIS Zoom Mode* is set, it will be stored with the drawing. Every time the viewport is zoomed or panned, either during editing, prototyping in the Builder or in an application at run time, the *GIS Zoom Mode* will be used to zoom or pan the map.

The *Unset GIS Object* option of the *Arrange* menu can be used if the regular zoom mode needs to be restored.

Zooming and Panning in the Run Mode

Once the viewport's *GIS Zoom Mode* has been set (as described in the previous section), map zooming and panning may be prototyped in the *Run Mode* of the GLG Builder to tests the run-time behavior of the map.

In the *Run Mode*, the map may be scrolled using integrated scrollbars of the viewport object, which are enabled by the viewport's *Pan* property. To zoom the map in the *Run Mode*, the viewport's *ZoomEnabled* property should be turned on to activate keyboard accelerators for zooming and panning.

The *ZoomEnabled* property controls only the use of the keyboard accelerators. At run time, an application can also zoom, pan and rotate the map programmatically using the *GlgSetZoom* method, which does not depend on the *ZoomEnabled* property setting.

Click on the *Hierarchy Up* button to go up to the *$Widget* viewport level. Display the *Properties* dialog for the *$Widget* viewport by selecting *Object, Properties* from the main menu, or using the right mouse button and selecting *Properties*.

In the *Properties* dialog, set *Pan=Pan XY, ZoomEnabled=YES* to activate scrollbars and keyboard accelerators, then close the dialog.

Click on the *Start* toolbar button, or select *Run, Start* from the main menu to start the *Run Mode*.

A dialog will appear prompting for an animation command. Erase the animation command, as we are not going to animate any resources.

Note: Instead of erasing the animation command, you may accept it and then ignore the error messages pertaining to the resources not being found.

Click on the *OK* button to close the dialog and start prototyping.

Try using the horizontal and vertical scrollbars to pan the map. In the RECTANGULAR projection, the vertical scrollbar will be enabled only when the map is zoomed in.

Map zooming, panning map rotation operations may also be performed via the keyboard bindings.

Since we are going to use keyboard events for zooming and panning, we need to set the keyboard focus to the map window by clicking on the map with the mouse. The viewport that has the focus will be zoomed or panned when the zoom and pan accelerators are used.

Press the *'i'* key to zoom in, use the '*o*' to zoom out.

To start the *ZoomTo* operation, press and hold the *Shift* key, then click and drag a rectangle on the map to select an area to zoom to. Alternatively, pressing the *'t'* key, then clicking and dragging may be used. When the mouse button is released, the map image will display the selected map region.

Note: Default key bindings for zoom and pan accelerators are documented on page 82 of the *Viewport* chapter of the GLG Builder User's Guide and Reference Manual. They include the keys for querying lat/lon coordinates and elevations of points on the map.

The map can be also scrolled by dragging it with the mouse. Press and hold the *Ctrl* key, then click and drag the map with the mouse.

Maps in the RECTANGULAR projection can also be rotated using the *'c'* and *'C'* keys.

Use the *'n'* key to reset the map to the whole world view.

The zoom and pan operations will change the GIS Object's *GISExtent* and/or *GISCenter* properties which will be stored in the drawing when the drawing is saved.

Click on the *Stop* toolbar button to stop the prototyping mode.

The zoom, pan and rotate operations may be also performed programmatically via the *GlgSetZoom* method of the GLG API.

GIS Object Resources

The properties of the GIS Object may be accessed as resources of the drawing without navigating down to the object level and selecting it. The resource mechanism is also used at application run-time to access GIS Object and its properties programmatically. Let's use the Builder's Resource Browser to familiarize ourselves with drawing resources.

Make sure that the top *$Widget* viewport is selected and select *Object, Object Resources* from the main menu (or press the *Object Resources* toolbar button) to bring the *Resource Browser*.

The resource browser will list all resources of the $Widget viewport. Since we named our GIS Object *"MapObject"*, the GIS Object will appear in the resource browser as the *MapObject* resource.

Double-click on the *MapObject* resource in the *Resource Browser* to open it up and view a list of the GIS Object's resources. Most of the properties of the GIS Object accessible through its Properties dialog are also accessible as resources through the Resource Browser.

Click on the *GISExtent* resource in the Resource Browser to bring its *Resource Object* dialog.

Notice that the value of the *GISExtent* resource was changed by the zooming operations performed in the prototyping mode. This value is defined in meters, since the current projection of the GIS Object is set to ORTHOGRAPHIC.

Let's change the object' projection to rectangular. Select the *GISProjection* resource and change the projection to RECTANGULAR in the *Resource Object* dialog.

Since the rectangular projection uses different units for *GISExtent*, the value of the *GISExtent* property will be reset to the default value *"360 180 0"* and the whole globe will be displayed in the rectangular projection.

Let's try zooming the map again and see how it affects the GIS Object's properties.

Press the *Start* toolbar button to start the prototyping mode, then click on *OK* to dismiss the animation command dialog.

Click on the map with the mouse to set focus into the map window.

Note: The zoom mode is automatically set to GIS Zoom Mode, since we used the Set as parent's viewport GIS Object option in the previous sections.

Press and hold the *Shift* key, then click and drag a rectangle for the *ZoomTo* operation, selecting the United States area to be displayed.

Click on the *Stop* toolbar button to stop *Prototyping* mode.

Click on the map to select the *$Widget* viewport, then press the *Object Resources* toolbar button to display resources of the *$Widget* viewport.

Double-click on *MapObject* resource in the *Resource Browser* to open it up, then click on the *GISCenter* resource to display its value.

Notice that the *GISCenter* value in the *Resource Object* dialog is defined in the lat/lon degrees and corresponds to the center of the map region we zoomed to. Keep in mind that the first coordinate of the value corresponds to the longitudinal value, and the second to the latitudinal value.

Click on the *GISExtent* resource in the *Resource Browser.*

Notice that the value of this resource is defined in the lat/lon coordinates as well, specifying the extent of the longitude and latitude in degrees of the displayed map region.

To summarize:

- The zoom and pan operations change *GISExtent* and *GISCenter* attributes to match the map region being zoomed to when prototyping *GIS Zoom* in the Run mode.
- The *GISCenter* attribute is always specified in *degrees*; the units of *GISExtent* depend on the projection -- it is specified in lat/lon *degrees* for the rectangular projection, and in *meters* for orthographic projection.
- When the lat/lon value is entered as a triplet of X, Y and Z coordinates, the X coordinate specifies longitude, the Y coordinate specifies latitude and Z coordinate must be set to 0.

The current zoom factor may be viewed as part of the verbosity output, if the value of the *GISVerbosity* resource is greater than zero.

Assuming that the *Resource Browser* dialog for the *MapObject* is still open, click on the *GISVerbosity* resource in the *Resource Browser* and set the resource's value to 1.

Press the *Reset* toolbar button to reset the drawing and notice the diagnostic output with the value of the zoom factor displayed on top of the map image.

Adding dynamic objects to the map

GIS Rendering Mode

The GIS Object can be used as a container that holds dynamic icons, polylines and other graphical objects that need to be drawn on top of the map. The objects are drawn on the map in the **GIS Rendering Mode**, in which the coordinates of the objects' control points are interpreted as lat/lon coordinates. This allows positioning of icons and lines on the map by defining their lat/lon coordinates directly, without any coordinate conversions. When the map is zoomed or panned, the objects on the map will be automatically adjusted to zoom and scroll with the map, with no application code required.

While the map rendered using the Map Server is static, the GLG objects rendered on top of the map are dynamic, enabling GIS applications to use such features as object dynamics, tooltips, object selection and interaction in conjunction with GIS maps.

GIS Editing Mode

The Graphics Builder supports the **GIS Editing Mode** for editing objects drawn on the map. In this mode, dynamic icons, polylines and other objects can be drawn or positioned on the map with the mouse in the lat/lon coordinates. The Builder automatically converts the mouse position to the lat/lon coordinates, which are stored in the object's control points. The Builder transparently handles GIS projections, which allows the user to draw polylines on top of the globe displayed in the ORTHOGRAPHIC projection. The GIS Object will automatically maintain the position of the graphical objects added to the map, displaying them at the specified lat/lon location when the map is zoomed or panned.

The *GIS Editing Mode* is activated by going down into the GIS Object using the *Hierarchy Down* button.

Since we are going to add other objects to the GIS Object, it makes sense to set its *HasResources* flag to ON, so that the added objects appear as resources of the GIS Object.

Assuming that the GIS Object is currently selected, bring the *Properties* dialog for it and turn the *HasResources* flag ON.

Press the *Hierarchy Down* button to go down into the GIS Object, activating the *GIS Editing Mode*.

Create a new polygon object by using the *Filled Polygon* button and clicking on the map a few times to define the polygon's points. Press *Escape* or the right mouse button to finish.

When the polygon's points are defined on the map with the mouse, the Builder automatically converts screen coordinates of the mouse to the lat/lon coordinates on the map in the current GIS projection. The lat/lon coordinates are stored in the polygon's control points and are saved with the drawing.

While the polygon object is still selected, select one of its control points using *Shift+Click*. In the *Control Point* dialog, notice that the coordinate values are shown as lat/lon coordinates. These values will be stored in the drawing when it gets saved and may be accessed programmatically via the GLG API.

Close the *Control Point* dialog.

Bring the *Properties* dialog for the polygon object we just created. In the *Name* field, enter *AreaPolygon* to name the object. Close the *Properties* dialog.

Click on the *Hierarchy Up* button to get to the GIS Object level.

Click on the *Object Resources* toolbar button to bring the *Resource Browser* with a list of resources of the GIS Object.

There is the *GISArray* resource, which is a group used by the GIS Object to hold all graphical objects added to the map.

Double-click on the *GISArray* resource in the *Resource Browser*. You will see the *AreaPolygon* resource, which is the polygon object we added to the map.

To add dynamic icons to the map, such as symbols representing an aircraft, use the *Reference* object with the *FixedSize* attribute set to *YES*. The size of the fixed size icons is defined in screen coordinates and remains unchanged when the map is zoomed. Refer to the *Reference* chapter on page 106 of the *GLG Builder User's Guide and Reference Manual* for details on the *Reference* objects and *FixedSize* attribute.

Close the *Resource Browser* Dialog.

Programmatic Access at Run-Time

At run-time, all objects added to the map may be accessed as subresources of the GIS Object's *GISArray* resource. For instance, in our example the following resource path may be used to change the edge color of *AreaPolygon*: *"MapObject/GISArray/AreaPolygon/EdgeColor"*.

At run-time, graphical objects may be added to the map via the GLG Extended API by adding them either to *GISArray* or directly to the GIS Object. The objects are positioned on the map by setting the lat/lon coordinates of their control points.

3. Map Data Setup

The GIS data and setup files used to generate map images are organized in a hierarchical directory structure, which is kept on a local filesystem for the library-based deployment of the Map Server. For the web-based deployment, where the map images are served over the web via a URL, the GIS data and setup files reside in a map data directory on the web server. The content of the map data directory is identical for either deployment option.

The following sections describe configuration files used by the Map Server.

Server Data File (SDF)

The top level directory of the GIS data contains a server data file (SDF), the main configuration file that defines the layers and fonts used to render the map. The name of this file is specified as a *GISDataFile* property of the *GIS Object*, or as the *-dataset* command-line option when the Map Server executable is used on a web server.

Sample map data provided with the GLG Toolkit is located in the GLG's *map_data* directory. Let's take a look at the content of this directory.

It contains a sample SDF file named *sample.sdf*, as well as *fonts* and *layers* directories. It also contains a few sample scripts that show the Map Server's command-line options needed to generate and save map images.

The following lists a partial content of the *sample.sdf* file:

```
#GLM ROOT DIR=/usr/local/glg/map_data
LAYER DIR=layers
FONT DIR=fonts

# Layer string aliases used by gis and airtraffic control demos
ALIAS=default_gis "earth,states"
ALIAS=default_air "political_filled,states_dcw,grid50,outline"

# Layers
LAYER=earth earth.lif
LAYER=earth_fallback earth_fallback.lif
LAYER=states states.lif

# Sample DCW layers
LAYER=political polit_bound.lif
LAYER=political_filled polbnd_polbnd.aft.gvf.lif
LAYER=states_dcw states_dcw.lif
LAYER=states_fill states_fill.lif

# Sample grid layers with different colors
LAYER=grid grid.lif
LAYER=grid30 grid30.lif
LAYER=grid50 grid50.lif
LAYER=grid70 grid70.lif

# Outline of the whole globe
LAYER=outline outline.lif

# Sample us cities layer
LAYER=us_cities us_cities.lif

# Fonts
VECTOR FONT=roman romans.data
VECTOR FONT=script scriptc.data
IMAGE FONT=moderna MgOpenModernaRegular.ttf
IMAGE FONT=moderna_b MgOpenModernaBold.ttf
IMAGE FONT=moderna_i MgOpenModernaOblique.ttf
IMAGE FONT=moderna_ib MgOpenModernaBoldOblique.ttf
```

The *LAYER DIR* and *FONT DIR* parameters point to the directories where the layer information files (LIFs) and font files are located, *layers* and *fonts* respectively.

The *GLM ROOT DIR* parameter defines where to look for the data files that are defined using relative paths. All files needed for the Map Server are relative to this directory, unless an absolute path is used. If this attribute is omitted, all files are relative to the location of the SDF file itself. For the example above, the *layers* directory defined by the *LAYER DIR* parameter is relative to the directory where the SDF file is located, since *GLM ROOT DIR* is commented out.

The *sample.sdf* file also lists all available layers that may be included in the map image. The *LAYER* token is used to describe each layer as follows:

```
LAYER=<layer_name> <.lif file>
```

where

<*layer_name*> is an arbitrary name given to a layer. This name is used to identify the layer when supplying the list of layers to be displayed, via the *GISLayers* property of the GIS Object or the layer string of a map server request.

<*.lif*> file is a *layer information file* (LIF), a configuration file that contains layer attributes, such as colors, zoom factor, transparency and so on. It also contains a name of the file with the GIS data used to render the layer.

The *ALIAS* parameter defines layer name aliases that may be used for convenience. An alias associates a single name, such as *default_gis* or *default_air*, with a list of several layers. When an alias name is included into the list of layers to be displayed, all layers listed in the alias will be displayed on the map. The alias also provides a logical layer name for an application, making it possible to change the layers displayed on the map by changing the layer names in the alias, without a need to change the application source code.

The SDF file also lists available fonts provided in the *fonts* directory. The map server may also use any True Type font available on the system.

Layer Information File (LIF)

Let's take a look at the content of the *states.slf* LIF file for the *states* layer, located in the *map_data/layers* directory:

```
TYPE=VECTOR
TRANS TYPE=OPAQUE
ALPHA=0.5
FILENAME=states.gvf
EDGE COLOR=0.89 0.89 0.89
DEFAULT ON=1
```

It includes the following layer attributes:

TYPE=VECTOR defines the layer as a vector layer, rendered using vector data. For raster layers, such as the *earth layer, TYPE* attribute is set to *IMAGE*.

TRANS TYPE=OPAQUE defines layer blending type to be *OPAQUE*. The layer will be fully opaque only if the layer's transparency is turned off by setting the layer's ALPHA attribute to 1 (default). If the value of the *ALPHA* attribute is between 0 and 1, the layer will be semi-transparent. For example, *ALPHA=0.5* makes the layer 50% transparent. If ALPHA is set 0, the layer will be completely transparent. OPAQUE is the most often used blending type, other blending types are used only for special types of rendering, such as rendering shaded relief images or transparent weather layers.

FILENAME=states.gvf specifies the file containing GIS data for rendering the layer. The data file is located in the same directory as *states.lif*. The format of the file (GVF) is detected from its extension, but it may also be specified explicitly using the *VECTOR FORMAT=GVF* directive. Refer to *the GIS Data Formats for Vector Layers* chapter for more information on the *GVF* and other supported vector formats.

EDGE COLOR=0.89 0.89 0.89 specifies the color for rendering the *states* layer outline - light grey. The color is defined using the OpenGL style, with RGB values in the range from 0 to 1.Since the *FILL TYPE* attribute is not specified, the default *EDGE* value is used, and the layer is rendered as an outline with no fill.

DEFAULT ON=1 defines the layer as a default layer, which will be automatically activated if the *'default'* layer name is included in the layers string of the map request. If this attribute is equal to zero or omitted, the layer is not turned on by default, and the layer's name should be explicitly included into a layers string of a map request to display the layer.

Let's change the states layer to add fill, define the fill color and change the line width of the outline.

Edit the *states.lif* file, located in the GLG's *map_data/layers* directory and add the following three lines:

```
FILL TYPE="FILL AND EDGE"
FILL COLOR=0.89 0.89 0.63
LINE WIDTH=3
```

Save the changes.

In the GLG Builder, click on the *Reset* toolbar button to reset the drawing. This will regenerate the map image using the new *states.lif* file. The *states* layer is now rendered with the brown fill color, light grey edge color and wider line width, reflecting the new settings in the *states.lif* file.

Edit *states.lif* file again to undo the changes. You may just comment out the lines we just have added using the *'#'* character as follows:

```
TYPE=VECTOR
TRANS TYPE=OPAQUE
ALPHA=0.5
FILENAME=states.gvf
EDGE COLOR=0.89 0.89 0.89
DEFAULT ON=1
#FILL TYPE="FILL AND EDGE"
#FILL COLOR=0.89 0.89 0.63
#LINE WIDTH=1
```

Save the changes. In the GLG Builder, click on the *Reset* toolbar button again to redraw the map. The *states* layer should appear as edge-only layer again.

The *states.lif* file contains only a subset of all available layer attributes. Refer to the *Configuration Variables* chapter on page 38 of the *GLG Map Server Reference Manual* for a complete list of layer attributes.

Using Zoom Factor Limits for Automatic Layer Switching

Individual map layers may be switched on/off automatically by specifying zoom factor limits in the layer's *.lif* file.

Let's enable the *us_cities* layer to display major US cities on the map.

In the GLG Builder, select the *$Widget* viewport by clicking on it.

Display viewport's resources by clicking on the *Object Resources* toolbar button.

Double-click on the *MapObject* resource in the Resource Browser to open it up and view its resources.

Click on the *GISLayers* resource and change its value to *'default,us_cities'*.

Click on the *GISVerbosity* resource and change its value to 1.

Close the *Resource Browser*.

Click on the *Reset* toolbar button to redraw the map.

Verbosity information on top of the map image will display the current value of the *Zoom factor*.

Notice that although the *us_cities* layer is included in the *GISLayers* resource, the city markers are not displayed on the map.

Start the prototyping mode by clicking on the *Start* toolbar button.

Click on the map with the mouse to set the keyboard focus into that window.

Press the *'g'* key to enable the *GIS Zoom* mode.

Press and hold the *Shift* key, then click and drag a rectangle on the map to define a map region to zoom to. Zoom on, let's say, New England or California. Notice the value of the *Zoom factor* displayed as part of the verbosity information on top of the map image.

If *Zoom factor* is less than 10., repeat the *ZoomTo* operation to zoom further, until the *Zoom factor* becomes greater than 10. You should see large US cities appearing on the map when the *Zoom factor* value exceeds 10.

Press the *'o'* key to zoom out -- city markers will disappear once *ZoomFactor* becomes less than 10.

The *us_cities* layer is enabled or disabled automatically, based on the map zoom level, which is achieved by the following setting in *us_cities.lif* file:

```
MIN ZOOM=10.
MAX ZOOM=-1.
```

A combination of *MIN ZOOM* and *MAX ZOOM* attributes determine the zoom factor range within which the layer is enabled. In our case, the *us_cities* layer will be enabled when zoom factor > 10 (zoom factor > MIN ZOOM). Since the maximum zoom is set to -1, there is no limit to the maximum zoom level.

This feature is very convenient for controlling the level of details to be displayed on the map. More detailed layers may be turned on at high zoom levels, while other layers may be turned off.

The *us_cities* layers also uses attribute thresholds attached to the *MIN ZOOM* attribute to control which cities and towns are displayed as the zoom factor increases. As the map is zoomed in further and further, smaller and smaller cities and towns get displayed, as directed by the *popul_to_zoom.tt* threshold table attached *MIN ZOOM*. While the global *MIN ZOOM* setting controls the zoom limit for the whole layer, the threshold table defines the minimum zoom limit for each city and town marker depending on it population attribute (defined in the GIS data file as an attribute with index 1).

Click on the Stop toolbar button to exit the prototyping mode when you are done experimenting with various zoom factors.

GIS Data Formats for Vector Layers

For vector data, the Map Server uses an open specification *GVF* vector data format, which is a simple vector format optimized for run-time performance. Files in this format usually have the *.gvf* extension. The Map Server also supports *shapefile* format, which may be used directly or converted to the *GVF* format for efficiency. A shapefile conversion utility is provided as the *-shp2gvf* option of the map server executable. Refer to the the *Tools and Utilities* chapter of the *GLG Map Server Reference Manual* for information on command-line options.

Other vector formats are supported using filters, which are invoked by the Map Server at run time to read vector data in custom formats. While the vector data filters can be used by the Map Server at run-time, the data conversion may be performed at the setup time to optimize the run-time performance. Filters for most common GIS vector formats - shorelines, CIA World DataBank (WDB), etc. - are provided in the *map_server/convert* directory. Please contact Generic Logic if you need assistance.

The rest of the vector formats may be imported using *gdal* converters, by converting them to the *shapefile* format first.

Data sets for the commonly used GIS data sets, such as the *Digital Chart of the World* (DCW / VMAP0) and *US Census Tiger Data*, are available in the GVF format from Generic Logic, in the ready-to-use form that includes all setup files.

Image Layers and Tiling

So far, we have been examining vector layers, such as *states* and *us_cities*, which used vector GIS data for rendering.

There are also *raster*, or *image* layers which are rendered using raster images, in either *TIFF*, *JPEG* or *PNG* format. This layers are defined with *TYPE=IMAGE* in the LIF file.

The *earth* layer is a good example of an image layer. Its LIF file, named *earth.lif* and located in the *map_data/layers* directory, has the following content:

```
# The earth layer
TYPE=IMAGE
IMAGE FORMAT=JPEG
TRANS TYPE=OPAQUE
# this is the filename or template for tiling
FILENAME=earth_tiff/earth_%_%.jpg
# A fallback layer in case the number of tiles exceeds
# memory requirements
FALLBACK LAYER=earth_fallback
MAX TILES=40 WIDTH=8192
HEIGHT=4096
NUM TILES X=20
NUM TILES Y=20
MIN LON=-180
MAX LON=180
MIN LAT=-90
MAX LAT=90
MIN ZOOM=-1.
MAX ZOOM=-1.
DEFAULT ON=1
```

IMAGE FORMAT provides information about the format of the image data file. This parameter is optional, if it is omitted, the format is determined by the filename extension. JPEG, TIFF and PNG formats are natively supported for the image data.

FILENAME specifies the name of the image file that is used to render the layer. If the layer's image is tiled, which is the case for the *earth* layer, the FILENAME provides a filename pattern for layer's tiles. The pattern must have two *'%'* characters which are replaced with the tile row and column to construct the tile name. In our case, *FILENAME=earth_tiff/earth_%_%.jpg* specifies that the tiles are located in the *earth_tiff* subdirectory of the layers directory and named *earth_0_1.jpg, earth_0_2.jpg, earth_0_3.jpg,* and so on.

The tiling mechanism is used to optimize performance when rendering layers with large amount of data. It would be very inefficient to process all layer data when only a small area of the layer is displayed. For example, the earth image has resolution of 8192 x 4096 pixels, which would require in excess of 100 MB of RAM to load it. Loading the entire image when only one country is shown on the map would waste a lot of memory and make Map Server unnecessarily slow. With tiling, only the tiles needed to render a requested map area are loaded, minimizing memory and CPU consumption and enhancing performance.

The tiles for the earth image were created using a tiling utility, which is provided as a part of the Map Server executable and is invoked using the *-tile* option. For example, the following command will split the *earth.tif* file into 20 x 20 rectangular tiles:

```
GlmMap -tile -image -filename earth.tif
           -template earth_tiff/earth_%_%.jpg
           -num-x-tiles 20 -num-y-tiles 20
```

GlmMap in the above command is the Map Server executable located in the GLG's *bin* directory; when used with the *-tile* option, it serves as a tiling utility.

The *-image* option specifies that image tiling is performed, as opposed to tiling vector data.

The *-template* option specifies a pattern for tile filenames. The used pattern, *earth_tiff/earth_%_%.jpg*, specifies that the generated tile files will be saved in the *earth_tiff* subdirectory using the *earth_%_%.jpg* pattern for filenames. The tile filenames will be formed by replacing the *'%'* characters in the pattern with the zero-based row and column indices of each tile.

The *num-x-tiles* and *num-y-tiles* options define the number columns and rows of tiles in the X and Y direction.

Note: In some datasets GIS data are grouped in non-rectangular categories. For example, the US Census Tiger dataset organizes GIS data by state and county using a tree-like directory structure. For such datasets the rectangular tiling becomes unusable, and hierarchical directory-based tiling option should be used. The hierarchical tiling is supported for both image and vector layers, and utilities for automatic generation of setup files for such datasets are provided as a part of the Map Server executable activated using the *-slf* option. Refer to the *Tools and Utilities* chapter of the *GLG Map Server Reference Manual* for more information.

The *MAX TILES* attribute specifies the maximum number of tiles (40 in this case) the layer can use for rendering. If the number of tiles needed to display a requested map region exceeds *MAX TILES*, the layers will not be drawn, and a fallback layer specified by the *FALLBACK LAYER* token will be drawn instead to avoid excessive CPU and memory consumption. If the fallback layer is not specified, the layer will not be drawn.

FALLBACK LAYER=earth_fallback defines the *earth_fallback* as a fallback layer to be drawn instead of the layer when the number of tiles needed to draw the layer exceeds the maximum number of tiles specified with MAX TILES. The *earth_fallback* is just another map server layer that uses an earth image of a smaller resolution, *earth_smaller.jpg*. The smaller resolution image is displayed when large area of the earth needs to be rendered, and using the high-resolution tiled image would require too many tiles to be loaded.

In some cases, one fallback layer may not be enough, and *FALLBACK LAYER* attribute may be used recursively, to create a chain of multiple layers, either tiled or untiled.

For example, the original *earth.tif* file uses a high resolution image with about 8000x4000 pixels. It was split into 20*20=400 rectangular tiles, with 400x200 pixels each. These tiles are used for the main *earth* layer, with the *earth.lif* LIF file.

We could also setup an intermediate fallback layer, called *earth_fallback_medium*, that would use a medium resolution image with 4000x2000 pixels split into 10*10=100 tiles, with 400x200 pixels each.

Finally, the *earth_fallback_small* layer would use a single untiled image of a smaller resolution, 2000x1000 pixels.

The LIF files for these layers would have the following content:

earth.lif for the *earth* layer:

```
#The main earth layer, high resolution.
FILENAME=earth_tiff/earth_%_%.jpg
FALLBACK LAYER=earth_fallback_medium
MAX TILES=15
```

earth_fallback_medium.lif for the *earth_fallback_medium* layer:

```
#The first fallback, medium resolution
FILENAME=earth_fallback_medium/earth_%_%.jpg
FALLBACK LAYER=earth_fallback_small
MAX TILES=20
```

earth_fallback_small.lif for the *earh_fallback_small* layer:

```
#The last fallback in the chain, no further fallbacks,
#smaller resoultion image.
FILENAME=earth_smaller.jpg
```

Once the number of requested tiles for each layer exceeds the number defined by *MAX TILES* attribute, its fallback layer gets activated. In the above example, when the number of requested tiles for the *earth* layer exceeds 15, its *earth_fallback_medium* fallback layer gets activated; when the number of requested tiles for this layer exceeds 20, the last fallback layer, *earth_fallback2,* will be used.

Transparency and Blending Types

Simple Transparent Layers

The *TRANS TYPE* token of the LIF file defines blending function (blending type) used for rendering the layer. The simplest blending 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 by setting the layer's APLHA attribute to 1 (default value), it will be fully opaque.

To make the whole layer transparent, the *ALPHA* attribute may be set to a value in the range from 0 to 1 as follows:

```
ALPHA=<value>
```

For layers with the *OPAQUE* blending type, the alpha value of 0 makes the layer completely transparent and 1 makes it completely opaque. Other values in the range make the layer semi-transparent.

For example, *ALPHA=0.5* makes the layer 50% transparent.

Layers with Transparent Background

In cases when an image has a background of a particular color which needs to be rendered as transparent, the *TRANSPARENT COLOR OPAQUE* blending type may be used.

The following parameters in the LIF file would remove green background from the image, while rendering pixels that have other colors:

```
TRANS TYPE=”TRANSPARENT COLOR OPAQUE”
#Define transparent color as green
TRANS COLOR=0. 1. 0.
#Define the color precision for JPEG images
TRANS COLOR PRECISION=0.1
```

The *TRANS COLOR* attribute defines which color to render as transparent in the *TRANSPARENT COLOR OPAQUE* rendering mode. In our example, the transparent color is green.

TRANS COLOR PRECISION defines a maximum distance in RGB color space (in the range from 0 to 1) used to determine a color’s transparency. This parameter is used with JPEG images to overcome artifacts of JPEG’s lossy compression, which slightly changes pixel colors. 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.

When possible, we recommend using TIF format for transparent images, so that color precision would not be an issue. For TIF images, TRANS COLOR PRECISION parameter may be omitted.

Special Blending Types

There are cases when more complex blending is required. For example, for transparent cloud layers the brightness of the pixel indicates the thickness of the cloud cover. A black pixel in the cloud image specifies no cloud cover, while a white pixel specifies full cloud cover. The *ALPHA WHITE OPAQUE* and *ALPHA BLACK OPAQUE* blending type may be used to render such layers with transparency that depends on the pixel value.

For the *ALPHA WHITE OPAQUE* blending type, the brighter the color, the more opaque it is. For the *ALPHA BLACK OPAQUE* blending type, the darker the color, the more opaque it is. The alpha parameter can still be used with these two transparency types to alter the transparency of completely opaque pixels.

If the blending type is set to *XOR*, the color will be XOR’ed with the pixel in the image. This causes lines to stand out, and may be useful for grid layers, where it is desirable to have grid lines that always contrast with the background.

The *BUMP MAP* blending type may be used for rendering shaded relief images.

Markers and Labels

Vector map layers may contain point features that need to be rendered using icons, or *markers*. The markers may also have labels displayed next to them.

The *MARKER ICON TYPE* parameter of the layer defines the icon type, which may include a combination of predefined marker types, such as *CIRCLE_EDGE, CIRCLE_FILL, BOX_EDGE, BOX_FILL, CROSS*, etc.

For example, the *us_cities* layer displays markers that represent US cities and towns using the following parameters in its LIF file (*us_cities.lif*):

```
MARKER ICON TYPE=”CIRCLE_FILL,CIRCLE_EDGE,DOT”
MARKER ICON SIZE=10
MARKER MARGIN=3
MARKER EDGE COLOR=1. 1. 1.
MARKER FILL COLOR=1. 0. 0.
```

The marker type is defined as a combination of predefined types and will be rendered as a round icon with a fill, edge and a dot in the center.

If predefined types are not sufficient to display an icon, a custom icon may be used in a form of a TIF or JPEG image. In this case, the following marker parameters may be used:

```
MARKER ICON TYPE=CUSTOM_ICON
ICON FILE=<icon filename>
```

For example, airports may be displayed on the map as custom symbols, using a custom *jpeg* file:

```
MARKER ICON TYPE=CUSTOM_ICON
ICON FILE=airport_symbol.jpg
```

The *TRANSPARENT COLOR OPAQUE* blending type described above may be used for layers that need to use icons with transparent background.

The *us_cities* layer also displays labels next to each city or town marker. It is achieved by using the following lines in the layer’s LIF file:

```
# TrueType font sample
FONT=moderna_b
FONT SCALE=10
FONT TYPE="MAPPED FLAT"
FONT ANGLE=0.
FONT ANCHOR=HLEFT VCENTER
LABEL EDGE COLOR=1. 1. 0.
# Outline around the text is used to make the label more readable.
LABEL STYLE=OUTLINE
LABEL OUTLINE COLOR=1. 1. 1.
# Simple label using place name (attr index=0):
LABEL FORMAT=<0>
# Enable layout negotiation
LAYOUT TYPE=GLOBAL
LABEL PRIORITY=0
```

The *FONT* and *FONT SCALE* parameters define the type and size of the font used to render the labels.

The *FONT TYPE* parameter specifies that the labels have to be rendered using a fixed size font that does not scale or move with the map when the map is zoomed in or zoomed out.

FONT ANGLE may be used to specify an optional text rotation angle.

The *FONT ANCHOR* parameter specifies anchoring of the text label relatively to the marker.

The *LABEL STYLE* parameter requests an outline to be drawn around each character to improve text readability on top of a complex map background. *LABEL OUTLINE COLOR* specifies the color of the outline.

The *LABEL FORMAT* parameter is used to specify what needs to be rendered in the label. The *LABEL FORMAT=<0>* setting requests the name of the city or town (the attribute of the point feature with index=0) to be displayed in the label. The attribute index is specified inside angle brackets.

To use attributes of GIS features, one has to know what attributes are provided in the GIS data file. Since the population attribute is stored in the data file as an attribute with index=1, the following setting may be used to display a place's name and population:

```
LABEL FORMAT="<0>, population: <1%.0lf>"
```

The place's population will be displayed using the "*%.0lf*" C format, which is appended to the attribute index with no separator characters.

LAYOUT TYPE=GLOBAL requests the label layout negotiation to be performed on labels to make sure the labels do not obscure each other. The label layout negotiation will be performed globally for labels in all layers.

LABEL PRIORITY=0 defines the highest label priority for the labels in this layer. This makes sure that these labels will never be overwritten by labels from a different layer with lower priority. In the actual *us_cities.lif* LIF file, an attribute threshold is also attached to the label priority resource of the layer to vary the label's priority depending on the city or town population. The next chapter shows how to attach attribute thresholds.

Dynamic Attribute Thresholds

In some cases, a finer control of the attribute value in LIF file is required. For example, the `MARKER ICON SIZE` for the *us_cities* layer is set to 10, which means that all city and town markers will be 10 pixels in diameter. It would be nice, however, to vary the marker size depending on the city's population, displaying large markers for large cities, smaller markers for medium cities, and so on. In addition, markers could use different colors to visually differentiate large cities from small towns.

To accomplish this, a threshold table may be attached to the marker size attribute of the layer to change the city's marker size depending on the population attribute of a city in the vector data file. Another threshold may be attached to the layer's fill color attribute to vary colors for cities and towns with different populations.

This could be accomplished in a different way, by splitting the layer data into several layers based of values of their attributes using provided vector data splitting utility (the *-split* option of the Map Server executable). For example, large cities may be extracted into one layer, medium cities into a second layer, and small towns into the third layer. Each of the layers may use different rendering attributes (color, marker size, etc.) and may be independently activated. This option will also yield better run-time performance than the attribute thresholds. However, it may not be practical when the thresholds for splitting data may change, in which case it is more convenient to use the threshold tables which can be easily modified without a need to re-run the splitting utility.

In the *us_cities.gvf* data file that is used for the *us_cities* layer, the population is stored as an attribute with index 1. The following LIF directives may be used to attach threshold tables to vary marker color and size depending on the place's population attribute:

```
ATTR MAP= 1 MarkerFillColor popul_to_color.tt ABS
ATTR MAP= 1 MarkerSize popul_to_size.tt REL
```

The first parameter of the *ATTR MAP* directive is the index of the population attribute in the data file, which is 1.

The second parameter, *MarkerFillColor*, is the name of the layer resource, which resembles the name of the corresponding *MARKER FILL COLOR* LIF parameter, but uses a different naming convention to differentiate LIF directives from layer resource names used at run time. A list of all layer resource names may be found in the the *Layer Resources* chapter of the *GLG Map Server Reference Manual*.

The third parameter, *popul_to_color.tt*, provides a filename of the threshold table.

The last parameter specifies a mode for applying thresholds: absolute (*ABS*) or relative (*REL*). In the absolute mode, the value from the threshold table replaces the value of the layer's resource. In the relative mode, the layer's resource value is multiplied by the threshold value, allowing the layer attribute to control the size of all city markers, while the threshold value controls the size of each city marker relative to the value defined in the LIF file.

Let's take a look at the content of the *popul_to_color.tt* file located in the GLG's *map_data/layers* directory*:*

```
INPUT DATA TYPE= GLM_D
OUTPUT DATA TYPE= GLM_G
0       1.  1.  1.
50000   1.  1.  0.
100000  0.  1.  0.
500000  1.  0.  0.
```

The *INPUT DATA TYPE* specifies the data type of the input attribute used to control the threshold table mapping. Since the population attribute uses double value, the matching GLM_D (double) type is used.

The *OUTPUT DATA TYPE* specifies the data type of the output. Since the threshold table maps the value of the population attribute to a marker color, the *GLM_G* (RGB triplet) type is used.

The remaining lines of the threshold table list the table's thresholds. For each threshold, an input value of `INPUT DATA TYPE` is listed first, following the output value of `OUTPUT DATA TYPE`.

According to this table, large cities with a population of 500K or more will be rendered using the red color, cities with population less than 500K but more or equal 100K will be displayed in green color, cities and towns with size from 100K to 50K will be displayed using yellow markers and the rest of the towns with population of 50K or less will be rendered using white markers.

The output value in this table, the value of *MarkerFillColor* resource, is specified as an absolute value of the marker's fill color.

Now let's take a look at the *popul_to_size.tt* file, located in the GLG's *map_data/layers* directory:

```
INPUT DATA TYPE= GLM_D
OUTPUT DATA TYPE= GLM_D
0       0.35
10000   0.45
50000   0.62
100000  0.8
500000  1.0
```

In this threshold table, the input value which is the city's population of type *GLM_D* (double), is mapped to the output value, a marker size of the marker representing a city (*GLM_D* as well). The output value will be used as a relative value of the marker size, since the *REL* parameter was used in the LIF's *ATTR MAP* directive.

According to this threshold table, the resulting value of the *MarkerSize* resource for towns with population less than 10K will be calculated as *MARKER ICON SIZE * 0.35*. Since *MARKER ICON SIZE=10* in the *us_cities.lif* LIF file, the towns with population less than 10K will be represented by small markers with the marker size of 10 * 0.35 = 3.5 pixels, which will be rounded to the integer value of 3.

The *us_cities* layer also attaches a threshold table to the *MinZoom* resource of the layer to display more cities and towns as the map is zoomed in. This threshold table is located in the *popul_to_zoom.tt* file located in the *map_data/layers* directory and is similar to the marker size threshold table described above.