# Table of Contents

## Chapter 1: Introduction
- Product Contents ................................................................. 2
- Product Audience ............................................................... 2
- Features ................................................................................. 2
- Benefits .................................................................................. 3
- Programming Advantages .................................................... 4
- What Can You Do with MapXtreme Java? ................................. 6
- MapXtreme Java Documentation Set ........................................ 6
- New in MapXtreme Java Edition 4.7 ........................................ 7
- Migrating MapXtreme Java 4.5 Applications to 4.7 ....................... 9

## Part A: Using MapXtreme Java ............................................. 10

### Chapter 2: Getting Started ................................................ 11
- System Requirements .......................................................... 12
- Installation .............................................................................. 12
- Uninstalling MapXtreme Java ................................................ 19
- Base Map Data Installation .................................................... 19
- Uninstalling Base Map Data .................................................... 24
- Tomcat Deployment Environment ........................................... 25
- Deploying MapXtreme in Other Web Server Environments .......... 25
- Installed Components .......................................................... 31
- Installing Fonts ...................................................................... 32
- Installing on UNIX without a Graphics Card ............................ 33
- Configuration Troubleshooting ............................................... 36
- Software Copy Protection ....................................................... 38

### Chapter 3: Planning Your Application .................................. 39
- Web Deployment ..................................................................... 40
- Deployment Options ............................................................. 41
- MapXtreme Java Overview ..................................................... 42
- Design Considerations .......................................................... 44

### Chapter 4: Mapping Concepts ............................................ 45
- Organizing Your Data and Maps: An Overview of Tables .......... 46
- Map Definitions ..................................................................... 47
- Map Features ........................................................................ 48
- Renditions .............................................................................. 48
### Chapter 5: Managing MapXtreme Java

- MapXtreme Java Manager ........................................ 51
- Map Definitions Panel ........................................... 52
- Web Applications Panel ......................................... 53
- Named Resources Panel .......................................... 53
- Connections Manager Panel ..................................... 54
- Adding Services to MapXtreme Java Manager ................. 54
- Managing Data with MapXtreme Java Manager ............... 55
- Geosets and Map Definitions ................................... 56
- Loading Existing Map Definitions .............................. 57
- Creating a Map Definition ...................................... 58
- Saving a Map Definition ......................................... 62
- Manipulating Layers with Map Tools .......................... 64
- Layer Control ...................................................... 69
- Display Options via Layer Control ............................. 74
- Labeling Options via Layer Control ........................... 77
- Managing Named Resources ..................................... 82

### Chapter 6: Web Application Builder

- Using the Web Application Builder ............................ 87
- Custom JSP Tag Library .......................................... 90

### Chapter 7: MapXtreme JavaBeans

- Overview ............................................................ 93
- VisualMapJ Bean .................................................. 93
- MapTool Beans .................................................... 94
- MapToolBar Bean ................................................ 95
- LayerControl Bean ............................................... 96
- AddTheme Bean .................................................... 98
- Creating an Application with MapXtreme JavaBeans ..... 101
- SimpleMap Applet Sample ...................................... 103
- Including MapTools in Your Applet or Application .... 104
- Configuring and Controlling the Standard Map Tools .. 106
- Creating Custom MapTools .................................... 109

### Chapter 8: Writing Your Own Servlets

- Servlet Overview ................................................ 114
- Working with Servlets .......................................... 114
- Thin Client/Thick Server ....................................... 114
- Thick Client/Thin Server ....................................... 116
- Medium Client/Medium Server ................................ 118
- MapXtremeServlet Data Flow .................................. 121
- Servlet Forwarding ............................................. 123
**Part B: Introduction to the MapJ API**  ................................................................................. 128

**Chapter 9: MapJ API** ................................................................................................................. 129

- MapJ Object .............................................................................................................................. 130
- Creating Your First Map ............................................................................................................ 130
- Controlling the Map View ...................................................................................................... 133
- Adding a Layer ......................................................................................................................... 135
- Saving Your Map Programmatically ...................................................................................... 135
- Named Maps ............................................................................................................................ 137
- Beyond the Basic Map ............................................................................................................. 139

**Chapter 10: Mapping In Layers** ............................................................................................... 141

- Maps as Layers ......................................................................................................................... 142
- The Layers Collection: Building Blocks of Your Map .............................................................. 142
- Defining a Layer with a Data Provider ..................................................................................... 143
- Creating Your Own Data Provider ............................................................................................ 146
- Caveats for Defining JDBC Layers ........................................................................................... 146
- Adding a Layer to a MapJ Layers Collection ............................................................................ 147
- Data Binding .............................................................................................................................. 150
- Annotation Layers ..................................................................................................................... 152
- Analysis Layers ........................................................................................................................ 153
- Named Layers .......................................................................................................................... 153
- Methods of the Layers Collection ........................................................................................... 155
- Zoom Layering .......................................................................................................................... 159
- Generating Labels For a Layer .................................................................................................. 160
- Raster Images ........................................................................................................................... 161
- Considerations for Importing Raster Images ........................................................................... 162
- Image IO Data Provider ............................................................................................................ 163
- Raster Style Tags ....................................................................................................................... 164
- Grid Images in MapXtreme Java ............................................................................................... 166

**Chapter 11: Rendering Considerations** ................................................................................... 168

- MapXtremeImageRenderer ........................................................................................................ 169
- LocalRenderer ........................................................................................................................... 169
- EncodedImageRenderer ............................................................................................................ 169
- Rendering Additional Layers with a Named Map .................................................................... 169
- Animated Images ....................................................................................................................... 171
- Raster Output Formats ............................................................................................................. 173
- SVG Output ............................................................................................................................... 173
- WBMP Support .......................................................................................................................... 175
- Composite Renderer ................................................................................................................. 177
- Progressive Rendering .............................................................................................................. 179
- Intra-Servlet Container Renderer ............................................................................................. 180
## Chapter 12: Accessing Remote Data
- Named Connections .......................... 183
- Named Connection Pooling ............... 183
- How to Create Named Connections ........ 184
- Accessing Pooled Connections ............ 186
- Managing Named and Direct Database Connections .................. 187

## Chapter 13: Features and Searches
- The Feature Object ......................... 189
- Creating Features Using FeatureFactory .... 193
- FeatureSet Collection ...................... 196
- Searching .................................. 197
- Search Methods ............................ 198
- Searching Layers Defined by SQL Queries .. 202
- Feature Editing ............................ 204
- Editing an Annotation Layer ............... 204
- Editing a JDBC Table Layer ............... 205
- Editing a Tab Layer .......................... 207

## Chapter 14: Labeling and Renditions
- Labeling Overview .......................... 209
- Thematic Labeling ......................... 209
- Per-Feature Label Renditions ............... 210
- LabelProperties Class ...................... 211
- Merging Label Properties .................. 216
- Label Code Example ....................... 217
- Rendition Overview ....................... 220
- Rendition Properties ...................... 220
- Named Renditions ......................... 232
- Per-Feature Renditions .................... 234

## Chapter 15: Theme Mapping and Analysis
- Thematic Mapping Overview ............... 236
- General Objects for Themes ............... 237
- OverrideTheme ............................. 238
- RangedTheme ............................... 238
- SelectionTheme ............................ 242
- IndividualValueTheme ...................... 243
- Theme Legends ............................ 244
- Cartographic Legends ..................... 245
- Working with Analysis Layers ............. 246

## Part C: Advanced Topics .................................. 248

## Chapter 16: XML Protocol
- MapInfo Enterprise XML Protocol ........ 250
- MapImageRequest ......................... 251
MapVectorRequest ................................................................. 254

Chapter 17: Customizing the AddLayer Wizard ............................... 258
  Overview ........................................................................ 259
  Changing the List of Data Sources ..................................... 259
  Specifying Named Database Connections .............................. 261
  Location of addlayerwizard.properties ............................... 262
  Specifying Default Values ................................................ 263
  Saving Passwords ............................................................ 264

Chapter 18: Creating Your Own Data Provider ............................. 265
  Introduction ..................................................................... 266
  Overview of Principal Data Provider Interfaces and Files .......... 266
  Steps for Implementing a DataProvider for Vector Data .......... 268
  Steps for Implementing a DataProvider for Raster Data .......... 270
  Adding Custom DataProviders to the AddLayerWizard .......... 270

Chapter 19: Uploading TAB Data to Remote Databases .................. 273
  Introduction to EasyLoader .............................................. 274
  Running EasyLoader ......................................................... 274
  EasyLoader Options ........................................................ 276
  Loading Oracle Spatial Data ............................................. 278
  Command Line Flags ....................................................... 279
  Mixing Command Line Flags with GUI ................................. 282

Chapter 20: Web Map Service .................................................... 283
  WMS Server Overview ...................................................... 284
  GetCapabilities ............................................................. 284
  GetMap ........................................................................ 287
  GetFeatureInfo. ............................................................. 289
  WMS Raster Data Provider ............................................... 291
  WMS Client .................................................................... 292

Part D: Appendices .................................................................... 294

Appendix A: Custom JSP Tag Library ........................................ 295
  Custom JSP Tags ............................................................. 296
  Creating a Custom JSP Tag ................................................. 305
  Creating an Add TAB Layer Tag ......................................... 307
  Adding Custom Tags to the Web Application Wizard ............... 309

Appendix B: Understanding MapBasic Style Strings ..................... 312
  Pen Styles ..................................................................... 313
  Brush Styles ................................................................... 314
  Symbol Styles .................................................................. 315
  Font Styles ..................................................................... 321
  Colors .......................................................................... 322
  Using MapInfo Professional to Determine MapBasic Styles ........ 323
<table>
<thead>
<tr>
<th>Appendix C: MAPINFO.MAPINFO_MAPCATALOG</th>
<th>324</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>325</td>
</tr>
<tr>
<td>MapXtreme Java SQL Scripts</td>
<td>326</td>
</tr>
<tr>
<td>MAPINFO.MAPINFO_MAPCATALOG Format</td>
<td>327</td>
</tr>
<tr>
<td>Appendix D: System Properties</td>
<td>328</td>
</tr>
<tr>
<td>Properties</td>
<td>328</td>
</tr>
<tr>
<td>Appendix E: System Logging</td>
<td>330</td>
</tr>
<tr>
<td>Logging Overview</td>
<td>331</td>
</tr>
<tr>
<td>Logging on the Server</td>
<td>331</td>
</tr>
<tr>
<td>Logging on the Client</td>
<td>331</td>
</tr>
<tr>
<td>Logging Levels</td>
<td>331</td>
</tr>
<tr>
<td>Appendix F: Custom Symbols</td>
<td>333</td>
</tr>
<tr>
<td>Symbols</td>
<td>333</td>
</tr>
<tr>
<td>Index</td>
<td>335</td>
</tr>
</tbody>
</table>
MapXtreme Java Edition is a mapping application development tool for organizations who recognize that data visualization and mapping can help you make better business decisions and manage assets and operations more effectively. Applications running on a managed server network offer huge economies of scale, including lower hardware and administrative costs while dramatically improving application performance, reliability, and security. Applications built with MapXtreme Java are appropriate both for corporate intranets and the public Internet.

In this chapter:

- Product Contents ................................................................. 2
- Product Audience ............................................................... 2
- Features .............................................................................. 2
- Benefits .............................................................................. 3
- Programming Advantages .................................................... 4
- What Can You Do with MapXtreme Java? .............................. 6
- MapXtreme Java Documentation Set ..................................... 6
- New in MapXtreme Java Edition 4.7 ..................................... 7
- Migrating MapXtreme Java 4.5 Applications to 4.7 ............... 9
Product Contents

MapXtreme Java is a set of 100% Pure Java classes (Java 2 compliant), which allow you to deploy your application on the system you have, whether it is Windows, UNIX, or both. The following components are included:

- MapXtremeServlet mapping engine
- MapJ API and Java class library
- MapXtreme JavaBeans
- MapXtreme Java Manager and Web Application Builder
- JSP custom tag library
- Java 2 VM (1.4.1)
- Sample applications
- Sample data
- MapXtreme Java Documentation set (this Developer Guide, Object Model poster, and complete MapJ API Javadocs)

Product Audience

This product is designed for developers who wish to create Web applications that incorporate live maps and the tools needed to interact with those maps.

For experienced Java programmers, use the MapJ API to programmatically design custom maps and functionality.

For novice Java programmers, we provide mapping functionality in the form of JavaBeans for easy drag and drop development in a visual IDE.

Even non-programmers can create prototypes of Web applications quickly with no programming by using the Web Application Builder. This tool walks you through the process of creating a mapping application that uses JavaServer Pages tags.

Features

MapXtreme Java is an enterprise-level application development tool based on the servlet architecture specified in the Java 2 Enterprise Edition. The key element of this product is MapXtremeServlet, a server-side component that manages the requests and responses for mapping services, including image requests (GIF, JPEG), vector data requests (query methods), and metadata requests (table information).
By using a servlet architecture, MapXtreme Java can focus on satisfying mapping requests, while the web server/servlet container handles other server-side issues such as load balancing, security, and fault tolerance. Additionally, the servlet model uses HTTP, the standard for communicating across the Internet. Other features of MapXtreme Java include:

- Sophisticated mapping capabilities such as selection behavior, theme mapping and analysis, and advanced labeling and rendering.
- Data source access through JDBC that allows you to maintain your spatial data on a secure RDBMS, yet still use it to its full potential to build maps.
- Conveniences for easier installation and configuration, development and deployment, such as JavaBeans, Web Application Builder, JSP custom tag library, sample applications, and Apache Web Server/Tomcat servlet container.

Benefits

MapXtreme provides a multi-platform, high quality, high performance, easy-to-use solution for your mapping needs.

Multi-Platform

Because of the security, reliability, and performance, many end-user industries such as telecom and insurance use UNIX, while other users in areas of the same organization are working on Windows systems. Often there is a need to deploy a similar solution on many platforms. Java-based mapping applications allow developers to write a single program for use on multiple platforms that support virtual machines.

With MapXtreme Java Edition running on the server side, existing UNIX or Windows resources can be used. Data can be stored and manipulated on one system and programmatically accessed from another machine running a Virtual Machine.

Highly Scalable

Organizations creating enterprise-wide mapping solutions with MapXtreme need applications that perform well and can support all users that need access. Since MapXtreme offers this component-based, reliable, and multi-threaded solution, its versatility is assured. Your application can grow with the needs of your organization.

Fast Deployment

MapXtreme allows you to install, develop, and deploy your applications in a timely manner. It includes a Web Application Builder wizard for rapid prototype development, and several sample applications that demonstrate the basics of MapXtreme Java. These sample applications can be specialized or built upon for your own applications. MapXtreme Java is compatible with all J2EE-certified Web servers/browsers, and there are no proprietary plug-ins.
Programming Advantages

Object Oriented
MapXtreme is object oriented with an easy-to-use object model hierarchy for mapping display, query, and analysis.

MapJ API
The MapJ API is the client-side API used to communicate with the mapping engine, MapXtremeServlet. Each client requesting maps from MapXtreme uses (or reuses) an instance of the MapJ object. MapXtreme has no proprietary plug-ins, so it delivers maps to any browser on any operating system.

MapXtreme is asynchronous, multi-threaded, and stateless for maximum performance.

Server-Side Java
Most vendors of application software for use on the Web are providing solutions originally created as state-full, client-side tools that have been force-fit into a server environment. In contrast, MapXtreme was designed as a server-side Java component from the beginning. It was developed specifically to support:

- many concurrent users
- clusters of machines
- multiple CPUs per physical server
- virtually any platform (including Windows NT and the numerous UNIX versions)
- database connection pooling
- security issues that are not relevant in client-side implementations

Thus, MapXtreme web-based mapping applications can scale to support the number of users that can be managed by your application server. MapXtreme will work efficiently with even the most heavily hit web site in conjunction with web servers such as Apache/Tomcat, JRun, or IBM WebSphere.

Intelligent Multi-Threading
MapXtreme uses intelligent Java threading provided by the servlet container/application server to serve multiple concurrent users efficiently. It has a low memory consumption per user and scalable user load as additional CPUs are added. Tests demonstrate that the MapXtreme server engine requires about 8 MB of memory in steady-state and 100 KB to 200 KB per concurrent user. For example, while one thread is processing a map request, three other threads could be streaming the results of three previous map requests via network I/O simultaneously.
Component-Based Flexibility

MapXtreme offers great deployment flexibility because of its component-based architecture. There are four high-level components: the MapJ object, map Renderers to display maps, Data Providers to access various data sources, and MapXtremeServlet. MapXtreme can be used in a two-tier intranet deployment that puts MapJ on the client side, or a three-tier configuration for the Internet with MapJ and your business logic in a middle tier.

Strong Connectivity to Remote Data Sets

MapXtreme embraces the trend to store spatial data in relational databases such as Oracle with the Spatial option and the Informix Dynamic Server with the SpatialWare DataBlade. This allows you to protect your mission-critical spatial data in an enterprise-level database management system while providing appropriate access to any user on the World Wide Web.

Compatible with any Web Environment

MapXtreme's open architecture is compatible with virtually any Web environment (especially three-tier architectures) and works with any Web server supporting ISAPI, NSAPI, or CGI gateways, such as Netscape, or Apache. As a servlet environment, it provides all the benefits of Sun's Java Servlet API.

MapInfo recommends an environment architecture that includes an application server that can generate instances of Java objects. Apache Web Server and Tomcat servlet container are provided on the product CD for your convenience.

Virtually any Web browser accepts MapXtreme-generated maps because MapXtreme can output map images as raster images such as GIF or JPEG, which can then be embedded in HTML. More capable browsers (or browsers with a 1.2 VM plug-in or higher) can also receive vector data and display maps from these vectors. MapInfo recommends the use of Netscape or Internet Explorer versions 4.x or greater.

Programming Conveniences

MapXtreme Java provides a number of tools and conveniences that help you develop web applications. The MapXtreme Java Manager includes a Web Application Builder wizard that walks you through the creation of a web application. This is particularly useful for rapidly creating prototypes as proof of concept applications. The wizard uses JavaServer Pages technology in the form of custom JSP tags that simplify the coding necessary for a mapping application. MapXtreme also includes a number of JavaBeans, discreet mapping components that can be dragged and dropped onto a visual IDE interface.
What Can You Do with MapXtreme Java?

Use MapXtreme Java to build two- and three-tier Web mapping applications that service requests from clients for map data. Its flexible, scalable architecture allows you to send as much or little software to the client for your mapping needs, control the access to sensitive data, and expand your application as the need arises.

Specifically, programmers can:

- design custom maps with only the features and information you need
- programmatically create static and dynamic objects
- customize the look, placement and behavior of map features
- listen for map events such as a user’s mouse clicks to initiate a change in the map

For end users who will use your mapping application, you can provide them with tools to:

- zoom and pan to change the view of the map
- select map features and draw search areas
- query map features for more information
- create thematic shading such as a color-coded map based on data in your database
- control the visibility and style of objects and labels

Practical Uses

While the capability of MapXtreme Java is vast, here are a few ways mapping can be incorporated into enterprise applications.

- Provide field technicians real-time access to corporate and customer data to better serve customers from the field.
- In telecom and transportation, maps provide a way to monitor networks to determine problem areas, bottlenecks, or repair status.
- In supply chain management, mapping applications can be used to view the distribution of goods, services, or people and redistribute as necessary.
- Create a tracking application for delivery trucks, network calls, or troop deployments.
- Self-service applications for quicker customer service, for example, for the public to access information about government programs.

MapXtreme Java Documentation Set

This Developer Guide is the best resource to start learning MapXtreme Java. There are several additional sources of information useful to MapXtreme developers, including:

- MapJ API Specification in HTML – "Javadocs" installed with MapXtreme
- Object Model poster of MapJ classes and methods
- MapXtreme Java on the Web – web resources at [www.mapinfo.com/mapxtreme](http://www.mapinfo.com/mapxtreme) including access to the Discussion Forum and KnowledgeBase
New in MapXtreme Java Edition 4.7

Some of the newest enhancements to the MapXtreme Java Edition include:

**TAB Library**
You can now edit native MapInfo TAB files. For more information, see *Features and Searches in Chapter 13 on page 188* and *Editing a Tab Layer on page 207*.

**Buffering**
MapXtreme Java provides a new method that allows you to buffer regions. You can create a buffered region from any input geometry (point, line, or region). For more information, see the section, *Buffering on page 190*.

**Embedded Raster Data URL**
The Rendition.SYMBOL_URL property now supports embedded raster data URLs. This allows you to have an image embedded within the rendition, making the rendition and the document it is in more transportable. For more information, see the section, *Image Symbols on page 227*.

**Gradient Fills and Strokes**
You can now set a rendition to use a gradient fill or stroke. You have the choice of using a linear or radial gradient. A linear gradient a transition through a series of colors along a straight line. A radial gradient a transition through a series of colors along a circular path. For more information, see the section, *Gradients on page 221*.

**JPEG2000 Support**
MapXtreme Java now provides the ability to load a JPEG2000 raster file registered within a TAB file. All raster styles are honored. This includes brightness, contrast, grayscale, transparency, and translucency.

**Graphical User Interface Enhancements**
MapXtreme 4.7 includes many usability improvements. These enhancements include:

- In the MapXtreme Java Manager Client, you can now use key combinations to switch tabs. For example, use Alt-M to switch to the Map Definitions tab.
- The Add Layer Wizard GUI for file-based layer types (Tab, GeoTIFF, Shape) has been enhanced so that it now allows you to select multiple files, from separate directories, within one use of the wizard. That is, you can run the wizard once, and load 10 layers from various directories.
- The MapXtreme Java Manager Client can now run in stand alone mode. This means that you can run the client without having started the server. Some functionality, specifically the Web Applications builder wizard, will not be available when you run in stand alone mode.
- The MapXtreme Java Manager Client now includes a fourth tab for the Connections Manager. This eliminates the need to run a separate client application to define your connections. The interface that appears on the Connections Manager tab is the same as the as the Connections Manager interface from previous versions.
- The MapXtreme Java Manager Client now supports a right-click menu, and you can now build a standard right-click menu directly into VisualMapJ, as well.
- When developing application, zoom layering options in the Layer Control dialog can now be restricted. You can use the setAllowZoomLayerEditing method in LayerControl to disable zoom layering options. Refer to the HTML API for more information.
Natural Break Ranges
When creating a ranged thematic map, you now have the option of using natural break ranges. Natural break ranges contain distributed values so that the average of each range is as close as possible to each of values in that range. For more information, see Distribution Types on page 239.

Pie and Bar Charts
MapXtreme Java provides you with the ability to include pie charts, side-by-side bar charts, and stacked bar charts. For more information, see Working with Analysis Layers on page 246.

Raster Layer Optimizations
MapXtreme Java now renders raster layers with improved efficiency. Significant improvements have been made to performance and the look and feel of raster layers. Enhancements are apparent in the following raster styles: brightness, contrast, grayscale, transparency, and translucency. MapXtreme uses a combination of Java's Advanced Imaging (JAI) API and Java 2 Platform SDK 1.4's new Image I/O (IIO) API to provide you with these enhancements.

Style Chooser Enhancements
The style chooser dialog allows you to select display properties such as line patterns and foreground color. MapXtreme Java now provides more support in creating and editing named renditions, or styles. The following enhancements have been made:

- The style chooser is available in more places. Specifically, the thematic mapping dialog now contains a button that launches the Select a Style dialog, so that you will be able to specify more than just the color.
- If you launch the Select a Style dialog from the Named Resources tab, new named styles can now be saved. For example, if you select a standard style (e.g. the “Railroad” line style), then customize it by changing its color, line thickness, etc., you may want to save the selected style as a new named style such as “favorites/my railroad”.
- In addition to the foreground color, the background color can now set.
- Through the use of a slider, you can now set opacity for a style.
- You can select a custom (GIF or JPG) image to use as a symbol.

For more information about working with styles, see Managing MapXtreme Java in Chapter 5 on page 50.

Scalable Vector Graphics (SVG) Output
MapXtreme Java Edition provides support for exporting map images in SVG, a graphics format a for describing two-dimensional graphics in XML. For more information, see SVG Output on page 173.

SVG JSP Tags
You can now use MapXtreme Java custom SVG JSP tags that also support exporting map images in SVG. For more information, see Custom JSP Tag Library in Appendix A on page 295.

Web Map Service
MapXtreme Java allows you to create an OGC compliant Web Map Service. For more information, see Web Map Service in Chapter 20 on page 283.
New Sample Data
The sample data for version 4.7 has been enhanced to include new chart samples. The samples include pie charts and bar charts. For more information, see the MapXtreme Java Sample Data Descriptions document.

Logging
MapXtreme Java can log various types of messages. For more information, see System Logging in Appendix E on page 330.

Migrating MapXtreme Java 4.5 Applications to 4.7

When migrating 4.5 applications to 4.7, be aware of deprecations to the API. For a complete listing of deprecated classes, fields, methods, and constructors, refer to the HTML API documentation.

Deprecated Layer Class
The most significant deprecation is that of the Layer class. MapXtreme Java now provides you with the class FeatureLayer to be used instead of Layer. A FeatureLayer represents a layer of data for a map. For example, a layer of region objects representing the world country boundaries is a FeatureLayer.

The methods in FeatureLayer allow you to set the visibility and appearance of the layer. For example, it allows zoom layering. Sometimes you want a map layer to display only at certain zoom levels. Zoom layering allows the display of a map layer only when the map's zoom level falls within a preset distance.

For example, you have a layer of streets and a layer of ZIP Code boundaries. When you zoom out past 10 or so miles, the appearance of the streets degrades. This is because the zoom (window width) is too wide to show detailed street maps. Use zoom layering to tell MapJ to display the street layer only when the zoom is set to distance that allows you to see the street detail properly, for instance, less than 5 miles. You can set a different zoom layering level for each layer.

The search methods of Layer work on the data referenced by the layer. All other methods relate to the appearance of the layer when displayed in a map. In some cases, there may be a many to one relationship of tables to a layer.

There are no public constructors for this class. It is created when the Layers.add method is called.

Recompiling Existing Applications
Though it may not be necessary, we recommend that you recompile existing applications, and use the -deprecation flag. If you have implemented your own custom DataProvider, you must recompile existing applications.
Part A: Using MapXtreme Java

Part A introduces you to MapXtreme Java and its many features and conveniences to aid you in your web application development.

**Topics:**

- **Chapter 2: Getting Started**
  Contains product contents, system requirements, installation and setup instructions.

- **Chapter 3: Planning Your Application**
  After installation, your next decision is what kind of web application to build. This chapter defines your realm of choices for configuration and deployment options.

- **Chapter 4: Mapping Concepts**
  This chapter covers the basic mapping concepts to prepare you well for building a mapping application.

- **Chapter 5: Managing MapXtreme Java**
  An administrative tool, the MapXtreme Java Manager encompasses the Map Definition Manager, Named Resources panel and Web Application Builder for prototyping applications.

- **Chapter 6: Web Application Builder**
  The Prototype wizard is further broken out in this chapter where the use of JavaServer Pages tags, the basis for the prototype builder, are further examined.

- **Chapter 7: MapXtreme JavaBeans**
  This chapter covers MapXtreme JavaBeans, pre-built Java components that allow you to provide mapping functionality in your application without writing code.

- **Chapter 8: Writing Your Own Servlets**
  This chapter is for developers who wish to write a servlet that calls MapXtremeServlet, as a way to extend even further the functionality of the web server.
Getting Started

This chapter describes the components included with MapXtreme Java Edition, how to install them, and how to initialize the components.

In this chapter:

- System Requirements .................................................. 12
- Installation ................................................................. 12
- Uninstalling MapXtreme Java ........................................... 19
- Base Map Data Installation .............................................. 19
- Uninstalling Base Map Data ............................................. 24
- Tomcat Deployment Environment ..................................... 25
- Deploying MapXtreme in Other Web Server Environments ...... 25
- Installed Components .................................................... 31
- Installing Fonts ............................................................. 32
- Installing on UNIX without a Graphics Card ..................... 33
- Configuration Troubleshooting ......................................... 36
- Software Copy Protection ................................................. 38
System Requirements

MapXtreme Java Edition allows you to develop mapping applications on any platform that supports a Java virtual machine. The following requirements are the minimum necessary to implement your mapping applications:

- Web server that supports servlets/Java Server Pages or web server with plug-in to support servlets/JSPs or stand-alone servlet container. The Servlet container or plug-in must support the 2.3 Servlet API specification and the 1.1 JSP API specification.
- A Java 2 Platform compatible virtual machine 1.4.1 or higher.
- A video card installed on the server. On Solaris, either a video card or X11 server (as required by Java 2D for enhanced graphics capabilities).
- 200 megabytes hard disk space for MapXtreme Java (~760 MB for installation).
- 400 megabyte hard disk space for sample map data.
- 256 megabytes of RAM available for MapXtreme.

Installation

MapXtreme Java can be installed on any system that supports a Java 2 Virtual Machine (1.4.1 or higher). The general procedure is:

1. Install MapXtreme Java: The installation program installs the MapXtreme Server, client, samples, documentation, and an example deployment environment (Tomcat).
2. Install base map data: This is a separate installation that will install up to 400 MB of geographic and demographic data for use in MapXtreme Java. Choose a full installation or custom to select which data sets you want. See Base Map Data Installation on page 19 for instructions.

MapXtreme Java Installation

To install MapXtreme Java:

1. Place the MapXtreme Java Software CD in your CD-ROM drive.
2. Run install.htm from the CD root. Select the link for your platform: Windows, Solaris, Linux, or HP-UX.
   Alternatively, for the appropriate platform:
   - **Windows** – At the CD root, go to \InstData\Windows and run install.exe.
   - **Solaris** – At the CD root, go to /InstData/Solaris and run install.bin.
   - **Linux** – At the CD root, go to /InstData/Linux and run install.bin.
   - **HP-UX** – At the CD root, go to /InstData/Solaris and run install.bin.
3. At the main MapXtreme dialog box, choose a language (German, English, Spanish, French, or Italian) and click OK.
4. At the Introduction dialog box, read the information in the panel and click **Next** to proceed.

5. At the Important Information dialog box, review the information and click **Next**.
6. Choose to accept the License Agreement. Click NEXT. The Choose Install Set dialog box displays.

7. At the Choose Install Set dialog box, choose either FULL INSTALL or CUSTOM INSTALL. Click NEXT. A full installation installs all product components, while custom gives you the option of which components to install. If you chose FULL INSTALL, proceed to step 9.

8. If you choose Custom Install in step 7, the Choose Install Set dialog box displays. Select the components you want to install. Your options are:
   - SERVER – installs the server components.
   - CLIENT – installs the client components.
   - SAMPLES – installs the product samples.
   - DOCUMENTATION – installs the product documentation.
   - EXAMPLE DEPLOYMENT ENVIRONMENT – installs an example web deployment environment. This includes Tomcat 4.1.18.

   Depending on which components you choose, some of the following steps may not pertain to your installation process. CLICK NEXT. The Choose Install Folder dialog box displays.
9. At the Choose Install Folder dialog box, accept the default location to install the MapXtreme Java or browse to another location. The Windows default location is `<SYSTEM_DRIVE>\Program Files\MapInfo\MapXtreme-4.7.0`. On Solaris, Linux, and HP-UX, the default location is `/user/home/mapinfo/MapXtreme-4.7.0`. Click NEXT.

10. At the Choose Shortcut Folder dialog box, accept the default location to install application icons or browse to another location, or choose one of the other available options. Click NEXT.
11. At the Choose Java Virtual Machine dialog box, select a Java VM from the available list. This must be a Java VM 1.4.1 or higher. Click NEXT.

If you attempt to select a JVM lower than 1.4.1, the following dialog box displays.
12. At the Setup Web Environment dialog box, enter the host name and port number. The default port is 8080. Click **NEXT**.

13. At the Add to Application icon to classpath dialog box, click the **ADD FOLDER** or **ADD ZIP/JAR** buttons to add files that you want on the classpath when you launch MapXtreme Java Manager. For example, include any JDBC drivers you need for accessing remote data. Oracle users must include classes12.zip. Click **NEXT**.
14. The Pre-Installation Summary dialog box displays. Review the information and click **INSTALL** or click **PREVIOUS** to make changes.

16. At the Install Complete dialog box, click **DONE** to leave the software installer.

---

**Uninstalling MapXtreme Java**

To uninstall MapXtreme Java from Windows, choose the uninstall shortcut on the Program menu under MapXtreme Java or run the uninstall.exe from the directory `<INSTALL_DIR>\UninstallMapXtremeJava 4.7`.

To uninstall MapXtreme Java from Solaris, Linux, or HP-UX, run the uninstall.sh from the directory `<INSTALL_DIR>/UninstallMapXtremeJava-4.7.0`.

---

**Base Map Data Installation**

You can install all or a portion of the data from five geographic regions (North America, South America, Europe, Asia, and Australia), StreetPro street data, and U.S. demographic data. The `world.mdf` is automatically installed on your system when you install MapXtreme Java.

The data sets are described in the MapXtreme Java Edition Sample Data Descriptions PDF, located in the `<INSTALL_DIR>/Program Files\MapInfo\maps` directory after installing the data.

The total data set requires approximately 400 MB of disk storage.

To install the map data:

1. Run `install.htm` from the maps directory. Select the link for your platform: Windows, Solaris, Linux, or HP-UX.
   Alternatively, for the appropriate platform:
   - **Windows** – At the CD root, go to `\maps\InstData\Windows` and run `sampdata.exe`.
   - **Solaris** – At the CD root, go to `/maps/InstData/Solaris` and run `sampdata.bin`.
2. At the initial dialog box, choose the appropriate language and click OK.

3. Read through the Introduction dialog box and click NEXT to continue.

4. At the Choose Install Folder dialog box, accept the default location to install the MapInfo MapXtreme Java or browse to another location. The Windows default location is \<INSTALL_DIR>\Program Files\MapInfo\maps. On Solaris, Linux, and HP-UX, the default location is /user-home/mapinfo/maps. Click NEXT.
5. At the Choose Install Set dialog box, choose COMPLETE to install the entire base map data set or choose CUSTOM to select the portion of data you wish to install. Click NEXT. If you choose COMPLETE, proceed to step 7.

6. If you chose CUSTOM in step 5, the Choose Product Components dialog box displays. Your options are:
   - **NORTHAMERICA** – Sample map data for North America. This includes data for United States, Mexico, and Canada.
   - **SOUTHAMERICA** – Sample map data for South America. This includes data for Brazil and Argentina.
   - **EUROPE** – Sample map data for Europe. This includes data for France, United Kingdom, Portugal, Spain, Germany, and Italy.
   - **ASIA** – Sample map data for Asia. This includes data for Japan, Israel, South Korea, and China.
   - **AUSTRALIA** – Sample map data for Australia.
   - **DEMOGRAPHICS** – Demographic data.
   - **STREETPRO** – StreetPro data.

   Clear any check box for data sets you do not want installed and click NEXT.
7. At the Choose Shortcut Folder dialog box, accept the default location to install application icons or browse to another location, or choose one of the other available options. Click NEXT.
8. At the Setup Web Environment dialog box, enter the host name and port number. The default port is 8080. Click **NEXT**.

![Setup Web Environment dialog box](image)

9. At the Pre-installation Summary dialog box, review the information and click **NEXT** to proceed, or click **PREVIOUS** to return to an earlier dialog box and make changes.

![Pre-installation Summary dialog box](image)
10. Click **INSTALL**. The install process proceeds to completion.

11. Click **DONE** to leave the data installer.

---

**Uninstalling Base Map Data**

To uninstall MapXtreme Java from Windows, choose the uninstall shortcut on the Program menu under MapXtreme Java edition Evaluation Base Maps or run the uninstall.exe from the directory `<INSTALL_DIR>\UnInstallerData`.

To uninstall MapXtreme Java from Solaris, Linux, or HP-UX, run the uninstall.sh from the directory `<INSTALL_DIR>/UnInstallerData`. 
Tomcat Deployment Environment

MapXtreme Java uses a servlet model to service requests. MapXtreme Java requires a web server/servlet container to run, such as Apache/Tomcat, iPlanet, WebLogic, etc. As a convenience to users, we install and integrate Tomcat 4.1.18 with MapXtreme Java as part of the product installation.

After installation, you will find the Tomcat directory under `<INSTALL_DIR>\MapXtreme-4.7.0\Tomcat-4.1` In the `<INSTALL_DIR>\MapXtreme-4.7.0\Tomcat-4.1\webapps` directory will be the servlet contexts for MapXtreme Java (`/mapxtreme47`), Mapviewer JSP sample (`/mapviewer47`), other sample applications (`/samples47`), and WMS Server (`/wmsserver111`).

Multiple Instances of Tomcat

Using multiple instances of Tomcat can increase MapXtreme Java’s performance. On some Solaris servers with four or more CPUs, MapXtreme's performance improves when there are three instances of Tomcat running concurrently. For example, your server might run Apache and three Tomcat instances, and the Tomcat setup (e.g., mod-jk.conf) can perform load balancing among the Tomcat instances.

If you have multiple Tomcat instances on the same computer, you will need to modify your Tomcat configuration so that each instance listens on a different port number. For example, you will want to have multiple copies of the server.xml file (e.g., `server2.xml`, `server3.xml`), each of which designates a unique port number and a unique workDir setting, so that the multiple Tomcat instances do not conflict. You will also need to start each Tomcat instance with the optional `-f` argument to designate which xml file you wish to use. For example:

```bash
tomcat start -f ../conf/server2.xml
```

Note that load balancing is not necessarily limited to one computer. For example, you might have Tomcat running on several computers, with Apache performing load balancing.

Deploying MapXtreme in Other Web Server Environments

As part of the installation process, web archive (war) files are created and placed in the `<INSTALL_DIR>\MapXtreme-4.7.0\wars` directory. The war files are:

- `mapxtreme47.war` – the main mapxtreme context
- `mapviewer47.war` – the JSP sample MapViewer web application
- `samples47.war` – servlet sample applications
- `wmsserver111.war` – context for Web Map Service (WMS)

These war files can be deployed in any servlet container that supports WAR files.

**Note:** These files have information in them that ties them to the host machine on which they were installed.
Using the WARFile Generator

To run the WARFile Generator:

1. Run install.htm from the war-utility directory. Select the link for your platform: Windows, Solaris, Linux, or HP-UX.
   Alternatively, for the appropriate platform:
   - **Windows** – At the CD root, go to \war-utility\InstData\Windows and run install.exe.
   - **Solaris** – At the CD root, go to /war-utility/InstData/Solaris and run install.bin.
   - **Linux** – At the CD root, go to /war-utility/InstData/Linux and run install.bin.
   - **HP-UX** – At the CD root, go to /war-utility/InstData/Solaris and run install.bin.

2. At the initial dialog box, choose the appropriate language and click **OK**.

3. At the Introduction dialog box, read the information in the panel and click **NEXT** to proceed.

4. At the Choose Product Features dialog box, select the **DEFAULT MAPXTRME WAR FILES** option to create war files that represent the MapXtreme servlet context and MapXtreme sample context or select **USER-SPECIFIED CONTEXT**. This option allows you to choose a folder that points to a servlet context and creates a war file to a specified location with a specified name. Click **NEXT**.
5. At the Choose Install Folder dialog box, if you chose **DEFAULT MAPXTREME WAR FILES** in the previous step, choose a directory to install the war files that represent the context files. Click **NEXT**.

   If you chose **USER-SPECIFIED CONTEXT** in the previous step, choose the directory where you want the war file to be created. Click **NEXT**.

6. At the Network Information dialog box, enter the host name and port number. Click **NEXT**.

   If you chose **DEFAULT MAPXTREME WAR FILES** in **step 4**, click **NEXT** and proceed to **step 9**.
7. At the Locate context dialog box specify the directory that contains the context to be made into a WAR file. Click NEXT.

8. At the Specify War file name dialog box specify the name to give to your custom WAR file. Click NEXT.
9. At the Add to Application icon to classpath dialog box, click the **ADD FOLDER** or **ADD ZIP/JAR** buttons to add files that you want on the classpath when you launch MapXtreme Java Manager. For example, include any JDBC drivers you need for accessing remote data. Oracle users must include classes12.zip. Click **NEXT**.

10. At the MapXtreme Logging file Location dialog box specify the location where log files will be generated. For more information about logging, see **System Logging in Appendix E on page 330**. Click **NEXT**.
11. The Pre-Installation Summary dialog box displays. Review the information and click **INSTALL** or click **PREVIOUS** to make changes.

12. The Installing dialog box displays.
13. At the Install Complete dialog box, click **DONE** to leave the installer.

**Installed Components**

The following outlines the key components included in MapXtreme Java. A short description of each follows.

**MapXtreme Java Manager**

This is a GUI tool that allows you to manage several aspects of MapXtreme Java, including create and load map definitions, manage named resources, manage JDBC connections to RDBMSs, and build prototypes and applications with the help of a wizard. MapXtreme Java Manager is discussed in **Chapter 5: Managing MapXtreme Java**.
MapJ API and Java Class Library

The MapJ API, class library, and Javadocs provide everything you need to build customized applications, applets, servlets, and JavaBeans. Part II of this Developer Guide covers the MapJ API.

JSP Tag Library

MapXtreme Java includes a set of custom JSP tags to assist you with rapid prototype and application building. See Chapter 6: Web Application Builder for more information.

Custom Symbols

MapXtreme Java installs a folder of custom symbol GIF images. For a thumbnail view of each image, see Appendix F: Custom Symbols. For information on how to use these symbols in an application, see Chapter 14: Labeling and Renditions.

MapXtreme Java also provides pre-defined renditions for pen, brush, and symbol fonts. Use these to customize your map layers. These are located in the mapxtreme47 servlet context under /resources/mistyles.

Samples

MapXtreme Java includes a variety of sample servlets and applets that demonstrate key elements of MapXtreme Java. Most can be found in the /examples directory after installation. Several of these sample applications include an HTML tutorial that explains the features and the code. In addition, pre-compiled versions can be found in the MapXtreme-4.7.0/lib/client/mxjclientsamples.jar and MapXtreme-4.7.0/lib/server/mxjserversamples.jar.

Installing Fonts

MapXtreme Java provides 11 TrueType font sets for use as map symbols. These fonts are installed and automatically registered on Windows. They are located in the <SYSTEM_DRIVE>/OS/fonts directory.

On platforms other than Windows, the installer copies the fonts into the <SYSTEM_DRIVE>/OS/fonts directory. After installation, you will need to register these fonts with your operating system in order to use them. The following instructions apply to Solaris. Instructions for your platform will be similar. The instructions assume that you have successfully installed Java 2 and MapXtreme Java.

1. Log on to Solaris as ROOT (i.e., superuser).
2. Execute the Solaris command xset -q, which displays all the directories that the system uses for font storage/usage.
3. Copy the MapXtreme Java fonts to one of these directories (i.e., /usr/openwin/lib/X11/fonts/TrueType).
4. Re-assign the permission on these font files by executing the following commands:
5. Edit the fonts.dir file. Go to the directory where you placed the fonts and provide an entry for each of the MapXtreme Java fonts. You can copy these entries from a file called fonts_sample.dir located in the mapxtremejava/server/fonts directory after installation. The entries are:

- `mapiau__.ttf -unknown-mapinfo arrows unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapicuc__.ttf -unknown-mapinfo cartographic unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapimu__.ttf -unknown-mapinfo miscellaneous unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapiogu__.ttf -unknown-mapinfo oil&gas unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapireu__.ttf -unknown-mapinfo real estate unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapisu__.ttf -unknown-mapinfo shields unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapitu__.ttf -unknown-mapinfo transportation unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapiww__.ttf -unknown-mapinfo weather unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapsu__.ttf -unknown-mapinfo symbols unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapsymu.ttf -unknown-map symbols unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`
- `mapispu__.ttf -unknown-map symbols unicode-regular-r-normal--0-0-0-0-0-p-0-iso8859-1`

6. In the first line of the fonts.dir file, increment the number by 10 to include the 10 new font entries you added.

7. Log off Solaris, and log on again, this time as any user.

8. Execute the Solaris command `xlsfonts`, which reads all existing fonts.dir files and displays a list of all the fonts available for use on the system.

9. Start the font server prior to starting the X server, if you are using one.

**Note:** Users who do not have ROOT access may also provide the following to get MapXtreme Fonts working.

Assuming the fonts are in `/data/my_data/MXTJ47/fonts`, set the environment variable `JAVA_FONTS` to:

```
/usr/openwin/lib/X11/fonts/TrueType:/data/my_data/MXTJ47/fonts
```

---

### Installing on UNIX without a Graphics Card

While it is possible to manual install the MapXtreme Java, MapInfo strongly recommends that you install on a machine that uses a graphics card. If you must perform a manual install, use the Xwindows option or the Console option as described in the following sections.
X-windows Installation Option

UNIX developers who do not have a graphics card on their UNIX boxes need to have X-windows (X11) to run the MapXtreme Java installers. Using an X emulator, such as eXceed or KEA!X, will let you run and install on remote systems, without being at the computer itself. It is possible to install the product in X, but problems may occur. In general, make sure that you are using the certified versions of this type of software. There could be issues with the display manager, if it is set incorrectly. As MapInfo does not produce these X managers, we recommend that you run the installer in GUI mode, then use a UNIX box which has a graphics card installed in it. Once you have done this, follow the Solaris installation steps in the section, Installation on page 12.

Console Installation Option

On a UNIX machine that does not have a graphics card to support the GUI install and does not have X-windows, you may perform a console install and manually configure the files. You can use MapXtreme Java, but will not be able to run the samples that come along with MapXtreme Java.

To run the installer in console mode from the command line, type the command:

```
install.bin -i console
```

When you perform a console installation of MapXtreme Java, the product is by default installed at `/<USER_HOME>/MapXtremeJava-4.7.0` The chart below describes the changes that need to be made to the web.xml file after performing a console install.

**Note:** When launching the MapXtreme Java installer in console mode, you will be asked to choose the locale in which you want to run the installer. Choose the number that points to your locale.

**Location:** The place where you installed the product.

On a UNIX platform the default installation location is `/<USER_HOME>/MapXtremeJava-4.7.0`.

**Files:** The files that need to be configured to use the MapXtreme Java.

**Original and Replacement Text:** MapXtreme Java installer configures some files to deploy the product on your machine. It does this by replacing text in the files with values specific to your environment. When you run the installer at the command line prompt, this configuration does not occur. Therefore, you will need to manually replace the text in each file. The files that need to be changed, and the text to replace in each file are as follows:

<table>
<thead>
<tr>
<th>Location and Files</th>
<th>Original Text</th>
<th>Replacement Text</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;USER_INSTALL_DIR&gt;/MapXtreme-4.7.0/bin-MapXtreme Java Manager Web Server.lax</code></td>
<td>lax.command.line.args= http://:/mapxtreme47/manager</td>
<td>lax.command.line.args= http://@hostname@:@port.number@ /mapxtreme47/manager</td>
</tr>
</tbody>
</table>
Configuation Considerations

Consider the following items before carrying out a console installation.

- The MapXtreme Java console-based installation does not configure the files needed to run the samples, so after running the installation none of the samples will work. For configuring them you will have to modify the specific files for each sample.

- For loading the sample data in the MapXtreme Java Manager applet, you will have to rename a directory under your examples directory at the root installation with your machine name. For example, change
  
  `<USER_INSTALL_DIR>/MapXtreme-4.7.0/examples/server/data/machine-name`

  to
  
  `<USER_INSTALL_DIR>/MapXtreme-4.7.0/examples/server/data/hostname`

  where hostname is your machine name.
Configuration Troubleshooting

If you encounter a configuration problem that prevents MapXtremeServlet from servicing mapping requests, once you have installed MapXtreme Java and set it up in a servlet container, use the following information. For example, if you try to deploy the sample applet, SimpleMap, the applet might report a FileNotFoundException referring to the URL of your MapXtremeServlet.

If you experience such an error, there are two main possibilities:

- MapXtremeServlet might not be configured correctly.
- You might not be specifying the correct URL to access MapXtremeServlet.

To troubleshoot this type of problem, make sure your server is started and run a "debug" URL. A status page will display indicating that MapXtremeServlet is running and the URL is correct. In your browser’s URL text field, type the URL to your MapXtremeServlet, followed by "?debug=true". For example:

    http://hostname:portnumber/mapxtreme47/mapxtreme?debug=true

The status page will show basic diagnostic information, including the version of the Java VM and the version of MapXtreme in use. If you do not see this status page, either the servlet is not running or the URL is not correct.

Verifying that MapXtremeServlet Is Running

In addition to running the debug URL, you can check if MapXtremeServlet is running by watching the startup process. As your servlet container launches, watch for any indication of errors. For example, if your servlet container has a console window or log that you can view, view the console or log for any messages that refer to "MapXtremeServlet" or "mapxtreme".

If your servlet container provides an administrator tool, you should be able to check the status of MapXtremeServlet by running your administrator. If your servlet container does not provide an administrator tool, try viewing a console window or log to see whether MapXtremeServlet started.

The exact text that appears in your console window or log will depend in part on which servlet container you use. In general, if MapXtremeServlet is initialized successfully, you should see an "init" message. For example, if MapXtreme has been correctly configured with Tomcat, the Tomcat console or log file shows the following text:

    Context log path="/mapxtreme47" :mapxtreme: init

These messages indicate that the web.xml file for MapXtreme was processed, and the MapXtremeServlet was initialized correctly. Note that the status message above says "mapxtreme: init" instead of "com.mapinfo.mapxtreme.MapXtremeServlet: init". This is because MapXtreme's installer assigns the MapXtremeServlet a registered name of "mapxtreme"; see the MapXtreme web.xml file in /mapxtreme47/WEB-INF.

If MapXtremeServlet does not initialize when you start your servlet container, it does not necessarily mean that there is a problem. It might simply mean that the MapXtremeServlet configuration did not set the Load On Startup option to True (and hence, MapXtremeServlet has
not yet been initialized because there have not yet been any requests for it). MapXtreme's Tomcat installer does set Load On Startup to True, so Tomcat users should see an "init" message, as shown above, when Tomcat is started.

If you are having difficulty verifying that MapXtreme is configured correctly, you should configure MapXtreme so that it is set to Load On Startup, and then restart your servlet container. In Tomcat, the Load On Startup option is specified in mapxtreme/WEB-INF/web.xml, using the <load-on-startup> tag. If your servlet container provides an administrator tool, you can probably set the Load On Startup option in the administrator.

If your MapXtremeServlet deployment descriptor is set to Load On Startup, and you still do not see a MapXtremeServlet "init" message in the servlet container's console or log, then there is probably an error in how MapXtreme is configured; in this case, review the steps that you used to configure MapXtreme, to make sure all configuration steps were performed correctly.

Verifying the Correct URL for MapXtremeServlet

Before you can take advantage of your MapXtremeServlet, you need to know its precise URL. For example, before you can run the sample applet, SimpleMap, you need to edit the HTML file that loads the applet, and specify the URL to your MapXtremeServlet.

**Note:** URLs are case-sensitive.

The general form of a MapXtremeServlet URL is as follows:

```
http://hostname:portnumber/path/registeredname
```

**hostname:** This depends on the name or IP address of your server. If you are running everything on one computer (e.g., for testing or development), you might use "localhost" for your hostname.

**portnumber:** This depends on how you set up your servlet container. If you ran MapXtreme's installer and accepted the defaults, this should be port 8080. If you have configured your servlet container to use port 8080, you can omit the port number (and the colon) when you specify the MapXtreme URL.

**path:** This depends on your servlet container, and may also depend on what sub-directories you created. See examples below.

**registeredname:** This is a name that you can specify, such as "mapxtreme", which acts as a shortcut to the com.mapinfo.mapxtreme.MapXtremeServlet class. It is not a strict requirement that you use a shortcut; you can specify com.mapinfo.mapxtreme.MapXtremeServlet as the final part of your URL. However, if you have set up a registered name for MapXtremeServlet, then you should use that registered name. (Any configuration options that you specified in your deployment descriptor, such as Load On Startup options, are associated with the registered name; if you use an URL that specifies the class name rather than the registered name, then those options will not take effect.)

**Example: Tomcat**

If you are using the example deployment environment (Tomcat) that comes with MapXtreme Java and you accepted the defaults, then your MapXtremeServlet URL should be:

```
http://stockholm:8080/mapxtreme47/mapxtreme
```
In this example, the /mapxtreme47 segment of the URL signifies the mapxtreme directory created inside the Tomcat directory. The ending segment, /mapxtreme, refers to the registered name that the installer places into [tomcat directory]/mapxtreme47/WEB-INF/web.xml, using the <servlet-name> tag.

LAX Files

The MapXtreme Java installer places several configuration files on your system that relate to the executables for MapXtreme Java Manager and Uninstaller. These files, with the extension .lax (e.g., MapXtreme Java Manager.lax) provide important configuration information about your system:

- lax.class.path – Add any JDBC drivers you need to the list of jar files here. For example, to access Oracle with Spatial Option, add classes12.zip to the classpath. The installer provides an opportunity for you to add any .zip or .jar files. However, you can add them here after installation if you need additional files.
- lax.stderr.redirect – For debugging, put "=console" at the end to send the information to a console window.
- lax.stdout.redirect – Put "=console" at the end for debugging, as above.
- lax.nl.current.vm – Check the version of the VM, if it’s not working.

Software Copy Protection

If your copy of MapXtreme Java Edition is copy protected (i.e., a watermark "Protected Software - Key Required for Permanent Use" is visible on maps displayed in MapXtreme Java Manager), you will need to apply for a license file that will remove the watermark and make MapXtreme Java available for permanent use.

To remove the watermark from the map display:

1. Do one of the following:
   - Fill out the copy protection key application entitled "Important Please Read" provided in the MapXtreme Java product package.
   - Download the application form from the MapXtreme website at http://testdrive.mapinfo.com/mapxtremejava. Click on the link labeled Support, then click the Documentation link and choose Software Copy Protection License Application.
2. Submit the completed form to your distributor or reseller. See form for contact information. Your distributor will process your application and send you a license file by e-mail.
3. After receiving the license file called mapxtremejava.key, place it in a directory that is on your classpath, for example, C:/Program Files/MapInfo/MapXtreme-4.7.0/lib/client. This file also needs to be on the server’s classpath, so for MapXtremeServlet, place it in the mapxtreme47/WEB-INF/classes directory as well. Create the /classes directory if you do not have one. Your maps will now display without the watermark.
This chapter provides you with an introduction to Web-based deployment infrastructure requirements and necessary skills, followed by an overview of MapXtreme Java components and common configurations.

In this chapter:

- Web Deployment .................................................. 40
- Deployment Options .................................................. 41
- MapXtreme Java Overview .......................................... 42
- Design Considerations ................................................. 44
Web Deployment

Web-based deployments provide many advantages including:

- Reduced cost of ownership: It is cost effective to distribute applications across the web because it eliminates the need to install components on every machine that will access the application.
- Scalability: Web systems are served from powerful servers. If the number of users increases, more servers can be added.
- Access to data and software: Each user of your application accesses the latest version. You introduce software and data updates to all users at once.
- Security: Since security is centralized, you have more control over how you want to implement it.

Web-based deployments, in general, work as follows. On the client side, the user interacts with the mapping application via HTML pages and/or an applet in his or her browser. The interaction is based on a request/response scenario. The user makes a request, for example, to zoom in on an area of the map. That request goes to the web server. A server-side application communicates with the map server to provide the requested information. The updated map is returned to the client's browser embedded in the HTML page or applet. Should any of the data that is needed to create the map reside in an RDBMS, calls are made to the database via JDBC to retrieve the data. For more detailed descriptions of web-based deployments, see Writing Your Own Servlets in Chapter 8 on page 141.

An example of a useful web-based mapping application serves the wireless telecommunications industry. Frequently, potential customers want to know if their location is inside or outside the company's coverage area. By providing coverage maps embedded in an HTML page that is deployed over the Internet, the telco company can provide an immediate response to customers in a self-service framework. The customer can interact with the map by using navigation tools such as zoom in or pan. Each click is a request that is sent to telco application to regenerate the map.

Infrastructure Requirements

In general, web-based mapping deployments utilize standard components, including a server machine running a web server, an application server (may be the same as the web server), and a database of map data for background maps, and custom data specific to the application.

MapXtreme Java web-based deployments require Java 2 support. Additionally, the web server must support servlets since the mapping server for MapXtreme Java is deployed as a servlet. Data for maps can be stored locally or accessed from an RDBMS via JDBC. The application that you are to build can be in the form of a servlet, JavaServer Pages, Enterprise JavaBeans, or an applet.

Necessary Skill Sets

MapXtreme Java is a product for Java developers experienced in writing servlets and applets. If you are building an application, skills in creating a Java GUI are also necessary.

Additionally, since MapXtreme Java is deployed on the web, you should have web development and HTML expertise available to your project. How much you will need is dependent on the development tool that you are using.
If you are accessing data in an RDBMS, skills in database usage and administration are also necessary.

Finally, because this is a mapping application you are creating, some familiarity with mapping concepts and/or MapInfo mapping products is helpful. The basics of mapping is covered in Chapter 4: Mapping Concepts.

Deployment Options

Deployment options for MapXtreme Java can be categorized into three types: thin client, medium client, and thick client. The difference is in how much software and data is sent to the client.

An overview of each type is presented below, followed by a discussion of the components that make up MapXtreme Java.

Thin Client

In a thin client deployment, the user interacts with HTML pages in a browser. The map is typically a GIF image embedded in the HTML. The map request processing occurs on the server. This is the classic Internet deployment that does not require Java on the client.

To build this type of application, you will need to know how to develop the server-side application that generates the HTML.

Thick Client

A thick client is at the other end of the spectrum. The client downloads a Java applet that provides a more sophisticated user interface than straight HTML. Additionally, MapXtreme Java can return vector data instead of a raster image. Because of the increased download time for the applet, this deployment is better handled on intranet systems where the client side can be better controlled.

To build this type of application, you will need to know how to build a Java applet and optionally use JavaBeans.

Medium Client

In between the thin and thick options is the medium client. Like the thick client, the medium client downloads an applet so the client must support Java. Like the thin client, the medium client receives a raster image of the map. The applet can give you a more sophisticated user interface than straight HTML and additional map tools, such as a marquee selection tool.

To build this type of application, you will need to know how to develop an applet and a server-side application that interact.
MapXtreme Java Overview

There are four main components to MapXtreme Java Edition: MapXtremeServlet, the MapJ object, Data Providers, and Renderers. These components work together to access geographic data, manipulate it, and provide a map or data to your application.

MapXtremeServlet

The MapXtremeServlet is the mapping server provided in the MapXtreme Java product. It services three types of client requests:

- requests for map images
- requests for vector map data
- requests for map metadata (e.g. the column names of a Layer in a map).

MapXtremeServlet responds to HTTP POST requests from MapJ objects. Additionally, you can write your own clients using MapXtreme Java’s XML protocol for communicating with MapXtremeServlet.

MapXtremeServlet is designed to leverage the capabilities of its parent servlet container. MapXtremeServlet is stateless, it relies on the client request to fully describe the state of the map. Image requests are handled within MapXtremeServlet by a multi-threaded "Renderer server". Similarly, requests for map data are handled by a multi-threaded "DataProvider server". These factors make MapXtremeServlet highly scalable when deployed within a parent servlet container.

While MapXtremeServlet focuses on fulfilling mapping tasks, its parent servlet container can handle load balancing, fault tolerance, and security management. Servlet containers are found in web servers such as Sun's JavaWebServer, and in application servers such as BEA's WebLogic. Web servers such as Apache's Web Server or Microsoft IIS do not include a servlet container. In these cases a separate servlet container plug-in such as JRun or Tomcat must be used.

MapJ Object

The MapJ object manages the state of a map. It maintains a map's center and zoom, coordinate systems, distance units, and the Layers that collectively comprise the map. MapJ is the topmost level of MapXtreme's client API.

MapJ objects can be configured to work with different types of Renderers and Data Providers. In the most typical configuration MapJ is a client of MapXtremeServlet. MapJ sends requests to a MapXtremeServlet instance and as part of the request provides the servlet with its current state. MapJ obtains map images and data from the servlet.

MapJ can also work stand-alone to directly obtain map data and produce map images. A strength of MapXtreme's component based design is that MapJ can be configured with other variations. For instance, MapJ can be configured to access map data via one or more instances of MapXtremeServlet, but still be responsible for displaying the map image.

Since MapJ’s primary purpose is to maintain map state it has a small memory footprint. This makes MapJ ideally suited for being deployed in the middle tier of n-tier architectures. See Design Considerations on page 44 for more on deployment configurations.
Renderers

Renderers display map data. There are five types of Renderers: LocalRenderer and MapXtremeImageRenderer, EncodedImageRenderer, IntraServletContainerRenderer and CompositeRenderer.

A LocalRenderer can be created from any Java Graphics 2D object, which are typically obtained from an AWT component or BufferedImage. LocalRenderer is "local to" or in the same process space as the MapJ object to which it is associated. It uses Data Providers to directly obtain map Features for each Layer in a map. The LocalRenderer then draws the Features into its Component's Graphics object.

A MapXtremeImageRenderer can be created from a URL reference to an instance of MapXtremeServlet. When MapJ uses a MapXtremeImageRenderer it signifies that it wants to defer map rendering to an instance of MapXtremeServlet. The servlet satisfies this request by returning a raster image to the MapJ client. Various raster formats including GIF, JPEG, and PNG are supported by MapXtremeServlet. Of note, MapXtremeServlet's "Renderer server" satisfies rendering requests by using instances of LocalRenderers and exporting images to the desired raster formats.

MapXtreme Java also provides variations on MapXtremeImageRenderer. The IntraServletContainerRenderer is used in servlet forwarding. EncodedImageRenderer allows you to render animated images for your layers. CompositeRenderer allows you to request that only layers with changed data be redrawn. It's particularly useful for creating "animation" layers. Renderers are discussed in Rendering Considerations in Chapter 11 on page 209.

Data Providers

Data Providers are the key link between your MapJ object and your map data. Each Layer object which is part of MapJ has its own internal Data Provider. Data Providers are used to access data sources and return vector data. Data Providers are also invoked during rendering when MapJ uses a LocalRenderer.

MapXtreme has Data Providers for accessing the following data sources:

- MapInfo tables
- Oracle with Spatial Option
- Informix Universal Server SpatialWare DataBlade
- SQL Server with SpatialWare
- JDBC compatible tables containing longitude and latitude columns
- ESRI Shapefiles
- Data Binding (where data from TAB files and JDBC data sources are joined)
- Raster files
- MapInfo Grid

A MapJ object has two ways of accessing a data source. The first approach is to directly access the data source using a LocalDataProviderRef.
The second method is to make a request to an instance of MapXtremeServlet to get the data. MapXtremeServlet will then use a MapXtremeDataProviderRef from its "Data Provider server" to directly access the data source. As MapXtremeServlet obtains data from the data source, it will stream the data back to the client MapJ object. MapXtremeServlet uses an extremely efficient compression scheme to stream the data. One of its capabilities is to take into account the needed resolution of the data. For instance, when the data is used for rendering a 640 x 480 image, the data can be transmitted at a much higher level of granularity than it may be stored.

Each Layer associated with MapJ specifies how it would like to access its underlying data source through a "Data Provider reference". A LocalDataProviderRef signifies that data access should occur "local to" or within the process space containing MapJ. A MapXtremeDataProviderRef denotes that a MapXtremeServlet instance will act as an intermediary in accessing the data source.

Data Providers are discussed in Mapping In Layers in Chapter 10 on page 177.

Design Considerations

Keep the following elements in mind as you plan your mapping application.

Client Side

- Are you deploying over the Internet or via a corporate intranet?
- What is the network bandwidth?
- Is the client an applet or stand-alone application? Will the client need additional software or resources, such as JDBC drivers, to run?
- Are you designing your application for a specific platform, for any platform, or a mixed environment?
- What browser will your users be using? Is Swing support or other browser plug-ins required?
- How much mapping functionality is needed client-side? How useful are the JavaBeans such as the AddThemeWizard and LegendContainer for your needs?

Server-Side

- How complex of an application are you building? Do you have the necessary hardware?
- How many users do you expect to use your application? What is the peak user load expected?
- What services do you want or need from a Web server and/or application server?
- Do you have the appropriate skill sets for the type of application you are building? It can include Java programming, database administration, web development, etc.
- Have you considered any security or network issues?
- What other software will your application need to interact with?
- What version of Java will be used? Do all the components support a common version?
This chapter provides an overview of many of the concepts you will be working with to create mapping application with MapXtreme.

In this chapter:

- Organizing Your Data and Maps: An Overview of Tables ........................................ 46
- Map Definitions ......................................................................................................... 47
- Map Features ........................................................................................................... 48
- Renditions ................................................................................................................ 48
- Labeling ..................................................................................................................... 48
- Map Data Analysis ..................................................................................................... 49
- Putting Your Data on the Map .................................................................................. 49
Organizing Your Data and Maps: An Overview of Tables

To use MapXtreme Java, you need the files that contain your records and maps. Your data can be in MapInfo format or in a format that supports spatial data. This chapter will explain the MapInfo file formats (.tab) and introduce you to Map Definitions, XML-based text files that can store MapInfo data or data from your JDBC database.

How Files Make Up a Table

MapInfo .tab files are organized into a group of files to build a map layer. MapInfo tables consist of the following types of associated files:

- *.tab: This file describes the structure of the MapInfo table. It is a small text file describing the format of the file containing the data.
- *.dat(.mdb, .dbf, .txt, .xls or .wks): These files contain the tabular data.
- *.map: This file describes the graphic objects (it will not exist if the table has no map objects).
- *.id: This file is a cross reference file that links the data with the objects (it will not exist if the table has no map objects).
- *.ind: This is an index file. The index file allows you to search for map objects (will not exist if the table has no index).

Together these files make up a single .tab layer in MapXtreme Java.

MapInfo Tables and MapXtreme Layers

Each mappable MapInfo table can be displayed in a MapXtreme application as a layer in a map. For example, you can display a layer of customers, a layer of streets, and a layer of county boundaries.

Think of these layers as transparencies where each layer contains a different part of the map. The layers are stacked one on top of the other and allow you to see all aspects of the map at the same time. Layers are drawn from the bottom up. They are listed in the Layer Control dialog with the topmost layer listed first.
Map Definitions

Map Definitions describe the data that makes up the map, including what data is displayed, where it is stored, how the data is organized into layers, and how the data displays (e.g., what colors are used, are features labeled, what is the map zoom settings).

MapXtreme Java ships with a wide variety of sample data covering world geography and demographics. It is provided in two formats: .mdf and .gst. (For descriptions of the data sets, see the PDF document MapXtreme Java Sample Data Set, located on the product CD.) Each type is explained below.

MapInfo Geosets

A collection of MapInfo .tab files is called a geoset. If you are familiar with MapInfo Professional, you will notice that geosets are similar in concept to a workspace. Workspaces are a saved configuration of MapInfo tables (.tab) and windows. Instead of opening individual .tab files, you can open the geoset (somefile.gst) and all the layers with their particular settings display.

Geosets are limited in their functionality in MapXtreme Java, however. They specifically refer to MapInfo .tab files which cannot be saved to a remote database.

MapXtreme cannot open a MapInfo workspace (.wor file type). If you are using MapInfo Professional, you may save your workspaces as geosets using the MapInfo Geoset Utility, then load them into MapXtreme Java using the MapXtreme Java Manager.

Map Definitions

MapXtreme Java provides an alternative data format that is far superior to geosets. Map Definitions are XML-based text files that contain map layer information that can be stored as a file or as a record in a JDBC database. They are more forward compatible, easily editable, and conform to the growing standard use of XML for data transfer. We strongly recommend using Map Definitions over geosets.

Map Definitions are created using the MapXtreme Java Manager that ships with MapXtreme Java. Instructions are provided in Chapter 5: Managing MapXtreme Java. When saved, they can be stored as a file using the extension .mdf, stored as a record in a JDBC database, or as a named map for easy retrieval.

Map Definitions created in MapXtreme Java 3.x will automatically be imported into MapXtreme Java 4.x. Map Definitions created in versions prior to 3.x are not supported. You will need to create new Map Definitions either by using the MapXtreme Java Manager or programmatically through the MapDefContainer interface, explained on page 135.

Note: MapXtreme Java 3.x cannot open version 4.x Map Definitions.

To get familiar with maps and MapXtreme, start by opening a sample .mdf in MapXtreme Java Manager. Experiment with different layer settings (described in Chapter 5: Managing MapXtreme Java). When you are ready to save your work, save it as a Map Definition file, as a record in a remote database table, or as a named map.
Map Features

Maps in MapXtreme are made up of layers of map objects. These map objects are accessed in MapXtreme through the Feature object. There are three basic types of features:

- **Regions**: closed objects that cover a given area. These include polygons, ellipses, and rectangles. Regions include country boundaries, postal code boundaries, and sales territories, among others.
- **Point objects**: represent single locations of data. Some examples include, customer locations, restaurants, and parking meters.
- **Line objects**: open objects that cover a given distance. These include lines, polylines, and arcs. Examples are streets, rivers, and power lines.

You can have each type of object in a separate layer (most common), or you can combine objects in the same layer. MapXtreme lets you customize and display these objects to make maps that meet your needs.

Renditions

All features in a MapXtreme Java map display with certain visible characteristics, such as the color or symbol type. This is known as a rendition or style. MapXtreme Java includes many display properties that exploit the rendering capabilities of the Java2D API, among them symbol paint for lines and regions, dashed and parallel lines, vector symbols, and scalable symbols and labels.

Renditions can be set programmatically or through the MapXtreme Java Manager’s Layer Control dialog. See Chapter 14: Labeling and Renditions for the Rendition API. See Chapter 5: Managing MapXtreme Java for more on the Layer Control.

Labeling

Labeling is a powerful feature in MapXtreme Java used to enhance your map and impart the proper message to the viewer. It is more than just adding text to describe the map feature. With MapXtreme Java you can control the font type, size, color, position, and use creative effects such as haloing and outlining, to create distinctive labels for each layer. Label content can be generated from expressions that bring in data from one or more columns in the layer’s data source. You can also create label themes if you need to differentiate among labels in the same layer.

Labeling is covered in Chapter 14: Labeling and Renditions.
Map Data Analysis

A powerful feature of mapping with MapInfo Corporation’s family of mapping software is the ability to analyze the map to learn more than if you only saw the data in row and column format. By displaying the data on a map you can make visual comparisons of your data which lead to better business decisions.

MapXtreme Java provides a number of ways to analyze the map, from using map tools to click on features, to creating thematic maps that display relationships among the features, to searching underlying data according to your own criteria.

Putting Your Data on the Map

Data Providers enable you to add data from other sources to your maps. For example, if you have an Oracle Spatial database table of sales by county, you could open that data and display it on a map in order to visually spot trends in sales patterns by county.

MapXtreme supports access to several different types of data sources, including:

- Oracle Spatial
- Informix Universal Server SpatialWare DataBlade
- SQL Server with SpatialWare
- JDBC Compliant databases (spatial data stored in X,Y columns)

Additionally, MapXtreme Java supports data binding, in which data from .tab files can be joined with JDBC data to make up a map layer.

For information on adding data from these data sources, see Chapter 10: Mapping In Layers.
Managing MapXtreme Java

This chapter explains how to manage data and named resources with the MapXtreme Java Manager.

In this chapter:

- MapXtreme Java Manager ................................................................. 51
- Map Definitions Panel ................................................................. 52
- Web Applications Panel ............................................................... 53
- Named Resources Panel ............................................................... 53
- Connections Manager Panel .......................................................... 54
- Adding Services to MapXtreme Java Manager ............................... 54
- Managing Data with MapXtreme Java Manager ................................. 55
- Geosets and Map Definitions .......................................................... 56
- Loading Existing Map Definitions ..................................................... 57
- Creating a Map Definition ............................................................... 58
- Saving a Map Definition ................................................................. 58
- Manipulating Layers with Map Tools ................................................. 64
- Layer Control ................................................................................. 69
- Display Options via Layer Control ..................................................... 74
- Labeling Options via Layer Control ................................................... 77
- Managing Named Resources ............................................................ 82
MapXtreme Java Manager

MapXtreme Java Manager is a Java server and client GUI that allows you to manage various aspects of MapInfo enterprise products, including MapXtreme Java map services, MapMarker J Server geocoding services, and Routing J Server routing services. Currently, only the MapXtreme Java services are enabled; however, you can add geocoding and routing services. See Adding Services to MapXtreme Java Manager on page 54 for more information.

The MapXtreme Java Manager server and client communicate with each other via XML. The client can be run as an application or as an applet in a browser. As an application, you can run in Web Server mode or Stand Alone mode. You can use the MapXtreme Java Manager client to:

- manage and configure map layers (Map Definitions panel)
- manage named resources, including maps, layers, and renditions (Named Resources panel)
- rapidly build prototype web applications (Web Applications panel).
- manage JDBC connections (Connections Manager panel).

About the MJM Elements

MapXtreme Java Manager (MJM) uses pre-built components to carry out specific mapping tasks. The buttons on the Map Definitions Panel are JavaBeans that you can add to your own application. See Chapter 7: MapXtreme JavaBeans for more information.

The Web application builder uses JavaServer Pages technology for rapid prototyping. The JSP tags in the custom tag library are available for your own use, as well. See Chapter 6: Web Application Builder and Appendix A: Custom JSP Tag Library.

Running MapXtreme Java Manager

Before starting the MapXtreme Java Manager, you should make sure that your Web application server (such as Apache/Tomcat) is running, since the Manager communicates with the server. You can run the Manager in stand-alone mode, without starting the server, but in that scenario some functionality (the Web Application Builder wizard) is not available.

To run the MapXtreme Java Manager Server:

- Start Tomcat. For Windows users, start Tomcat from the Program menu or run startup.bat from the tomcat/bin directory. UNIX users run the startup.sh.

Once the MapXtreme Server is running, you may then start the MapXtreme Java Manager client as an applet or application.

To start MapXtreme Java Manager client as an applet:

- Open a Web browser and point it at the URL of MapXtreme Java Manager servlet. For example: http://stockholm:8080/mapxtreme47/manager

To start MapXtreme Java Manager as an application:

- On Windows, choose the MapXtreme Java Manager shortcut on the Program menu. For others, or if you wish to run it from the command line, use this syntax.
java com.mapinfo.mjm.client.MJMClient <MJM Servlet URL>

### Security Considerations

If you are deploying a thicker MapXtreme Java client application that talks directly to MapXtremeServlet, be aware that this also allows access to the MapXtreme Java Manager servlet since it is in the same context. If you expose the mapxtreme47 servlet context outside of your firewall, this exposes MapXtreme Java Manger, as well. Someone could, by accessing http://host/mapxtreme47/manager, browse the file system and even save files there.

The solution for these thicker clients is to simply configure your Web server to not allow access to the /manager servlet.

### Map Definitions Panel

The Map Definitions panel is the location where you build your map. From the Map Definition Panel you can:

- Load and save map layers in two formats: mdf (Map Definitions) or .gst (Geosets)
- Add map layers from a file, database or named resource and edit their property settings
- Pan and zoom, change display and labeling settings
- Rearrange the order of layers that will appear in the map
- Create thematic layers, and save all settings for loading into your application whenever necessary

By default, all map layers created by the MapXtreme Java Manager use MapXtremeDataProviders, however, you have the option of using either MapXtremeDataProviders or LocalDataProviders.

The following illustration shows the MapXtreme Java Manager client interface with a Map Definition loaded. To load or create a map definition, see the instructions beginning on Loading Existing Map Definitions on page 57.
Web Applications Panel

The Web Applications panel of the MapXtreme Java Manager provides a wizard for developers to construct custom web applications quickly and easily. Using Sun’s JavaServer Pages technology, a key component of the Java 2 Platform Enterprise Edition, this rapid prototype developer allows you to create web pages with both static and dynamic content that is platform independent. JSP technology separates the user interface from content generation, thus enabling web designers to change the overall page layout without altering the underlying dynamic content.

This prototype builder uses custom JSP XML-like tags that act like widgets that you simply choose from a list and add to the builder’s layout frame. We have provided a variety of typical mapping widgets, such as a toolbar, map, layer control, and legend widgets in a JSP Library (discussed in Appendix A: Custom JSP Tag Library). Also included are instructions on how to construct your own custom JSP tags and add them to the wizard to increase the pool of widgets from which to build applications.

For instructions on using the Web Applications Panel, see Chapter 6: Web Application Builder.

Named Resources Panel

The MapXtreme Java Manager’s Named Resources panel is the location for managing resources that have been given a unique name, or alias. Here you can create contexts for named maps and named layers, and access named renditions (styles) from the mistyles context that is created during installation. Named resources are discussed on Saving a Map Definition on page 62.
Connections Manager Panel

The Connections Manager provides a user interface to manage JDBC connections (i.e., edit the contents of miconnections.properties). These are handled by the miconnections.properties file and are managed on the Connections Manager panel. See Chapter 12: Accessing Remote Data for more information.

Adding Services to MapXtreme Java Manager

With a standard installation, the MapXtreme Java Manager is automatically pre-configured with information about the MapXtremeServlet service and is ready to use that service (e.g., applications created using the Web Applications wizard will use the MapXtremeServlet to draw maps).

The MapXtreme Java Manager can also be configured with information about other services, such as geocoding (via MapMarker J Server) or routing (via Routing J Server). However, you must configure MapXtreme Java Manager manually, as described below, before it can make use of those other services.

Note: MapMarker J Server and Routing J Server are products available for purchase from MapInfo Corporation.

Adding MapMarker Support to MapXtreme Java Manager

Once you have a working MapMarker server configured, you can make MapXtreme Java Manager aware of it by adding the following initialization parameter to the MapXtreme Java Manager servlet section of webapps/mapxtreme47/WEB-INF/web.xml:

```xml
<init-param>
  <param-name>mapmarker</param-name>
  <param-value>mapmarker://testbed:4141</param-value>
</init-param>
```

The parameter’s name is mapmarker. Note its value is a pseudo-URL (starting with "mapmarker://" instead of "http://"). Use this URL to specify the host and port (separated by a colon) for your MapMarker server.

It is expected that a future version of MapMarker will operate through a conventional HTTP URL, instead of through a host and port. When that future version of MapMarker becomes available, you can replace the pseudo "mapmarker://" URL with a conventional "http://" URL to your MapMarker server.

Restart your servlet container. The next time you run the MapXtreme Java Manager, the Web Applications wizard will include additional widgets (e.g., buttons that go inside the toolbar widget) that allow you to add geocoding support to your JSP application.
Adding Routing J Server Support to MapXtreme Java Manager

Once you have a working Routing J Server configuration, you can make MapXtreme Java Manager aware of your routing server by adding the following initialization parameter to the MapXtreme Java Manager servlet section of webapps/mapxtreme47/WEB-INF/web.xml:

```xml
<init-param>
    <param-name>routing</param-name>
    <param-value>http://testbed:85/routing/servlet/RoutingServlet</param-value>
</init-param>
```

The parameter's name is routing. Its value is the URL to your Routing J Server servlet.

Restart your servlet container. The next time you run the MapXtreme Java Manager, the Web Applications wizard will include additional widgets (e.g., buttons that go inside the toolbar widget) that allow you to add routing operations to your JSP application.

Managing Data with MapXtreme Java Manager

One of the primary functions of the MapXtreme Java Manager is to manage the data you want to display in your map. MapXtreme Java displays this data as a collection of layers, each layer (ideally) made up of one type of map feature, such as political boundaries, customer locations, wireless networks, or highways.

These map layers can be saved and reused by creating Map Definitions with the MapXtreme Java Manager. Map Definitions describe what map features are to be included in the map, as well as the display settings. Map Definitions are XML-based text descriptions of the collection of map layers. Map Definitions can be saved in file format as .mdf, in records of remote databases or as named maps in the named resources repository.

Use MapXtreme Java Manager to create the appropriate base maps that will serve as reference information for your map. For example, you might want a layer of regional boundaries, a layer of streets, and a layer of store locations.

Additionally, you will want to include map data that is specific to your application, such as customer locations, cellular towers or delivery routes. Any record that has a location component to it, such as customer address, can be used as map data.
To begin building your map set, start by loading a pre-defined Map Definition in.mdf format that ships with MapXtreme Java. Instructions begin on Loading Existing Map Definitions on page 57. A sample world.mdf has been installed under /examples/server/data/machine-name.

Note: Descriptions of the sample base map data set can be found in the online PDF document MapXtreme Java Sample Data Set that is located on the product CD or in the /maps directory after installation.

Once you have opened some layers, you can customize the way in which layers display, add or delete layers, or re-order them. The tools you use to customize are described beginning on Manipulating Layers with Map Tools on page 64.

When you are satisfied with your layers’ display characteristics, you can save them as a Map Definition for loading into your application whenever necessary.

Geosets and Map Definitions

This section provides explanations of geosets and map definitions.

Geosets

MapXtreme Java’s sample data set includes MapInfo .tab format files that are grouped into geosets (extension .gst), similar in concept to a workspace. For example, there is the world.tab in the sample data and there is a world.gst. The geoset World.gst, is a type of metadata file that describes a collection of .tab files that include world.tab and others.

Geosets are one type of file that you can load into MapXtreme Java Manager. Note, however, they are limited to MapInfo .tab format and cannot be stored in a remote database or the named resources repository. Additionally, renditions in geosets cannot be changed.
**Map Definitions**

To work around the limitations of geosets, MapXtreme Java provides the sample data set in .mdf format, as well. These XML-based Map Definitions are text files that describe the map features and settings. Map Definitions can be saved as a file (extension .mdf) or stored as a record within an RDBMS, or as a named map in a named resources repository. We strongly recommend that you use Map Definitions. If you start by using geosets, save your map settings as Map Definitions when you are through.

Map Definitions that could be opened in MapXtreme Java version 3.x will automatically be imported into version 4.x. Map Definitions created in versions prior to 3.0 are not supported. You must create new Map Definitions either by using the MapXtreme Java Manager or programmatically through the MapDefContainer interface (see *Saving Your Map Programmatically on page 135*).

**Loading Existing Map Definitions**

You can open map data that is stored in four ways:

- in files (.mdf or .gst)
- as named maps (collection of layers and settings that were previously given a unique name)
- as a recently accessed map definition (MRU)
- in a remote database

The instructions that follow cover each situation.

To display an existing Map Definition:

1. Run the MapXtreme Java Manager Servlet and Client as described on *Running MapXtreme Java Manager on page 51*. The Map Definitions panel displays.
2. Click the **OPEN** button on the Map Definitions panel. The Load Map Definition screen displays. It has the following tabs: File, Recent, Named, and Database.
3. To load a Map Definition stored as a file, click the **FILE** tab and navigate to the location of the .mdf or .gst. Click **LOAD**.

   **Note:** The files are stored on the MapXtreme Java server, so even if you are running MapXtreme Java Manager as an applet on a remote computer, you will be able to navigate to the directories on the server. You do not need to be at the server computer to access the map files.

4. To load a previously opened Map Definition that was saved to the Recent tab, click **RECENT**, select the appropriate Map Definition, and click **LOAD**.
5. To load a previously named map, click on the **NAMED** tab. In the left pane click on the appropriate context, if necessary, and navigate to the location of the named map. Highlight the map in the **RESOURCES** list and click **LOAD**. (For instructions on creating a named map, see *Named Maps on page 83*.)
6. To load a Map Definition from an RDBMS, click the **DATABASE** tab.
In the CONNECTION box, choose the connection from the list of previously saved connections (if any). A named connection must be previously defined before you can save a map definition to it. See Chapter 12: Accessing Remote Data for more on creating named connections.

7. In the TABLE OR QUERY group, choose from:
   - USE MAPINFO.MAPDEFINITIONS TABLE (default)
   - USE TABLE: provide name of table, name column, and Map Def column
   - USE SQL: provide your own SQL query statement

8. In the MAP DEFINITION group click the REFRESH button to display a list of existing Map Definitions. Choose a Map Definition file and click LOAD. A Map Definition displays and is ready for you to customize.

Creating a Map Definition

The following section describes how to create a map definition using the MapXtreme Java Manager.

Using the Add Layer Wizard

You can create Map Definitions, either by adding new layers or by saving existing Map Definitions or geosets with new settings and new names. MapXtreme Java assists you with this task by providing an Add Layer wizard to walk you through the process. You will build your Map Definition one layer at a time.

Note: In the Add Layer Wizard, “remote” means you are potentially browsing a remote machine, although the data could be local as well.
The procedures below describe how to add new layers from file-based data sources, JDBC data sources, named layers, and data bound layers, using the wizard. To create a new Map Definition from an existing one or a geoset, use the Save As... button.

Adding MapInfo TAB Files and Other File-Based Map Data

To create a new Map Definition using file-based data:

1. From the Map Definitions panel, click the NEW button. Any existing layers in the map are removed leaving a blank map. Click the LAYER CONTROL button.
2. From the Layer Control dialog box, click ADD. The Add Layer Wizard appears.
3. To add a MapInfo .tab file, choose MAPINFO TAB from the list provided in the Select a Data Source panel. Click NEXT.
4. At the Specify MapInfo Table Information dialog box, click the ADD button to navigate to the file location, and then select one or more files. Click OPEN to return to the Add Layer Wizard, then click NEXT to continue.
5. In the Specify Other Layer Settings dialog box, give the layer a name, if desired.
6. Click FINISH. You are returned to Layer Control. It displays your .tab file in the Layer/Theme list. Click OK to view the layer in the map.
7. Customize the layers by setting display, label properties and layer properties, re-ordering the way in which layers display, removing or adding additional layers, and setting whether layers are visible, or contain labels. See Layer Control on page 69 for more information.
8. Click OK to leave the Layer Control dialog box when you are satisfied with the settings. Your newly created map displays. Return to the Layer Control panel, if necessary, to change display settings or rerun the Add Layer Wizard to add more layers.

When you have finished creating your map you'll need to save it as a Map Definition. See Saving a Map Definition on page 62.

Note: The above procedure applies similarly to adding GeoTIFF Raster or ESRI Shape files. To add a MapInfo Grid file, choose the TAB data source and open the .tab file that is associated with the Grid file you want to add.
Adding JDBC Map Data

To create a new Map Definition using JDBC data:

1. From the Map Definitions panel, click the NEW button. Any existing layers in the map are removed leaving a blank map. Click the LAYER CONTROL button.
2. From the Layer Control dialog box, click ADD. The Add Layer Wizard appears.
3. Select the JDBC data source, such as ORACLE WITH SPATIAL OPTION, from the list. Choose default property values or previous settings. Click NEXT.
4. Enter the connection information for your data source.
5. In the Specify Table or Query panel for your data source, choose to add a table, or you can query your database. Click NEXT.
6. Specify whether MapXtreme Java should query the MAPINFO_MAPCATALOG for settings or specify your own. Click NEXT.
   If you have feature- or label-level rendition information stored in your table, specify how the information is stored and the name of the Rendition column from the MAPCATALOG that it is stored in. The default behavior is None. Click NEXT.
7. At the Specify Other Layer Settings panel, specify a layer name (optional). Click FINISH.
   You are returned to the Layer Control dialog box. The layer you added displays in the list.
8. Repeat to add more JDBC layers.
9. Customize the layers by setting display and label properties, re-ordering the way in which layers display, removing or adding additional layers, and setting whether layers are visible, or contain labels. See Layer Control on page 69 for more information.
10. Click OK to leave the Layer Control dialog box when you are satisfied with the settings. Your newly created map displays. Return to the Layer Control dialog box, if necessary, to change display settings or rerun the Add Layer Wizard to add more layers. When you have finished creating your map you'll need to save it as a Map Definition. See Saving a Map Definition on page 62.

Adding a Named Layer

A named layer is a layer that you have previously saved with a unique name. You can retrieve named layers via the Add Layer Wizard like any other layer. For instructions on creating a named layer, see Named Layers on page 84.

To add a named layer:

1. From the Map Definitions panel, click NEW. Any existing layers in the map are removed leaving a blank map. Click the LAYER CONTROL button.
2. From the Layer Control dialog box, click ADD. The Add Layer Wizard appears.
3. Choose NAMED LAYER from the list of data sources provided in the Select a Data Source panel. Click NEXT.
4. Click the CHOOSE NAMED LAYER button to display the NAMED RESOURCES dialog box. In the left pane, click on the context that contains your named layer. Highlight a named layer in the Resources panel and click LOAD.
5. At the Select a Named Layer panel, click FINISH. You are returned to the Layer Control dialog box showing the layer you added. Click OK to view the layer in the map.
6. Continue to add other layers to build your map. Customize the layers by setting display and label properties, re-ordering the way in which layers display, removing or adding
additional layers, and setting whether layers are visible, or contain labels. See Layer Control on page 69 for more information.

7. Click OK to leave the Layer Control dialog box when you are satisfied with the settings. Your newly created map displays. Return to the Layer Control dialog box, if necessary, to change display settings or rerun the Add Layer Wizard to add more layers. When you have finished creating your map you'll need to save it as a Map Definition. See Saving a Map Definition on page 62.

Adding a Data Binding Layer

The Data Binding feature allows you to build a layer by joining the feature geometry in a MapInfo .tab file with the attribute information in a JDBC database.

To create a new Map Definition by adding a data bound layer:

1. From the Map Definitions panel, click the NEW button.
2. From the Layer Control dialog box, click ADD. The Add Layer Wizard appears.
3. Choose DATA BINDING from the list of available data sources. Click NEXT.
4. At the Specify MapInfo Table Information panel, click the ... button to navigate to the .tab file location. Click OPEN to return to the Add Layer Wizard, then click NEXT to continue.
5. Specify the Bind Layer data source information (JDBC driver class name, connection URL, user name and password). Click NEXT.
6. Provide the table or query that you want to bind with the MapInfo .tab geometry. Optionally provide an owner name for the table or ID Column for the query. Click NEXT.
7. At the Geographic and Join Layer Columns panel, enter the column name for the geographic layer (MapInfo .tab file) and click ADD. For the Join layer from the JDBC data source, specify the column name that correlates to the geographic column and click ADD. If you want to join using additional columns, add them here. Click NEXT.
8. Optionally, give the layer a name. Click FINISH. You are returned to Layer Control dialog box. The data binding layer is displayed in the Layer/Theme list. Click OK to view the layer in the map.
9. Customize the layer by setting display and label properties, re-ordering the way in which layers display, removing or adding additional layers, and setting whether layers are visible, or contain labels. See Layer Control on page 69 for more information.
10. Click OK to leave the Layer Control dialog box when you are satisfied with the settings. Your newly created map displays. Return to the Layer Control dialog box, if necessary, to change display settings or rerun the Add Layer Wizard to add more layers. When you have finished creating your map you'll need to save it as a Map Definition.

Creating an Analysis Layer

You can use the Add Layer Wizard to create a pie or bar chart as an Analysis Layer. To create an Analysis Layer:

1. From the Map Definitions panel, click the OPEN button. Open the file to which you want to add an Analysis Layer. Click the LAYER CONTROL button.
2. From the Layer Control dialog box, click ADD. The Add Layer Wizard appears.
3. Choose ANALYSIS LAYER from the list of available data sources. Click NEXT.
4. At the Specify Layer and Columns to Analyze panel, select the layer, the columns to analyze and the type of analysis. The bars in a side-by-side chart display vertically, while the bars in a stacked bar chart display horizontally. Click **NEXT**.

5. At the Specify Pie Chart Properties panel, select the wedge colors, chart size, orientation, and position. Click **NEXT**.

If you selected one of the bar chart types in **step 4**, select the bar colors, chart size, and orientation. Click **NEXT**.

6. At the Specify Other Layer Settings dialog, provide a name for the new layer. Click **Finish**.

**Saving a Map Definition**

You have several choices in how you can save your Map Definition.

You can save to a file, as a named map, or to a remote database.
1. To save your map, from the Map Definitions Panel click the SAVE button. The Save Map Definition dialog box displays with four tabs: File, Recent, Named, and Database. Go to the appropriate tab for the type of Map Definition you are saving.

2. If you are saving to a file, click the FILE tab and provide the filename and location where you want the file stored. Click SAVE. It is stored in .mdf format.

3. If you are saving the map as a named map, from the Named tab, click on the context where the map definition will be stored. Provide a name for the map definition in the Name box. Click SAVE. The information is now saved as a Map Definition in XML format in the specified context.

   If you do not have a context to hold your named map, you must set it up. Go to the Named Resources panel in the MapXtreme Java Manager client. For more information, see Managing Named Resources on page 82.

4. To save the Map Definition to a remote database, click the DATABASE tab.
5. In the Connection box, choose the connection from the list of previously saved connections (if any). A named connection must be previously defined before you can save a map definition to it. See Chapter 12: Accessing Remote Data for more on creating named connections.

6. In the Table or Query group, choose from:
   - **Use MAPINFO.MAPDEFINITIONS Table** (default)
   - **Use Table:** provide name of table, name column and Map Def column
   - **Use SQL:** provide your own insert/update SQL statement

7. In the Map Definitions group specify a map name and click **Save**. The information is now saved as a Map Definition in XML format in the database.

---

**Manipulating Layers with Map Tools**

Once you have your map layers open in MapXtreme Java Manager, you will want to customize their display for your own needs. Use the buttons on the Map toolbar to control how your map displays.

The following tools are discussed: Zoom In, Zoom Out, Pan, Ruler, Info, Selection tools, Preferences, Map Options, Create Theme and Layer Control.

**Zoom In Tool**

Use the Zoom In tool to get a closer area view of a map or a layer.
To use the Zoom In tool, click the Zoom In cursor in the center of the area you want to zoom in on. This magnifies the area by a linear factor of two. This point will be at the center of the map in the zoomed-in view. Repeat this procedure until you have the appropriate level of enlargement.

To zoom in on a rectangular area draw a marquee in the map or layout by diagonally dragging the Zoom In mouse cursor. The area within the marquee is enlarged.

**Zoom Out Tool**

Use the Zoom Out tool to get a wider area view of a map or a layer.

To use the Zoom Out tool, click the Zoom Out cursor on the center of the area you want to zoom out on. This enlarges the area by a linear factor of two. This point will be at the center of the map in the zoomed-out view.

**Pan Tool**

Use the Pan tool to reposition a map within its window.

To move or adjust the map display, click an area of the map and while holding down the mouse button, drag the map in the appropriate direction. When you release the mouse button, the map is redrawn in its new location.

**Ruler Tool**

Use the Ruler tool to measure distance between points and total distance among multiple points.

To use the Ruler tool, click on the starting point on the map. Double-click on the ending point. The Ruler window shows the distance between the two points.

To show the distance of intermediate points along a route, click on a starting point and continue to click on additional points. The Ruler window will show the last segment distance and the total distance. To end the operation, double-click on the last point you wish to measure or press the Esc key.

**Info Tool**

The Info tool provides a view of the attributes that are associated with map objects that exist at a given point.

To use the Info tool, click on a map feature. The Info tool window displays.
If the objects for only one layer are selected, the Info tool window displays the attributes associated with the map object at that point. Every attribute column for that record is viewable. You may need to enlarge the window to see the complete list.

If the map contains multiple layers, the Info tool window displays the list of layers. Highlight a layer and double-click to view the layer level attribute information. Click the List button to return to the list of layers.

Selection Tools

MapXtreme Java enables you to select individual or multiple features in a map layer in order to perform additional operations on them. MapXtreme Java Manager provides five selection tools. The Selection tool allows you to select individual features. The other four tools provide different confining areas (rectangle, radius, polygon, boundary) within which features are selected.

A behavior common to the five selection tools is the ability to add selected features by using the Shift key. For example, click with the Select tool to select a feature, shift-click to select another feature that you want included with the first selection.

All features are selected from the topmost selectable layer in the map. Layers are not selectable by default. You must check the Selectable box in Layer Control to indicate the layer you wish to select from.

Once your features are selected, you can save them to a Selection theme. See Chapter 15: Theme Mapping and Analysis for more information on Selection themes.

Select Tool

To use the Select tool choose the Select tool by your preferred method and click on the feature you want to select. Shift-click to select additional features.

Select Within Rectangle Tool

To use the Select Within Rectangle tool, choose the tool and click and drag the mouse to form a rectangle around the objects you wish to select. Shift-click and drag the mouse again to select features in another rectangle that you wish included with the first selection.

Select Within Radius

To use the Select Within Radius tool, choose the tool and click and drag to form a circle around the objects you wish to select. Shift-click to create an additional circle that will add the selected features to the first selection.
Select Within Polygon Tool

To use the Select Within Polygon tool, choose the tool and click on the map to start creating the polygon that will enclose the features you wish to select. Continue to click at additional points to form the polygon. Complete the polygon by clicking close to the starting point or by double-clicking. Be sure that the polygon has at least three nodes.

Select Within Boundary Tool

To use the Select Within Boundary tool, choose the tool and click on a boundary object to select all features within that boundary. Shift-click in another boundary to add those selected features to the collection.

Other Map Tools

This section describes other tools available to you such as preferences and map options.

Preferences

Click the Preferences button to set the default map directory, MIME type (image/gif is the default), and whether MapXtreme Java is to render maps remotely (default) or locally.

Local rendering is mainly used for testing purposes such as to confirm that local rendering works for your application. It should not be used if the client does not have all the required resources available for your application or applet. For example, some maps might not display properly if the client does not have required fonts installed.

Map Options

To change settings for distance units and projection use the Map Options button.
Choose the distance unit from the drop-down list of the Map Options dialog.

To change the coordinate system, click the Display CoordSys drop-down list to display the Choose Coordinate System dialog.

Coordinate system data is stored in a projection file called micsys.txt This file lists hundreds of supported coordinate systems and the parameters that define them.

The file micsys.txt is contained in micsys.jar. When MapXtreme Java Manager runs as an application, micsys.jar is loaded from MapXtreme’s lib/common directory. When MapXtreme Java Manager is run as an applet, micsys.jar is loaded from the servlet container’s webapps/mapxtreme47/client directory.

**View Entire Layer Button**

Use the View Entire Layer button to view the extents of the selected layer. Choose the layer in the View Entire Layer dialog, and click OK. The map redraws at a recalculated zoom level so that the extents of the selected layer are visible on the map, allowing you to see the entire layer.

**Note:** You may not be able to use the View Entire Layer dialog to view the extents of a query layer. Specifically, if your layer is defined as a query, and the layer does not have a QueryBuilder, then you will see an error message if you attempt to perform View Entire Layer on that query layer.

**Add Theme Tool**
The Add Theme Wizard is a guided tool to allow you to add a Feature theme or Label theme to your map.

For Feature themes, you can create either a Ranged theme that groups features according to a range of values, or an Individual Value theme that shades features based on a value.

The theme can be based on any supported column and layer in the current map. Currently there is support for creating a theme based on numeric, string and date column data, and on point, line and region layers.

Part of the operation will be to create a default theme legend that is associated with the new theme. The legend can be customized to change the title, fonts, insets, descriptive text, and colors.

For Label themes, MapXtreme Java supports three types: Ranged, Individual Value, and Selection. For example, to differentiate between major and minor cities in the same layer, use a Ranged Label theme to group the cities based on their corresponding data, such as population.

Label themes are further discussed in Chapter 14: Labeling and Renditions.

For more on the Add Theme Wizard, see the AddTheme Bean on page 98.

Layer Control

The real power in controlling the display of your map lies in Layer Control. Its options enable you to display, remove, add, edit, select, zoom layer, and label your layers. You can also change the order of map layers and themes.

To access the Layer Control dialog, click the LAYER CONTROL button on the Map Toolbar.
The Layer Control dialog shows all the layers that make up the current map and the status of the layer attributes. These attributes are: Visible, Selectable, and Autolabel. The icons above each check box column represent the attributes. It is easy to change the attributes for one or more layers using the check boxes.

You also have options available to change the display, label, and theme settings, and to reorder, add, edit, or remove layers. These settings will be maintained when you save the Map Definition. You can also save layers as named layers via Layer Control’s Save button.

The following sections discuss the features that comprise the Layer Control dialog.

**Layer/Theme Visibility**

The Visible attribute in Layer Control controls whether a layer or theme is visible on the map. For example, to make a layer invisible, clear the layer’s Visible check box. The layer will no longer display in the map (however, it will still be included in the Map Definition when you save it). This feature is especially useful when you want to focus on only one or two layers of a multi-layered map.

**Layer Selectability**

MapXtreme Java enables you to select map objects for further manipulation and analysis. The Selectable check boxes allow you to control whether a layer’s map objects can be selected. By default, layers that are added to the map are not selectable.

This feature is useful when you wish to select objects from a lower layer. Turn off the selectable status of all the layers that are above the layer you wish to select from.

**Autolabeling**

Any layer that has a check in the Autolabel checkbox will display labels, provided the zoom level settings for the label are appropriate. Clear the check box for any layer that you do not wish to display labels. By default, layers that are added to the map are not autolabeled.
See Labeling Options via Layer Control on page 77 for details on setting and controlling the display of labels.

**Reordering Layers and Themes**

Map layers and themes display in the order that they are listed in the Layer Control dialog, with the bottom layer drawn first and the top layer drawn last. It is important to order your layers correctly so that map features are not hidden by others.

For example, you have a layer of customer points and a layer of census tracts. If the layers are incorrectly ordered in the map, your application might draw the customer points first and then display the census tract layer second. Your customer points would be obscured by the census tract layer.

To change the order in which layers are displayed, use the Up and Down buttons. Select the layer(s) you want to reorder and click either the Up or Down button to move the layer(s) to a position above or below its current position. To reorder a theme, you must reorder the base layer it is associated with. If there are multiple themes for the base layer, you can reorder a theme within the list of themes.

**Ordering Objects in a Layer**

MapXtreme does not allow you to control the front-to-back ordering of objects within a single map layer. If you need to control this kind of ordering (e.g., you need to make sure that your lines display on top of your regions), put the different object types in separate layers. Put your line objects in one layer, and put your region objects in another layer. Then use the Layer Control dialog to order the layers.

**Adding Layers: Add Layer Wizard**

Layer Control provides an Add Layer Wizard to make adding layers easy. You can add layers from a number of sources, including:

- files
- remote databases
- previously saved named layers
- data bound layers (two sources)

To add a layer to the map, click the Add button and follow the prompts (instructions are on Using the Add Layer Wizard on page 58).
The data source information comes from a properties file from which the Wizard initializes. The list of data sources can be modified through the settings in the `addlayerwizard.properties` file. This file also contains default values for certain controls in the wizard. It also saves the values that you have entered for a specific data source and offers the option of using those values the next time you run Add Layer Wizard.

The `addlayerwizard.properties` file is located in MapXtreme-4.70/lib/client. You can modify it as needed to change the configuration of the Add Layer Wizard to best suit your needs. This assumes that you are running the MapXtreme Java Manager client as a standalone application.

Another copy of `addlayerwizard.properties` is stored in mjmappletsup.jar in webapps/mapxtreme47/client; this copy is used when you run the MapXtreme Java Manager as an applet. When run as an applet MapXtreme Java Manager uses slightly different security settings, so it needs its own copy of the properties file.

If you need to edit the `addlayerwizard.properties` file, make sure you edit the appropriate copy of the file for the application you plan to use.

For customizing the `addlayerwizard.properties` file, see Chapter 17: Customizing the AddLayer Wizard.

**Editing Layers: Edit Layer Wizard**

The Edit Layer Wizard displays when you click the Edit button from Layer Control. Use the wizard to view and/or edit a layer’s properties, such as JDBC connection information, coordinate system and geometry settings, and layer name. Use the wizard when you wish to view what type of layer it is, as that is not displayed in Layer Control. With the Edit Layer wizard you can make minor changes, such as the name of the layer as it will display in Layer Control, or major changes such as host and port for a remote data source. All types of layers except Annotation layers, can be edited.
The Edit Layer Wizard looks and behaves like the Add Layer Wizard, except that the fields in the Edit Layer Wizard are already populated with information. You can change any property for the layer except the type of layer it is. For example, you cannot take a TAB layer and redefine it as a JDBC layer.

When you finish going through the wizard, the selected layer will be rebuilt with the new properties and will replace the layer you had selected originally.

**Note:** The Edit Layer Wizard sometimes discards a layer’s themes and custom labeling expressions. This can happen if you use the wizard to re-define the layer completely, so that the resulting table does not have the same columns as the original.

If you are concerned about the possible loss of themes, examine the Layer Control dialog when you are finished with the Edit Layer Wizard. If the themes were maintained, you will see them listed (you may need to click the layers’ expand/contract icon to display them). If they are not there and you want to keep them, click Layer Control’s Cancel button to cancel the changes you made in the Edit Layer Wizard.

### Removing Layers and Themes

To remove layers, select the layer you want to remove and click the Remove button. The selected layers are removed from the Layer list. Click OK or Apply in the Layer Control dialog to redisplay the map. These layers are not saved in the Map Definition.

**Note:** Remove does not delete any files. It only prevents the layer from being displayed in this particular map.

### Theme Button

If you have already created a feature theme or label theme, you can control the zoom range through Layer Control’s Theme button. Additionally, distance units can be changed here. (Use the Add Theme tool on the Map Definition interface to create a theme.)

### Save Button

To save a layer as a named layer, use Layer Control’s Save button. You will be brought to the Save Named Layer dialog where you can specify a name for the layer, as well as its stored location.
Display Button

The Display button brings up the Display Options dialog where you can customize the display for each layer in a map, including setting the zoom range and overriding the style.

Label Button

The Label Options dialog is accessible from Layer Control’s Label button, where you can create a wide variety of label styles for your map layers. Label options are presented on Display Options via Layer Control on page 74.

Display Options via Layer Control

The Display Options dialog enables you to customize the display for each layer in a map, including setting the zoom range and overriding the style. In Layer Control, select a layer and click on the Display button to bring up the Display Options dialog.

Zoom Range

Set the zoom at which a layer displays in this dialog. You can also set the units for the layer. Zoom layering controls the display of a map layer so that it displays only when the map’s zoom level falls within a preset distance.

For example, your map contains a layer of streets and a layer of postal code boundaries. When you zoom out past 30 or so kilometers, the appearance of the streets degrades. This is because the zoom (window width) is too wide to show detailed street maps.

Use Zoom layering to tell MapXtreme to display the street layer only when the zoom is set to a distance that allows you to see the street detail properly, for instance, less than 8 km.

Different layers in the same map can be displayed at different zoom levels. For example, your map contains a layer of streets, a layer of county boundaries, and a layer of state boundaries. You want the streets layer to be visible only when the zoom level is less than eight miles. You want the county boundary layer to display when the zoom level falls between 20 miles and 200 miles. You want the states boundary layer to be visible only when the zoom level is greater than 100 miles.
Style Override

One of the Display options available to you is the ability to override the display of a layer with a different style, such as a new color for regions, new symbol for points or a new line type for line segments or region borders. These style overrides are set in the Display Options dialog.

You can override the display with pre-defined styles that ship with MapXtreme. Or you can customize these named renditions in a variety of ways to give your map features the exact look you want. When you save your map as a Map Definition the style changes will be saved.

Setting a Style Override

To change the style of a map feature, highlight the layer in Layer Control and click the Display button. At the Display Options dialog, click the Select Style button to reveal the Select a Style dialog. Click on the mistyles context in the left pane to display the folders of available styles, then choose the appropriate folder: either brushes, fonts, or pens.

Use the brush styles to change the pattern, fill, or foreground color of map features. Choose Brush_002 to choose a solid fill or Brushes_003-034 for patterns. To set a transparent fill, choose Brush_001. The brush numbers display as tool tips when your cursor hovers over a style swatch.

Use the font styles to override the style for symbols, including symbol type, color, size, and unit. Units can be either paper units, such as millimeters, or geographic units, such as miles, that allow your symbols to resize themselves at different zoom levels. Nine font sets are installed with MapXtreme Java.

Use the pen styles to change the line style, color, and thickness of roads or other linear features. Pen styles also are used to change the border of polygon features.

Overriding with a Pre-Defined Named Rendition

To override the layer’s display style with a pre-defined style, choose the style from the swatch panel. The style retains its pre-defined name in the Name box. Click Load to return to the Display Options dialog. Click OK twice to view the effect of the style override on your map.
For example, to change all stars to triangles, choose the triangle symbol from the Map Symbols font set. Click Load, then OK twice to view the map. The symbol now displays as a triangle at the original point size. Note that the only change here is to the symbol. The symbol still displays with the same point size because you did not override the point size (doing so would be considered a custom style change, explained below).

**Overriding with a Custom Style**
To override the layer’s display style with a custom style, begin by choosing a pre-defined style and make changes to it. Any changes to color, size and unit, and line thickness change the pre-defined style to a custom style.

For example, to change postal code boundaries to blue hatched features with thick borders, from the Brush swatches, choose the hatch pattern of your choice. (If you loaded this pattern at this point, it would still be considered a pre-defined named rendition). Click on the Fill color box and choose blue from the color picker. The name of the style in the Name box has now changed from brush_001 to "Custom Style." In the Line Thickness box, specify a new thickness that will be applied to the border of the boundaries. Click Load, then OK twice to view the style change.

**Style Overrides for Layers with Multiple Feature Types**
You can also set style overrides if you have more than one feature type in the same layer. For example, to change the color of your postal code boundaries and the color of the symbol used to display post offices that are contained in the same layer, start by setting the style override for the boundaries. Choose a brush pattern and color.

To change the color of the symbol style click the Symbol tab to access the Symbol color picker. Choose a new color. Note that the symbol style in the Selected Style window shows a square symbol with the new color. This is a default symbol style to indicate the color change. The symbol type will not change in this example. When you are through making changes for both feature types, click Load and OK twice to view the changes in your map.

**Overriding with a Gradient Fill or Stroke**
You can set a style to use a gradient fill or stroke. You have the choice of using a linear or radial gradient. A linear gradient a transition through a series of colors along a straight line. A radial gradient a transition through a series of colors along a circular path. For a complete discussion of gradients, refer to [Gradients on page 221](#).

To use gradient fills or strokes, in the Select a Style dialog, select Radial Gradient or Linear Gradient for the fill or stroke property you want to override. You can then click the ellipse button to display the Specify Radial Gradient or Specify Linear Gradient dialog.
You can now specify the aspects of the gradient to meet your needs. Once you are satisfied with the gradient, click OK.

**Saving New Named Styles**

MapXtreme comes with a large assortment of standard named styles, such as brush_002 and pen_118. You also can create your own custom named styles. For example, you could create a named style called “BigBlueStar” representing a 24-point, blue star symbol.

To save a new named style resource:

1. Run the MapXtreme Java Manager client.
2. Go to the Named Resources tab.
3. Click the **CREATE A NEW STYLE** button. The Select a Style dialog box appears.
4. Select one of the existing named styles, such as brush_002, as a starting point for your custom style.
5. Apply whatever custom style properties (color, etc.) you want.
6. In the Contexts tree at the left, click on the folder where you want to save your new named style.
7. In the Name textbox, type the name you want to use for your style. Short but descriptive names are best, such as “bluestar”.
8. Click the **SAVE** button. Your new named style is saved on the server.

*Note:* The Select a Style dialog only provides a Save button when you launch it from the Named Resources tab. If the Select a Style dialog is launched from another location, such as the Layer Control dialog, then the selected style can only be applied to the map, it cannot be saved to a named style resource. (This restriction exists for the sake of security. If the Select a Style dialog always allowed users to save named styles, then any application or applet that used the Layer Control bean would allow any user to create or overwrite named styles on the server.)

**Labeling Options via Layer Control**

MapXtreme Java offers a wide variety of label properties to give your maps a distinctive and sophisticated look. To change the label properties for a layer, select the layer and click the Label button to display the Label Options dialog.
The Label Options dialog has three panels to enable different label features:

- **Text** (column name or expression, multiline display, font attributes)
- **Visibility** (zoom settings, duplicate/overlapping labels)
- **Position** (offset from anchor point, horizontal and vertical alignment)

**Label Text Tab**

To set the column for labeling, from the Text tab, select the column from the Label With: drop-down list. This list defaults to the first column of the table.

Additionally, you can create an expression using column data or static text, or a combination of both. To combine both you would create an expression for the label.

For example, you want to create a label that begins with the static text "Pop:”, followed by the actual population values from a column called POP_2000. From the Label with drop-down list, choose Expression. The Expression dialog displays.

Type "Pop:” in the Expression window. Be sure the text is in quotes. Choose the column POP_2000 from the Column list. It will be entered automatically in the Expression window. Note that " " + is inserted between the static text and the column entry. Click OK to leave the Expression dialog. Click OK to leave Layer Control to view the results of the label expression.

If you create an expression for your labels and wish to display it on multiple lines, in the Expression window add the characters "\n" at the point in the expression where you want the line to break.

For example:

```
Pop_2000 + " " + \n + Pop_Grw_Rate
```
will display a label with two lines of text describing the population and growth rate for each feature.

**Label Style**

To change the style of the label, click the Styles box to display the Font options dialog. Choose from an assortment of style enhancements including: font name, size, units, color, backgrounds (halo or box), outline, bold, underline, and italic.

Click on the Text Color box to reveal the color picker. You can either pick the color from the swatches, or specify the exact color values you want in the RGB (Red, Green, Blue) or HSB (Hue, Saturation, Brightness) tabs. Note that when you pick a color swatch, the RGB and HSB values are updated in their respective tabs.

**Label Visibility Tab**

You can configure labels to display only within a specific zoom range, the same way that you display map layers within a certain zoom range. To specify a zoom range for labels, check the Display within Zoom Range box. Then, set the maximum zoom and minimum zoom at which the labels will display.
To keep from creating a cluttered map when labeling all layers, set the label zoom at different values so that as you zoom in or out, the appropriate labels display. For example, set the zoom for the world layer at 0 miles minimum and 2000 maximum. Set the capital city layer to display labels between 200 and 500 miles. Set the city layer to display labels only when zoomed in below 50 miles.

Check or clear the boxes to control whether you have duplicate or overlapping labels. Choose Allow Duplicate Text if you need to label a map object in more than one location with the same label. If you do not allow duplicate or overlapping labels, MapXtreme Java will only label objects that do not violate this setting. Keep in mind that a map with a lot of overlapping or duplicate labels will be hard to read.

Labeling for multiple layers is carried out according to a priority and whether duplicate or overlapping text settings are set. Those layers drawn last will take precedence over lower layers drawn first. To override the priority, check the Override Label Priority box and give the layer a new priority number. The higher the number, the higher the priority. A label’s default priority is calculated by (number of layers - layer position) * 10. The illustration above shows a priority of 30 for the world layer since it is one of four layers, listed second (layer position = 1) (position 0 is the topmost layer).

Intragroup priority is used if the priority between labels is tied.

To set labels to display on more than one line, set the Multiline Text mode to On. Existing line breaks in the label text are respected. Off (default) means that all text will display on one line. Compute means that MapXtreme Java will determine on the fly if the text needs to be displayed on multiple lines. The Compute operation is the slowest of the three operations because of the large runtime overhead.
Label Position Tab

To control the label position, in the Position tab set the offset distance that a label will be from the label point of the map object. Enter the screen coordinates in pixels in the X and Y boxes that you want used for the new label position.

You may also set the horizontal and vertical alignment for the label. Labels can be aligned horizontally to the left, center, or right of the map centroid and/or vertically from the baseline, top, bottom, or center of the centroid. Alignment refers to the edge of the bounding box of the label that is nearest the label point. Left alignment means the left edge of the bounding box is nearest the label point, which means the label displays to the right.

You can also set the label to rotate with the line. In the Label Options dialog, the mini-panel will update to show placement of the new settings relative to an anchor point.

In addition to straight-line labels, you can set labels for polylines or polygons to follow the curved path of the object. Labels will be drawn along the polyline or along the polygon boundary when you select this option. Keep in mind that curved labels are computed on the fly, taking into consideration all the label settings, such as horizontal alignment, creative effects, and visibility settings.
Some settings do not apply to curved labels. Multiline text mode is not yet supported for splined labels. Geometry calculation mode is ignored since placement for splined labels is always dynamically computed.

**Geometry calculation mode** allows you to control the label placement when you change the map view. For example, if you zoom in very closely it’s likely that the label point, which was set to the approximate centroid of the object, will now be out of view. To remedy that, set the calculation mode to Compute so that MapXtreme Java redraws the labels on the fly in the new locations. The locations are calculated based on the clipping of the view rectangle. The default behavior (Static) is that no recalculation of the label point occurs. The original label point is used for every view.

### Managing Named Resources

Named resources allow a name to be attached to a resource. This offers many advantages:

- Allows a resource to be known by its name and not by its properties.
- Allows a resource to be located in one spot but be referenced from many locations, which makes administration of resources easier.
- To change the look or behavior of applications or data, the resource only need be changed, not each application or data file.

MapXtreme Java supports several named resources that can make storing and retrieving map information easier. The MapXtreme Java Manager provides a Named Resources panel for managing named maps, named layers, and named renditions. The Connections Manager is the administration tool for named database connections. The Named Resources panel, as well as named maps, layers, and renditions are discussed below. Named database connections and the Connections Manager panel are discussed in Chapter 12: Accessing Remote Data.

### Named Resources Panel

The Named Resources panel on the MapXtreme Java Manager is the location for managing all supported named resources. The illustration below shows the panel with contexts for three named resources. The mistyles context is automatically available after MapXtreme Java is installed and
contains a variety of pre-defined styles for fill, lines, and symbols. The contexts for named layers and named maps are user-created. If you wish to save any layers or maps as named resources, you must first create their contexts here.

The Named Resources Repository Location at the top of the panel is, by default, the URL for MapXtremeServlet. If you create resource repository elsewhere, you can set the path here so that named resources at that location will display. The location may be either a file-based URL (e.g., file:///c:/mapdata/mymaps) or an HTTP URL which points to a running NamedResourcesServlet.

The Context pane provides buttons for creating and deleting contexts. The Resources pane shows the named resources for the highlighted context. To view the properties of a named resource, click on the Info button. To delete a resource click on the Delete button.

**Note:** The resources in the mistyles context are read-only and cannot be deleted.

**Named Maps**

Named maps are maps referred to by an alias, such as "myeurope," "salesterr," that represent a unique collection of map layers. The information that defines a named map is stored as XML map definitions. Named maps are created in the MapXtreme Java Manager, as explained below, or programmatically (see Chapter 9: MapJ API).

**Creating a Named Map**

To create a named map using the MapXtreme Java Manager:

1. From the Map Definitions panel of the MapXtreme Java Manager Client, open an existing map definition or geoset to display a map. You can also build a map by clicking on the **Layer Control** button and adding layers.
2. Make any adjustments to the display of the map, such as zoom, labeling, addition of thematic layers, etc., using the tools provided in the MapXtreme Java Manager.
3. Click the **Save** button to save the map as a named map. The Save Map Definition dialog box displays. Click on the **Named** tab if it is not already displayed.
4. On the left, click on the subfolder under the root context for storing your named map. If you do not have a context for storing the named map, you must first create one in the Named Resources panel of MapXtreme Java Manager before you can save a named map. See Named Resources Panel on page 82.

5. In the Name text field, type a name for the map. Click **SAVE**.

Named Layers

Named layers are map layers that you give a unique name to, so that you can recall the layer by name in the future. To save a map layer as a named layer, click the Save button in Layer Control and provide a name for it in the Save Named Layer dialog box.

To retrieve a named layer, from Layer Control, click the Add button to display the Add Layer Wizard. Choose the named layer from the list of data sources provided and follow the wizard’s prompts.
Named Renditions

MapXtreme Java provides a wide variety of fill, line, and symbol styles that you can select to override an existing rendition for a layer. These styles are accessed by clicking on the Display button in Layer Control. For a discussion of the Display Options dialog, see Display Options via Layer Control on page 74.

The pen, brush, and symbol styles that are used to draw lines and borders, fill regions, and represent point locations are XML files that reside, by default, in the mapxtreme47/resources/mistyles directory.

These styles correspond to standard styles used in MapInfo Professional/MapBasic. To view thumbnails of these styles, see Understanding MapBasic Style Strings in Appendix B on page 312.

You can customize the pre-defined styles by changing the color, line thickness and symbol size, depending on the elements of the style. To save these custom styles, you must save the Map Definition, which will write out the XML for the custom styles. You cannot save them as new named renditions in a named resources repository to be re-used in another Map Definition.

To save custom renditions programmatically, see the MapJ API and Chapter 14: Labeling and Renditions for more information.

Named Database Connections

Named database connections are simply connections to remote databases that are referred to by an alias instead of by cumbersome JDBC details such as the driver, URL, username, and password. These named connections are stored in an miconnections.properties file that creates a pool of connections that MapXtremeServlet or MapJ can access.

Named database connections are managed by the Connections Manager panel of the MapXtreme Java Manager. See Chapter 12: Accessing Remote Data for more information.
This chapter details the wizard that MapXtreme Java provides for prototype development for Web applications.

In this chapter:

- Using the Web Application Builder ........................................... 87
- Custom JSP Tag Library ......................................................... 90
Using the Web Application Builder

The Web Application Builder panel of the MapXtreme Java Manager provides a wizard for developers to construct custom web applications quickly and easily. Using Sun’s JavaServer Pages technology, a key component of the Java 2 Platform Enterprise Edition, this rapid prototype developer allows you to create and maintain web pages with both static and dynamic content that is platform independent. JSP technology separates the user interface from content generation, thus enabling web designers to change the overall page layout without altering the underlying dynamic content.

This prototype builder uses custom JSP XML-like tags that act like widgets that you simply choose from a list and add to the builder’s layout frame. We have provided a variety of typical mapping widgets, such as a toolbar, map, layer control, and legend widgets in a custom JSP Library (discussed on Custom JSP Tag Library on page 90). Instructions for constructing your own custom JSP tags and adding them to the wizard are located in Chapter 17: Customizing the AddLayer Wizard.

The application that you create is saved as a .JSP file in a web archive file (.war).

The Web Application Builder wizard walks you through the steps to create a prototype web application. To begin:

1. Launch the MapXtreme Java Manager client (be sure the MapXtreme Java Manager Servlet is running).
2. Click on the WEB APPLICATIONS tab to display the first dialog of the Web Applications Builder wizard. Click the CREATE NEW WEB APPLICATION button. Provide a name and description for your prototype and click NEXT.
3. At the Application Layout dialog, click in the LAYOUT frame on the right side of the dialog to activate the layout operations.
4. Choose a widget from the left hand listbox and click the **ADD WIDGET** button. The widget appears in the Layout frame.

5. To add additional widgets, click the **SPLIT CELL** button and choose how you wish the layout frame to be divided, either horizontally or vertically. You can adjust the size of the split cell to suit your needs.

6. With the new cell active, highlight a widget and click the **ADD WIDGET** button.

7. Continue to split cells and add widgets until you have all the elements you want included in the application. Click **NEXT**.

8. You may also edit the properties of any widget included in the layout by selecting it and clicking the **EDIT WIDGET** button or by simply double-clicking on the widget. See **Editing Widgets on page 89** for more information.

9. At the Application Deployment dialog, provide information to create the application including the name of the web archive (*somefile*.war), main JavaServer Page (*mymapapp.jsp*), and associated map file (click the ... button to browse to the location).

   ![Web Application Wizard](image)

   **Note:** To change the deployment directory, first select the **CHOOSE CUSTOM DEPLOYMENT LOCATION** option.

10. If you deploy into the same container as the MapXtreme Java Manager servlet, you may opt to enable Servlet Forwarding, an optimized method of rendering maps. See **Code Sample: Servlet Forwarding on page 180** for more on this feature.

11. Click **FINISH** to create the application.

12. To test the application, restart your servlet container to make the new application available. In a web browser, type the URL that is listed in the deployment dialog.

   **Note:** If you are using Tomcat and you created the .war file in the default location, simply restart Tomcat to deploy the .war file. Other application servers may require additional steps before a .war file will be deployed.
Using Prototype Builder Templates

You can save any layout you create and use it as a new template for future application building. Once you are satisfied with the application layout, click the **SAVE TEMPLATE AS** button and give the template a name. The next time you build a prototype you can select the template name from the drop-down list and modify the layout further if you choose.

Editing Widgets

The Application Layout dialog of the Web Applications Builder contains an **EDIT WIDGET** button that provides access to editor dialogs for each widget. Here you can control a variety of attributes that affect the look and behavior of the map widgets. Once you have added the widget to the layout frame, you can click the **EDIT WIDGET** button to display the Editor dialog.

For example, to add, remove, or reorder tools from the Toolbar widget:

1. Click the **TOOLBAR** widget in the Layout frame and click the **EDIT WIDGET** button or double-click on the widget in the layout cell. The Toolbar Tag Editor dialog displays.

2. Highlight a tool in the left pane and click the **>** button to add it to the toolbar.

3. To remove a tool from the Toolbar, highlight it in the Tools to Include list and click the **<** button to remove it from that list.

4. Continue to add or remove tools as necessary. Reorder the tools as you wish using the **A** and **V** buttons on the right side of the dialog.

5. Change the toolbar layout from horizontal to vertical, if desired, or adjust the spacing of the tool icons on the toolbar.

6. Click **OK** when finished.

The following section on the Custom JSP Tag Library provides more information on the tag attributes that you can customize.
Deploying Applications Built with Prototype Builder

If you deploy your application into the same container as the MapXtreme Java Manager servlet, you may choose to enable servlet forwarding, an optimized way to render maps. This is accomplished by the IntraServletContainerRenderer, which allows MapXtremeServlet to write its raster data directly back to the client.

Custom JSP Tag Library

MapXtreme Java provides a library of custom JSP tags that you can insert into a .JSP file using a text editor or IDE. These tags display as widgets in the MapXtreme Java Manager’s Web Applications Builder, where you can select the element you need and add it to a layout frame that will be saved as a .JSP file. At run time, the .JSP communicates with a servlet that carries out the business logic of the application. If changes to the display of the application are necessary, the widgets can easily be rearranged, added, or deleted in the Web Applications Builder to create a new .JSP without affecting the content generation operations of the servlet.

These custom tags have been designed for use in an MVC (Model/View/Controller) JSP-servlet architecture. The resulting .JSP (View) contains forms, and the forms submit to a generic servlet (Controller). The controller servlet redirects to the appropriate Java bean (Model) that carries out the necessary business logic, such as creating themes, performing radius searches, etc. and will forward the request back to the .JSP file, which displays the updated map.

For a listing and description of the Custom JSP Tag Library that ships with MapXtreme Java, see Appendix A: Custom JSP Tag Library.

Using the JSP Tag Library

All of the MapXtreme custom JSP tags are contained in one main tag, called the mapapp tag and nested inside the <mapinfo:mapapp> </mapinfo:mapapp> tag block.

When building a custom application with the MapXtreme custom JSP tags, be aware of the following client-side browser requirements:

- JavaScript must be turned on (client-side Java is not required).
- Browser version should be relatively recent (e.g., Netscape 4.7, Internet Explorer 4.0 or higher).
On the server side, the web server/app server/web-app server for deploying a custom application using these tags must be J2EE compliant; that is, it must support the JSP 1.1 API and the Servlets 2.2 API.

For the necessary initialization parameters, run the Web Applications Builder wizard and look at the web.xml created by the wizard.

**JSP Tag Library Sample: MapViewer**

MapXtreme Java provides a sample map viewer application built from custom JSP tags. The application, mapviewer.war, can be found in examples/client/jsp/mapviewer after installation. This sample allows you to use all the features available in the JSP library as well as show how you can customize the behavior of the tags. Additionally, this example shows a more advanced layout that cannot be created using MapXtreme Java Manager.

![MapXtreme Java JSP Custom Tag Sample](image)

**Note:** This application does not have the geocoding and routing features enabled. You must remove the comments in the web.xml and map.jsp to enable them. You must also have MapMarker J Server and Routing J Server to run those features.
MapXtreme Java Edition offers MapXtreme JavaBeans that allow you to easily embed live vector maps in a Java applet or application.

In this chapter:

- Overview ................................................................. 93
- VisualMapJ Bean ....................................................... 93
- MapToolBeans ......................................................... 94
- MapToolBarBean ...................................................... 95
- LayerControlBean .................................................... 96
- AddThemeBean ......................................................... 98
- Creating an Application with MapXtreme JavaBeans .......... 101
- SimpleMapAppletSample ........................................... 103
- Including MapTools in Your Applet or Application .......... 104
- Configuring and Controlling the Standard MapTools ...... 106
- Creating Custom MapTools ........................................ 109
Overview

The MapXtreme JavaBeans utilize the latest Java technology and are based on Java’s Swing components, part of the Java 2 Standard Edition Platform. If you are distributing applets using these JavaBeans, end users will need a suitable plug-in if their browsers do not support J2SE.

MapXtreme JavaBeans are easy to customize. They can be used to build applets that run in a user’s browser or can be built into a stand-alone application. The MapXtreme Java Manager that ships with MapXtreme Java is an example of an application that uses many of the JavaBeans discussed in this chapter.

MapXtreme JavaBeans are easy to use in a visual development environment such as JDeveloper, JBuilder, Visual Age, etc. The components needed to create your mapping applet can be easily dropped onto a form. Once on a form, the properties can be set.

Developing with MapXtreme JavaBeans allows you to create a mapping applet quickly that provides more robust functionality than a regular HTML page with an embedded map can offer. They are best used in intranet environments with powerful servers and networks to handle the increased download time that it takes for an applet to load. For a summary of configuration options using applets, see Chapter 3: Planning Your Application.

Additionally, use the MapXtreme JavaBeans for rapid prototype development.

There are seven types of MapXtreme JavaBeans that can be used in your applet or application.

- VisualMapJ Bean
- MapToolBar Bean
- MapTool Beans
- LayerControl Bean
- AddTheme Wizard Bean
- LegendContainer Bean
- ViewEntireLayer Bean
- ZoomPanel Bean

Each MapXtreme JavaBean is explained in the following sections.

VisualMapJ Bean

The VisualMapJ Bean is the main mapping component that displays maps within its content pane. It is built on top of the existing MapJ class library. VisualMapJ allows the most common mapping operations to be done in a visual development environment, rather than programmatically. Operations include adding map definitions or geosets, controlling zoom, controlling center, controlling bounds, etc. Less common, more sophisticated actions can be accomplished using the underlying MapJ object from the API.
VisualMapJ displays a map image based on the state maintained by its internal MapJ reference. VisualMapJ maintains a list of registered Map Tools, and manages tool selection and deselection. It forwards mouse and keyboard activity to the currently selected Map Tool, and relies on the tool to manipulate its state.

VisualMapJ can be used in conjunction with a MapXtremeServlet, or stand-alone. The **StartupMapDefinition** property allows you to specify a map definition or geoset file to use to initialize VisualMapJ (MapJ). The Layers defined within this file will specify whether data access should be done directly or through an instance of MapXtremeServlet. When VisualMapJ is used within an applet you will want to set up data access through MapXtremeServlet to remain within the security constraints imposed by the applet environment.

The **MapRenderer** property is used to specify whether the map is rendered locally (within the process space of the applet or application embedding VisualMapJ), or remotely through an instance of MapXtremeServlet.

The **ShowToolTips** property specifies whether VisualMapJ will allow popup text to be displayed when a mouse hovers within its extents. It is up to individual MapTools to set the content of the popup text.

Finally, VisualMapJ has **Center** and **Zoom** properties to specify the map’s initial bounds.

**MapTool Beans**

MapTool Beans are a collection of tools that allow you to carry out basic map navigation and selection operations. Tool operations include pan, recenter, zoom, measure distance, get information and select features at a point or within confining boundaries. A list of tools is provided in the table below.

All of the MapTool Beans extend Swing’s Action Interface. This makes them easy to add to a toolbar or menu. If you add the tools to both, they will remain synchronized.

MapTools act on VisualMapJ in the following areas by:

- specifying a tool tip to display
- specifying the cursor to use
- providing custom rendering on top of the map image (e.g., marquee selection box, ruler rubberbanding).
- modifying any properties of MapJ (center, zoom, Layers, Themes)
- prompting VisualMapJ to repaint.

The table below describes the available MapTools.

<table>
<thead>
<tr>
<th>MapTool</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>PanMapTool</td>
<td>Use to reposition a map within its window by dragging the map.</td>
</tr>
</tbody>
</table>
The Selection MapTools allow users to select Features in multiple layers through individual mouse clicks, and through more complex mouse drag operations. These mouse interactions may create SelectionThemes on a per layer basis to denote the selection that was made with the tool in the VisualMapJ map. For more on SelectionThemes, see Chapter 13: Features and Searches.

### MapToolBar Bean

You can add your map tools to any Swing JToolBar. There is a special MapToolBar component included with the MapXtreme JavaBeans. By default, the MapToolBar includes the ZoomInMapTool2, ZoomOutMapTool, PanMapTool, RulerMapTool, and InfoMapTool. The toolbar also contains a button to activate the LayerControl Bean. These default items can be removed, and additional tools can be added to the toolbar.
A requirement of using the MapToolBar Bean is that VisualMapJ and the MapToolBar have the
same parent component and they must be added in a certain order to allow for proper registry. You
should first add a VisualMapJ object to the form and then the MapToolBar Bean. When you add
the components in this order, the MapToolBar Bean will register all of its MapTools with the
VisualMapJ component. If you add the MapToolBar Bean first, you will need to manually associate
the MapTools with VisualMapJ.

Once the toolbar has been added, users can make a tool the currently selected tool by clicking on
its corresponding button.

For more information on working with MapTools, see page 104.

LayerControl Bean

To give your users the ability to control the layers in the map, include the Layer Control Bean in the
ToolBar of your application or applet.

The Layer Control Bean provides a user interface that allows you to set layer visibility, selectability,
and autolabeling, as well as add, edit, remove and reorder layers. Additionally, you can set the
layer display characteristics and label properties, and manage themes.

If you include the default MapToolBar in your GUI, you can display the Layer Control by clicking on
the LAYER CONTROL button.

For a complete description of the elements that make up the Layer Control, see the Layer Control
on page 69.
AddLayer Wizard

An important element of the Layer Control is the ability to add layers to your map. Clicking the **ADD** button displays the Add Layer Wizard, which guides you through the process of adding layers from your data sources, whether they are locally stored tab files, data sources accessed via JDBC, named layers, or data binded layers.

**Note:** In an applet, some types of layers cannot be added, including .TAB file layers, because applet security prevents any applet from accessing the local file system. This restriction does not apply to applications.

The Add Layer Wizard is configurable to help you help the user add a layer with ease. The configuration information is stored in the `addlayerwizard.properties` file, located in the `/MapXtreme-4.7.0/lib/client`. In the properties file, you can define pre-set values for commonly used settings (e.g., Oracle Spatial host and port). The wizard remembers the last used values for each data provider. Specific data providers and named resources can be added or removed from the wizard through the properties file. Storing passwords in the file is optional (default is to not store passwords).

For more on the addlayerwizard.properties file, see Chapter 17: Customizing the AddLayer Wizard via the addlayerproperties.file. See Adding Layers: Add Layer Wizard on page 71 for a walk-through the Add Layer Wizard in MapXtreme Java Manager. For information on named resources, see Chapter 12: Accessing Remote Data.

EditLayer Wizard

Once a layer has been added to the map, the properties that define it can be viewed or changed via the EditLayer Wizard. From Layer Control, click the **EDIT** button to display the wizard. Proceed through the panels (identical to those in the AddLayer Wizard) to view and/or make any changes. When you click **FINISH**, the layer will be re-built with the new properties you specified.
AddTheme Bean

The AddTheme Bean is a guided tool to allow you to add a Feature theme or Label theme to your map.

Feature Themes

For Feature themes, you can create either a Ranged theme that groups features according to a range of values, or an Individual Value theme that shades features based on a value.

Feature themes can be based on any supported column and layer in the current map. Currently there is support for creating a theme based on numeric, string and date column data, and on point, line and region layers.

Part of the operation will be to create a default theme legend which is associated with the new theme. The legend can be customized to change the title, fonts, insets, descriptive text and colors.

Through the Wizard Bean the user can choose a name for the theme. For RangedThemes, the user can also choose the number of ranges (bins), distribution method (equal count, equal ranges, etc.), and the value that all break-points (bin upper-bound values) should be rounded to. Users can also specify the rendition (color) of the points, lines or regions that will be thematically shaded by setting the rendition for the first and last ranges. Appropriate renditions will then be computed for the ranges in between.

For IndividualValueThemes, the user can choose which values in a layer should be given a distinct shading to distinguish them from other values.

At run-time, the AddTheme Bean hooks up to an existing VisualMapJ instance. In order for this to work, VisualMapJ must be a child of the component that the AddTheme Bean was added to. For example, if the AddTheme Bean is added to a JPanel to which a VisualMapJ instance was already added, then the VisualMapJ instance will be found and hooked up to. If the automated hookup cannot occur, then you must use the setVisualMapJ(VisualMapJ) method of the AddTheme Bean to seed the Bean with a working instance of VisualMapJ.
The AddTheme Bean extends AbstractAction, so it can be added to a JMenu or JToolBar (menu and toolbar will stay synchronized). When the menu item or toolbar button is clicked, the AddTheme Wizard will display.

For information on creating Ranged and Individual Value Feature themes programmatically, see Chapter 15: Theme Mapping and Analysis.

Label Themes

MapXtreme Java supports four types of label themes: Ranged, Individual Value, Override and Selection. Use a label theme when you wish to have different properties for labels in the same layer.

For example, you are labeling your layer of world countries with population growth rate. You want the countries with the largest growth rate to stand out from the countries with smaller growth rates. Create a Ranged Label theme via the Add Theme Bean whereby you group the values for the layer into ranges and apply a color and/or font size spread to the labels.

Label themes are further discussed in Chapter 14: Labeling and Renditions.

LegendContainer Bean

As a way of managing how legends are laid out and displayed, MapXtreme Java includes a LegendContainer Bean. It can be dropped in along-side a VisualMapJ object and display any legends that are associated with the Layers in VisualMapJ. If the LegendContainer detects the addition or removal of a theme, it updates its display accordingly.

Legends are Swing JPanels. This allows the legends to be laid out within the LegendContainer and render themselves into a LegendContainer.
ViewEntireLayer Bean

The ViewEntireLayer Bean allows you to quickly re-center and zoom the map so that you are zoomed out far enough to see the entire layer. The ViewEntireLayer Bean can be added to the MapToolBar. You can click the VIEW ENTIRE LAYER button on the MapToolBar to display the View Entire Layer dialog. From there the user chooses which layer to view.

![ViewEntireLayer Bean](image1)

The SimpleMap sample applet provides an example of using the ViewEntireLayer Bean. Find it in /examples/client/Java/simplemap. For more on the SimpleMap applet see SimpleMap Applet Sample on page 103.

ZoomPanel Bean

The ZoomPanel Bean provides a text field that displays the current map zoom. It also allows the user to type in a new zoom width.

![ZoomPanel Bean](image2)

For another example of ZoomPanel bean, see the SimpleMap sample applet in the /examples folder.
Creating an Application with MapXtreme JavaBeans

This section describes the steps to create a stand-alone mapping application using the MapXtreme JavaBeans.

This example uses JBuilder Personal version 6.0 as its RAD environment. For other environments, use this discussion as a guide, but refer to your documentation for specific instructions.

If you want to draw your user interface visually, in a WYSIWYG-like manner, you must first incorporate MapXtreme's Java Beans onto one of JBuilder's component palettes, as described below. Along the way, we will also define a MapXtreme "library" (a set of jar files), which may be useful to you even if you do not want to create user interfaces in WYSIWYG manner.

Adding MapXtreme Java Beans to the JBuilder 6 IDE

1. On the File menu, click New to display the Object Gallery.
2. Select Application and click OK. JBuilder creates a new project with a standard sample application.
3. Click the Design tab to switch into GUI editing mode. An assortment of tabbed palettes appears at the upper right (Swing, Swing Containers, etc.)
5. Click Add to define a new page; give the page a name such as "MapXtreme."
6. Select the "MapXtreme" item in the "Pages" list at the left of the dialog.
7. Click the Add Components tab.
8. Click the Select Library button. The Select a Different Library dialog appears.
9. Click the **NEW** button. The New Library Wizard appears.

10. Type a name for the new library, such as "mapxtreme_lib".

11. Click the **ADD** button, then select MapXtreme's lib/client/mxjbeans.jar file. The Library Paths list should now include mxjbeans.jar.

12. Click the **ADD** button again to add a directory to the Library Paths list — the MapXtreme lib/client directory.

**Note:** Be sure to only select the directory. Do not expand it to expose its contents.

13. Click the **ADD** button again and add the following .jar files to the Library Paths list (most of which are in lib/common directory): mxj.jar; micsys.jar; miutil.jar; jdom.jar; xercesImpl.jar; mxjtabdp.jar; mistyles.jar; mxjloc.jar, and xml-apis.jar.

14. Click **OK** to close the New Library Wizard.

15. Make sure your new "mapxtreme_lib" library is selected, then click **OK** to close the Select a Different Library dialog.

16. In the Component Filtering box, click the **JAVA BEANS ONLY** option.

17. Click the **ADD FROM SELECTED LIBRARY** button. The Browse For Class dialog appears, displaying a class tree starting with "com".

18. Expand the com tree until you see the contents of the com.mapinfo.beans.vmapj package. Select the class VisualMapJ, and click **OK**. A "Results" dialog appears, confirming that you have added the VisualMapJ class to your palette.


20. Click **OK** to close the Palette Properties dialog.

21. In the tabbed palette area at the upper right part of the screen, click the "MapXtreme" tab. (You may have scroll to view the MapXtreme tab, depending on your screen size.) In the MapXtreme palette you will see an icon for each of the Java Beans that you added — in this example, one icon for VisualMapJ and another for MapToolBar.

You can add more of MapXtreme's Java Beans to the palette, but for simplicity's sake this example will use just the VisualMapJ bean and the MapToolBar bean.

Now that you have added MapXtreme Java Beans to JBuilder, you can use those beans to "draw" a Swing user interface based on MapXtreme Java beans.

1. In the frame at the lower left, click the gray rectangular icon labeled "contentPane (borderLayout)."

2. Click the VisualMapJ icon on the MapXtreme component palette.

3. Draw a rectangle on the gray rectangle that represents your application. This places a VisualMapJ component on your application (in the "center" spot of your BorderLayout).

4. While the VisualMapJ1 item is selected at the lower left, click to the right of the "startupMapDef" property in the property inspector panel at the right. Then click the "..." button to launch the Startup MapDef dialog.

5. Click the **CHANGE STARTUP MAP DEFINITION** button, then select a .mdf file, such as world.mdf (located under MapXtreme's examples/server/data directory).

6. Click the MapToolBar icon on the component palette.

7. Click directly on the "contentPane (borderLayout)" item in the frame at the lower left. This places a MapToolBar in the "north" spot of your BorderLayout.

8. On the Run menu, click **RUN PROJECT**. Your application is launched, allowing you to view a map and interact with the map (zooming in, etc.) using MapXtreme's toolbar buttons.
SimpleMap Applet Sample

Provided as one of the sample applications with MapXtreme Java is an example of an applet called SimpleMap that uses the MapXtreme JavaBeans. This applet requires that MapXtreme is running in a web server.

This section is an overview to the sample. For further information on configuring and running SimpleMap, see the readme.html in the /examples/client/Java/simplemap directory.

SimpleMap is an applet that provides basic map-viewing capabilities to:

- Display a map (using the MapXtreme JavaBean, VisualMapJ).
- Display a toolbar containing tools for map navigation, an Info tool, and a Layer Control tool (using the MapToolbar Bean).
- Allow the user to add thematic shading by clicking an Add Theme button (using the AddTheme Bean).
- Display a corresponding theme legend in a panel to the right of the map (using the LegendContainer Bean).
- Provide a text field that displays the current map zoom, and allows the user to type in a new zoom width (using the ZoomPanel Bean).

The SimpleMap sample consists of following files:

- Readme.html – Document describing the applet and steps to take to deploy it.
- SimpleMap.java – The main code module. The source code is very simple, because it uses MapXtreme Beans – specifically, the VisualMapJ Bean, Map Toolbar Bean and AddTheme Bean – to do most of the work.
- simplemap.html – A sample HTML file that shows how to run the applet using Sun's Java Plug-in. Instead of using the <APPLET> tag, this HTML page uses the <OBJECT> and <EMBED> tags that are used by Sun's Java Plug-In. If you do not already have the Java Plug-In installed, viewing this HTML page will prompt you to download the Plug-in.
SimpleMap Applet Setup

This is a summary of the procedure you will need to carry out to set up SimpleMap. Refer to the readme.html in the SimpleMap directory for complete details.

1. Configure and launch the servlet container used to run MapXtremeServlet. (Chapter 2: Getting Started provides steps.)

2. Copy the following files: simplemap*.class files, simplemap.html, addlayerwizard.properties, resources-config.xml, encoding-map.xml, a map definition file such as world.mdf and the MapXtreme jar files to a directory that is accessible through your web server.

3. If you have chosen to use a map other than world.mdf, modify the tags in simplemap.html to specify the name of the map file you wish to use.

4. Load the applet by opening the HTML file in your browser.

Keep in mind the following must be configured properly for SimpleMap to run:

- The MapXtremeServlet must be running (i.e., your servlet container must be running).
- The applet must be compiled correctly.
- The web browser must have Java Plug-in installed.
- The HTML file used to load the applet must specify appropriate param tags.

Map Definitions and SimpleMap

If you configure SimpleMap to load a Map Definition, make sure the map definition was constructed using the ACCESS DATA VIA REMOTE MAPXTREMESERVLET option in the Add Layer Wizard dialog. If an attempt is made to access local resources, a security exception will occur. For more on creating map definitions, see Chapter 5: Managing MapXtreme Java.

Including MapTools in Your Applet or Application

Most mapping applications allow the user to perform various tasks by interacting with the map with mouse actions (such as clicking and perhaps dragging). If you want to support this type of map interaction, you will want to add map tools to your application or applet.

If your development environment supports visual GUI design, you can create and manipulate GUI elements, such as toolbars and tools, visually. For example, a JDeveloper user can draw GUI elements onto a form when in “design mode.” If your development environment does not support visual GUI building, you will need to create map tools by hand, by typing in code, as shown below.

Creating a MapToolBar

You can add a set of map tools to your code very quickly and easily by instantiating a MapToolBar object, and adding it to your layout. The default MapToolBar object provides several common map tools, including Zoom In, Zoom Out, Pan, Info and Ruler, as well as a button that launches the Layer Control dialog.

You can instantiate a MapToolBar very simply, with no arguments:
MapToolBar mapToolBar1 = new MapToolBar();

Or, if you want to make major changes to the set of map tools included in the tool bar, you can use the other MapToolBar constructor, which takes a Vector of map tools.

Once you have created the MapToolBar, add it to the same container to which you added the VisualMapJ object.

this.getContentPane().add(visualMapJ1, BorderLayout.CENTER);
this.getContentPane().add(mapToolBar1, BorderLayout.NORTH);

### Associating Map Tools with the Map

The tools in the MapToolBar must be associated with the VisualMapJ object. If they are not properly associated, then the tools will not have any effect on the map.

The simplest way of associating the tools with the VisualMapJ object is to add the MapToolBar object to the same container (e.g. the same JPanel) where you placed VisualMapJ. When the MapToolBar and VisualMapJ objects are in the same container, the tools are automatically associated with the map.

If your map tools are not automatically associated with the map (for example, because your GUI layout required that you place the toolbar in a different JPanel), you will need to call your VisualMapJ object's add method, which registers a map tool with the map; see below.

### Adding Individual Tools

If you find that you need more map tools than are provided with the default MapToolBar object, you can add more tools. For example, the default MapToolBar object does not include selection tools, such as the RadiusSelectionMapTool, so you might want to add that tool to the toolbar. To add individual tools to the toolbar:

1. Create any tool objects that you plan to use.
   
   ```java
   RadiusSelectionMapTool radTool = new RadiusSelectionMapTool();
   ```

2. If necessary, call VisualMapJ's add method to associate the tool with the map. (As described above, this step is not necessary if your MapToolBar is on the same container as the VisualMapJ.)
   
   ```java
   myVisualMapJ.add(radTool); // may not be required...
   ```

3. Add your tools and/or your toolbar to your GUI. Typically, you add your tools to a MapToolBar, and add that MapToolBar to the same component (e.g. the same JPanel) where you added your VisualMapJ object.
   
   ```java
   mapToolBar1.add(radTool);
   ```

The order of statements is important. If you want your map tools to be automatically associated with the map, you should create your map tools, add them to your MapToolBar, and then add your MapToolBar to the same container as the VisualMapJ, in that order.

You can also add map tools to a menu (e.g. by calling a JMenu object's add method). For an example of how map tools appear when on a menu, see the MapXtreme Java Manager, which displays tools on its Map menu. If you have added tools to both a tool bar and a menu, the tool bar and menu are kept in sync; when you select a tool on the toolbar, the same tool appears selected on the menu.
Removing Individual Tools

You can remove individual tools from the MapToolBar by calling its `removeMapTool` method. For example, if you want to use the default MapToolBar object, except that you do not want to include a Ruler tool, you could remove the ruler tool as follows:

```java
for (Enumeration e = mapToolBar1.getMapTools(); e.hasMoreElements(); ) {
    MapTool mt = (MapTool) e.nextElement();
    if (mt instanceof RulerMapTool) {
        mapToolBar1.removeMapTool(mt);
        break;
    }
}
```

Alternately, you could build a List containing only the map tools you do want to use, then pass that List to the MapToolBar constructor.

Configuring and Controlling the Standard Map Tools

Once you have included a map tool in your application, you may want to adjust how the tool behaves. Some examples of configuration options include:

- You could disable a map tool.
- You could reconfigure the InfoMapTool to not display its secondary "Info Tool" window.
- You could control whether the RadiusSelectionMapTool saves a circle in an annotation layer.

As a general rule, you control the behavior of a map tool by setting its properties. Map tool properties that you can set are described below. If you are using a visual development environment that supports property sheets the tool properties can be set by using the appropriate property sheet editors. Otherwise you can adjust the properties programmatically by calling a property’s appropriate set method. For example, to configure the enabled property, you would call the `setEnabled` method.

General Settings

All map tools share some common properties:

- **enabled** – To enable or disable a tool, call its `setEnabled` method.
- **cursor** – To assign a custom cursor, call the `setCursor` method.
- **selected** – To select a tool, call its `setSelected` method. When your application runs, the tool will appear selected automatically. However, this will not stop the user from selecting a different tool.

```java
radTool.setSelected(true);
```

Zoom In / Zoom Out Settings

The three zoom tools (ZoomInMapTool, ZoomInMapTool2, and ZoomOutMapTool) can be customized with the following properties:
• **zoomFactor** – this property controls how far in or out the map is zoomed when the user clicks the map. The default setting is 2; in other words, if your map shows an area 500 miles wide, and you click to zoom out, the resulting map will show an area 1000 miles wide.

• **zoomMode** – controls whether the map is recentered at the point where the user clicked, or whether the map is only zoomed in or out while remaining centered at the same location.

### RulerMapTool Settings

The Ruler tool provides the following properties:

- **calculationMethod** – controls whether distances are measured using Cartesian or Spherical calculations.
- **distanceUnit** – specifies the unit, such as miles or kilometers, used for displaying distances.
- **distanceWindowVisible** – controls whether the Ruler tool opens a secondary window to display distance measurements. To suppress the secondary window, call `setDistanceWindowVisible(False)`.

The RulerMapTool also has a bound “Distance” property so that listeners can be notified as the distance changes.

### InfoMapTool Settings

The Info tool provides the following properties:

- **searchMode** – controls which layers the InfoMapTool searches. For example, to search all map layers, call:
  ```
  myInfoMapTool.setSearchMode(SearchMode.FULL);
  ```
- **infoWindowVisible** – controls whether a secondary "Info Tool" window is opened to display the text returned by the Info tool. In some cases, you might want to suppress the secondary window, and display the resulting text somewhere else in your GUI. To prevent the Info tool from displaying its secondary window, call:
  ```
  setInfoWindowVisible(false)
  ```

Once you have suppressed the Info Tool window, you will want to display the results in another place. Create an object that implements the InfoObtainedListener interface (meaning that the object provides an infoObtained method), then register your listener with the InfoMapTool. When the user selects the InfoMapTool and clicks the map, your infoObtained method will be called, and an InfoObtainedEvent object will be passed in. You can use the methods on the InfoObtainedEvent object to extract the information returned by the InfoMapTool. It is then up to you to display that information somewhere in your GUI.

### SelectionMapTool Settings

MapXtreme provides several selection tools: BoundarySelectionMapTool, PolygonSelectionMapTool, RadiusSelectionMapTool, RectangleSelectionMapTool, and ObjectSelectionMapTool. If you include one or more of these tools in your application, the user will be able to choose the selection tool, then click (or, depending on the tool, click and drag) on the map to select one or more features on the map.
The selection map tools have several properties that you can configure, as follows. (Not all properties apply to the ObjectSelectionMapTool.)

- **saveAnnotation** – a boolean value that either permits or prohibits the automatic creation of an annotation layer whose single Feature depicts the boundary of the selection area being evaluated. All selection tools except for ObjectSelectionMapTool allow you to save annotations. For example, the RadiusSelectionMapTool can save a circle annotation representing the search radius that the user selected.

- **searchType** – controls the search criteria that are used to determine whether a feature should be selected. For example, if a region is only partially inside a search radius, you might or might not want the region to be selected. To control this option, call the selection tool's setSearchType method, and specify one of these values:
  - `SearchType.mbr` – returns features whose minimum bounding rectangle intersects the search region. This is the least restrictive of the search types, and returns the maximum number of features.
  - `SearchType.partial` – returns features that intersect the search region, at least partially.
  - `SearchType.entire` – returns features that are completely contained within the search region (default); this is the most restrictive search type.
  Both partial and entire are absolute, valid searches that compare the actual geometry of the feature and the search region. These search types must be used when a true and accurate result is needed. An mbr search is a quick approximation that simplifies the geometry to allow much faster comparison. In certain datasources, such as Oracle Spatial, the mbr search (SDO_FILTER) is actually implemented by intersecting the minimum bounding rectangle of the feature with the minimum bounding rectangle of the region, which results in a larger number of "hits" than using the actual search region.

- **selectionMode** – this property controls how many layers are searched. To control which layers are searched, call setSelectionMode, and specify one of the following:
  - `SelectionMode.VISIBLE` – searches all selectable layers that are currently visible. This option skips layers that are currently invisible due to zoom layering.
  - `SelectionMode.FULL` – searches all selectable layers.
  - `SelectionMode.FIRST` – searches all visible and selectable layers; however, the search stops as soon as one layer has features within the search area.

- **selectionThemeType** – controls whether a SelectionTheme is created to highlight selected features, and also controls how the theme assigns colors to the selected features. To control the shading of the selected features, call setSelectionThemeType with one of these options:
  - `SelectionMapTool.SELECTIONTHEME_OVERRIDE` – the selected features will be shaded, and the shading will use the color that was explicitly specified by the themeOverrideColor property.
  - `SelectionMapTool.SELECTIONTHEME_NONE` – no theme will be created; the appearance of the selected features will not change after being selected.
  - `SelectionMapTool.SELECTIONTHEME_DEFAULT` – selected features will be shaded using a color that is the inverse of the original feature color.

- **themeOverrideColor** – Sets the override color (the color value used to display selected features when and if selectionThemeType is set to SELECTIONTHEME_OVERRIDE). For example, if you want all selected features to appear in green, call
setThemeOverrideColor(Color.green), and also call setSelectionThemeType (SelectionMapTool.SELECTIONTHEME_OVERRIDE).

- **coordinateType** – the Radius and Rectangle selection map tools allow you to specify whether the selection area (the circle or rectangle drawn by the user) should use screen coordinates or map coordinates. Set this property to one of the following values:

  ConfiningSelectionMapTool.COORD_MAP – radius and rectangle search tools will create features in "map coordinates" (i.e. a radius may not look circular on the screen, depending on the map projection, just as the earth itself does not appear round in some map projections, but the radius will accurately represent the total area within a specified distance of a center point).

  ConfiningSelectionMapTool.COORD_SCREEN – radius and rectangle search tools will create features in "screen coordinates" (e.g. a radius will look circular on the screen).

All SelectionMapTools fire SelectionToolEvents when a selection occurs, so further customization is possible if you create and register a SelectionToolListener.

### Creating Custom MapTools

If you need functionality that is not provided by any of the standard MapXtreme map tools, you can create your own custom tools. To define a custom tool, you must implement your own Java class which satisfies the MapTool interface. Once you have created this class, you can instantiate your custom tool, and add it to your application, the same way that you added standard tools (such as the Radius tool example shown above).

#### Custom Tool Example: SimpleRulerMapTool

An example of how to create a custom tool is provided in the /examples/client/beans directory, which contains the following modules:

- **SimpleRulerMapTool.java** – an example of a custom map tool. This class implements a customized ruler tool, for measuring distances on the map. Unlike the standard Ruler tool, this custom tool displays the current distance as a ToolTip when you stop moving the mouse.

- **SimpleRulerMapToolBeanInfo.java** – a BeanInfo class, which controls how the tool's properties are exposed (e.g. when you inspect properties using a visual development environment).

- **SimpleToolFrame.java** – a class that extends JFrame. This is the class that displays a map and a toolbar. This is also the place where we instantiate the SimpleRulerMapTool object.

- **SimpleToolApp.java** – a very simple class that provides the main method.

Before studying the source code that defines the SimpleRulerMapTool, you may want to familiarize yourself with how the tool behaves. You can see the custom tool in action by compiling all four java classes, then running the SimpleToolApp class.

**Note:** Before compiling, you may need to edit SimpleToolFrame.java to change the filename and/or path specified in the loadGeoset call, so that they identify the name and location of a file on your system.
When the application runs, the custom SimpleRulerMapTool appears at the end of the toolbar. The user can select the tool, click the map once to start measuring a distance, move the mouse, then pause; when the user pauses, the current distance is displayed as a ToolTip. The user can click again to start measuring from a different location. The user can cancel the current line segment altogether by pressing the Escape key.

**Basic MapTool Requirements**

Map tools are **Action** objects, so that they can be added to JMenu or JToolBar objects. Since map tools are actions, any custom map tool that you create must define an *actionPerformed* method. Whenever the user selects your custom tool, either from the toolbar or the menu, your custom tool class's *actionPerformed* method is called.

Also, any custom map tool class that you create must implement the **MapTool** interface (com.mapinfo.beans.tools.MapTool). This means that you must provide the following methods:

- **getCursor** – Returns the java.awt.Cursor object associated with this MapTool.
- **isSelected** – Returns True or False to indicate whether this MapTool is currently selected.
- **setCursor** – Sets the tool's Cursor object.
- **setSelected** – Sets whether the tool is currently selected.

**Note:** That selecting one tool automatically de-selects the previously selected map tool.

For examples of these methods, see SimpleRulerMapTool.java.

Once you have built a custom tool that addresses these basic requirements, you might consider adding more functionality to your custom tool. Different functionality that you can add is described in the following sections.

**Responding to Click and Drag Actions**

Since map tools allow the user to interact with the map, most map tools implement the **MapMouseListener** interface (com.mapinfo.beans.vmapj.MapMouseListener). Implementing this interface means providing the following methods, which correspond to various mouse actions:

- **mouseClicked** – called when the user completes a Click action (pressing and releasing the mouse button)
- **mouseDragged** – called when the user moves the mouse while holding down the mouse button
- **mouseEntered** – called when the mouse enters the VisualMapJ display area
- **mouseExited** – called when the mouse exists the VisualMapJ display area
- **mouseMoved** – called when the mouse moves, and the mouse button is not being held down
- **mousePressed** – called when the user presses down on the mouse button
- **mouseReleased** – called when the user releases the mouse button

The SimpleRulerMapTool class demonstrates how some of these mouse event methods are used. This tool allows the user to click to set the starting point for measuring a distance; therefore, the **mouseClicked** method contains code that stores starting coordinates.
Next, the user can move the mouse, causing the SimpleMapRulerTool to display a rubberband line. To produce this rubberband effect, the `mouseDragged` and `mouseMoved` methods both call the `handleStretch` method.

In the SimpleRulerMapTool example, the other mouse methods are included only to satisfy the MapMouseListener interface; therefore, those methods are empty. For example:

```java
public void mouseEntered(MapMouseEvent e) {}
```

Depending on how you want your custom tools to behave, you may want to add code to those empty methods.

**Responding to Keyboard Input**

In some cases, you may want your custom map tool to handle keyboard input. For example, the SimpleMapRulerTool allows the user to press the Escape key to cancel the display of the rubberband line.

If your custom map tool needs to respond to keystrokes, implement the `KeyListener` interface (`java.awt.event.KeyListener`), which entails implementing the following methods:

- `keyPressed` – called when the user presses down on a key
- `keyReleased` – called when the user releases a key
- `keyTyped` – called when the user finishes typing a key (pressing and releasing a key)

The SimpleMapRulerTool checks for the Escape key in the `keyReleased` method. If it was pressed, the `keyReleased` method calls the cleanup method, which resets the tool.

In the SimpleRulerMapTool example, the `keyPressed` and `keyTyped` methods are included only to satisfy the `KeyListener` interface, so those methods are empty.

**Displaying ToolTip Text**

Map tools can have two different types of ToolTip text:

- When the user positions the mouse over the tool on the toolbar, a ToolTip appears, displaying the name of the tool. To set the text that appears in this type of ToolTip, call `putValue`; for example:
  ```java
  putValue(SHORT_DESCRIPTION, "Ruler");
  ```
- When the user has selected your custom tool, and the cursor is over the map, you can display ToolTip text. This is accomplished by implementing the `ToolTipTextSetter` interface (com.mapinfo.beans.tools.ToolTipTextSetter) by providing a `getToolTipText` method.

For example, the SimpleMapRulerTool uses ToolTip text to display the distance between the current cursor location and the location where the user last clicked. Accordingly, SimpleMapRulerTool.java provides a `getToolTipText` method, which simply returns a string representing a distance measurement (for example, "123.45 mi").

**NOTE:** Even if your custom tool implements the `ToolTipTextSetter` interface, you will not see ToolTips unless you call the `VisualMapJ` object's `setShowToolTips` method. For an example, see SimpleToolFrame.java, which makes the following call:

```java
visualMapJ1.setShowToolTips(true);
```
Drawing on Top of the Map

Some map tools make temporary changes to the appearance of the map while the user is using the tool. To draw on the map in this manner, implement the MapPainter interface (com.mapinfo.beans.vmapj.MapPainter), and its `paintOnMap` method.

For example, the SimpleMapRulerTool draws a rubberband line on top of the map, and updates this line as the user moves the mouse within its paintOnMap method.
Writing Your Own Servlets

This chapter shows you how you can deploy your mapping application using servlets (server-side Java programs).

In this chapter:

- Servlet Overview .................................................. 114
- Working with Servlets ............................................. 114
- Thin Client/Thick Server ......................................... 114
- Thick Client/Thin Server ......................................... 116
- Medium Client/Medium Server ................................. 118
- MapXtremeServlet Data Flow ................................. 121
- Servlet Forwarding ................................................. 123
- Sample Servlet: HTMLEmbeddedMapServlet .............. 123
- Sample Servlet with Thematic Capabilities ................. 126
- Using the Servlet Utility Library (MapToolkit) ............ 126
Servlet Overview

Servlets are Java components that are used to extend the functionality of web servers. Servlets are to servers what applets are to browsers, except servlets have no graphical user interface.

The mapping engine in MapXtreme Java is deployed as a servlet. It conforms to the Java 2 Enterprise Edition architecture and must be run from within a J2EE-certified servlet container. By doing so, the servlet container manages non-mapping tasks such as load balancing, threading, and fault tolerance, and allows MapXtremeServlet to do what it does best — handle requests for maps.

Working with Servlets

Your servlet can contain a variety of mapping functionality to meet the needs of your application. For example, you can construct a servlet that offers basic map navigation such as pan, zoom in, zoom out, and a measure of distance between points. If you need a more sophisticated application, consider offering selection or thematic mapping capabilities.

Note: Before compiling a servlet source program, you may need to configure your Java IDE to include the J2EE servlet class files (e.g., servlets.jar) in your IDE’s classpath.

MapXtreme Java is suited to two- and three-tier web applications. The difference between them is where the MapJ component is located, on the server or client. The following sections describe the different configuration options. Examples of servlet architectures can be found in the MapInfo Knowledge Base at http://testdrive.mapinfo.com/kbase_by_product.

Thin Client/Thick Server

The following illustration shows the most common configuration — a browser as the client, your user defined servlet, which uses MapJ objects in the middle tier, the MapXtremeServlet in the middle tier, and the database in the final tier. Your application may utilize any combination of servlets, JavaServer Pages, or Enterprise JavaBeans.

When the client issues a request through the browser, your web server forwards the request to your user defined servlet, which in turn, may update the state of a MapJ object. The MapJ object is then used to communicate a mapping request to MapXtremeServlet. If the request is for an image, the MapXtremeServlet will return a raster image of the map to the user defined servlet. The user defined servlet can then embed this image within an HTML page and return the page to the end-user's browser.

In the middle tier, user defined servlet can also leverage MapXtreme's servlet library to aid in building the HTML page. For instance, there are library methods for creating an HTML-based layer control.
This three-tier architecture has the following characteristics:

- MapJ is deployed in the middle tier within the user defined servlet.
- MapXtremeServlet is deployed in middle tier.
- Java is not required on client. Client can send HTTP requests and can receive HTML pages in response.
- Produces minimal network traffic: Applets are not required, so there is no applet to download. Vector data is not sent to the client, only HTML pages with embedded raster images. Raster formats such as GIF typically produce map images that are 15–25K in size.

These characteristics make the three-tier architecture ideally suited to Internet deployments in which you, as the Web application developer, have little control over your client’s configuration. This deployment is a "lowest common denominator" approach and can be used to satisfy clients that do not have Java capable browsers and/or have low network bandwidth.

Additionally, MapXtreme Java provides a library of custom JSP tags that you can use to create the middle-tier servlet in a rapid application development (RAD) environment such as the Web Application Builder in the MapXtreme Java Manager.

**Thin Client Architecture**

This section illustrates the architecture of a thin client.

The following table highlights the main points of this architecture.
The following steps correspond to the numbers in the thin client architecture diagram:

1. Create MDF file on server machine that accesses local datasources using local data providers.
2. The MapJ object created for each connecting client will be initialized using the MDF file.
3. User defined servlet uses MapXtremeServlet to fabricate map image.
4. Map image returned to user defined servlet.
5. User defined servlet map incorporates images into HTML for return to client.

Important points to remember about thin client architecture include:

- No programming is needed for the client. It is just returned interpreted by the browser.
- Communication occurs through HTTP POST request. The only exception is the first request, which is an HTTP GET request.
- MapJ resides on the server within the user defined servlet.
- Remote rendering takes place on the server.
- The MDF file on the server uses LocalDataProviders.

### Thick Client/Thin Server

In a two-tier or thick client configuration, the MapJ object and business logic are deployed on the client-side, typically as an applet within a browser. A main advantage with this type of deployment is that it allows you to use MapXtreme Java Edition's JavaBeans. Applications can be created much more rapidly working with MapXtreme Java Edition’s JavaBeans in a visual RAD environment than working at the lower level MapJ API. MapXtreme Java Edition's JavaBeans provide visual map tools, toolbars, wizards, and map display components ready for inclusion in your application.

In a two-tier deployment, a client will first download an applet containing the JavaBeans from your Web server. The two-tier architecture has the following characteristics:

- MapJ is deployed on the client-side within an applet.
- MapXtremeServlet will be deployed in the middle tier for map rendering and data procurement.
- Java required on client: The client browser must have support for a Java 2 Platform VM v. 1.4.0 or higher (or have a suitable plug-in).
• Heavier network traffic: The applet containing the JavaBeans must be downloaded. Vector data may be sent to the client and the size of the vector data is much more variable than the size of raster files.

These characteristics make the two-tier architecture most suited to intranet deployments in which the deployment environment is more homogenous and controlled. When the applet is responsible for rendering and data access, a high network bandwidth is required. The client may also require more powerful machines.

**Thick Client Architecture**

This section illustrates the architecture of a thick client.

The following table highlights the main points of this architecture.

<table>
<thead>
<tr>
<th>Client Side</th>
<th>Communication</th>
<th>Server Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>• HTML browser</td>
<td>• HTTP protocol</td>
<td><strong>MapXtreme Java servlet</strong></td>
</tr>
<tr>
<td>• applet w/ VMapJ</td>
<td>for remote rendering</td>
<td>• remote rendering using server resources</td>
</tr>
<tr>
<td>• MapJ on client side</td>
<td>for remote data access</td>
<td>• procurement of data via MapXtremeDataProviders</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MDF file that is downloaded to client side</td>
</tr>
</tbody>
</table>

The following steps correspond to the numbers in the thick client architecture diagram.

1. Create the MDF on server machine that accesses local datasources using MapXtremeDataProviders.
2. Put the MDF file in a publicly accessible directory on the applet’s Web server.
3. Stream the MDF contents to applet from the Web server.
   
   **Note:** MDFs must use MapXtremeDataProviderRefs for every layer.

4. Send the MapJ settings to the server for datasource access and map image creation.

5. Fabricate map image using the datasources on server machine.

6. Stream the image back to client for display.

Important points to remember about thick client architecture include:

- The client requires extensive programming and contains mapping functionality.
- MapJ resides on the client side.
- The MDF file is downloaded to client applet, such that its layers are defined using only MapXtremeDataProviders.
- Remote rendering takes place on the server with the resultant image being streamed back to the client.

---

**Medium Client/Medium Server**

In a medium client/medium server configuration, the applet may be configured to go through a user defined servlet to obtain its data and map images. Specifically, this configuration is used when an applet serves as the client and when layer data source data is contained in an RDBMS. In such cases, a level of indirection as provided by the user defined servlet in the middle tier allows for RDBMS access.

**Medium Client Architecture**

This section illustrates the architecture of a medium client.
The following table highlights the main points of this architecture.

<table>
<thead>
<tr>
<th>Client Side</th>
<th>Communication</th>
<th>Server Side</th>
</tr>
</thead>
</table>
| HTML browser      | HTTP protocol
applet                  | **MapXtreme Java servlet**
                  |      | remote rendering using server resources |
|                   | handshake request to obtain sessionID
                   | **User Defined Servlet**
                  |      | concurrent client connections accepted |
|                   | serialized objects for passing data                                           | each connecting client is assigned its own MapJ per object |
|                   |                                                                                | MapJ saved to session |
|                   |                                                                                | local MDF file with LocalDataProviders |

The following steps correspond to the numbers in the medium client architecture diagram:

1. The MDF file created on the server machine uses **local** data providers for each layer.
2. The HTML browser downloads the applet HTML page to begin applet execution.
3. Serialized Java objects are used for sending information between the applet and user defined servlet.
4. An HTTPSession will be created with the user defined servlet for each connecting client which will be used to hold the client’s MapJ object.

Important points to remember about medium client architecture include:

- The client requires programming, but still serves to depict/modify a map that is programmatically held on the server.
- Communication takes place through serialized objects. This allows for a greater volume of meaningful data to be passed between client / server.
- MapJ resides on the server side.
- Remote rendering takes place on the server.
- The MDF file on the server uses LocalDataProviders.
Medium Client Communication

This section illustrates the communication process between a client applet and the user defined servlet within a medium client/medium servlet architecture.

Loading the Applet

1. When the index.html file is downloaded into the browser, it forces the browser (via special HTML tags) to download the applet's class files and begin its execution.

2. The class files that are being downloaded are strictly those that exist in the Java 2 Platform Standard Edition SDK, version 1.4.1_02. No MapInfo related class files are being used in the client side applet.

Handshake Request

3. The handshake request is the first request that is sent from a client to the user defined servlet. It tells the servlet to create a new "servlet session" for the concerned client, and to

Map Rendering Request

4. The servlet creates the session and sends a handshake response back to the client.

5. The applet then sends a map rendering request for a map image.

6. The MapXtreme server processes the request and sends the map image bytes to the client.

7. The Java Plug-in (applet) then renders the map image.

8. The applet appends the map image bytes to a MapXtremeImageRenderer Request/Response object.

The following steps correspond to the numbers in the medium client communication diagram. Steps 1-4 happen one time only, while steps 5-8 can happen as many times as needed.

1. When the index.html file is downloaded into the browser, it forces the browser (via special HTML tags) to download the applet's class files and begin its execution.

2. The class files that are being downloaded are strictly those that exist in the Java 2 Platform Standard Edition SDK, version 1.4.1_02. No MapInfo related class files are being used in the client side applet.

3. The handshake request is the first request that is sent from a client to the user defined servlet. It tells the servlet to create a new "servlet session" for the concerned client, and to
return a special URL that the client can use when making subsequent requests. This special URL contains an embedded session ID value that the server will use in order to obtain a client's specific servlet session. A servlet session is simply a chunk of memory (or perhaps a file on the file system if the session is permanent) that contains a client state.

4. The handshake response is sent back to a client from the user defined servlet. It contains a special URL that has an embedded session ID value, such that the client uses this new URL when making subsequent requests to the server.

5. After the handshake request, which is the first request sent from a client, every request sent thereafter will be a MapImage request. The MapImage request contains a description of the image that is to be created by and returned by the MediumServer. Specifically, it gives details such as the pixel dimensions of the image to create, as well as any map navigation operations (zoom-in / zoom-out / pan) that need to be applied beforehand.

6. After the MapJ object is modified as needed, the user defined servlet makes a request to the MapXtremeServlet for map rendering. This is called remote rendering.

7. After the MapXtremeServlet has fabricated the map image, it sends the bytes of the image back to the user defined servlet for return to the client.

8. The user defined servlet sends a MapImage response back to the client, which contains the bytes of the map image itself. The client applet then extracts the image bytes from the response message and draws the map image on the GUI.

MapXtremeServlet Data Flow

When developing a MapXtreme Java 4.x enabled client application, a user's map state is held in an object of class type com.mapinfo.mapj.MapJ. At runtime, both rendering operations and search operations can be performed against the MapJ object in order to obtain true mapping functionality within an application. Based on the settings within the MapJ object, as well as how the MapJ object is being used, data flow scenarios between a client application and the MapXtremeServlet can differ.

The general factors that contribute to differing data flow scenarios are as follows:

- What types of data providers are being used in the layers of the MapJ object?
- Is a map image being rendered?
- Is a search on a layer being performed?
- If a map image is being rendered, is local rendering or remote rendering being used?

Data Provider Operations

This section describes each data flow scenario as it relates to the type of data provider and operation being performed.

LocalDataProviderRef: Remote Rendering

These are the steps for remote rendering with a LocalDataProviderRef.

1. The client sends its MapJ object to the MapXtremeServlet.
2. The MapXtremeServlet reads the datasource data directly.
3. The MapXtremeServlet creates a map image using this data.
4. The MapXtremeServlet returns the bytes of the map image to the client.

**LocalDataProviderRef: Local Rendering**
These are the steps for local rendering with a LocalDataProviderRef.

1. Within the client, MapXtreme Java classes read the datasource data directly.
2. Within the client, MapXtreme Java classes create a map image using this data.
3. Within the client, the map image is then transcribed into a graphics context.

**LocalDataProviderRef: Layer Search**
These are the steps for a layer search a LocalDataProviderRef.

1. Within the client, MapXtreme Java classes read the datasource data directly.
2. Within the client, MapXtreme Java classes perform a layer search using this data.
3. Within the client, MapXtreme Java classes yield the results of the search.

**MapXtremeDataProviderRef: Remote Rendering**
These are the steps for remote rendering with a MapXtremeDataProviderRef.

1. The client sends its MapJ object to the MapXtremeServlet.
2. The MapXtremeServlet reads the datasource data indirectly via the MapXtremeServlet that was targeted by the URL of the MapXtremeDataProviderRef. These MapXtremeServlet instances can be different and reside on different machines. If the MapXtremeDataProviderRef URL simply references the current MapXtremeServlet instance (i.e. to which the MapJ object was sent), then it is this current MapXtremeServlet instance that accesses the datasource data directly.
3. The MapXtremeServlet fabricates a map image using this data.
4. The MapXtremeServlet returns the bytes of the map image to the client.

**MapXtremeDataProviderRef: Local Rendering**
These are the steps for local rendering with a MapXtremeDataProviderRef.

1. Within the client, MapXtreme Java classes read the datasource data indirectly via the MapXtremeServlet that was targeted by the URL of the MapXtremeDataProviderRef.
2. Within the client, MapXtreme Java classes create a map image using this data.
3. Within the client, the map image is then transcribed into a graphics context.

**MapXtremeDataProviderRef: Layer Search**
These are the steps for a layer search with a MapXtremeDataProviderRef.

1. Within the client, MapXtreme Java classes read the datasource data indirectly via the MapXtremeServlet that was targeted by the URL of the MapXtremeDataProviderRef.
2. Within the client, MapXtreme Java classes perform a layer search using this data.
3. Within the client, MapXtreme Java classes yield the results of the search.
Servlet Forwarding

To leverage servlet forwarding that is provided for in the J2EE 2.2 specification, MapXtreme Java offers the `IntraServletContainerRenderer`. This feature is an optional way to return raster images to the client. This renderer does not require socket connections between the renderer and `MapXtremeServlet`, as is necessary with the `MapXtremImageRenderer`.

The benefit of this deployment option is that the raster image can be sent directly to the client. `MapXtremeServlet` does not need to write the image to the middle-tier and then have the middle-tier re-write it back to the client. The limitation, however, is that your application must be deployed in the same container as `MapXtremeServlet`. Take this into consideration when you are planning your application.

The `IntraServletContainerRenderer` constructors take as input the information necessary for the middle-tier servlet to obtain a `RequestDispatcher` object to `MapXtremeServlet`. The `RequestDispatcher` object handles the servlet forwarding. The information that is needed includes:

- The alias used by `com.mapinfo.mapxtreme.MapXtremeServlet`, e.g., "mapxtreme47"
- `MapXtremeServlet`'s ServletContext object, or a URI to the servlet context, e.g., "/mapxtreme47"
- The `HttpServletRequest` and `HttpServletResponse` objects that `MapXtremeServlet` will use to satisfy the request
- The mime type for the raster image
- Whether the image should be multi-part, and an update interval for multi-part

For more on `IntraServletContainerRenderer` see Chapter 11: Rendering Considerations.

Sample Servlet: HTMLEmbeddedMapServlet

The HTMLEmbeddedMapServlet is a sample servlet that ships with MapXtreme Java. You can find it in `/examples/server/Java/servlet`. A pre-compiled version is also provided in the `mxjserversamples.jar`.

HTMLEmbeddedMapServlet provides an HTML page embedded with these basic mapping elements:

- Map frame that displays the map
- Radio buttons for Zoom In, Zoom Out, Pan
- Map Width box for user to type in new map width
- Apply button to apply the map width
- Layer Settings link that displays a table of layers that can be turned on or off
- Toggle for enlarging or reducing the map
- Scalebar

You can run HTMLEmbeddedMapServlet by opening a browser and typing the URL to `MapXtremeServlet`, such as:
http://stockholm:8080/samples47/htmlmap

Your URL may be different depending on how you set it up.

To customize this servlet change the map width or height by modifying the appropriate variables in HTMLEmbeddedMapServlet.java and recompiling. Whenever you modify and recompile the servlet, you will have to copy its class files into the appropriate directory. For example, with Tomcat you might need to copy the .class files to Tomcat's webapps/samples47/WEB-INF/classes directory (or, if you are running from a jar, you might copy the classes to WEB-INF/lib/mxjserversamples.jar).

If you compile the samples, be sure to delete, move, or rename mxjserversamples.jar, otherwise, your servlet container may use the HTMLEmbeddedMapServlet.class file from the jar file instead of using the class file that you compiled.

The HTMLEmbeddedMapServlet code sample allows you to change many settings without recompiling. The servlet loads many of its settings using standard servlet init parameters. For example, the name of the map to load (e.g., world.gst) can be overridden using an init parameter; thus, if you simply want to change the name of the map to load, you do not have to modify the servlet source code at all, you can simply edit the init parameter.
The following table describes the most important init parameters that are expected by the HTMLEmbeddedMapServlet sample. For a complete list of the init parameters used by the sample servlet, view the comments in HTMLEmbeddedMapServlet.java.

<table>
<thead>
<tr>
<th>Init Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>filetoload</td>
<td>The full path to the map file that will be displayed (either a .gst or .mdf file).</td>
<td>C:/mxt/maps/world.gst</td>
</tr>
<tr>
<td>mappath</td>
<td>The path to the directory where geoset (.gst) map files are installed on the server. Not applicable for map definitions.</td>
<td>C:/mxt/maps</td>
</tr>
<tr>
<td>mapxtremeurl</td>
<td>The MapXtremeServlet URL</td>
<td>/mapxtreme47/mapxtreme</td>
</tr>
</tbody>
</table>

If you use a servlet container that provides an Administrator tool (such as JRun or JavaWebServer), use that tool to set up your initialization parameters, as described above. Some servlet containers may require you to specify init parameters in an XML file; the next two sections describe both.

**Editing Init Parameters in Tomcat**

You can modify init parameters for Tomcat by using a text editor to edit a web.xml file. The following example shows a <servlet></servlet> block that defines three init parameters for HTMLEmbeddedMapServlet -- the mappath, filetoload, and mapxtremeurl init parameters.

```xml
<servlet>
  <servlet-name>htmlmap</servlet-name>
  <servlet-class>HTMLEmbeddedMapServlet</servlet-class>
  <init-param>
    <param-name>mappath</param-name>
    <param-value>C:\mxt\maps</param-value>
  </init-param>
  <init-param>
    <param-name>filetoload</param-name>
    <param-value>C:\mxt\maps\world.gst</param-value>
  </init-param>
  <init-param>
    <param-name>mapxtremeurl</param-name>
    <param-value>/mapxtreme47/mapxtreme</param-value>
  </init-param>
</servlet>
```
Sample Servlet with Thematic Capabilities

Provided in the examples directory under /examples/server/java/thematic is an enhanced version of the sample servlet HTMLEmbeddedMapServlet that incorporates thematic shading. See the file HTMLThemeServlet.java to learn more about adding thematic support to servlets.

Using the Servlet Utility Library (MapToolkit)

MapXtreme Java provides a library of helper methods for building a servlet. Given a MapJ object, the MapToolkit class helps you to construct the common elements of a map-enabled web page, such as a Layer Control form. The methods used in the sample application HTMLEmbeddedMapServlet are taken from this class.

Note: This library is only of interest to people writing servlet source code. Most developers prefer using JSP applications; if you plan to write your applications in JSP, you do not need to use the servlet utility library. However, if for some reason you cannot use the JSP library (e.g., if your clients cannot adhere to the JavaScript requirements imposed by the JSP library), then you may want to use the servlet utility library.

Use this class to simplify the development of servlets. The following elements are included in the library:
Map Tools - radio buttons that identify map navigation elements (zoom in, zoom out, pan)

Zoom box - text field that displays the current map zoom and allows user to type in new zoom.

Layer Control - an HTML page that displays the layers in the map and allows the user to check or clear settings for selectability, visibility, and autolabeling.

Scale Bar - a viewable element that shows the scale for the map.

Map Size Toggle - link to enlarge or reduce the size of the map.

Methods in the Servlet MapToolkit

The MapToolkit provides the following methods. For a complete description of the MapToolkit class, see the HTML Reference installed on your machine under /docs/devsupport.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getHTMLLayerListControl</td>
<td>Returns a string representing an HTML page that acts as a &quot;layer control dialog.&quot;</td>
</tr>
<tr>
<td>applyLayerSettings</td>
<td>Updates the map to reflect all options selected by the user in the Layer Control page.</td>
</tr>
<tr>
<td>getHTMLZoomControl</td>
<td>Returns a string representing a &quot;zoom control&quot; — a set of HTML tags that provide a text field for displaying the current map zoom width, and a Submit button to apply any new zoom width the user types in.</td>
</tr>
<tr>
<td>getHTMLMapToolsControl</td>
<td>Returns a string representing a set of radio buttons for Zoom In, Zoom Out, and Pan.</td>
</tr>
<tr>
<td>getHTMLScaleBar</td>
<td>Returns a string representing HTML syntax that defines a map scale bar. Specify a certain width, or specify zero and the scalebar will be sized to a round number approximately 1/4 the width of the map image.</td>
</tr>
<tr>
<td>getToolNumber</td>
<td>Returns an integer that represents which tool number corresponds to which tool name.</td>
</tr>
<tr>
<td>getStr</td>
<td>Returns a string resource from a resource bundle.</td>
</tr>
</tbody>
</table>
Part B: Introduction to the MapJ API

Part B of this Developer Guide describes the MapJ API, for programmatically creating and controlling the map information you want for your application.

Topics:

- **Chapter 9: MapJ API**
  This chapter describes the starting point of all MXTJ mapping applications: the MapJ object. This chapter defines MapJ, how to create a basic map, render an image, control map view and introduces you to features, renditions, themes.

- **Chapter 10: Mapping In Layers**
  This chapter describes the basic element of a map, the collection of layers that hold map features and information. Also described are "special" layers: Annotation and Raster.

- **Chapter 11: Rendering Considerations**
  This chapter contains more information about MapXtremeImageRenderer and LocalRenderer, as well as other rendering options.

- **Chapter 12: Accessing Remote Data**
  This chapter discusses database connection pooling for accessing remote data in an efficient manner.

- **Chapter 13: Features and Searches**
  This chapter defines features and FeatureSets and offers a number of search methods to use when mining for information.

- **Chapter 14: Labeling and Renditions**
  Here are the details about customizing and controlling the visibility and placement of labels. Also renditions, or styles, are covered for features and labels.

- **Chapter 15: Theme Mapping and Analysis**
  To get even more information out of your map, use the powerful thematic mapping feature that allows you to, at a glance, make visual comparisons and decisions about your data.
This chapter introduces the MapJ API and the topics to be covered in the following chapters.

In this chapter:

- MapJ Object ................................................................. 130
- Creating Your First Map .............................................. 130
- Controlling the Map View ............................................ 133
- Adding a Layer .......................................................... 135
- Saving Your Map Programmatically .............................. 135
- Named Maps .............................................................. 137
- Beyond the Basic Map .................................................. 139
MapJ Object

MapJ is a small light-weight component that provides an interface for the creation of maps by MapXtremeServlet or by itself. MapJ can make two types of requests: a request for data in vector form, called features, or a request for a map image file. MapJ’s job is to maintain the state of the map, including keeping track of the layers, coordinate system, distance units and map bounds.

MapJ objects can be configured to work with different types of Renderers and DataProviders. In the most typical configuration MapJ is a client of MapXtremeServlet. MapJ sends requests to a MapXtremeServlet instance and as part of the request provides the servlet with its current state. MapJ obtains map images and data from the servlet.

MapJ can also work stand-alone to directly obtain map data and produce map images. A strength of MapXtreme’s component-based design is that MapJ can be configured with other variations. For instance, MapJ can be configured to access map data via one or more instances of MapXtremeServlet, but still be responsible for displaying the map image. See Chapter 3: Planning Your Application for more information on configuration options.

The MapXtreme Java object model poster included with the product shows that almost every object, property, and method is derived from the MapJ object. Every method shown underneath the MapJ object will contribute to building the overall MapJ object. Primarily, the Data Provider, Layers, and Feature objects define each MapJ object. The other objects on the MapXtreme diagram contribute to the creation and rendering of the MapJ object; these are the Data Provider objects and the Renderer objects.

Creating Your First Map

The following outlines the general process of creating a map using the MapJ API. In this case, MapJ is communicating its request to MapXtremeServlet. These are the steps and page numbers on which they are described.

1. Initialize a MapJ Object on page 130.
2. Load Map Data on page 130.
4. Render the Map on page 132.

Initialize a MapJ Object

Before you can use a MapJ object or its methods in your application, you must first initialize MapJ. This is done simply with the following Java code:

```java
myMap = new MapJ();
```

Load Map Data

Once the MapJ object has been created, you must load map data.
You can load a geoset or Map Definition. See Chapter 5: Managing MapXtreme Java to learn more about opening and creating Map Definitions and geosets.

There is no default map definition setting in MapXtreme Java Edition. Therefore, as part of your initialization process, you must set a default map definition.

**Loading a Geoset**

The following is an example of loading a geoset file:

```java
myMap.loadGeoset(geosetName, dataDir, servletURL);
```

where

- `geosetName` is the full path to the geoset
- `dataDir` is the location on the server machine of the .tab files referenced in the geoset (may not be the same machine as MapXtremeServlet)
- `servletURL` is the path to the MapXtremeServlet when MapJ is using a remoteDataProviderRef (if using LocalDataProviderRef parameter is null).

For example:

```java
myMap.loadGeoset("c:\\mapxtreme\\maps\\world.gst", 
    "c:\\mapxtreme\\maps", "http://stockholm:8080/mapxtreme47/ 
    mapxtreme");
```

**Loading a Map Definition**

You can load Map Definitions that are stored in a file, as a record in a database, or as a named map in a named resources repository.

To load a map definition, you must first create a MapDefContainer which is an abstraction that represents where the map definition is stored.

To create a MapDefContainer for a Map Definition stored in a file:

```java
MapDefContainer mdc = new FileMapDefContainer(dir)
```

where `dir` = full path to the directory containing the Map Definition files

For example:

```java
MapDefContainer mdc = new FileMapDefContainer("c:\\mapxtreme\\maps")
```

To create a MapDefContainer for a Map Definition stored in a record in an RDBMS:

```java
MapDefContainer mdc = new JDBCMapDefContainer( 
    driver, url, user, password)
```

where `driver`, `url`, `user` and `password` are database connection parameters.

For example, the following loads from an Oracle RDBMS:

```java
MapDefContainer mdc = new OraSoMapDefContainer( 
    "oracle.jdbc.driver.OracleDriver", 
    "jdbc:oracle:thin:@ machinename:1521:dbSid", "username", 
    "password", "tableName", "Name", "Map_Definition");
```

To create a MapDefContainer for a Map Definition stored in the named resources repository:

```java
MapDefContainer mdc = new NamedMapDefContainer(providerUrl, path);
```
where providerUrl = the file:/// or http:// URL that points to the root of the named resources repository and path = the path to the named map, relative to the root of the named resources repository.

For example:

```java
MapDefContainer mdc = new NamedMapDefContainer(
    "http://torpedo:8080/mapxtreme47/namedresource", "mymaps");
```

will reference the named maps in "mymaps" via the NamedResourceServlet at the above URL.

To load the named map definition:

```java
myMap.loadMapDefinition(mapDefContainer, name)
```

where `mapDefContainer` = the above defined container class and `name` = the map to load from the container (the name used in the saveMapDefinition command).

For example:

```java
myMap.loadMapDefinition(mdc, "Asia");
```

### Set Map Device Bounds

Set the size of the rendered map image using `MapJ.setDeviceBounds()`. This is set before the map is rendered. The Device Bounds set the dimensions in pixels of the image that will be returned from the Renderer. For example, you may want to return a map that is 800x600. The default image size is 640x480.

To set the Device Bounds, use the setDeviceBounds method of the MapJ object. This method does not need to be called if you accept the default bounds.

```java
myMap.setDeviceBounds(new DoubleRect(0, 0, 800, 600));
```

### Render the Map

To render the map you must instantiate a renderer object. The following example uses a MapXtremeImageRenderer and renders the image as a GIF file.

Specify the URL to the MapXtreme servlet which remotely connects to the map engine.

```java
String mapxtremeServletUrl =
    "http://stockholm:8080/mapxtreme47/mapxtreme";
```

Create an ImageRequestComposer:

```java
ImageRequestComposer imageRC = ImageRequestComposer.create(
    myMap, 256, Color.blue, "image/gif");
```

Create a MapXtremeImageRenderer:

```java
MapXtremeImageRenderer renderer =
    new MapXtremeImageRenderer(mapxtremeServletURL);
```

Render the map:

```java
renderer.render(imageRC);
```

Export the rendered map to the file:

```java
renderer.toFile("comp.gif");
```
MapXtremeServlet returns an image of all of the specified layers. When a the render() method of a Renderer method is called, a request is made to MapXtremeServlet, which then produces an image on the server. The image is returned to the user only when a toFile, toStream, or toImage method is invoked.

Alternatively, if you have configured MapJ to work as stand-alone to directly obtain map data and produce images, instead of MapXtremeImageRenderer, you would use LocalRenderer to render the image locally.

Interaction can occur between the MapJ client and the MapXtremeServlet without using the Renderer. However, the Renderer is the only way that a map image will be returned to the user. You could create and initialize a MapJ object and execute several methods that manipulate the object or query a map, but in order to see the current map, you must use the render method. This is useful if you would rather create the map in several steps, and then display it.

MapXtremeImageRenderer and LocalRenderer are further discussed in Chapter 11: Rendering Considerations.

Controlling the Map View

Once your map is displayed, you will likely want to change its view to see map detail closer up, or to gain a wider view. MapJ has several methods for controlling the map view: setZoom(), setCenter(), and setZoomAndCenter().

Setting the Zoom Level

The zoom level is the distance across the map. You may change the zoom level to any distance. The units used will be the current distance units. The zoom level is first set when the geoset or map definition file is loaded. To change the zoom level of the map, use the setZoom method. The following example sets the zoom level:

```java
// Assuming that the current distance units are kilometers, this command will set the map zoom to 500 kilometers.
myMap.setZoom(500);
```

Recentering the Map

Part of controlling the map view is setting the center of the map. You may want to center on a found location or a particular coordinate. The setCenter method accomplishes this. You must pass a DoublePoint to the setCenter method. A DoublePoint is defined by a pair of XY coordinates.

The point location, if it is the result of a user clicking on the map at a certain location, is typically returned in pixels. MapJ requires the location to be in numeric coordinates, so a conversion method, transformScreenToNumeric, is necessary. MapXtreme Java uses the numeric coordsys as the base coordinate system for internal computation.
The following example creates a screen point, converts it to a "real world" point and sets the center of the map. DoublePoint is a point defined by coordinates for x and y.

    // Create the screen point
    screenpoint = new DoublePoint(event.getX(), event.getY());
    // Create the real world point
    worldpoint = myMap.transformScreenToNumeric(screenpoint);
    // Set the center of the map
    myMap.setCenter(worldpoint);

**Setting Zoom Level and Centering the Map**

The `setZoomAndCenter` method sets the current zoom level and center of the map. The zoom level is set in distance units. The center is set using the numeric coordinate system. The following example sets the zoom level and centers the map.

    // Create the screen point
    screenpoint = new DoublePoint(event.getX(), event.getY());
    // Create the real world point
    worldpoint = myMap.transformScreenToNumeric(screenpoint);
    // Set the zoom to be twice the current zoom
    // and center on the point where the user clicked
    myMap.setZoomAndCenter(currentZoom * 2, worldpoint);

**Setting the Map Bounds**

Use `setBounds` to set the bounding rectangle for the map. The method takes a `DoubleRect`, which is defined by coordinates that represent either the two opposing corners, or its center point, width, and height. Both ways are illustrated below.

This example uses opposing corners to set the bounds to a zoomed-in map area:

    DoubleRect bounds = new DoubleRect(
        -1.969272, 50.560410, 1.443363, 52.315529);

This example uses the center point, width, and height to set the bounds for the world:

    DoubleRect bounds = new DoubleRect(new DoublePoint(0,0), 360, 180);
    myMap.setBounds(bounds);

**Setting the Coordinate System**

Coordinate system data is stored in a projection file called micsys.txt that can be found in micsys.jar located in MapXtreme-4.7.0/lib/common. The micsys.txt file lists hundreds of supported coordinate systems and the parameters that define them.

Coordinate systems are set through the MapJ method `setDisplayCoordSys`.

    String csProj = new String(""\"Azimuthal Equidistant (North Pole)\", 5, 62, 7, 0, 90, 90");
    CoordSys ts = CoordSys.createFromPRJ(csProj);
    myMapsetDisplayCoordSys(ts);

Additionally, you can set the coordinate system using `createFromMapBasic` to read MapBasic strings and through some pre-defined constants.

For more information see the CoordSys class in the HTML API Reference.
Setting the Map Distance Units

Units are set through the MapJ method `setDistanceUnits`

```java
LinearUnit distUnit = LinearUnit.kilometer;
myMap.setDistanceUnits(distUnit);
```

Adding a Layer

One of the more used methods of the MapJ API is the `addLayer` method of the Layers object which allows you to bring additional data into your map. While the addLayer method is simple enough to call, there are a number of steps that must precede it to describe what data to add, where to get it from and how to get it. Data can come from local or remote data sources, in the form of files or records from a database. In order to manage this operation, MapXtreme Java uses Data Providers, which are explained in Chapter 10: Mapping In Layers.

To access the Layers object, use MapJ's `getLayers` method.

Saving Your Map Programmatically

You can save your map programmatically for future use. You have the option of saving it either as a Map Definition in a file or database, or as a named map, where the collection of map layers is given a unique name.

Saving a Map Definition to a File

To save a Map Definition to a file, create a FileMapDefContainer (as you would for loading a Map Definition) and call `saveMapDefinition`, passing in the container and a name for the Map Definition.

Storing Map Definitions in a Database Table

Map Definitions are stored in remote databases as long text strings (in XML format) in any Character Large Object (CLOB) type field of any table in your remote database. MapXtreme Java can store Map Definitions in any column large enough to hold them. It is possible to store a very limited Map Definition in a VARCHAR column. The column must be able to hold a string of the size of the Map Definition. CLOB columns can hold up to 2 or 4Gb of data, depending on the database. All of the major databases have a CLOB type, or something very similar.

MapXtreme Java requires that any table that will hold Map Definitions have at least the following:

1. A CHAR, VARCHAR, or similar type column in which the Name of the Map Definition can be stored.
2. A CLOB column in which the actual XML text can be stored.

It may, of course, have any number of other columns in addition to those listed above.
The example that follows is a CREATE TABLE statement that creates a table called MAPDEFINITIONS (with the current user as the owner of the table):

```
CREATE TABLE MAPDEFINITIONS (NAME VARCHAR(40), MAPDEF CLOB)
```

Executing this statement creates a MAPDEFINITIONS table in your remote database. This statement specifies that the NAME column stores strings up to 40 characters, and the MAPDEF column stores CLOB (large text) values.

The MapXtreme Java installer installs sample SQL scripts that can be used to create a MAPDEFINITIONS table in your remote database, with an owner of MAPINFO (located in /MapXtreme-4.7.0/examples/server/scripts). These scripts execute a CREATE TABLE statement very similar to the one described above, but create a MAPDEFINITIONS table with the following recommended structure.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>VARCHAR(40)</td>
</tr>
<tr>
<td>MAPDEF</td>
<td>CLOB</td>
</tr>
</tbody>
</table>

**MapDefContainer Interface**

Storing Map Definitions programmatically calls on the MapDefContainer interface, which provides three container implementations, depending on how the information will be stored:

- FileMapDefContainer stores Map Definitions to a file.
- JDBCMapDefContainer saves Map Definitions to a database.
- NamedMapDefContainer stores Map Definitions to a named resources repository.

Additionally, there are two database-specific implementations for JDBCMapDefContainer: OraSoMapDefContainer for Oracle and IUSMapDefContainer for Informix. All other JDBC databases (for example, SQL Server) should use JDBCMapDefContainer. Named maps are further discussed on page 137.

The MapDefChooser dialog in the MapXtreme Java Manager, as shown in the illustration below, prompts you to specify how to build either the FileMapDefContainer, JDBCMapDefContainer, or NamedMapDefContainer.
To write an XML Map Definition to a MapDefContainer, use `saveMapDefinition()` in the MapJ class. Specify the name of the container the Map Definition should be saved to and the name of the map you are saving.

The following code illustrates how to save a Map Definition to the MAPINFO.MAPINFO_MAPDEFS table.

```java
String driverName = "oracle.jdbc.driver.OracleDriver";
String dbUrl = "jdbc:oracle:thin:@machinename:1521:dbSID";
String usrName = "username";
String pwd = "password";
OraSoMapDefContainer mdc = new OraSoMapDefContainer(
    driverName, dbUrl, usrName, pwd);
map = new MapJ();
map.loadGeoset("c:\maps\asia.gst", "c:\maps", null);
map.saveMapDefinition(mdc, "Asia");
```

### Named Maps

For convenience in retrieving a saved map, you may wish to call the map by a unique name. This section describes how to save a map as a named map and retrieve it programmatically. To manage named maps through the MapXtreme Java Manager Named Resources panel, see [Chapter 5: Managing MapXtreme Java](#).
Storing Named Maps with NamedMapDefContainer

To store named maps programmatically, you must use the version of MapJ.saveMapDefinition() that accepts a MapDefContainer and a map name. For a named map, you will need to create a NamedMapDefContainer. To create a NamedMapDefContainer, you need a JNDI Context (javax.naming.Context). This can be either an InitialContext (javax.naming.InitialContext), or a sub-context of the InitialContext (obtained by doing a lookup() via the InitialContext). For more on Contexts and InitialContexts, see the Javadocs for the JNDI API.

To create an initial Context, you need to know the provider URL, which is (most likely) the URL of NamedResourceServlet. Then invoke the createInitialContext(providerURL) factory method of the NamedResourceContextFactory class, as shown below:

```java
Context initCtx = NamedResourceContextFactory.createInitialContext
               ("http://torpedo:8080/mapxtreme47/namedresource");
```

You can now use the InitialContext to create a NamedMapDefContainer:

```java
NamedMapDefContainer container = new NamedMapDefContainer(initCtx);
```

Now decide where, relative to the root of your named resources repository, you want to store your named map. (Remember that NamedResourceServlet knows where the root of your named resources repository is.

For example, if you have already created a directory beneath the root of your repository called "mymaps", and you want to save the current state of MapJ in that sub-directory as "my map". You would do so like this:

```java
// mapJ was previously initialized
mapJ.saveMapDefinition(container, "mymaps/my map");
```

In this example, "mymaps/my map" represents a compound name. When specifying compound names, each component of the name must be separated with a forward slash "/".

**Note:** It is recommended that you always store named resources (maps) in a sub-directory of the root of the repository.

Requesting an Image of a Named Map

Once a named map is stored in the named resources repository, you can request an image of it from MapXtremeServlet. To do so, use the ImageRequestComposer constructor that accepts a map name, rather than an entire MapJ:

```java
// bounds represents the desired bounds for the resulting map image
ImageRequestComposer imgReq = ImageRequestComposer.create(
    providerUrl, "mymaps/my map", mapCoordSys, bounds, 640, 480, 256,
    Color.white, "image/gif");
```

This creates a request for a 640x480 256-color GIF with a white background.

To send this image request off to MapXtremeServlet:

```java
// create the remote renderer
MapXtremeImageRenderer renderer = new MapXtremeImageRenderer(
    "http://torpedo:8080/mapxtreme47/mapxteime");
// render the map
renderer.render(imgReq);
// export to a file
```
renderer.toFile("c:/temp/my map.gif");

**Beyond the Basic Map**

MapXtreme Java provides the API that allows you to control every aspect of the map. This section introduces you to some of them.

**Features**

A record is a set of related columns of information. For example, a database of customers will have a record for each customer that includes columns for name, address, interest, etc. A feature is simply a record that combines tabular data and geometric information.

For example, the file World.tab from the MapXtreme sample data is a MapInfo format database. For each country in the database, there is a record. Each record includes several columns of tabular data as well as a reference to the geometric information that describes the shape and location of each country. This allows it to be displayed on the map. The tabular data is referred to as attribute data, and the geometric data is referred to as the geometry. These two types of data make a feature.

Features are not directly connected to the MapJ object, but are important for several reasons. As explained earlier, MapJ is the base for all of the map functions in your program. It sits at the top of the object diagram.

A Feature object sits at the lowest level of your program and deals with specific information. It is one of the most specific objects in the object model and relates to record level information. It is at the Feature level that the graphic objects can be given different display characteristics. The characteristics that specify the appearance of a graphic object are set by the Rendition object.

Information on Feature objects can be found in the Chapter 13: Features and Searches.

**Renditions**

Every feature has a rendition associated with it that describes how it is to look on a map. Rendition properties can be grouped into three general categories: fill, stroke, and symbol. The fill properties control how a region is filled. The stroke properties control how a line (either a line geometry or the edge of a region) will be drawn. The symbol properties control how symbols are drawn for either point geometries, line markers, or symbol fills.

The portion of the MapJ API that controls renditions is the Rendition class. The combinations of renditions that you can achieve are practically unlimited due to the variety of methods available. Renditions can be assigned to Features, Layers, Labels, and Themes and can be used to override symbology. For more on renditions, see Chapter 14: Labeling and Renditions.
Themes

Whether layers are added by the Layers addLayer method, MapJ loadGeoset method or loadMapDefinition method, each layer will have its own characteristics such as a line’s color, its width, etc. These characteristics are based on the information in the data source. Usually these settings are consistent for an entire layer. For example, if you load World.tab from the sample data, each country displays with a solid, green fill pattern. Every feature in the layer appears the same way.

Themes allow you to programmatically change the appearance of some or all of the features in a layer based on some criteria. For example, if you wanted to change the color of all of the world countries that have a population over fifty million, you could accomplish that with a theme. There are four Theme classes available:

- OverrideTheme – for changing the rendition of an entire layer
- RangedTheme – for grouping data into ranges and shading based on range value
- IndividualValueTheme – for shading groups of features which share a specific attribute value
- SelectionTheme – applies a rendition to a user-defined list of selected features

The features in each theme all have a rendition associated with them. The Rendition object encapsulates the style properties for both graphic and text displays.

Themes can also be created for labels to differentiate among labels in the same layer. For example, create a ranged label theme for showing sized labels associated with growth rate; larger labels can mean higher growth rate. For more on theme mapping, see Chapter 15: Theme Mapping and Analysis.
This chapter presents the relationship between tables and maps and how they are layered to create the level of detail you want.

**In this chapter:**

- Maps as Layers ........................................... 142
- The Layers Collection: Building Blocks of Your Map ........................................... 142
- Defining a Layer with a Data Provider ........................................... 143
- Creating Your Own Data Provider ........................................... 146
- Caveats for Defining JDBC Layers ........................................... 146
- Adding a Layer to a MapJ Layers Collection ........................................... 147
- Data Binding ........................................... 150
- Annotation Layers ........................................... 152
- Analysis Layers ........................................... 153
- Named Layers ........................................... 153
- Methods of the Layers Collection ........................................... 155
- Zoom Layering ........................................... 159
- Generating Labels For a Layer ........................................... 160
- Raster Images ........................................... 161
- Considerations for Importing Raster Images ........................................... 162
- Image IO Data Provider ........................................... 163
- Raster Style Tags ........................................... 164
- Grid Images in MapXtreme Java ........................................... 166
Maps as Layers

You have already been introduced to the concept of computer maps as a collection of layers. Each table that contains graphic objects can be displayed as a layer in a map image.

Think of these layers as transparencies where each layer contains a different part of the map. The layers are stacked one on top of the other and allow you to see all aspects of the map at once. For example, one layer may contain country boundaries, a second layer may have symbols that represent capitals, and a third might consist of highways. Laying these transparencies on top of each other builds a map.

Layers are made up of geographic features and associated data. For example, a layer of country boundaries has regions that define each country’s boundary and it might have attributes that represent the population of each country, literacy rate, or average household income. By creating a map of layers that have information attached, you can go beyond the pretty map and query the layer for information that you can analyze and display. That kind of map is much more effective in showing relationships among map data.

This chapter will focus on how to handle layers programmatically, such as defining a layer and adding a new layer to a map. This chapter also covers special types of layers including annotation layers and raster images. To learn how to load layers using MapXtreme Java Manager’s Map Definition GUI, see Chapter 5: Managing MapXtreme Java.

The Layers Collection: Building Blocks of Your Map

The Layers collection is accessible from MapJ and contains Layer objects. These Layer objects, which are built from tables, make up your map. Each layer contains different map features, such as regions, points, or lines. The Layers collection has methods used to perform operations such as adding and removing Layer objects from the collection. Layer objects have search methods that allow you to locate specific information on a layer.

The Layer object represents data made up of map features that usually have a predominant feature type, such as regions, lines, or symbols. Typically, a Layer object corresponds to the geographic objects from one table. Each of the Layer objects in a Layer collection behave independently of each other. Their styles may be changed, zoom layering altered, etc., on an individual basis, without affecting any of the other layers.

The Layer object makes use of several related classes such as ThemeList, LabelProperties, FeatureSets, ColumnStatistics, and TableInfo. Besides methods for accessing these objects, the Layer object also has search methods that allow you to locate specific information on a layer. Through the Layer object, you can take advantage of most of the MapXtreme functionality.
How to Build a Layers Collection

To build your map, you begin by adding layers to a Layers collection. In the previous chapter we walked through the code that loaded the map data using `MapJ.loadMapDefinition` and `MapJ.loadGeoset`. The Layers collection is defined by the layers of this map definition. When a map definition is loaded, the display coordinate system is updated and any previous layers are removed.

Once you have created a Layers collection, you can add more layers. When you use the `Layers.addMapDefinition` method, the layers are added to the current map and the existing coordinate system settings are maintained. Layers can also be added individually with the `Layers.addLayer` method. The addLayer method puts the layer at the end of the collection. Use `insertLayer` to control the position.

How Layers Are Drawn

Map layers in a Layers collection display in increasing order of a zero-based index: Layer(0) is the top layer, Layer(1) is the layer underneath Layer(0), etc., with the bottom layer drawn first and the top layer drawn last. It is important to order your layers correctly.

For example, you have a layer of customer points and a layer of census tracts. If the layers are incorrectly ordered in the Layers collection, MapXtreme will draw the customer points first and then display the census tract layer second. Your customer points would be obscured by the census tract layer. See Methods of the Layers Collection on page 155 for a code example that illustrates positioning layers.

Defining a Layer with a Data Provider

To add a layer to a Layers collection, you must first define it. The key to defining a layer is the Data Provider.

Each layer has an internal object, a Data Provider, that is responsible for data access. Data Providers are not created directly by users, but their description defines a layer. The following three interfaces are used to describe a Data Provider (and thus a layer):

- `TableDescHelper` – describes the data
- `DataProviderHelper` – defines the source of the data
- `DataProviderRef` – describes how to get the data

MapXtreme Java’s DataProviders allow you to create map layers for the following data sources:

- MapInfo tab format (.tab)
- Oracle with Spatial Option
- Informix Universal Server SpatialWare DataBlade
- SQL Server for SpatialWare
- JDBC compatible tables containing longitude and latitude columns
- GeoTIFF and MIGrid Raster
• ESRI Shapefiles
• Data Binding $^1$
• Annotation $^2$

**TableDescHelpers**

The TableDescHelper is an interface that helps to describe the data that you are accessing. There is a TableDescHelper for each different type of data source that MapXtreme can access. Each one has constructor parameters specific to the data source.

For example, the TABTableDescHelper that is used to describe a MapInfo table such as world.tab needs only the table name to describe it. The OraSoTableDescHelper, used to describe Oracle data, is defined by either a table name or SQL query. Code examples are presented later in this chapter. Further details on each TableDescHelper can be found in the online Javadocs Reference.

**DataProviderHelpers**

DataProviderHelpers define the data source. In the case of MapInfo TAB files, the directory containing a .tab file is the data source for that file. Therefore, theDataProviderHelper for tab files, TABDataProviderHelper, takes a directory as its only parameter.

Consider the example where a map consists of several tables, all of which are stored within an instance of Oracle database. This database is the data source for each of the tables. The OraSoDataProviderHelper takes parameters that describe the data source, such as server host name, server port number, user name, and password. The sameDataProviderHelper can be used for different tables from the same data source. Further details on eachDataProviderHelper can be found in the online Javadocs Reference.

---

1. Allows you to retrieve data from .tab and JDBC data sources to display as a single layer.
2. Not a typical Data Provider as the information is not stored in the database, but held in memory. See Annotation Layers on page 152 for more information.
**DataProviderRef**

Data Providers have a built-in remoting capability. The DataProviderRef describes who is responsible for accessing the data source. There are two possibilities: 1) the application (process) that contains MapJ and this Layer can directly access the data source, or 2) the application can defer to an instance of MapXtremeServlet to access the data source, and then have MapXtremeServlet transport the data back to the application. The direct way is carried out via the LocalDataProviderRef, while the indirect method uses the MapXtremeDataProviderRef.

Using a **LocalDataProviderRef** signifies that the application will directly communicate with the data source. This means that any resources needed to access the data source must be available to the application. For instance, a JDBC driver must be on the classpath of the application in order for it to directly access data within an RDBMS.

![LocalDataProviderRef Diagram](image)

A **MapXtremeDataProviderRef** is used when the client defers to MapXtremeServlet for the data. This is the typical case when using three-tier deployments. In this case only a "stub" data provider will exist in the client and the real data provider will be created by MapXtremeServlet, which then accesses data from the data source. With this deployment the resources needed to access the data source are only put in the middle tier. This avoids the need to have JDBC drivers deployed in the client, but still allows MapXtremeServlet to access data within an RDBMS.

![MapXtremeDataProviderRef Diagram](image)

**When Is Data Accessed?**

TableDescHelpers, DataProviderRefs, and DataProviderHelpers are used to define aDataProvider. When a Layer is first defined and created, no data access occurs. DataProviders only access their underlying data source in response to a specific request such as a call to Layer’s getTableInfo, a Layer’s search method, or a render request.
Creating Your Own Data Provider

MapXtreme Java supports custom Data Providers as well. For information on how to construct one, see Chapter 18: Creating Your Own Data Provider.

Caveats for Defining JDBC Layers

Each JDBC data source has several corresponding DataProviderHelper constructors. While some constructors are easier to use than others, we strongly recommend setting up your JDBC Layers to use connection pooling. This requires using the most generic form of the DataProviderHelper constructor. Connection pooling is discussed in Chapter 12: Accessing Remote Data. See also a code example in the Javadocs under any JDBCDataProviderHelper.

The TableDescHelper objects for JDBC Layers all share some common parameters. When a JDBC Layer is defined as a table, rather than as a SQL statement, several of these parameters are optional. These include the spatial column, coordinate system of the geometry, dimension of the geometry, and the table level Rendition. If these parameters are not present, the MapInfo.MAPINFO_MAPCATALOG is searched for the information. If you know these values we strongly recommend that you supply it when constructing the object. This will eliminate extra queries to the MAPCATALOG and increase the performance of your application.

When the primary key column(s) is not specified, MapXtreme interrogates the table schema definition in the database to find a column or columns suitable for use as a key. It will attempt to find the column(s) formally defined as the table primary key or, if not present, it will choose a column that the database guarantees is unique to each row, that is not a pseudo column and whose type is character or numeric. Identifying the primary key column(s) in the TableDescHelper eliminates this overhead and guarantees expected behavior when the primary key is used (e.g. in the Selection class). Additionally, the primary key column(s) also receive special treatment when adding features to a JDBC layer (see PrimaryKey of Annotation Layer Features on page 204), so specifying it in this case is also important.

JDBC Layers supports per-Feature Renditions where each Feature stored in the database can be given its own Rendition. The rendition column for the database table is included in the MAPINFO_MAPCATALOG under a column called RENDITIONCOLUMN. If you upload TAB files to a remote database using EasyLoader, this column is created for you. EasyLoader is available on the MapInfo website.

Per-feature label renditions can also be specified for JDBC layers. This allows you to create different style labels for a single layer. The actual Rendition is stored in a column in the table defining the layer. In this case, however, EasyLoader and the MAPCATALOG do not support per-feature label renditions at this time. You must provide the column and type when creating a TableDescHelper for a data source.
Adding a Layer to a MapJ Layers Collection

This is the general procedure to add a layer:

1. Create the TableDescHelper
2. Create the DataProviderHelper
3. Create the DataProviderRef (requires the DataProviderHelper as input)
4. Use Layers.addLayer method (takes DataProviderRef and TableDescHelper as input). This puts the layer at the bottom of the collection, by default. You can also use Layers.insertLayer.

TableDescHelper and DataProviderHelper implementations exist for each type of data source MapXtreme Java supports. The following table is a summary. See the HTML Reference (Javadocs) for more information.

<table>
<thead>
<tr>
<th>Data Source</th>
<th>TableDescHelper</th>
<th>DataProviderHelper</th>
</tr>
</thead>
<tbody>
<tr>
<td>MapInfo Tables</td>
<td>TABTableDescHelper</td>
<td>TABDataProviderHelper</td>
</tr>
<tr>
<td>Oracle with Spatial Option</td>
<td>OraSoTableDescHelper</td>
<td>OraSoDataProviderHelper</td>
</tr>
<tr>
<td>Informix Universal Server SpatialWare DataBlade</td>
<td>IusSpwTableDescHelper</td>
<td>IusSpwDataProviderHelper</td>
</tr>
<tr>
<td>SQL Server for SpatialWare</td>
<td>SQLServerSpwTableDescHelper</td>
<td>SQLServerSpwDataProviderHelper</td>
</tr>
<tr>
<td>JDBC compatible tables containing longitude and latitude columns</td>
<td>XYTableDescHelper</td>
<td>XYDataProviderHelper</td>
</tr>
<tr>
<td>Annotation Layers</td>
<td>AnnotationTableDescHelper</td>
<td>AnnotationDataProviderHelper</td>
</tr>
<tr>
<td>GeoTIFF Raster</td>
<td>GeoTIFFTableDescHelper</td>
<td>GeoTIFFDataProviderHelper</td>
</tr>
<tr>
<td>ESRI Shapefiles</td>
<td>ShapeTableDescHelper</td>
<td>ShapeDataProviderHelper</td>
</tr>
<tr>
<td>Data Binding *</td>
<td>DataBindingTableDescHelper</td>
<td>DataBindingDataProviderHelper</td>
</tr>
<tr>
<td>MapInfo Grid</td>
<td>MIgridTableDescHelper</td>
<td>MIgridDataProviderHelper</td>
</tr>
<tr>
<td>Northwood Grid</td>
<td>NWGridTableDescHelper</td>
<td>NWGridDataProviderHelper</td>
</tr>
</tbody>
</table>

* Special Data Provider for binding data from .tab and JDBC data sources in the same MapJ layer. See Data Binding on page 150 for more information.

Code examples for TAB, Oracle, and JDBC compatible data sources follow. These and other examples are also provided in the online Javadocs Reference.

**TAB Data Provider Example**

The following is an example of creating a TABDataProvider and assigning the layer to MapJ.

```java
// specify the url to the MapXtreme servlet which remotely connects us // to the map engine
String mapXtremeURL = "http://stockholm:8080/mapxtreme47/mapxtreme";
```
// create the tab TableDescHelper
TABTableDescHelper tabTDHelper = new TABTableDescHelper(
        new File("mytab.tab").getName());

// create the tab DataProviderHelper
TABDataProviderHelper tabDPHelper =
        new TABDataProviderHelper("d:\\maps");

// Create the Remote DataProviderRef needed to access the Data
MapXtremeDataProviderRef mxtDPRef = new MapXtremeDataProviderRef(tabDPHelper, mapXtremeURL);

// assign it to MapJ
map.getLayers().addLayer(mxtDPRef, tabTDHelper, "tabLayer");

Oracle Data Provider Example

The following is an example of creating an OracleDataProvider and assigning the layer to MapJ. Be sure that the JDBC driver is in your classpath.

// Specify the URL to the MapXtreme servlet that will be used to access // the data
String mapXtremeURL = "http://stockholm:8080/mapxtreme47/mapxtreme";

// Create the DataProviderHelper
// Using pooled connections (Recommended)
CommonDataProviderHelpers

// Using Database specific DataProviderHelper
OraSoDataProviderHelper oraDPHelper = new
        OraSoDataProviderHelper("dbName", 1521, "dbSid", "mary", "mary123",
                            DriverType.thin, "oracle.jdbc.driver.OracleDriver");

// Create a String array with the name(s) of the column(s) to use as a unique key for records in the table
String[] idColumn = {"mi_prinx"};

// Create a TableDescHelper
// Required constructor when using a tablename
OraSoTableDescHelper oraTDHelper = new OraSoTableDescHelper(
        "states", false, idColumn, "geoloc", null, RenditionType.none,
        null, RenditionType.none, CoordSys.longLatWGS84, 2, "mary");

// Required constructor when using a query
OraSoTableDescHelper oraTDHelper = new OraSoTableDescHelper(
        "select pop_1980, pop_1990, state_name, geoloc, mi_prinx from states where pop_1990 < pop_1980 * 1.03", idColumn, "geoloc", null,
        RenditionType.none, null, RenditionType.none,
        CoordSys.longLatWGS84, 2);

// Reference the remote DataProvider needed to access the data
MapXtremeDataProviderRef mxtDPRef = new MapXtremeDataProviderRef(oraSoDPHelper, mapXtremeURL);

// Add the layer (assume mapj is a MapJ object)
m_myMap.getLayers().addLayer(
        mxtDPRef, oraTDHelper, "Oracle Spatial Layer");
**XY Data Provider Example**

This code sample creates a Data Provider for a JDBC data source where the spatial information is stored in X, Y columns. Be sure your JDBC driver is in your classpath.

```java
// Specify the url to the MapXtreme servlet that will be used to access
// the data
String mapXtremeURL = "http://localhost:8080/mapxtreme47/mapxtreme";

// Create the DataProviderHelper
// Using pooled connections (Recommended)
// Using database specific DataProviderHelper (in this case the XY data
// is located in an Oracle database)
XYDataProviderHelper xyDPHelper = new XYDataProviderHelper(
        "oracle.jdbc.driver.OracleDriver",
        "jdbc:oracle:thin:@serverName:1521:dbSid", "mary", "mary123");

// Create a String array with the name(s) of the column(s) to use as a
// unique key for records in the table
String[] idColumn = {"city_name"};

// Create a TableDescHelper
// Required constructor when using a tablename:
XYTableDescHelper xyTDHelper = new XYTableDescHelper(
        "city125", "mary", false, "longitude", "latitude", null,
        RenditionType.none, null , RenditionType.none , idColumn,
        CoordSys.longLatWGS84);

// Required constructor when using a query:
XYTableDescHelper xyTDHelper = new XYTableDescHelper(
        "select longitude, latitude, city_name from city125 where pop_1990
> 50000", idColumn, "longitude","latitude", null, RenditionType.none, null ,RenditionType.none,
        CoordSys.longLatWGS84);

// Reference the remote DataProvider needed to access the data
MapXtremeDataProviderRef mxtDPRef = new MapXtremeDataProviderRef(
        xyDPHelper, mapXtremeURL);

// Add the layer (assume mapj is a MapJ object)
mapJ.getLayers().addLayer(mxtDPRef, xyTDHelper, "XY Layer");
```

**SQL Server Data Provider Example**

The following is an example of creating a SQL Server DataProvider and assigning the layer to MapJ. Be sure that the JDBC driver is in your classpath.

```java
// Specify the url to the MapXtreme servlet
String mapXtremeURL = "http://localhost:8080/mapxtreme47/mapxtreme";

// Create the DataProviderHelper
// Using pooled connections (Recommended)
// Using database-specific DataProviderHelper
SQLServerSpwDataProviderHelper sqlDPHelper = new
SQLServerSpwDataProviderHelper("machineName",1526,"mary",
"mary123","com.merant.datadirect.jdbc.sqlserver.SQLServerDriver");

// Create a String array with the name(s) of the column(s) to use as a
```
// unique key for records in the table
String[] idColumn = {"sw_member"};

// Create a TableDescHelper
// Required constructor when using a tablename:
SQLServerSpwTableDescHelper sqlTDHelper = new
  SQLServerSpwTableDescHelper("states", "mary", false, idColumn,
  "sw_geometry", null, RenditionType.none, null, RenditionType.none,
  CoordSys.longLatWGS84);

// Required constructor when using a query:
SQLServerSpwTableDescHelper sqlTDHelper = new
  SQLServerSpwTableDescHelper("select state, statecap, sw_member,
  sw_geometry from states where pop_1990 > 2000000", idColumn,
  "sw_geometry", null, RenditionType.none, null, RenditionType.none,
  CoordSys.longLatWGS84);

// Reference the remote DataProvider
MapXtremeDataProviderRef mxtDPRef = new MapXtremeDataProviderRef(
  sqlDPHelper, mapXtremeURL);

// Add the layer (assume mapj is a MapJ object)
mapj.getLayers().addLayer(mxtDPRef, sqlTDHelper, "SQLServer Layer");

Data Binding

MapXtreme Java supports data binding, in which a MapInfo .tab file can be combined with a JDBC data source into a single layer in MapJ. Use data binding to access the geometry data in the .tab file and combine it with the attribute data from database records that lack spatial information.

Data binding is handled through a data provider, located in the com.mapinfo.dp databinding package.

To combine two tables programmatically:

1. Create a TableDescHelper for each table.
2. Create a DataBindingTableDescHelper whose constructor takes the two TableDescHelpers created in the previous step. In addition, the column names that link the two tables will be required. Column names do not need to be identical. One or more columns can be used to bind the two data sources.
3. Create a DataProviderHelper for each table.
4. Create a DataBindingDataProviderHelper whose constructor will take the two DataProviderHelpers created in the previous step.
5. Create a DataProviderRef passing in the DataBindingDataProviderHelper.
6. Add the layer to the Layers collection.

The DataBindingDataProviderHelper handles dispatching searchXXX calls to the appropriate DataProvider and synchronizing the two results.

Additionally, you can add a data binding layer to VisualMapJ through the Layer Control bean.
For a walkthrough of the Add Layer Wizard in which you can add a data binding layer, see Adding a Data Binding Layer in Chapter 5: Managing MapXtreme Java.

**Data Aggregation**

Data aggregation is the process by which multiple records/features are combined as a result of joining the two data sources.

MapXtreme Java contains a class (com.mapinfo.dp.databind.Aggregation) that determines how column values from the attribute source are to be aggregated in the data binding layer. For example, consider the following two tables:

- A geometry .tab file contains sales regions with an attribute column for the region name.
- An Attribute database contains the region name plus sales for each sales person. There can be more than one sales person for each sales region.

Now if you perform a search at point and return sales region A and there are two sales people from that region, there will be two Features returned in the FeatureSet. Here is a possible result:

<table>
<thead>
<tr>
<th>REGION NAME</th>
<th>SALES</th>
<th>SALESPERSON</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100000</td>
<td>Smith</td>
</tr>
<tr>
<td>A</td>
<td>200000</td>
<td>Peterson</td>
</tr>
</tbody>
</table>

Now lets define an aggregation, say sum. The code would look like this:

```java
dbDesc.addAggregation(Aggregation.SUM);
```

If we performed the same search, we would now only get back one Feature. Here is the possible result:

<table>
<thead>
<tr>
<th>REGION NAME</th>
<th>SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>300000</td>
</tr>
</tbody>
</table>

MapXtreme Java will aggregate the data in the following ways:

- **sum** – returns a sum of the values for all records in the group
- **count** – returns a count of the number of records in a group
- **average** – returns a calculated average of the values for all records in the group
- **minimum** – returns the lowest value (or first value in a non-numeric column) for all records in the group
- **maximum** – finds the highest value (or last value in a non-numeric column) for all records in the group

The valid aggregations for non-numeric columns are minimum, maximum, and count.

You can define one aggregation for each column. It is also possible to remove aggregations.

If you perform a search and ask to have columnA and columnB returned but only define an Aggregation for columnB, an exception will be thrown. Either all the returned columns must have an aggregation defined or none can have an aggregation defined.

All the methods to work with aggregations are on the DataBindingTableDescHelper.
The records that will be returned with a particular search include all geometries, even if there isn’t a corresponding record in the database (attribute table).

**Code Example: Data Binding**

```java
String tabDir = "c:\\maps\\";
TABDataProviderHelper tabDPH = new TABDataProviderHelper(tabDir);
String tabFile = "states.tab";
TABTableDescHelper tabDesc = new TABTableDescHelper(tabFile);
String dbUrl = "jdbc:oracle:thin:@spatial.mapinfo.com:1521:maps";
String user = "spatial";
String password = "spatial123";
String driverClass = "oracle.jdbc.driver.OracleDriver";
JDBCDataProviderHelper jdbcDPH = new JDBCDataProviderHelper(
    dbUrl, user, password, driverClass);
String tableName = "USA";
JDBCTableDescHelper jdbcDesc = new JDBCTableDescHelper(
    tableName, user);
DataBindingDataProviderHelper dbDPH = new 
    DataBindingDataProviderHelper(tabDPH, jdbcDPH);
String[] tabBindCol = {"State", "State_Name"};
String[] jdbcBindCol = {"STATE", "STATE_NAME"};
DataBindingTableDescHelper dbDesc = new 
    DataBindingTableDescHelper(tabDesc, jdbcDesc, tabBindCol, 
    jdbcBindCol);
String servletUrl = "http://serverhost:8080/mapxtreme47/mapxtreme"
MapXtremeDataProviderRef dpRef = new MapXtremeDataProviderRef(
    dbDPH, servletUrl);
mapj.getLayers().addLayer(dpRef, dbDesc, "DataBinding Layer");
```

**Annotation Layers**

Annotation layers are special map layers that contain features which mark or place emphasis on certain areas of the map. Typically, features returned from a search method are added to the Annotation layer.

For example, to select and highlight features within a certain radius of a point, use `FeatureLayer.searchWithinRadius` method which returns a circular feature at the point. To display the search radius use the `createCircularRegion` method of the FeatureFactory. Once the feature has been created, use the `addFeature` method to add the new feature to the Annotation layer. You can create the Annotation layer before or after the search.

You may have more than one Annotation layer. The table for an annotation layer resides in memory. It is created using the AnnotationDataProviderHelper, the Annotation TableDescHelper and a LocalDataProviderRef. Once created, it can be treated like any other layer.

You save an Annotation layer just like you’d save any other layer. Any Features added to the Annotation layer will be stored in the Map Definition as an OGC SimpleFeature with the geometry stored according to the GML 2.0 specification.
Here's an example of creating an Annotation Data Provider and adding it to the MapJ Layers collection:

```java
// create the annotation table desc helper
AnnotationTableDescHelper annTDHelper = new AnnotationTableDescHelper("annLayer");

// create the annotation data provider
AnnotationDataProviderHelper annDPHelper = new AnnotationDataProviderHelper();

// An Annotation layer requires the use of local memory space, so we
// create a LocalDataProvider Ref
LocalDataProviderRef localDPRef = new LocalDataProviderRef(annDPHelper);

// assign it to MapJ - note getLayers()
mapJ.getLayers().addLayer(localDPRef, annTDHelper, "AnnLayer");
```

### Analysis Layers

MapXtreme Java provides you with the ability to include pie charts, side-by-side bar charts, and stacked bar charts. The bars in a side-by-side chart display vertically, while the bars in a stacked bar chart display horizontally. These charts are represented by the AnalysisLayer class. For more information, see Working with Analysis Layers on page 246.

### Named Layers

Named layers are a type of layer that you give a unique name. Named layers have the same benefits as other named resources in MapXtreme Java:

- The resource is known by its name and not by its properties.
- The resource resides in one location but can be referenced from many locations.
- To change the look or behavior of applications or data the resource only need be changed, not each application or data file.

And, like other named resources (maps, renditions), named layers use the Java Naming and Directory Interface (JNDI) Application Programming Interface (API). The NamedLayer object handles all of the interaction with JNDI transparently.

To manage named layers through the MapXtreme Java Manager Named Resources panel, see Chapter 5: Managing MapXtreme Java.
Storing Named Layers with NamedResource

Any layer (TAB, annotation, JDBC) can be saved programmatically as a named layer via the JNDI API. The JNDI Context (javax.naming.Context) contains two methods that can be used to save named resources into the named resources repository. They are:

- `bind(String name, Object obj)`
- `rebind(String name, Object obj)`

Use the `bind()` method to save a brand new resource (does not yet exist) into the repository. Use the `rebind()` method to update a pre-existing resource in the repository.

The first thing you need is a JNDI Context. This can be either an InitialContext (javax.naming.InitialContext), or a sub-context of the InitialContext (obtained by doing a `lookup()` via the InitialContext). For more on Contexts and InitialContexts, see the Javadocs for the JNDI API.

To create an initial Context, you need to know the provider URL, which is (most likely) the URL of the NamedResourceServlet. Then invoke the `createInitialContext(providerURL)` factory method of the NamedResourceContextFactory class, as shown below:

```java
```

Now you just need to decide where, relative to the root of your named resources repository, you want to store your named layer.

Let's say you've already created a directory beneath the root of your repository called "my layers", and you want to save a particular layer in that sub-directory as "my states". You would do so like this:

```java
// mapJ was previously initialized
// fetch the "states" layer from the layers collection
Layer states = mapJ.getLayers().get("states");

// create a named resource out of the layer
NamedResource resource = new NamedResource(states);

// Now save it via the container we obtained above
container.bind("my layers/my states", resource);
```

In this example, "my layers/my states" represents a compound name. When specifying compound names, each component of the name must be separated with a "/" (forward slash).

**Note:** You must always store named resources (layer) in a sub-directory of the root of the repository. They should never be stored directly in the root.

Adding a Named Layer to the Layers Collection

To add a previously stored named layer to a Layers collection, you must use one of the following methods of the Layers collection:

- `AbstractLayer addNamedLayer(String providerURL, String path, String resourceName)`
- `AbstractLayer addNamedLayer(Context context, String resourceName)`
• AbstractLayer `insertNamedLayer(String providerURL, String path, String resourceName, int pos)`

• AbstractLayer `insertNamedLayer(Context context, String resourceName, int pos)`

The versions of `addNamedLayer()` and `insertNamedLayer()` that accept a providerURL and path rather than a JNDI Context do not require you to make any JNDI calls at all. So, to add a layer named "my states" which is in a "my layers" sub-directory of the root of your named resources repository, you would do the following:

```java
// mapJ was previously initialized
Layers layers = mapJ.getLayers();
layers.addNamedLayer("http://torpedo:8080/mapxtreme47/namedresource",
    "my layers", "my states");
```

The path argument is optional (you may specify null). If you specify a path, then the `resourceName` argument must be relative to the given path. If no path is given, then the `resourceName` must be relative to the root of the named resources repository. So you could also substitute this `addNamedLayer()` call for the call above:

```java
layers.addNamedLayer("http://torpedo:8080/mapxtreme47/namedresource",
    null, "my layers/my states");
```

The versions of `addNamedLayer()` and `insertNamedLayer()` that accept a JNDI Context obviously require that you first obtain a valid JNDI Context. See the section on Storing Named Layers above for information on how to obtain a JNDI Context.

### Methods of the Layers Collection

Now that you’ve added some layers, you will likely need to make some changes to the Layers collection. This section describes several methods to help. In your applications you will be frequently referencing objects and methods through the Layers collection.

#### Get the Name of Layers in a Collection

This example tells you the number of items, in this case the number of layers, in a collection. This is used if you want to cycle through each item in the collection, for example, getting the names of each item:

```java
Layers layers = myMap.getLayers();
AbstractLayer layer;
String layerName;
for (int i=0; i < layers.size(); i++)
{
    layer = layers.get(i);
    layerName = layer.getName();
}
```
Get a Layer from the Collection

The `get(string name)` method gets a specific Layer object from the collection. The `get(string name)` method returns one of the layers as an object. You can reference layers by their names, such as Highways or Cities. You may also reference a layer by its position. The `get(index at)` method returns the layer at a given position in the collection, such as 0, 1, 2, and so on. The index is zero-based. The following examples demonstrate both uses:

```java
AbstractLayer myLayer;
myLayer = myMap.getLayers().get("highways");
myLayer = myMap.getLayers().get(5); // gets the 6th layer
```

Insert a Layer

The `insertLayer` method adds a feature layer to the Layers collection given DataProvider information and the position at which to place the layer. Similar to adding a layer, when inserting a layer you must provide a DataProviderRef and TableDescHelper. Any layers in the collection that come after the inserted layer are shifted down one position.

```java
// inserting a layer at position 5
layers.insertLayer(dataProviderRef, tableDescHelper, 5, "newLayer");
```

Move a Layer

The `move` method repositions a layer in the Layers collection. The first parameter is `From` position (the top layer = 0) and the second parameter is the `To` position.

```java
// moving a layer from the bottom to the top
layers.move(layers.size() - 1, 0);
```

Remove a Layer

The `remove` method removes a specified layer from the map.

```java
// removing a layer by position (top layer)
layers.remove(0);
// removing a layer by name
layers.remove("highways");
```

Remove All Layers

The `removeAll` method removes all layers from the map.

```java
// removing all layers
layers.removeAll();
```

The MapJ HTML Reference has a complete listing of Layers collection methods and properties.

Retrieving the Bounds of a Layer

Use the `getBounds()` method to retrieve the bounds of a layer, which, in effect, allows you to view an entire layer.
Previously, retrieving the bounds of a layer was a computationally intensive operation, especially for JDBC and raster layers. It required querying all the Features and then merging their bounds.

When invoked on a TAB layer, getBounds() queries the .map file for the bounds information. For JDBC layers, getBounds() queries the MapCatalog (if no bounds information exists, null is returned). If successful, a DoubleRect in the numeric CoordSys of the map is returned. This can then be used to set the bounds of the map to the bounds of the layer, a more efficient way than previously available for viewing the entire layer.

This method also provides the underlying functionality in the new View Entire Layer Maptool bean. A working example of the View Entire Layer bean can be seen in the MapXtreme Java Manager interface.

**Code Example**

Here is a sample application that uses Layer.getBounds() to set the bounds of the map to a layer or all layers in the MapJ.

/* setting the MapJ's bounds to the DoubleRect that encompasses all of the features in a particular layer obtain a reference to a layer in the MapJ */
DataLayer lyr= (DataLayer)mapj.getLayers().elementAt(0);

//get the layer's bounds
DoubleRect lyrBounds=lyr.getBounds();

//check if the layer's bounds are null
if (lyrBounds!=null)
{
    mapj.setBounds(lyrBounds);
}

// setting the MapJ's bounds to the DoubleRect that encompasses the bounds of all of the layers in the MapJ

//get all of the layers in the MapJ
Layers ls= mapj.getLayers();

//create the bounds for all the layers in the MapJ
DoubleRect allLayerBounds=new DoubleRect();

//initialize it
allLayerBounds.initBounds();
for (int layerCount=0;layerCount<ls.size();layerCount++)
{
    //get the next layer
    DataLayer l =(DataLayer)ls.get(layerCount);

    //get the bounds of the layer
    DoubleRect lyrBounds=l.getBounds();

    //if the layer's bounds are not null, merge it with the bounds for // the entire MapJ
    if (lyrBounds!=null) {
        allLayerBounds.merge(lyrBounds); }
}
When `getBounds()` Returns Null

But what if `getBounds()` returns null? If it is important that you get the bounds information, you might consider using the previously mentioned "brute force" method of querying all the features and merging the bounds information. Know, however, that for large JDBC tables or queries or raster images, this operation can have a significant impact on performance. For example, for raster images that are not gif, jpeg, png, or tiff, MapXtreme Java must read the entire image into memory, then get the width and height, extremely expensive in terms of CPU time. You must decide if the trade-off of CPU time for viewing an entire layer is worth it to you.

To get the bounds for a layer using the brute force method, follow this code example:

```java
DoubleRect layerBounds = new DoubleRect();
Feature feat = null;
Geometry geom = null;
DoubleRect featBounds = new DoubleRect();

// create a new queryparams that returns geometry only
QueryParams qp = new QueryParams(
    SearchType.entire, true, false, false, false, false);
FeatureSet fs = null;
try {
    // return geometries only
    fs = lyr.searchAll((List)null, qp);
    // initialize layerBounds
    layerBounds.initBounds();
    // get next feature in fs
    feat = fs.getNextFeature();
    while (feat != null) {
        // get the bounds of each geometry in the FeatureSet
        geom = feat.getGeometry();
        if (geom != null) {
            // build the bounds of the layer by merging the
            // bounds of the feature... with the total
            // rectangle
            layerBounds.merge(geom.getBounds());
        }
        // get next feature in fs
        feat = fs.getNextFeature();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
    finally {
        try {
            if (fs != null) {
```
fs.dispose();
}
}
catch (Exception ex1) {}
}

// add a bit of padding
if (layerBounds != null && layerBounds.area() != 0.0 && layerBounds.area() != Double.POSITIVE_INFINITY)
{
    layerBounds.set(layerBounds.center(), layerBounds.width() * 1.05,
                    layerBounds.height() * 1.05);
}

## Zoom Layering

Sometimes you want a map layer to display only at certain zoom levels. Zoom layering controls the display of a map layer only when the map's zoom level falls within a preset distance. You can set a different zoom layering level for each layer.

For example, if your map includes a street layer, you may find that the streets become illegible when the user zooms out too far. Using Zoom Layering, set up your map so that MapXtreme does not display the streets whenever the user zooms out beyond a certain distance, for example, five miles.

### Note:
Zoom layering is one of the most important factors in increasing the performance of rendering the map.

The following sample code sets up Zoom Layering by modifying the Layer object's properties so that the layer only displays if the map's zoom is between 10 and 30 km.

```java
// set layer for zoom layering from 10 to 30 kilometers
layer.setZoomLayer(true);
layer.setMinZoom(new Distance(10.0, LinearUnit.kilometer));
layer.setMaxZoom(new Distance(30.0, LinearUnit.kilometer));
```
You can set a different zoom level for every layer in your map. For example, you have a layer of streets, a layer of county boundaries, and a layer of state boundaries. You want the streets layer to be visible only when the zoom level is less than eight miles. You want the county boundary layer to display when the zoom level falls between 20 miles and 200 miles. You want the states boundary layer to be visible only when the zoom level is greater than 100 miles.

### Generating Labels For a Layer

A map is practically useless without labels that describe its features. MapXtreme supports a wide variety of label attributes to give your map a distinctive look and impart helpful information.

Straightline labels are drawn based on the location of the geographic object’s label anchor point. This approximates, but is not necessarily the object’s centroid. Splined labels follow the curve of the polyline or polygon path.

Labels are dynamically connected to their map objects. If the data or geographic information changes, the labels change. The content of the label is determined by the data associated with the geographic object.

Layers can be set to be automatically labeled using FeatureLayer.setAutoLabel method. The method isAutoLabel returns True or False if the layer will be autolabeled.

In addition to label content, you control the display and style of automatic labels by using methods of the LabelProperties class. You can set conditions for displaying labels, in what style they will display, and what priority they have over all the objects in the layer.

Labeling is a powerful feature in MapXtreme, enough so that it warrants a separate discussion. See Chapter 14: Labeling and Renditions.
Raster Images

Raster images are another type of layer you can include in your map. Rasters are computerized pictures consisting of row after row of tiny dots (pixels). These are sometimes known as bitmaps. Aerial photographs and satellite imagery are common types of raster data found in GIS.

You can display raster images in your MapXtreme Java application as backdrops to the maps you create. You then can overlay additional data, such as street maps and customer locations, on top of the image.

**Note:** This section refers to bringing raster images into MapXtreme Java. This is not to be confused with the raster image output of a complete map that is returned by MapXtremeServlet in an HTML page. Raster outputs are discussed in Chapter 11: Rendering Considerations.

To display a raster image as a map layer, the image must contain geographic registration information, which are coordinates that correspond to earth locations. This will define the proper placement of the image in a map.

MapXtreme Java supports two types of raster images:

- Images that use an associated .tab file containing the geographic registration information. Raster images of this type include TIFF, JPEG, GIF, BMP, PNG, XBM, and MIG (MapInfo Grid).
- Images that have registration information contained in special tags in the image file. Formats of this type include GeoTIFF and MIG\(^1\).

To register an image as a geographically correct image, you can bring the image into MapInfo Professional and register it there. Many USGS map images come with an associated .tab file.

\(^1\) While it is not necessary for a MIG file to have an associated .tab file, you cannot open a MI Grid image with MapXtreme Java Manager directly. Open the associated tab file instead.
Adding Raster Layers to MapJ

Raster images are brought into a map in the same way other map layers are added — by creating a Data Provider that describes the image and its location. In the case of rasters with associated .tab files, you would create a TABDataProvider. For GeoTIFF images, you would create a GeoTIFFDataProvider.

MapXtreme Java implements a flexible raster handling scheme to allow data providers to be created dynamically. In the case of rasters registered in a TAB file, the TAB data provider will read RasterDataProviderFactory.xml to determine what raster data provider to use to load the raster file with. The RasterDataProviderFactory.xml is organized by raster data provider. Each raster data provider entry has a list of file suffixes (e.g. gif, jpeg, etc) indicating what type of raster formats that the specific raster data provider can read. Examining the RasterDataProviderFactory.xml that ships with MapXtreme Java reveals one raster data provider called com.mapinfo.dp.jai.JaiDataProvider is used to read GIF, JPG, PNG and JPEG2000 raster files. JaiDataProvider is the only raster data provider that is supplied with MapXtreme Java. This class uses the Java Advanced Image (JAI) API's for loading raster files. The list of suffixes associated with the JaiDataProvider in RasterDataProviderFactory.xml is not exhaustive and may be augmented by the user to read any raster format that JAI can read.

**Note:** If an image cannot be handled by one of the above raster Data Providers, then an exception is thrown indicating the specific file that cannot be handled.

GeoTIFF files can be added to a MapJ just like any other layer using the GeoTIFF data provider.

The following code illustrates the how to create a GeoTIFFDataProvider and add the GeoTIFF image to a map. In this case, the image is stored on the local system and retrieved by the LocalDataProviderRef.

```java
// Create a TableDescHelper that points to the Tiff image
GeoTIFFTableDescHelper geoTiffTDHelper = new GeoTIFFTableDescHelper("e:\image\dcquad.tif");

// Create DataProviderHelper
// (**note this constructor takes no parameters)
GeoTIFFDataProviderHelper geoTiffDPHelper = new GeoTIFFDataProviderHelper();

// If the data is local, use a LocalDataProviderRef
LocalDataProviderRef localDPRef= new LocalDataProviderRef(geoTiffDPHelper);

// Insert the layer into the map layer collection
map.getLayers().addLayer( localDPRef, geoTiffTDHelper, "GeoTIFF Layer");
```

Considerations for Importing Raster Images

The following sections offer things to keep in mind when importing raster images.
Set Display to Raster Coordinate System

When adding a raster image to your map, make sure to set MapJ's display coordinate system to the raster layer's coordinate system since MapXtreme Java does not re-project raster images.

This code example shows how to determine the coordinate system for a raster layer and set the display coordinate system accordingly:

```java
TableInfo ti= rasterLayer.getTableInfo();
CoordSys cSys= ti.getCoordSys();
myMapJ.setDisplayCoordSys(cSys);
```

Rasters and Performance

Due to the added system demands in displaying and using raster imagery, we recommend that you start your server or application with an expanded maximum heap size of 64 MB or more depending on your application and the types of raster files you use.

For example, to increase the maximum heap size when you are loading layers using the MapXtreme Java Manager, from the command line, type:

```
java -mx64M com.mapinfo.mjm.client.MJMClient <MJM Servlet URL>
```

Render Raster Images Locally

Any handling of all raster images must be done locally. The Renderer object and the image file must exist on the same machine because they need to be read as a random access file.

Image IO Data Provider

MapXtreme Java includes a data provider that allows third parties to display custom raster image formats in their mapping applications. This is through support of Sun's ImageIO interface that is included in the Java 2 Platform SDK 1.4. These custom formats can be read into MapXtreme Java via the IIODataProvider, provided that there is an associated .tab file containing the geographic registration information. The IIODataProvider is contained in com.mapinfo.dp.imageio.

To use the MapXtreme Java ImageIODataProvider, you must first create a plug-in, following the Sun's Image IO guidelines as found on their Web site. Then you must modify the rasterhandlerfactory.properties file as follows. Prepend com.mapinfo.dp.imageio.IIODataProvider to the list of available raster data providers. Be sure to add the IIODataProvider before the JDKRasterDataProvider in order to use IIO, as they read the same formats. See Adding Raster Layers to MapJ on page 162 for more information on rasterhandlerfactory.properties.
Raster Style Tags

Raster style tags are contained in the same .tab file that holds the raster’s registration information. These tags describe certain display effects of the image, including brightness, contrast, grayscale, transparency, translucency, and grid.

**Note:** MapXtreme Java can only read these style tags. To make any style changes, you must open the raster image in MapInfo Professional and change the style in the Adjust Image Style dialog. When you save the image, the .TAB file will be updated with the new style information.

Raster styles can affect the speed of the rendering process. Styles are rendered in MapXtreme Java by adding filters to the rendering chain. Translucency and transparency effects are the most time intensive operations.

These raster images should not be confused with the output raster images that MapXtreme Java renders. Those images are made up of layers of features that may or may not contain a registered raster image.

Below is an example of a raster image registration file. The raster style tags are shown in bold.

The table that follows explains the tag numbers and values.

```
!table
!version 300
!charset WindowsLatin1
Definition Table
File "conus13.tif"
Type "RASTER"
...
RasterStyle 1 62
RasterStyle 2 40
RasterStyle 3 1
RasterStyle 4 1
RasterStyle 7 1525779
RasterStyle 8 221
```

The table below describes the valid raster style tag numbers and corresponding values. The default values are used when no tag for the style is present in the .TAB file.

<table>
<thead>
<tr>
<th>Raster Style</th>
<th>Description</th>
<th>Tag Number</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brightness</td>
<td>Affects how light or dark the image will display.</td>
<td>1</td>
<td>0 -100, Default is 50.</td>
</tr>
<tr>
<td>Contrast</td>
<td>Affects the amount of difference between shaded areas.</td>
<td>2</td>
<td>0 -100, Default is 50.</td>
</tr>
<tr>
<td>Grayscale</td>
<td>Toggle: Determines whether image displays in shades of black and white, or color.</td>
<td>3</td>
<td>0 = off, 1 = on, Default is 0.</td>
</tr>
</tbody>
</table>
Brightness and Contrast

Brightness and contrast affect the overall display of a raster image. Changing the brightness and/or contrast can help to differentiate image features better. Brightness affects how light or dark the image is. The higher the brightness value, the lighter the image will appear. Contrast affects the shading between areas. A higher value means certain areas of the image stand out more than others.

Brightness and contrast values range from 0 to 100 percent. The default value is 50 if the tag is not present. This means do not apply any contrast or brightness to the raster image.

Grayscale

An image that displays in shades of black and white is known as a grayscale image. If the raster image .TAB file includes a rasterstyle tag of 3 1, MapXtreme Java will display the color image as grayscale. This is useful if you are printing the image to a grayscale printer.

Transparency

MapXtreme Java supports one-color transparency, in which a single color in the raster image will be transparent, or invisible when the image is displayed. This is useful if you wish to show features in other layers that would normally be obscured by that color in the image.

<table>
<thead>
<tr>
<th>Raster Style</th>
<th>Description</th>
<th>Tag Number</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparency</td>
<td>Toggle: Determines whether one color will be rendered transparent.</td>
<td>4</td>
<td>0 = off, 1 = on, Default is 0.</td>
</tr>
<tr>
<td>Grid</td>
<td>Determines if the raster is a grid.</td>
<td>6 *</td>
<td>1 = yes, 0 = no. Default is 0.</td>
</tr>
<tr>
<td>Transparency BGR</td>
<td>Determines what color is to be made transparent.</td>
<td>7</td>
<td>BBGGRR, the color values for blue, green and red, respectively, that represent the color to be made transparent. No default.</td>
</tr>
<tr>
<td>Translucency</td>
<td>Determines the degree of transparency of the displayed image.</td>
<td>8</td>
<td>0-255, as determined by the alpha channel value of the image. 0 = 100 percent translucent (invisible), 255 = 0 percent translucent (opaque). No default.</td>
</tr>
</tbody>
</table>

* Tag number 5 has been phased out.
Transparency is represented in the .TAB file by two style tags: tag 4 1 indicates that transparency is turned on, and tag 7 plus a number in the form BBGGRR represents the color to be made transparent. BBGGRR is the hexadecimal value, expressed as an integer, for blue, green and red that make up the color. For example, the value 1525779 for RasterStyle tag 7 represents a shade of dark green.

**Note:** The number 1525779, is the decimal form of hex value 174813, which represent blue 23(0x17), green 72(0x48), and red 19(0x13), respectively.

The image on the left below shows a highway that runs top to bottom in the center of the map, that was later made transparent in MapInfo Professional, shown at right.

Grid

A grid is an image with continuous color gradations that typically represent elevation or temperature changes. MapXtreme Java can determine if a registered raster image is a grid if the raster style tag 6 1 is supplied in the associated .TAB file. If the tag is not present, the image is treated as a regular raster.

Translucency

Translucency is the amount of light passing through an image that gives it a semi-transparent or "sheer" appearance. It is represented by the RasterStyle tag 8 followed by a number between 0 and 255. The number indicates the image's alpha channel, a part of the image that describes its transparency. The lower an image's alpha channel value is, the more translucent it will display. An alpha number of 0 will render the image completely invisible (100 percent translucent). With an alpha value of 255, the image will be completely opaque (0 percent translucency).

Of all the supported raster styles in MapXtreme Java, translucency has the greatest affect on the rendering speed of the image. This is because the operation affects every pixel in the image, regardless of the color. Rendering is also inherently slow in Java for values between 0 and 255.

Grid Images in MapXtreme Java

Grid images are a special type of raster image that show continuous gradation of color that represents some value. Grid maps are created from the interpolation of point data from the source table into a collection of grid cells.
What makes Grid images different from regular raster images is that MapXtreme Java can return the value stored in a cell, whether it is the interpolated value in a continuous grid or the name associated with the cell in a classified grid. Use FeatureLayer.searchAtPoint(...) or the InfoTool to retrieve the information.

Two types of Grid images are supported:

- MI Grid
- Vertical Mapper Grid

**MI Grid Rasters**

MI grid raster files (.MIG) are thematically shaded grid maps created with MapInfo Professional. MIG files can be added to a MapJ either using a TAB data provider to load a TAB file that references the MIG file or directly using the MIGrid data provider. Grid files for US rainfall, temperature, and elevation are included in the MapXtreme Java sample data. For more information on creating MI grid files, see the MapInfo Professional documentation.

**Vertical Mapper Grid Support**

MapXtreme Java reads and displays Vertical Mapper grid files. These files are created and modified in MapInfo Professional using Marconi’s Vertical Mapper add-on.

To display a Vertical Mapper grid file in MapXtreme Java, use the TAB data provider or Northwood Grid data provider. You cannot load a Vertical Mapper grid without its .TAB file. Be sure that NwSGridReader.jar and rgrid.jar are in your system classpath. The TABDataProvider recognizes the file as a Vertical Mapper grid and calls the NWGridDataProvider.

Vertical Mapper grids can be one of two types: continuous (.grc) or classified (.grd). Continuous grids show a color gradation across the map that represents change from cell to cell, for example, in an elevation map. It allows you to estimate the value of a location between actual data collection points.

A classified grid map contains cells whereby the value in the cell maps directly to a color that represents a class of features. For example, in a forestation grid map showing distribution of tree types, the color blue could represent blue spruce trees, red could represent oak, and yellow could represent birch trees.
Rendering Considerations

Rendering is the process of generating a map image. The final step of creating a basic map is to render the image. This chapter presents a variety of rendering options and considerations.

In this chapter:

- MapXtremeImageRenderer ................................................................. 169
- LocalRenderer ................................................................................. 169
- EncodedImageRenderer ................................................................. 169
- Rendering Additional Layers with a Named Map ................................ 169
- Animated Images ............................................................................ 171
- Raster Output Formats .................................................................... 173
- SVG Output ..................................................................................... 173
- WBMP Support ............................................................................... 175
- Composite Renderer ....................................................................... 177
- Progressive Rendering .................................................................... 179
- Intra-Servlet Container Renderer ..................................................... 180
MapXtremeImageRenderer

Remote rendering is handled using the MapXtremeImageRenderer.

Remote rendering means that the application containing the MapJ object will defer to an instance of MapXtremeServlet to create the map. The MapJ client communicates its request (how and what to render) to MapXtremeServlet, which processes the request, and returns the result to MapXtremeImageRenderer as one of these three types:

- Bitmap file (GIF, JPEG, etc.) on the local system using toFile() – this is typically used to store images on the middle-tier and then have the browser request the file from the server.
- Java output Stream object using toStream() – stream can represent a file (in which it would be the same as above), or in memory storage (as the Java Image object, below); for two-tier, use toStream so the client can keep the image in memory.
- Java Image object using toImage() – this keeps the raster in memory, ready to be directly displayed.

MapXtremeImageRenderer is the most common way to render a map because it relies on system resources such as the JVM and fonts that the client system may not have.

LocalRenderer

The LocalRenderer renders a map to a Java2D Graphics2D or BufferedImage object. Graphics2D objects typically come from BufferedImage's or a Swing component. All rendering occurs on the client's machine. As all rendering is done on the client the resources needed for rendering a map must reside on the client machine (fonts, video card, etc.). By default, VisualMapJ uses LocalRenderer.

EncodedImageRenderer

MapXtreme Java provides a special renderer that is used when you wish to create a map with animated images. See Animated Images on page 171 for more information.

Rendering Additional Layers with a Named Map

Named maps were originally introduced in MapXtreme Java 4.0. They enable you to save a collection of layers with a name for easier recall later. This is handled via the MapXtreme Java Manager’s Save Map Definition (see Chapter 5: Managing MapXtreme Java) or programmatically using NamedMapDefContainer (see Chapter 9: MapJ API).
The programmatic interface has been enhanced to allow you to recall a named map and include a list of layers that will be rendered on top of the named map. This is useful when you have an annotation layer in which you want certain features to be highlighted, such as a route between points. This is also a more efficient way to request a map as it sends a “thinner” render request to the server. Previously the process required rendering the named map on the client, then adding the list of layers to the map and sending all the layers to be rendered to the server.

Programmatically, the client, using the new factory method on the ImageRequestComposer, creates an Enterprise XML map image request that includes the name of the named map, the MapJ, and an overlayIndex, which is a list of layers from the MapJ object that the client wants to be rendered over the named map. Layers can be of any supported type in any combination (annotation, TAB, database, query builders, or data binding layers).

**Code Example**

In this example, a base map has been created and saved as a named resource. Another map containing multiple layers has been created and saved as a map definition file. The MDF will be loaded into the MapJ object. When the ImageRequestComposer object is created, the named resource will be overlaid by the number of layers specified by the integer value passed in the constructor.

```java
// Set mime type
private static String mimeType  = "image/gif";

// Specify a map
private static String mymapPath = "C:\\CODE_SAMPLES\\NAMEDMAPS\\POINTS.mdf";
private static String mapName   = "RENSS_CTY";
private static String providerURL = "file:///C:/cat3345/webapps/mapxtreme47/resources/codesample";
private static String mymxtURL = "http://stockholm:8080/mapxtreme47/mapxtreme";
private static final int ovrlayIndex = 1;
public void service(HttpServletRequest request,
    HttpServletResponse response)throws IOException, ServletException
{
    response.setContentType("image/gif");
    ServletOutputStream sos = response.getOutputStream();
    MapJ myMap = new MapJ();
    myMap.loadMapDefinition(mymapPath);
    myMap.setDeviceBounds(new DoubleRect(0,0,800,600));
    try
    {
        //myMap.setZoom(zoom);
        ImageRequestComposer irc =
            ImageRequestComposer.create(providerURL,mapName,myMap,
                                         ovrlayIndex,ImageRequestComposer.MAX_COLORS_TRUECOLOR,
                                         Color.white,mimeType);
        MapXtremeImageRenderer renderer = new
            MapXtremeImageRenderer(mymxtURL);
        renderer.render(irc);
        renderer.toStream(sos);
        renderer.dispose();
    }
    catch (Exception e)
Animated Images

MapXtreme Java provides you with the classes that you need to construct a map with animated images. This capability is available for point features only.

An animated image in MapXtreme Java terms is called an overlay image. An overlay image is a property of the Rendition object. When the Rendition.SYMBOL_MODE property is set to Rendition.SymbolMode.OVERLAY_IMAGE, the Rendition.SYMBOL_URL property is used to retrieve an image from the specified URL. (This is the same behavior for producing any type of image symbol.)

To signal that animated image information is desired, the MapImageRequest in the Enterprise XML protocol has been enhanced to return an image (image/gif, image/jpeg, etc.) and a MapImageResponse. The response is an XML document containing a base map and a list of point overlays. Each point overlay element contains information that describes its style and position relative to the base map.

The MapImageResponse is generated by a new renderer, EncodedImageRenderer. It takes a new MIME type known as application/encodedimage+xml;image/xxx where xxx can be gif, jpg, png, etc.

The process when using the MapJ API is to:

1. Set the rendition of the point features to be animated (Rendition SYMBOL_MODE property set to Rendition.SymbolMode.OVERLAY_IMAGE).
2. Specify the render response will be application/encodedimage+xml with a MIME type of application/encodedimage+xml;image/xxx.
4. Retrieve MapImageResponse document that contains the base static map and the list of overlay image information.

With this information, you are then ready to create your map.

You can also generate this information wholly outside the MapJ API as it conforms to our public Enterprise XML protocol. In that case, the procedure would be similar:

1. Set the rendition of the point features to be animated (Rendition SYMBOL_MODE property set to Rendition.SymbolMode.OVERLAY_IMAGE).
2. Specify the render response will be application/encodedimage+xml.
3. Send request to MapXtremeServlet.
4. Parse the XML response.
Usage Scenario for Animated Images

To illustrate how animated images can be used, consider this scenario. You wish to display point features with animated GIFs in a web browser, such as waving flags marking worldwide office locations. You create a JSP/Servlet that would render the map using the MapImageResponse information you received previously by following the process outlined above. The static base map and its images to be animated are placed inside an "image server" to be accessed later. Then the JSP/Servlet creates a DHTML page with layers. The bottom layer will have the static base map with a URL pointing to the image server. The second layer will have the animated images containing URLs that access the image server. The effect in the browser is that the images display over the base map.

The following is an example of a DHTML page describing such layers.

```html
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>Animated Icons Test</title>
</head>
<body>
<div ID="wa_flag" STYLE="position: absolute; left: 75; top: 20;">
<img SRC="http://stockholm:8080/mapxtreme/servlet/imageserver?name=washington_md_wht_8760.gif" >
</div>
<div ID="ca_flag" STYLE="position: absolute; left: 75; top: 150;">
<img SRC="http://stockholm:8080/mapxtreme/servlet/imageserver?name=california_full_md_wht_7881.gif" >
</div>
<div ID="ny_flag" STYLE="position: absolute; left: 500; top: 75;">
<img SRC="http://stockholm:8080/mapxtreme/servlet/imageserver?name=new_york_full_md_wht_7193.gif" >
</div>
<img SRC="http://stockholm:8080/mapxtremeserver/servlet/imageserver?map=some_unique_id" height=361 width=589 align=LEFT>
</body>
</html>
```

This DHTML document was created in the middle tier for display by a web browser. This application has several standard animated images registered with a user created "image server". After rendering by the middle-tier the resulting static map is registered with the image server for download later by the client's web browser. The name of the static map image is given a unique name as it is unique to the client. The DHTML page is returned to the client's browser where the browser will then try to download the images referenced in its `<IMG>` tags. The effect of animation occurs because of the `<DIV>` tags, which allow layers of HTML to overlay the base HTML.

Code Example

Provided in the examples/server/java directory is a sample application that demonstrates how overlay images can be displayed on a map in a thin client scenario. In this example, the animated images are star symbols that highlight world capitals. To run the servlet, open index.htm located in webapps/samples47 and click on the link to the Overlay Image Sample Servlet.
Raster Output Formats

MapXtreme Java supports a number of raster output formats, including GIF, JPEG, PNG, and WBMP. Output formats of raster images are specified in the ImageRequestComposer by MIME type. MIME is a format standard for non-textual data such as images. The following guidelines can help you decide which type is appropriate for your needs:

- `image/jpeg` – JPEG – good for layers with more than 256 colors.
- `image/gif` – GIF – good for layers with up to 256 colors.
- `image/png` – PNG – a replacement for GIF format; more than 256 colors.
- `image/wbmp` – WBMP – a speciality format for producing graphics in handheld devices such as wireless phones and PDAs.
- `image/svg+xml` – SVG – a format for describing two-dimensional vector graphics in XML.

For example, to output a JPEG, use the ImageRequestComposer, as shown here:

```java
ImageRequestComposer.create(mapj, maxColors, bgColor, "image/jpeg");
```

When using raster files, we suggest you use PNG output. GIF output is limited to a maximum of 256 colors and raster files generally have at least 256 RGB or grayscale colors. Adding a vector layer may bring the total number of colors to greater than 256. If this happens, the colors must be reduced, which is a time intensive operation.

Setting the Quality of a JPEG Image

To control the quality of the JPEG output on the server, set the parameter `jpegQuality` in your servlet container. For example, in Tomcat, edit the web.xml file under the `/mapxtreme47/WEB-INF` directory to contain a value for JPEG quality.

```xml
<init-param>
    <param-name>jpegQuality</param-name>
    <param-value>85</param-value>
</init-param>
```

The `jpegQuality` value ranges from 0–100, with the default at 75. A lower number means the image quality is reduced, but results in a smaller image size.

SVG Output

MapXtreme Java Edition provides support for exporting map images in SVG, a graphics format for describing two-dimensional vector graphics in XML. To specify the export format as SVG, set the MIME type in the ImageRequestComposer. For example:

```java
ImageRequestComposer.create(
    mapj, maxColors, bgColor, "image/svg+xml");
```
SVG support in MapXtreme Java Edition is SVG 1.0 compliant. SVG provides a facility for maps to be rendered on small devices and provides a resolution independent rendering format.


Adding JavaScript Events to SVG

SVG documents also have an event handler facility similar to the event handlers in HTML documents. SVG documents can signal several types of events like mouse move, resizing, etc. This facility allows interaction, for example, via a browser with JavaScript. To use SVG events the SVG document should be embedded in some other document like an HTML document than provides the actual event handler implementations. MapXtreme Java allows the user to specify event handlers for all of SVG's events via the com.mapinfo.xmlprot.mxtj.SvgConditions class. The following example demonstrates how to specify what function to call for the SVG onmouseover event:

```java
// create a Map that associates a function with an SVG event
Map eventMap = new HashMap();
eventMap.put("onmouseover", "OnMouseOver_Event");

// create an SvgConditions object
SvgConditions svgConditions = new SvgConditions(eventMap);

// create an ImageRequestComposer that signifies we want SVG as our
// render format
ImageRequestComposer irc = ImageRequestComposer.create(
    mapj, 257, Color.RED, "image/svg+xml");

// set the SvgConditions
irc.setSvgConditions(svgConditions);

// render
renderer.render(irc);
```

See MapXtreme Java's SVG JSP map viewer for more examples of how harness the power of SVG with events.

MapXtreme Java's SVG documents are SVG v1.0 compliant.

Limitations of SVG in MapXtreme Java

The following limitations exist when working with SVG and MapXtreme Java.

- SVG images cannot be rendered locally.
- The toImage() method of MapXtreImageRenderer will not work with SVG in this version of MapXtreme Java.
- Progressive Rendering will not work with SVG in this version of MapXtreme Java.
WBMP Support

MapXtreme Java Edition provides support for exporting map images in WBMP, a graphics format for use in handheld devices such as wireless phones and PDAs. To specify the export format as WBMP, set the MIME type in the ImageRequestComposer. For example:

\[
\text{ImageRequestComposer.create(mapj, maxColors, bgColor, "image/wbmp");}
\]

WBMP support in MapXtreme Java includes two modes of output: thresholding for fast map display, or dithering which results in a nicer, but slower map display than thresholding. Each is explained below.

Threshold Method for WBMP Output

Thresholding is a rudimentary technique whereby each color pixel in the map is converted to grayscale and then compared to a set threshold, yielding a 0 (black) or 1 (white) representation. The threshold value is configurable in the servlet init routine.

The algorithm used to determine if a particular pixel is translated into a 0 or 1 is as follows: The color is converted to grayscale \((R+B+G/3)\), whereby R, B, and G represent the color values of the pixel in red, green, and blue \((0, 255)\). If the value is greater than the threshold value, the pixel is set to 1 (white). If the value is less than or equal to the threshold value, the pixel color is 0 (black). The default threshold value is 127.

By default, MapXtreme Java performs a threshold conversion of the map automatically when you specify the MapXtremeImageRenderer MIME type as image/wbmp.

To change the threshold value from 127 to generate a different image quality, you must specify the new value in the init routine of your servlet container.

For example, the following is the code addition when using Tomcat.

\[
\text{<servlet>}
\text{  <servlet-name>mapxtreme</servlet-name>}
\text{  <servlet-class>com.mapinfo.mapxtreme.MapXtremeServlet</servlet-class>}
\text{  <load-on-startup>1</load-on-startup>}
\text{  <init-param>}
\text{    <param-name>wbmpThreshold</param-name>}
\text{    <param-value>95</param-value>}
\text{  </init-param>}
\text{</servlet>}
\]

The following illustration approximately represents the original color GIF image (left) and the same image exported to black and white WBMP using the threshold method.
Dithering Method for WBMP Output

Dithering the map image will yield a higher quality map display than thresholding; however, it will take longer to draw.

Dithering takes into account the color of the pixel that is converted and creates a dithered pattern that is applied to the area where the color is used. Dithering is a process that adjusts adjacent pixels of different colors to give the illusion of a color that is not in the current color palette for the browser. The current dithering routine in MapXtreme Java is the Floyd-Steinberg Error Diffusion routine. This method distributes the error during pixel quantization (color reduction) among neighboring pixels. This increases the apparent color resolution of the image that dithering alone does not do.

To set the image export to dithered, you must set the dithering parameter in the init routine for the servlet. For example, the following code sample for a Tomcat servlet is:

```xml
<servlet>
  <servlet-name>mapxtreme</servlet-name>
  <servlet-class>com.mapinfo.mapxtreme.MapXtremeServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
  <init-param>
    <param-name>wbmpDither</param-name>
    <param-value>1</param-value>
  </init-param>
</servlet>
```

The illustration below approximately represents the original online color GIF image (left) and the same image exported to black and white WBMP using the dithering technique.
Composite Renderer

MapXtreme Java’s renderer, called Composite Renderer, allows you to specify which layers are to be redrawn when the map needs updating. This is useful when only some of the layers contain information that has changed. CompositeRenderer allows the layers in MapJ to be separated into statically and dynamically rendered layers. The static layers will only be drawn once and stored locally as a bitmap. The dynamic layers will be redrawn for every render request.

Composite Renderer offers practical applications, such as showing geocoded points on a map or showing a moving vehicle along a road.

To show geocoded points on a map. Create an Annotation layer over a base map to hold the pushpin symbols that represent geocoded locations. With the CompositeRenderer, the layers of the base map could be rendered with labels, and the pushpins would appear over the labels. Without the CompositeRenderer, labels are always drawn last, thus potentially obscuring information below them.

To show a moving object on a map. A GPS application could show a truck moving along a highway on a map. The base map would only have to be drawn once while the truck symbol would be drawn whenever it was needed (e.g., with each location change).

Keep in mind the following information when using CompositeRenderer:

- The static layers are rendered to a bitmap that is stored locally. A lot of memory is used; a 640 x 480 image will consume at least 2.7 MB.
- Changing the zoom on the MapJ will cause the static layers to be distorted. Lines will display “staircasing” or “jaggy” lines.
- Changing the center on the MapJ will result in panning off the edges of the static layers. If this occurs, the static layers can also be regenerated.
Code Sample: "Animation" Layer

This example shows how to create a moving object on a map. You can also find this code sample in the online Javadocs under CompositeRenderer.

```java
try {
    /* ASSUMPTIONS:
    * The variable mapj is of type MapJ and has loaded a map
    * The variable req is of type HttpServletRequest
    * The variable res is of type HttpServletResponse
    * The variable dp is of type DoublePoint
    */
    // Add annotation layer - this layer will consist of one image symbol
    // to "animate"
    AnnotationTableDescHelper atdh = new AnnotationTableDescHelper("Animation_Layer");
    AnnotationDataProviderHelper adph = new AnnotationDataProviderHelper();
    LocalDataProviderRef ldpr = new LocalDataProviderRef(adph);

    // Add the annotation layer
    Layer animate_layer = mapj.getLayers().insertLayers(ldpr, atdh, 0, "Animation_Layer");

    // Create the rendition for the point
    Rendition r = RenditionImpl.getDefaultRendition();
    r.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.IMAGE);
    r.setValue(Rendition.SYMBOL_URL, "file:///C:/images/car.gif");

    // Create the label rendition for the point
    Rendition lr = RenditionImpl.getDefaultRendition();

    // Create the point
    FeatureFactory ff = mapj.getFeatureFactory();
    // An array containing a single int Attribute
    Attribute[] aAIntAttribute = {new Attribute(33)};
    // create a new Primary Key an an integer
    PrimaryKey pkey = new PrimaryKey(aAIntAttribute);
    Feature f = ff.createPoint(dp, r, lr, aAIntAttribute, pkey);
    PrimaryKey pk = animate_layer.addFeature(f);

    // Create the ImageRequestComposer
    ImageRequestComposer imageRC = ImageRequestComposer.create(
        mapj, ImageRequestComposer.MAX_COLORS_TRUECOLOR, Color.blue,
        "image/gif");

    // Create the composite renderer
    // Render the image
    // Stream the image back to the client
    CompositeRenderer compositeRenderer = new CompositeRenderer(
        "http://localhost:8080/mapxtreme47/mapxtrme", 0);
    compositeRenderer.render(imageRC);
    javax.servlet.ServletOutputStream sos = res.getOutputStream();
    compositeRenderer.toStream(sos);

    // Set this attribute to false so that the bottom image is not rendered //
    next time
    compositeRenderer.setRedrawBottom(false);
} catch(Exception e) {
```

© 2003 MapInfo Corporation. All rights reserved.
Progressive Rendering

Progressive rendering is MapXtreme Java’s ability to send partial map images to the client followed by successively more complete images at specified time intervals, until the complete map image is received. This allows you to receive some aspects of the image sooner than if the entire map image was sent once it was completely rendered.

Progressive rendering is useful when you need to regenerate layers that take a long time to access the underlying data. With progressive rendering, the total time it takes for the entire image to display is slightly longer than if one final image was sent.

The client controls whether an image is progressively rendered and at what interval it should be returned via a special MIME type used with the ImageRequestComposer. This MIME type takes the form:

```
multipart/image;imagetypexx;interval=yyy
```

where xxx is the MIME of the image to be returned (for example, image/gif, image/jpg, etc) and yyy is an update interval specified in milliseconds.

The methods toStream(), toFile(), and toImage() on the MapXtremeImageRenderer class return the next image chunk in the stream if progressive rendering is turned on. A boolean isDone() method allows you to loop and retrieve each successive image in the stream until they are exhausted. The following code sample illustrates the use of isDone().

```java
// create a MIME that tells the MapXtremeImageRenderer
// to render the map to PNG every three quarters of a second.
String progressiveMime =
    "multipart/image;imagetypex=image/png;interval=750";

// standard ImageRequestComposer using MapJ
ImageRequestComposer irc = ImageRequestComposer.create{
    mapj, MAX_COLORS, Color.WHITE, progressiveMime};

// create MapXtremeImageRenderer
MapXtremeImageRenderer renderer = new
    MapXtremeImageRenderer(MXJ_SERVLET_URL);

// while we are not rendering
while (!isDone()) {
    // get an image from the MapXtreme Servlet
    Image image = renderer.toImage();

    // do something with the image
    ...}
```
Intra-Servlet Container Renderer

To leverage servlet forwarding that is provided for in the J2EE 2.2 specification, MapXtreme Java offers the IntraServletContainerRenderer. This feature is an optional way to return raster images to the client. This renderer does not require socket connections between the renderer and MapXtremeServlet, as is necessary with the MapXtremeImageRenderer.

The benefit of this deployment option is that the raster image can be sent directly to the client. MapXtremeServlet does not need to write the image to the middle-tier and then have the middle-tier re-write it back to the client. The limitation, however, is that your application must be deployed in the same container as MapXtremeServlet.

The IntraServletContainerRenderer constructors take as input the information necessary for the middle-tier servlet to obtain a RequestDispatcher object to MapXtremeServlet. The RequestDispatcher object handles the servlet forwarding. The information that is needed includes:

- The alias used by com.mapinfo.mapxtreme.MapXtremeServlet, e.g., "mapxtreme47"
- MapXtremeServlet's ServletContext object, or a URI to the servlet context, for example, "/mapxtreme47/servlet"
- The HttpServletRequest and HttpServletResponse objects that MapXtremeServlet will use to satisfy the request
- The mime type for the raster image
- Whether the image should be multi-part, and an update interval for multi-part

Code Sample: Servlet Forwarding

If the IntraServletContainerRenderer is deployed in a different servlet context than MapXtremeServlet, you may or may not have security issues when dispatching requests.

For Tomcat 3.x users, to use the IntraServletContainerRenderer correctly, you will need to set the following attribute to true in the conf/server.xml file, for any context that has a servlet deployed which uses the IntraServletContainerRenderer. It should look like:

```xml
<Context crossContext="true" ....</Context>
```

Failing to do this will cause the ServletContext of the MapXtremeServlet to be null when returned.

```java
/* Assumptions:
The variable mapj is of type MapJ and has loaded a map.
The variable req is of type HttpServletRequest.
The variable res is of type HttpServletResponse.
*/
// Retrieve the current servlet's ServletConfig object
ServletConfig thisServletConfig = getServletConfig();

// Retrieve the current servlet's ServletContext object from the
//thisServletConfig object
ServletContext thisServletContext =
    thisServletConfig.getServletContext();

/* NOTE: Retrieve the MapXtremeServlet's ServletContext object from the
thisServletContext object. The value of getContext is dependent on how you
deploy the MapXtremeServlet in your servlet container, and is the URI of the
```
MapXtremeServlet. To access the MapXtremeServlet in this deployment you would use http://stockholm:8080/mapxtreme47/mapxtreme.

/*
ServletContext mxtServletContext = thisServletContext.getContext(
        "/mapxtreme47/mapxtreme");

// Check to make sure the mxtServletContext is not null
// (This may be null due to servlet container security)
if(null == mxtServletContext)
{
    // Take appropriate error handling steps
}

// Create the IntraServletContainerRenderer
try
{
    /* NOTE: The argument of "mapxtreme" on the next line is the name of the
    MapXtremeServlet, as defined by the deployment of the servlet
    */
    IntraServletContainerRenderer isRenderer = new
            IntraServletContainerRenderer(mxtServletContext, "mapxtreme", req, res);

    // Create the ImageRequestComposer
    ImageRequestComposer imageRC = new ImageRequestComposer(
            mapj, 256, Color.blue, "image/gif");

    // Render the image
    isRenderer.render(reqEnv);
} catch(Exception e)
{
    // Take appropriate error handling steps
}
MapXtreme Java allows you to access data from remote sources for map rendering and analysis. This chapter discusses named connections and connection pooling, an efficient way to retrieve remote data.

In this chapter:

- **Named Connections** ................................................................. 183
- **Named Connection Pooling** ...................................................... 183
- **How to Create Named Connections** ......................................... 184
- **Accessing Pooled Connections** ............................................... 186
- **Managing Named and Direct Database Connections** ............... 187
Named Connections

A named connection is a resource that describes a connection to a JDBC database using an alias. These connections can be pre-started and pooled with MapXtremeServlet to make them available for immediate use by the client application.

Important note: We strongly recommend that you implement named connections and pooling, as it is much more efficient than creating separate connections for each client request.

Information that describes named connections is stored in a file called miconnections.properties. See How to Create Named Connections on page 184 for more information.

Security Benefits of Named Connections

Beside the efficiency of connection pooling, a significant security benefit exists as well. For three-tier deployments, JDBC connection information stored in the miconnections.properties file remains on the server and is only known to MapXtremeServlet. Clients access the JDBC connections by alias; sensitive information that describes the connection, such as user name and password, is not transmitted over the network.

Named Connection Pooling

Establishing JDBC connections to remote databases can be time consuming and resource intensive. The standard way to minimize these costs is to use named connection pooling. In a connection pooling scheme a group of named database connections are created and then reused and shared among many users.

How Connection Pooling Works in MapXtreme

Typically connection pooling will be used server side with MapXtremeServlet. Additionally, MapJ can be configured to use connection pooling. Whichever way it is used, the behavior will be identical.

When a JDBC connection is needed to access data from a remote database, an attempt will be made to retrieve a connection from the pool. If the pool has an available connection, the connection will be provided to the application. Otherwise, a new connection will be created. When the task has finished using the connection it must be returned to the pool.

Connection Pooling with MapXtremeServlet

MapXtremeServlet will use connection pooling when an miconnections.properties file is found on its classpath (the classpath of its context within its parent servlet container). This file lists the named connections that are available. Any connection in the miconnections.properties can be configured to be pre-started during MapXtremeServlet's init method. This guarantees that the connections will be ready and available to the first clients that visit MapXtremeServlet.
When connection pooling is used by MapXtremeServlet, all open connections are automatically closed when MapXtremeServlet is destroyed by the servlet container. Open connections are also closed when the connection has reached a pre-defined period of inactivity, this is configured as a timeout in the miconnections.properties file.

Connection Pooling with MapJ

Pooling also occurs automatically on the client tier if an miconnections.properties file is located on the classpath of the application using MapJ. Only one connection pool will be created per application and this will be utilized by all MapJ instances created within the application. A MapJ client may need to access a remote data source in the following circumstances: 1) to perform a search method on a Layer doing local data access, 2) to obtain metadata information on a Layer doing local data access, and 3) to do local rendering. If a connection pool is in place, it can be used for each of these tasks.

How to Create Named Connections

Information that describes named connections is contained in a file called miconnections.properties. Information includes the alias or nickname for easy referral to the data source, data source name, host, port, username, password, etc. Multiple JDBC connections can be set up within this file.

Named connections can be created using the Connections Manager panel in the MapXtreme Java Manager, or by hand editing the miconnections.properties file in a text editor.

Once created, put the miconnections.properties file on the classpath so that MapXtremeServlet will pre-start the connections when the init method is invoked.

Example of miconnections.properties File

The following lines represent a sample entry that could be found in the miconnections.properties file.

```java
Connection1_name=Pantheon
Connection1_driver=oracle.jdbc.driver.OracleDriver
Connection1_url=jdbc:oracle:thin:@hostmachine:port:sid
Connection1_user=mapxtreme
Connection1_password=secret
Connection1_is_xy=false
Connection1_prestart=4
Connection1_max=15
Connection1_timeout=300
Connection1_defaultRowPrefetch=75
```

The first line specifies the name of the connection. Clients will use this name to get connections from the pool.

The next four lines specify the standard information needed to establish a JDBC connection: the JDBC driver to use, the connection URL of the database, user name and password.
The next line informs MapXtreme Java whether the data source whose connections are being pooled contains spatial objects or X and Y columns of spatial data.

**Note:** MapXtreme Java cannot use a single named connection to access both X, Y and spatial object data.

The next three settings are for managing the number of connections to pre-start, the maximum number of connections the pool can hold, and the length of time a connection may remain unused before the connection is closed and its resources are returned to the application.

Additional database-specific settings may appear at the end of the list. For example, the Oracle connection shown above has an additional Oracle-specific key called "defaultRowPrefetch." Database-specific keys must be name/value pairs that the connection for the database will understand.

**Connections Manager**

The Connections Manager panel of the MapXtreme Java Manager provides a user interface to manage JDBC connections (i.e., edit the contents of miconnections.properties).

To test your JDBC connection from the Connections Manager, be sure your JDBC drivers are in the classpath.

The Connections Manager will initialize the list of named connections from the miconnections.properties file. You can create new connections or edit or remove existing connections.

The Edit dialog of Connections Manager provides three tabs for supplying information. The General tab collects the name, driver, data source URL, user and password. It also provides a Test Connection button so you can make sure the connection is good (test will fail if the appropriate driver is not in the classpath).
The Custom tab provides a place to set custom properties and values, such as defaultRowPrefetch used in the example on page 184. The Pool tab contains the number of pre-start and maximum connections allowed, and the timeout period for idle connections.

### Accessing Pooled Connections

Individual Layer objects within MapJ must be created in a certain way to make use of pooled named connections. All DataProviderHelpers for JDBC layers share a common constructor type that takes the following input parameters:

- String URL
- Properties connectionProps (user, password, pre-fetch, etc.)
- String driverClassName

For a layer to retrieve a named connections from the pool it must use this form of the DataProviderHelper constructor and follow a special naming convention. The connection URL must be in the form:

\[
\text{jdbc:mipool:connection\_name}
\]

When you are referencing the data source by its alias, the other input parameters (connection properties and driver name) are ignored and should be null. For example, to connect to the Oracle data source which is set up as the "Pantheon" named resource, you would use the following:

```java
OraSoDataProviderHelper oraDpHelper = new OraSoDataProviderHelper("jdbc:mipool:Pantheon", null, null);
```

For more code examples of using connection pooling with DataProviderHelpers, see the code example link "Connection Pooling URL Example" in the Javadocs.
Managing Named and Direct Database Connections

Applications built with MapXtreme can share JDBC connections with MapXtreme. Previously, if an application needed to access the connections directly, it would create and manage its own set of connections. These direct connections would co-exist with any connections maintained by MapXtreme. This resulted in over-allocated resources and inefficiency.

Managing Connections

You can create and manage both types of connections via the ConnectionPool interface. To do this, write an implementation of this interface and identify the class name in the system property com.mapinfo.connpool. Mapxtreme will use this class to get all of its database connections (both types: named connections and direct connections).

When the ConnectionPool receives a request, named connections are retrieved from the pool (assuming the miconnections.properties file is on the classpath). If the request is for a direct connection, that connection is made from scratch.

For more information on the ConnectionPool interface see the MapXtreme Javadocs under the com.mapinfo.dp.conn package.

Differences Between Named and Direct Connections

Named connections are returned to the pool when the user is through and ready for the next request. Direct connections are discarded when no longer needed.

Named connections are requested using their alias following the form jdbc:mipool:resource_name. Direct connections require the complete connection information for each request (URL, connection properties, and driver name).
This chapter introduces the Feature object and various operations you can perform with features, such as searches.

In this chapter:

- The Feature Object ......................................................... 189
- Creating Features Using FeatureFactory ................................ 193
- FeatureSet Collection ......................................................... 196
- Searching ................................................................. 197
- Search Methods ............................................................... 198
- Searching Layers Defined by SQL Queries ......................... 202
- Feature Editing .............................................................. 204
- Editing an Annotation Layer ............................................. 204
- Editing a JDBC Table Layer .............................................. 205
- Editing a Tab Layer ......................................................... 207
The Feature Object

A map Feature is a geographic object on a map such as a point, line, or region. For example, a map of the World could contain regions as countries, lines as highways, and points as cities. In MapXtreme, a map Feature is represented as a Feature object. For example, the UK could be a Feature object of type region, the A10 highway a Feature object of type line, and London a Feature object of type point.

Anyone who has worked with databases is familiar with a record. A record is set of related columns of information. For example, a database of customers will have a record for each customer that includes columns for name, address, interest, etc. A Feature is simply a record that combines tabular data and geometric information. For example, the file World.tab from the MapXtreme sample data is a MapInfo format database. For each country, there is a record. Each record includes several columns of tabular data as well as a reference to the geometric information that describes the shape and location of each country; this allows it to be displayed on the map. The tabular data is referred to as attribute data and the geometric data is referred to as the geometry. These two types of data make a Feature. The following illustration is a conceptual view of a Feature:

<table>
<thead>
<tr>
<th>Country</th>
<th>Capital</th>
<th>Pop_1994</th>
<th>Gr_Rt</th>
<th>Pop_Male</th>
<th>Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>China</td>
<td>Beijing</td>
<td>1,136,429,638</td>
<td>2.2</td>
<td>584,836,207</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>Mexico City</td>
<td>81,249,645</td>
<td>2.2</td>
<td>39,893,969</td>
<td></td>
</tr>
<tr>
<td>United States</td>
<td>Washington, D.C.</td>
<td>257,907,937</td>
<td>0.8</td>
<td>125,897,610</td>
<td></td>
</tr>
</tbody>
</table>

Methods of the Feature Object

The Feature object has methods that obtain information about the tabular and geometric data. The following table lists these methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAttribute</td>
<td>Gets the specified attribute given the column index.</td>
</tr>
<tr>
<td>getAttributeCount</td>
<td>Gets the number of attributes associated with this Feature.</td>
</tr>
<tr>
<td>getGeometry</td>
<td>Gets the associated Geometry or null if the Feature has no geometry.</td>
</tr>
<tr>
<td>getLabelRendition</td>
<td>Gets the Rendition specified for this Feature's label. A null value will be returned if there is no rendition for the Feature.</td>
</tr>
</tbody>
</table>
Attributes
Each Feature can have one or more Attribute objects. Attribute objects represent a column of tabular data for the Feature. This object consists of the type and value of information. For example, an attribute may have a type of double and a value of 2.2 that represents growth rate.

Geometries
Each Feature can have a Geometry object. The Geometry object allows access to all of the geometric information for the Feature. The geometric information may be VectorGeometry or PointGeometry. The VectorGeometry is used for Features that are polylines or regions. The PointGeometry is used for points.

Buffering
MapXtreme Java provides a new method that allows you to buffer regions. You can create a buffered region from any input geometry (point, line, or region). The method takes the geometry object to buffer, a distance object giving the buffer distance, and an integer specifying the resolution as input. A VectorGeometry is always returned as output. The syntax is as follows:

```java
public static VectorGeometry buffer(Geometry geom, Distance dist, int resolution) throws Exception
```

You can supply a buffer distance (scalar value and LinearUnit) to the operation. Geometries in any coordinate system (both spherical and cartesian) can be buffered. You can also specify the resolution of the operation; the number of points used to approximate an arc around each point in the original geometry. For more information, refer to the GeometryUtils class in the online Javadocs.

Feature and Label Renditions
Each Feature can have a Rendition object. The Rendition object describes the display characteristics of the Feature. The Feature object can only get existing rendition information. It tells you how the Feature is being displayed. To change rendition information for existing Features, you should use a theme, such as the OverrideTheme object. Similarly, the Feature can have a a label rendition.

### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getPrimaryKey</td>
<td>Gets a PrimaryKey object (unique ID) for this Feature. A null value will be returned if the Feature does not have a PrimaryKey.</td>
</tr>
<tr>
<td>getRaster</td>
<td>Will return the raster object associated with the Feature if one exists or null if the Feature does not have a Raster.</td>
</tr>
<tr>
<td>getRendition</td>
<td>Returns the rendition for this Feature. A null value will be returned if there is no rendition for the Feature.</td>
</tr>
</tbody>
</table>
Raster

Each Feature can have an MIRaster object. When a raster image is associated with a Feature, you can retrieve binary information that describes an image. When an object has a raster, it may also have a geometry in which case the geometry describes the bounds of the raster image.

Code Sample: Getting Information from a Feature

The code example below demonstrates how to use some of the methods to get information about Features. See also the Feature topic in the Javadocs.

```java
List columns = new ArrayList();
Feature ftr;
Geometry geom;
DoubleRect rect;
DoublePoint dblPnt;
PointGeometry pntGeometry;
VectorGeometry vectorGeometry;
PointList pntList;
Attribute attrib;
int attribCount;

// Get the Table information from the FeatureLayer
TableInfo tabInfo = m_Layer.getTableInfo();

// fill vector with Column names
for (int i=0;i<tabInfo.getColumnCount();i++)
{
    columns.add(tabInfo.getColumnName(i));
}

// Perform a search to get the Features(records) from the layer
RewindableFeatureSet rFtrSet = new
    RewindableFeatureSet(m_Layer.searchAll(columns, null));

// get the first attribute
ftr = rFtrSet.getNextFeature();

// then loop through all features in the layer
while (ftr!=null)
{
    /* get the first attribute (columnData) from the feature Note: If
    you want to re-use the Attribute object later on (after the
    getNextFeature loop), you would need to make a copy of the
    Attribute object, using the copy constructor. */
    attrib = ftr.getAttribute(0);

    // get a count of all attributes in the layer
    attribCount = ftr.getAttributeCount();

    // get the reference to the geographic information from
    // the feature
    geom = ftr.getGeometry();

    // check to see if the geographic object is a Point
    if (geom.getType()==Geometry.TYPE_POINT)
```
{ // Cast the general geometry to a point geometry
pntGeometry = (PointGeometry)geom;

// get the minimum bounding rectangle for the feature
rect = pntGeometry.getBounds();

// get the x,y location where the feature’s label will be
// anchored
dblPnt = pntGeometry.getLabelPoint(null);
} else {

// Cast the general geometry to a Vector geometry
vvectorGeometry = (VectorGeometry)geom;

// get the minimum bounding rectangle for the feature
rect = vvectorGeometry.getBounds();

// get the x,y location where the feature’s label will be
// anchored
dblPnt = vvectorGeometry.getLabelPoint(null);
double[] pnts;
int offset=0;
int numPts;

// Loop through all the point groups and then put the
// points into an array
for (int i=0;i<vvectorGeometry.getPointListCount();++i)
{

// Get the next Point List
pntList = vvectorGeometry.getNextPointList();

// determine the number of Points in the point group
numPts = pntList.getPointCount();

// Create the point array large enough to hold all the
// points
pnts = new double[numPts];

// Call getNextPoints which will put the points into the
// array
pntList.getNextPoints(pnts, offset, numPts/2);
}

// Get the next feature
ftr=rFtrSet.getNextFeature();

// Rewind the FeatureSet to prepare for future use
rFtrSet.rewind();
Creating Features Using FeatureFactory

MapXtreme allows you to create, modify, or delete Features (points, lines, polylines, regions) and add them to an Annotation layer, TAB, or JDBC layer. There are two ways to create new map Features. You can create a Feature using the FeatureFactory object, or create Features by retrieving existing Features using search methods of the Layer class.

FeatureFactory Methods

The methods of the FeatureFactory object let you create new map Features that represent points, lines, polylines, and regions. They are:

- `createPoint`
- `createPolyline`
- `createRegion`
- `createCircularRegion`
- `createEllipticalRegion`

These methods return stand-alone Feature objects. To create any Feature through the FeatureFactory you specify a Rendition, Label Rendition, an array of Attributes associated with that Feature, the Geometry for the Feature, and a PrimaryKey.

**Note:** For TAB files, the PrimaryKey will be assigned when the feature is added to the layer. Also, there is no way to change the primary key.

When creating a Feature to be added to a JDBC table layer, the Feature will ultimately be stored as a row in the database table. The array of Attributes provided to the FeatureFactory are the column values for this new row. These values must exactly match the ordering of the column names retrieved from the layer's TableInfo, i.e. every name in the TableInfo column name array must have a corresponding value in the feature Attribute array. If the JDBC feature Attribute array contains null references, those column values will be inserted as NULL values into the database. Attribute objects that represent NULL values, e.g. `new Attribute((Double)null)`, can also be used in these cases.

Features also need to specify a non-null PrimaryKey value. This does not need to be done for TAB files. Make sure that the PrimaryKey value is part of the Attribute array specified for the Feature.

The Geometry for the JDBC feature must be in the coordinate system of the database table or an exception is thrown.

To create a point, specify the centerpoint, its Rendition, such as symbol size, font and color, and array of Attributes.

For describing the geometry of a circle, in addition to specifying rendition and attributes, you have additional parameters including resolution, and whether the circle is described using display or numeric coordinates. The resolution defines the number of nodes to use when making the approximating polygon and the first parameter controls whether the Feature is a display or numeric circle. The circle is either drawn in the coordsys of the screen (display) or of the map (numeric). The display version will generally look better (i.e., more like a circle) on the screen, whereas the numeric one may appear oblong since it is following the earth’s curve.
In addition to the parameters for a circle, an elliptical region also takes a double to describe the x and y radii.

Creating a region or polyline involves creating a double point array to describe the geometry.

**Code Example**

This example illustrates how to create each of the Feature types in the FeatureFactory: points, circles, lines, ellipses and polylines. To create a region, follow the polyline example to create an array of points that gets passed to createRegion().

This code example can also be found in the Javadocs under FeatureFactory class.

```java
// Get Feature Factory reference
//map is a MapJ object
FeatureFactory ff = map.getFeatureFactory();

// Set up Attribute object
Attribute att[] = new Attribute[1];
att[0] = new Attribute("Feature1");

// Set up rendition object
/* This will not work for tab files. For tab files you must use the com.mapinfo.tab.TABStyleFactory to create an appropriate Rendition object. */
Rendition rend = RenditionImpl.getDefaultRendition();

// For circles, specify the edge and fill color.
rend.setValue(Rendition.STROKE, Color.cyan);
rend.setValue(Rendition.FILL, Color.green);

// For points, specify the symbol size, font, and color
rend.setValue(Rendition.SYMBOL_STRING, "@");
rend.setValue(Rendition.FONT_SIZE, 16);
rend.setValue(Rendition.FONT_FAMILY, "MapInfo Shields");
rend.setValue(Rendition.SYMBOL_FOREGROUND, Color.blue);

// For lines, specify the line color and width
rend.setValue(Rendition.STROKE, Color.green);
rend.setValue(Rendition.STROKE_WIDTH, 4);

// For ellipses, specify the fill color and opacity
rend.setValue(Rendition.FILL, Color.blue);
rend.setValue(Rendition.FILL_OPACITY, new Float(0.40));

// Set the center point features for the circle
DoublePoint dp = new DoublePoint(-104, 45);

// Create Circular region
int circType=1;
int circRadius=25;
int circResolution=25;

// For Elliptical region
// Create an integer variable to set the coordinate system of the region. 0 sets for map coords, 1 sets screen coords
int type = 1;
```
// Create a DoublePoint variable to set the center of the ellipse
DoublePoint centerPt = new DoublePoint(-73.702000, 42.682599);

// Create a double variable to set the rotation angle of the ellipse in
// radians
double xRadius = 1.0;
double yRadius = 0.25;

// Create an integer variable to specify the number of points that will
// describe the ellipse. If 0, the default of 12 will be used.
int resolution = 0;
Feature retFeature;
try
{
    // Create an Annotation layer for storing the features we're
    // creating
    AnnotationTableDescHelper annTableDesc = new
    AnnotationTableDescHelper("Annotations");
    AnnotationDataProviderHelper dpHelper = new
    AnnotationDataProviderHelper();
    LocalDataProviderRef dpRef = new LocalDataProviderRef(dpHelper);
    Layer annotLayer = map.getLayers().insertLayer(
        dpRef, annTableDesc, 0, "Annotations");

    //PrimaryKey is taken as an argument by all the create methods and
    //cannot be null
    PrimaryKey pkey = new PrimaryKey(att[0]);

    // Create a Circular Region
    retFeature = ff.createCircularRegion(circType, dp, circRadius,
        LinearUnit.mile, circResolution, rend, null, att, pkey);

    // Add the new feature to the annotation layer
    PrimaryKey pk = annotLayer.addFeature(retFeature);

    // Create Point
    retFeature = ff.createPoint(dp, rend, null, att, pkey);
    pk = annotLayer.addFeature(retFeature);

    // Create PolyLine using a 2 x 6 double matrix
    // Row 1 contains 3 points(x,y,x,y,x,y) and 2 line segments.
    // Row 2 contains 3 points(x,y,x,y,x,y) and 2 line segments.
    double pts[][] = { {-104, 45, -102, 46, -100, 45},
        {-100, 45, -98, 44, -96, 46} }
    retFeature = ff.createPolyLine(pts, rend, null, att, pkey);

    //Create Elliptical Region
    retFeature = ff.createEllipticalRegion(type, centerPt, angle,
        xRadius, yRadius, LinearUnit.mile, resolution, rend, null, null, pkey);
    pk = annotLayer.addFeature(retFeature);
}
catch (Exception e)
{
    e.printStackTrace();
}
FeatureSet Collection

A FeatureSet is a collection of Features. In MapXtreme, the different layers that make up your map usually have the same Feature type within each layer. For example, the “World” layer has region Features to represent each country, the “US Highways” layer has line Features to represent major U.S. highways, and the “World Capitals” layer has point Features to represent each country’s capital city. The search methods of the Layer object return a FeatureSet collection from a layer.

**Note:** The features in the collection will not be presented in any particular order.

The following methods allow you to manipulate the FeatureSet object:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dispose</td>
<td>Disposes the resources used by the FeatureSet. This must be called once you are done with the FeatureSet.</td>
</tr>
<tr>
<td>getNextFeature</td>
<td>Gets the next Feature in the set.</td>
</tr>
<tr>
<td>getRendition</td>
<td>Gets the base Rendition for all Features in this FeatureSet.</td>
</tr>
<tr>
<td>getTableInfo</td>
<td>Gets the TableInfo (metadata) describing this FeatureSet.</td>
</tr>
<tr>
<td>isRewindable</td>
<td>Determines the rewindable status for this object.</td>
</tr>
<tr>
<td>rewind</td>
<td>Rewinds the FeatureSet prior to the first Feature.</td>
</tr>
</tbody>
</table>

To minimize memory allocation, MapXtreme may reuse the same internal memory when returning a Feature from the getNextFeature method. If you need to hold on to all or parts of a Feature beyond the next call to getNextFeature, make a copy of the object(s) you want to persist. This means that FeatureSets can only be traversed in a forward direction, and that once you pass a Feature you cannot return to it.

Some FeatureSets may be rewindable, which means that the FeatureSet can be reset to its first Feature. This is an implementation detail of each Data Provider. If a FeatureSet is not rewindable you can create a rewindable FeatureSet from a non-rewindable one, then the FeatureSet can once again be traversed.

Here is an example of rewinding a FeatureSet:

```java
if(!fs.isRewindable() )
{
    fs = new RewindableFeatureSet(fs);
}
```

When you are done using FeatureSets the dispose method should always be called.
Searching

One of the most powerful capabilities of MapXtreme is searching. Searching allows you to retrieve specific data according to geographic information. For example, if you were looking for all of the cellular towers within a 25 mile radius, you would perform a search.

Searches are methods of the Layer object. They return FeatureSet objects. A fundamental function of MapXtreme is selecting Features on the map, so that you can perform additional tasks on them. Users can click on the map to select one or more Features (points, lines, regions, etc.). Search results are often interpreted as selections.

The following methods of the Layer object provide various ways to search a Layer and return a FeatureSet collection.

- searchAll
- searchWithinRadius
- searchWithinRegion
- searchWithinRectangle
- searchAtPoint
- searchByAttribute
- searchByAttributes

All searches are passed the names and query parameters of the columns to be returned. The names of the columns you want returned from any search should be put into a List object.

**Limiting the Information Returned in a Search**

The characteristics of the Features returned from a search on a Layer depend on several optional parameters. By default, a Feature's associated Geometry, Rendition, Label Rendition, preferred label position, and raster data are returned with any query. If you wish to limit the information that is returned for a Feature, use the QueryParams class. This will improve query performance.

The QueryParams class also specifies the SearchType for the query. The Features returned from a query are dependent on the search type specified as part of the query. Queries using a search type of \textit{mbr} return Features whose minimum bounding rectangle intersects the search region. This search type is least restrictive and returns the maximum number of Features.

Queries using a search type of \textit{partial} return Features that intersect the search region.

Queries using a search type of \textit{entire} return Features that are completely contained within the search region. It is the most restrictive search type. If you don’t use QueryParams, the SearchType defaults to \textit{mbr}.

Both partial and entire are absolute, valid searches that compare the actual geometry of the feature and the search region. These search types must be used when a true and accurate result is needed. An mbr search is a quick approximation that simplifies the geometry to allow much faster comparison. In certain datasources, such as Oracle Spatial, the mbr search (SDO_FILTER) is actually implemented by intersecting the minimum bounding rectangle of the feature with the minimum bounding rectangle of the region, which results in a larger number of “hits” than using the actual search region.
This example shows how the QueryParams object limits the information returned in a search:

```java
// find all Features entirely within a given search region, return a
// single Attribute column and no Rendition information.
List cols = new ArrayList();
cols.addElement("County");
Feature searchFeature =
    mapj.getFeatureFactory().createRegion(points, rend, attribs, null);
QueryParams queryParams = new QueryParams(SearchType.entire, true, false,
    true, true, false);
FeatureSet fs =
    layer.searchWithinRegion(
        cols, searchFeature.getGeometry(), queryParams);
```

**Note:** Due to certain limitations in the implementation of SQLServer Data Provider,
searchWithinRegion() requests may have degraded performance if the SearchType is not mbr.
This is especially apparent when the search region and the geometry being searched are both large, complex regions.

**Vector Response in GML Format**

MapXtreme Java provides the ability to get the Vector response from a search in GML format, an
XML encoding that describes the geometry of geographic objects.

MapVectorRequests can be used by MapJ client applications and non-clients. For clients created
from MapJ classes, requests are built programatically via a new VectorRequestComposer
class. This class contains factory methods for all supported search methods on both MapJ layer
objects and named layers.

For non-MapJ client applications, a MapVectorRequest document must be created by the
developer’s own means in accordance with the DTD syntax and other policies that govern the
content of the XML document.

A MapVectorRequest will return geometry unless the QueryParams object passed to the
VectorRequestComposer’s createFeature() method is configured to not return geometry, where
Feature represents a point, polyline, or region.

**Code Example**

For a code example of how VectorRequestComposer is used during a search, see the online
example referenced in the API docs for the VectorRequestComposer class.

**Search Methods**

This section defines each search method available and code to demonstrate its use.

**searchAll**

Returns a FeatureSet collection with all Features from the layer. Use this search if your application
requires you to loop through an entire layer.
//Assume fs is a FeatureSet object. 
//Assume columnNames is a List of the columns to be 
//returned. 
//Assume qp is the QueryParams object. 
//Assume myLayer is a FeatureLayer object. 
try { 
    fs = myLayer.searchAll(columnNames,qp); 
} catch(exception e) { 
    e.printStackTrace(); 
}

searchWithinRadius

Returns a FeatureSet collection made up of Features within a specified distance of a point object. 
This search can be used to locate the nearest dealer to a given location, or it could return the 
number of customers within a certain radius of a store. 

//Assume fs is a FeatureSet object. 
//Assume columnNames is a List of the desired columns 
//to be returned. 
//Assume qp is the QueryParams object. 
//Assume myLayer is a FeatureLayer object. 
DoublePoint dblPt = new DoublePoint(-73.889444,42.765555); 
double dRadius = 10.03; 
try { 
    fs = myLayer.searchWithinRadius 
        (columnNames,dblPt,dRadius,LinearUnit.mile,qp); 
} catch(exception e) { 
    e.printStackTrace(); 
}

searchWithinRegion

This search method returns a FeatureSet collection made up of Features within the geometry of a 
Feature. Use this method to return the number of customers in a specific region, such as postal 
code, or return the Features that fall within a region created with the FeatureFactory. 

private boolean layerSearchWithinRegion() 
{ 
    //Assume fs is a FeatureSet object. 
    //Assume columnNames is a List of the columns to be 
    //returned. 
    //Assume qp is the QueryParams object. 
    //Assume myLayer is a FeatureLayer object. 
    //Assume vGeom is a VectorGeometry of TYPE_REGION 
    try { 
        fs = myLayer.searchWithinRegion(columnNames,vGeom,qp); 
    } 
    catch(Exception e) { 
        e.printStackTrace(); 
        return false; 
    } 
    return true; 
}
searchWithinRectangle

This search method returns a FeatureSet collection within the bounds of the specified rectangle. This method could be used to search within the given map window or to pretest a zoom level to see if it will incorporate certain points of interest.

```java
//Assume fs is a FeatureSet object.
//Assume columnNames is a List of the columns to be
// returned.
//Assume qp is the QueryParams object.
//Assume myLayer is a FeatureLayer object.
DoubleRect dRect = new DoubleRect(-74.092662,42.765555,-73.668898,42.856420);
try {
    fs = myLayer.searchWithinRectangle(columnNames,dRect,qp);
} catch(exception e) {
    e.printStackTrace();
}
```

searchAtPoint

This search method returns a FeatureSet collection that is made up of Features at a specified point. This method could be used to test for all objects intersecting a certain point. It could be used to test whether a given location falls within a certain coverage area.

```java
//Assume fs is a FeatureSet object.
//Assume columnNames is a List of the columns to be
// returned.
//Assume qp is the QueryParams object.
//Assume myLayer is a FeatureLayer object.
DoublePoint dp = new DoublePoint(12.3456,-67.890);
try {
    fs = myLayer.searchAtPoint(columnNames,dp,qp);
} catch(exception e) {
    e.printStackTrace();
}
```

searchByAttribute

This search method returns a FeatureSet collection whose attribute matches the given attribute. This method could be used to select all Features with a common piece of attribute information. For example if you had a table that contained a column of household income, you could do a searchByAttribute to return all of the records where household income is greater than $100,000.

```java
// Assume fs is a FeatureSet object.
// Assume columnNames is a list of the columns to be
// returned.
// Assume myLayer is a FeatureLayer object.

try {
    // return Features where "Annual_Income" equals $100,000
    Attribute mySearchAttr = new Attribute(100000);
    String searchCol = "Annual_Income";
```
searchByAttributes()

This release of MapXtreme Java provides a new search method for layers called `searchByAttributes()`. This method allows you to compare values in several columns and returns the features that match the criteria. Currently values are compared using the equality operator.

This method replaces `searchByPrimaryKey` (now deprecated). The columns that define the key are now included in the `attNames` parameter of `searchByAttributes()`. The values that represent the key are put into the `AttTuple` objects in the `attValues` argument. The `AttTuple` object replaces the Primary Key as the holder of the values.

With `searchByAttributes()`, a Feature is returned in a FeatureSet when it satisfies the search condition for any `AttTuple` object. Features are also returned if conditions are implied by multiple `AttTuple` objects that are evaluated using the OR operator.

The syntax for `searchByAttributes()` is as follows:

```java
public FeatureSet searchByAttributes (List columns, List attNames, List attOperators, List attValues, QueryParams queryparams)
throws Exception {...}
```

The general form of the search method is:

"return the requested columns from all features where (colA = v1 and colB = v2) OR (colA = v3 and colB = v4) OR ... (colA = v9 and colB = v10)."

The parameters are defined as follows:

- `attNames = {colA, colB}` // string objects
- `attOperators = {"eq", "eq"}` // `AttOperator` objects
- `attValues = {v1, v2}, (v3, v4), ....(v9, v10)` // `AttTuple` objects that contain `Attribute` objects

Columns can be any columns in the table where a simple operator comparison is valid and whose value can be represented by the `Attribute` class.

An empty list may be returned, but not null.

Because of the deprecation of `searchByPrimaryKey()`, a new constructor on the `QueryParams` interface has been made available. Instead of including the key columns in `QueryParams`, they should now be included in the list of columns defined by `attNames` in `searchByAttributes()`.

For users who are writing custom queries, note that the method in the `QueryBuilder` interface that previously used the PrimaryKey has been deprecated in favor of `queryByAttributes`.

The following code example demonstrates searching for all features in a layer whose value in the "Capital" field equals "Albany."

```java
fs = myLayer.searchByAttribute(
    columnNames, searchCol, mySearchAttr, null);
}
```

```java
catch(Exception e) {
    e.printStackTrace();
}
```
Searching Layers Defined by SQL Queries

MapXtreme Java executes user-defined SQL queries without making any modifications to the query. Referred to as “pass-through” queries, MapXtreme will execute them as written and retrieve all the Features into the layer. The query could return many undisplayed Features, for example, when zoomed in on a densely featured layer.

Pass-through queries are intended for more advanced users of MapXtreme Java who need custom queries to construct layer data and understand how to include the appropriate limiting conditions.

QueryBuilder Interface

To assist power users with the limitations posed by pass-through queries, MapXtreme Java provides an interface that allows you to write your own call back objects to create modified query strings when rendering or performing searches on layers defined by pass-through queries. A QueryBuilder object is given to a pass-through layer, which invokes its methods when needed.
During map rendering, if MapXtreme Java encounters a layer defined by a pass-through query containing a QueryBuilder, the QueryBuilder method `queryInRectangle` is invoked to provide the query string that is passed to the Renderer. The QueryBuilder is provided with all the data needed to construct a new query string that contains the limiting geometric condition that limits the Features returned to only those visible in the display viewport. If that layer does not have a QueryBuilder, it will likely cause significant inefficiency when rendering it as many more Features may be returned than are displayed. (You can determine the number of Features returned that were not rendered by logging the output via com.mapinfo.util.Logger class with a level of INFO or higher.)

Searching a pass-through layer by invoking any of the search methods requires the query to be modified by adding a where clause and/or changing columns in a select clause. Each search method invokes its counterpart method on the QueryBuilder interface and uses the new query string to perform the search. Without the QueryBuilder, a pass-through layer search will throw an exception. A QueryBuilder must be applied to any layer created by a pass-through query in order to perform a search.

To set the QueryBuilder on a layer object, follow this example:

```java
Layer.setQueryBuilder(QueryBuilder myQB);
```

### QueryBuilder Considerations

- The QueryBuilder interface is a power user feature and should only be used when the table definition of a layer is not sufficient.
- New QueryBuilder references can be saved/restored from a Map Definition with the QueryBuilderXMLHandler interface.
- The QueryBuilder interface can only be used in client-side applications, that is, where data access is done where the MapJ instance resides. QueryBuilder objects are not sent to the server.
- You can use the same QueryBuilder reference for more than one layer.
- Queries returned from a QueryBuilder are executed exactly the same as all pass-through queries.
- Using a QueryBuilder does not change any data that defines the layer. The returned query is executed once and discarded; it does not replace the original query from the TableDescHelper that was used to construct the Layer object.
- The data returned by the QueryBuilder query must have the same primary key definition, dimension, coordinate system, and spatial column (if any) as originally identified in the TableDescHelper. (This is a limitation on the QueryBuilder that may be relaxed in later releases.)

### Example Code

Provided in the `/examples/client/QueryBuilders` directory of MapXtreme Java is an example implementation of OracleQueryBuilder that Oracle users can use as a starting point.

You will also find sample code for IdentityQueryBuilder, which returns the original input query unchanged. This is useful as a base class for new QueryBuilder development.
Feature Editing

MapXtreme allows you to add, modify, or delete Features (points, lines, regions, etc.) that make up an Annotation layer, a layer populated from a table in a JDBC data source, or a TAB layer. This is done using the FeatureLayer class methods:

- addFeature
- addFeatureSet
- replaceFeature
- removeFeature

Features that are added to a Layer can be created in the FeatureFactory or can be the result of performing a search (searchWithin, searchBy, etc.) See Creating Features Using FeatureFactory on page 193 for more on the FeatureFactory or Searching on page 197 for a discussion on searching a layer.

Feature editing can only be performed in a client-side application. Be sure that any layers that you plan or may need to edit are created using the LocalDataProviderRef.

Editing an Annotation Layer

Annotation layers contain Features that can be used to mark or place emphasis on certain areas of the map. Annotation layers are not associated with any persistent data source so changes to Annotation layers are only reflected in the current MapJ. Editing annotation layers only changes the image rendered for that layer.

PrimaryKey of Annotation Layer Features

Features added to the Annotation layer must have a non-null PrimaryKey value, which is made up of one or more Attribute values that are also specified as part of the Feature’s (non-null) Attribute array.

Subsequent Features added to the Annotation layer after the first must match the PrimaryKey structure of the first Feature (same number and type of Attribute values defining the PrimaryKey value).

CoordSys of Annotation Layer Features

For Annotation layers, the Features are expected to be in the numeric coordinate system of the MapJ object. When using the FeatureFactory to create Features, it is necessary to specify the input ordinate arrays in the MapJ numeric CoordSys. When taking Features from a FeatureSet returned by a search method, the Feature is already in the MapJ numeric CoordSys.
Editing a JDBC Table Layer

JDBC layers can be defined by either a database table name or a database pass-through query. However, only those layers defined by a database table name can be edited (Features added, replaced or deleted) since the change to the layer is actually a change to the source database table.

The changes made to the table must be in accordance with any constraints defined in the table's schema definition. For example, certain columns may be required to be non-null, unique, have numeric values within a certain range, have numeric values greater than zero, or have string values within a certain length, etc. Violation of these constraints will cause the database to throw an Exception.

Additionally, you must have permission to make changes to the database table.

Persistence of Changes to JDBC Table Layer

A successful change to a JDBC table layer results in a change in the source database table. The change will then be visible in the MapJ the next time the layer data is refreshed from the database. MapXtreme treats each Feature edit request as a separate transaction and will immediately commit the change to the database after each request has successfully completed (or will immediately rollback if a change fails). When the request is addFeatureSet, a commit (or rollback) occurs for each individual Feature in the set.

Rules for Editing Features in a JDBC Layer

The following rules must be followed when adding Features to a JDBC layer. These rules also apply when adding a FeatureSet and when replacing features. For more information see FeatureLayer.addFeature() in the Javadocs.

Feature Definition

All JDBC features to be added must be defined exactly according to the TableInfo returned from calling FeatureLayer.getTableInfo(). In particular the Attribute array of the feature must align exactly with the column name array of the TableInfo, that is, every name in the TableInfo column name array must have a corresponding value in the feature Attribute array. If the JDBC feature Attribute array contains null references, those column values will be inserted as NULL values into the database. Attribute objects that represent NULL values, for example, new Attribute((Double)null), can also be used in these cases.

Feature Geometry

The geometry of the JDBC feature must be in the coordinate system of the table; if not, an exception is thrown.

Feature Renditions

Renditions will only be inserted if the rendition type of the table column is RenditionType.MAPXTREME; if not the rendition will be ignored.
Auto-Increment Columns

If any column in the database is a "read-only" or "auto-increment" column, i.e. its value can only be set by the database, Mapxtreme will ignore these columns. These columns types are used in some databases for the primary key value. All columns, including the primary key columns, must be represented in the JDBC feature Attribute array. MapXtreme will not supply missing values. If a unique value is needed for an integer key column it is possible to use getColumnStatistics() to find the max value in the column and add one to get a unique value.
Editing a Tab Layer

MapXtreme Java provides the ability to edit native MapInfo TAB files. Only TAB files that Java has permission to edit can be updated. That is, files marked as read-only or owned by another user or group will not be editable. Also modifying files from an applet will not be permitted unless special permission is given in the Java security manager.

The Feature being added should have the same table structure as the underlying TAB file.

The column values must adhere to the rules defined in the table. For instance, column with of character columns or precision of decimal columns.

Rules for Editing Features in a Tab Layer

The following rules must be followed when adding Features to a TAB layer.

Feature Renditions
Feature renditions must be created using the com.mapinfo.tab.TABStyleFactory class. This will allow for renditions that are consistent with the styles defined in MapInfo Professional.

Feature Geometry
All geometry objects added are assumed to be in the same coordinate system as the TAB file.

PrimaryKey of Tab Layer Features
When adding new features to a TAB layer, the PrimaryKey will be assigned automatically. The PrimaryKey for TAB files can be returned in one of two ways. The first is by using the deprecated QueryParams. The second is by requesting the pseudo column name com.mapinfo.dp.tab.TABTableDescHelper.KEY_COLUMN_NAME. The Attribute value from this column can be placed in a new PrimaryKey instance.
This chapter describes how to set a variety of label properties and renditions using the API.

In this chapter:

- Labeling Overview ................................................................. 209
- Thematic Labeling ................................................................. 209
- Per-Feature Label Renditions .................................................... 210
- LabelProperties Class ............................................................ 211
- Merging Label Properties ......................................................... 216
- Label Code Example ............................................................... 217
- Rendition Overview ............................................................... 220
- Rendition Properties ............................................................. 220
- Named Renditions ................................................................. 232
- Per-Feature Renditions ......................................................... 234
Labeling Overview

Label elements can be controlled through the MapXtreme Java API. Additionally, these properties can be manipulated via the Label button in the Layer Control dialog in the MapXtreme Java Manager or through the Layer Control Bean. This chapter focuses on the API. For a discussion about labeling via the Layer Control dialog see Chapter 5: Managing MapXtreme Java.

Thematic Labeling

Labels are drawn based on the properties that are set for them via the LabelProperties object. In previous versions only one LabelProperties object could be set for a layer. So all labels for the layer had to look alike.

Now you may have multiple LabelProperties objects per Layer with the introduction of label themes. Similar in concept to Feature thematics, label themes operate on the LabelProperties object. But they are more powerful than Feature themes in that they not only modify the Rendition object of the label, they can affect the label’s position, priority, text, and scalability.

And like Feature thematics, several types of label themes are available: RangedLabel themes, IndividualValueLabel themes, OverrideLabel and SelectionLabel themes. Use these themes as you would their underlying Feature themes: when you wish to treat a collection of labels with a different style, placement, text for labels.

For example, to label a layer whereby the prominence of each label is controlled by the values associated with the features. Create a RangedLabel theme to display labels for features at the higher end of the range with a different color or font size than labels that fall into the lower ranges.

To change the LabelProperties of features returned from a search, use SelectionLabelTheme. This is done by first adding the FeatureSet to a Selection, then associating the Selection with the SelectionLabelTheme.

Multiple label themes can be applied to a layer. Themes at the top of the Theme list may override settings of lower ordered themes. Properties are provided in the LabelProperties object to define how the various label themes are merged to display the final label characteristics.

Code sample: LabelProperties

This sample uses the default position settings, multiline text, and is labeled using multiple columns from the table and labels that follow the path of the feature.

```java
FeatureLayer thisLayer = (FeatureLayer) mapj.getLayers().getLayer("States");
LabelProperties labelProps = new LabelProperties();

// Set the Label offset and alignments to their default positions.
labelProps.setOffset(LabelProperties.DEFAULT_OFFSET);
labelProps.setHorizontalAlignment(
    LabelProperties.HORIZ_ALIGN_DEFAULT);
```
labelProps.setVerticalAlignment(LabelProperties.VERT_ALIGN_DEFAULT);

// Turn the multiline option on and set the columns to be used for
// labeling.
labelProps.setMultiLineTextMode(LabelProperties.MULTILINE_TEXT_ON);
labelProps.setLabelExpression("NAME:["+state_name+"]\nABBREV:["+state+"]\n");

// Set LabelFollowingPath to true to enable label splining
boolean bFollowPath = true;
labelProps.setLabelFollowingPath(bFollowPath);

// Set a Rendition for the Labels and create an OverrideLabelTheme in
// order to display the options set.
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.FONT_FAMILY, "Arial");
rend.setValue(Rendition.FONT_WEIGHT, 2);
labelProps.setRendition(rend);
OverrideLabelTheme orLabelTheme = new OverrideLabelTheme(
    labelProps, "Theme Name");
thisLayer.getLabelThemeList().add(orLabelTheme);

// Get the States Layer, set its LabelProperties object, and turn
// labeling on.
BaseLabelProperties base_label = new BaseLabelProperties(labelProps);
thisLayer.setLabelProperties(base_label);
thisLayer.setAutoLabel(true);

### Zoom Settings for Labels

Previously, when only one LabelProperties object was supported per layer, this object specified
zoom properties. Now the Label Theme specifies its own zoom settings, with each theme for a
layer having its own zoom settings. The zoom properties on LabelProperties have been
deprecated. Feature themes have also been enhanced to include zoom and visibility settings.

### Per-Feature Label Renditions

Map features are rendered in part by using information such as color, line width, fill pattern, and
symbol style. These renditions are returned by calling the `getRendition` method of the Feature
interface. Introduced in MapXtreme Java 3.x, each feature could have its own rendition.

Per-feature labeling allows you to supply a pre-determined Rendition object to describe the label
for each feature. These objects are accessed when you create the appropriate TableDescHelper
for the layer and supply a per Feature Label Rendition column and information describing how to
interpret the column’s data (for example, MapBasic clause, XML format, etc.). The
`getLabelRendition` method is used to return the rendition properties for a feature’s labels.
Renditions are explained beginning on page 220.
LabelProperties Class

The LabelProperties class contains methods that control how labels are drawn for each layer. With the methods in this class, you can control the content, visibility, appearance, position, and relative importance of labels.

Label Column

The text of the label comes from an attribute that is associated with the map feature. These two elements are dynamically linked. If the underlying attribute changes, the label text will change as well.

To control which attribute column will be used for the label for a layer, use the `setLabelColumn` method in the LabelProperties class and specify the column by name, not by index.

Note: The `setLabelColumn` method that took an integer representing the column index has been deprecated.

For example, to make your map more meaningful to your audience, you might label your school district regions with school-age population instead of the name of the school district.

Label Expressions

MapXtreme Java provides the ability to label your map features using either a column of information or static text, or a combination of both. To combine both you would create an expression for the label. For example, you want to create a label that prepends the static text “Pop:” to the actual population values from a column called POP_2000. Use the following method:

```java
labelprops.setLabelExpression("Pop:" + POP_2000)
```

The use of `setLabelExpression` is preferable to using `setLabelColumn`; it can be used with just a single label column.

Label Style

Label style covers a variety of font appearance elements such as the font used for the label text, size, foreground and background color, and special effects.

Any font that is supported by the Java 2D Platform, such as Type 1 or TrueType fonts, can be used for labeling. The Rendition associated with the layer’s LabelProperties object (which is retrieved from its `getRendition` method) is used to control the font, its color and size, the effects for the label, including bold, underline, italic, and background color for box, halo, or outline.

For example, it’s customary to label capital cities of countries larger than labels used for other cities. To draw out the prominence of these capital cities, they might also be labeled with a halo effect that makes them stand out from the other surrounding cities.

This example indicates that the label text will be changed to bold, red, italic text:

```java
//Change the Rendition
LabelProperties labelProp = layer.getLabelProperties();
LabelProperties labelProp = new LabelProperties;
Rendition labelRend = labelProp.getRendition();
```
Creating a Scalable Label

Rendition.FONT_SIZE can take distance units that either describe the size in paper units, such as millimeters, or geographic units, such as miles or kilometers. By using geographic units you can create scalable labels that re-size themselves appropriately when the map zoom changes. As you zoom in on your map, the label would display larger. This also applies to symbol size.

The above code sample uses geographic units. For another example, see page 227 in the Rendition section of this chapter.

The following code example illustrates how to set the font size:

```java
// This example sets a font size of 18 for the layer's labels
Distance distance = new Distance(18, LinearUnit.mile);
labelRend.setValue(Rendition.FONT_SIZE, distance);
labelProp.setRendition(labelRend);
```

Multi-Line Text

Text formatting has been enhanced with the addition of multi-line text properties. There are three modes of operation regarding multi-line text. The method `setLabelMultiLineTextMode` can be used to specify one of the following modes:

- **MULTILINE_TEXT_OFF** – Labels are formatted in a single line of text.
- **MULTILINE_TEXT_ON** – Existing line breaks within label text are respected, resulting in multi-line labels
- **MULTILINE_TEXT_COMPUTE** – Label text is dynamically evaluated by the labeling engine, which may decide to format label text on multiple lines. This mode has the largest run-time overhead.

The default behavior is for multi-line text to be off.

The code sample on page 209 illustrates the use of multi-line text labels.

Label Visibility

The MapJ API includes several ways to control the visibility of your labels: setting a zoom range, whether to allow duplicate and overlapping text, and setting the label priority.

Setting the zoom range for a label is similar to setting the zoom for the layer. You determine at which scale (distance across the map) you want the label to display and set the minimum and maximum values in the Layer's BaseLabelProperties.

**Note:** This zoom setting is not for the label properties associated with Label themes.

If you have two features with the same name, use the `setDuplicationAllowed` method. It allows you to label both features. For example, you may have a state boundary called New York and a city boundary called New York.
The method **setOverlapAllowed** permits multiple labels in a concentrated area to be visible. The default behavior for overlapped labels is False. Use caution with overlapped labels as a crowded map can be harder to read.

To control the density of labels in an area, you can set a priority level for each layer. The **setOverridePriority** method sets whether to use the default priority or an override value. The override value is set by the **setPriority** method. By default, labels for layers toward the top of the layer list have priority in drawing when setDuplicationAllowed or setOverlapAllowed are set to True.

The **setPriority** method changes the priority of labeling for the layer. The default label priority value is given by the equation: (Number of layers - layer position) * 10. So, if a Layers object contained 20 layers, the default label priority for the layer at position 5 is 150. A layer's default label priority may change when other layers are added or removed from its containing Layers object. Higher values have greater priority. Labels with greater priority will be rendered in the case of overlaps or duplicates.

The following is an example of setting the several LabelProperties methods that affect label visibility:

```java
/* This example allows overlap, uses the second column to label, increases the label priority for this layer to 200, and sets zoom labeling from 10 to 30 kilometers.
 */
LabelProperties labelProp = new LabelProperties();
labelProp.setOverridePriority(true);
labelProp.setPriority(200);
labelProp.setLabelColumn(1);
labelProp.setOverlapAllowed(true);

// create a BaseLabelProperties
BaseLabelProperties baseLabel = new BaseLabelProperties(labelProp);
baseLabel.setZoom(true);
Distance maxDist = new Distance(30.0, LinearUnit.kilometer);
Distance minDist = new Distance(10.0, LinearUnit.kilometer);
baseLabel.setZoomMax(maxDist);
baseLabel.setZoomMin(minDist);
layer.setLabelProperties(baseLabel);
```

**Label Position**

The LabelProperties class provides methods for controlling the position of labels, including:

- alignment to the label point
- offset from the label point
- if label is rotated with a line feature (optional)
- if label follows the path of the line feature (optional)
- whether the label position is re-calculated when the zoom changes (optional)
A Feature’s label position is calculated using the following algorithm: the initial position is at the label point returned by the Geometry’s `getLabelPoint` method. If the method returns null (which may occur for some features), then a label point is computed that corresponds roughly to the centroid of the region or mid-point of the line. The initial position is then adjusted for alignment and offset.

If, however, the Follow Path property is set the label position is not calculated from the `getLabelPoint`, but calculated dynamically for the path. See Labeling Along a Path on page 214.

**Alignment**

Labels are horizontally and vertically aligned to the label point. Alignment here refers to the edge of the bounding box of the label that is nearest the label point. Left alignment means the left edge of the bounding box is nearest the label point (label displays to the right).

Horizontal alignment can be specified as Left, Center, or Right aligned, or if not specified, a default alignment is used. VectorGeometries are Center aligned and PointGeometries are Left aligned by default.

Vertical alignment can be set to the Baseline, Top, Bottom, or Center. A top vertical alignment means that the top edge of the bounding box is closest to the label, thus the label would display below the label point. The default behavior is Baseline, if the vertical alignment is not specified.

**Offset**

The second element of label positioning is the offset value. The values of the Offset position are in device units with respect to user space. Note that in Java’s standard user space, the positive y-axis is beneath the x-axis (positive y goes down). If the offset is unspecified, a default value is used. The default offset for regions is (0, 0); the default for lines takes into account the line width and is (0, -w/2) where w is the width. The default for points takes into account the point’s symbol size, and is (s/2 + 2, -s/2 + 2) where s is the symbol size.

**Rotate Label with Line**

Another label position element controls whether the label will follow the slope of a callout line, its default behavior. It can be overridden by using the `setLabelLabelHorizontal` method.

**Labeling Along a Path**

Labels can be set so that they follow the path of the feature geometry when they are rendered. Called splined labels, this operation computes the path on the fly, taking into consideration other label properties settings, such as alignment, offset, and style. Splined labels are best for polylines; however, they can be used with polygons as well. In that case the label follows the path of the polygon boundary.
All properties for straight-line labels are available for curved labels with the exception of multi-line text. In addition, some properties are ignored as they do not apply to curved lines, for example, the force labels horizontal property. The path a label follows is always dynamically computed based on its geometry, clipped to the current view. So the geometry compute mode always defaults to compute for splined labels.

If the label text length exceeds the length of the feature geometry, part of the label will be truncated. The label will be cut off only at whitespace characters in between words. If less than two-thirds of the label's characters fit on the path, the label will not be drawn. This is consistent with the rules for text placement as laid out in the specification for Scalar Vector Graphics, an XML-based export format that MapXtreme Java will be supporting in the near future.

Several creative effects can be applied to curved labels. Haloing, boxing, and underlining are available options for curved labels as well as straight-line labels. Multi-line text for curved labels is not supported.

Setting labels to draw along a curved path will have an impact on the speed at which the map is drawn. Curved labels are computed on the fly and any additional computation will affect the rendering speed. So you must consider a number of factors when setting label properties. It is much more resource expensive to ensure that splined labels do not overlap than their straight-line counterparts. If labels are densely packed at a given zoom level and overlaps are not allowed, overall map rendering will degrade. So consider using splined labels for smaller map extents to obtain the best compromise between a visually rich map and fast rendering time.

Two new methods in the LabelProperties class have been added to the MapJ API to support splined labels. Use the method isLabelFollowingPath() to determine whether labels follow the path of the label geometry, or are placed relative to the label geometry centroid. A return of True means the labels follow the label geometry path.

Use setLabelFollowingPath(boolean bFollowPath) where bFollowPath is set to True for splined labels. The default behavior is for labels to use the centroid of the label geometry as the placement point (no splined labels).

Code Example

```java
// Set LabelFollowingPath to true to enable label splining
boolean bFollowPath = true;
labelProps.setLabelFollowingPath(bFollowPath);
```
Geometry Calculation Mode

The `setGeometryCalculationMode` method specifies whether labels are re-positioned when the map zoom changes. Labels are re-positioned when the value is set to `GEOMETRY_COMPUTED`. It is particularly useful when zooming in from a wide view to a close in view. Your labels will readjust to new positions, the calculation based on the new clipping view of the map. The default behavior is `GEOMETRY_STATIC`; the position will not be recalculated at another zoom.

This example uses alignment, offset, and geometry calculation mode properties.

```java
// change the horizontal and vertical alignments and offset and set the
// Geometry Calculation Mode to Compute.
LabelProperties labelProp = layer.getLabelProperties();
labelProp.setHorizontalAlignment(LabelProperties.HORIZ_ALIGN_RIGHT);
labelProp.setVerticalAlignment(LabelProperties.VERT_ALIGN_TOP);
labelProp.setOffset(new DoublePoint(10, -15));
labelProp.setGeometryCalculationMode(
    LabelProperties.GEOMETRY_COMPUTED);
BaseLabelProperties base_label = new BaseLabelProperties(labelProps);
layer.setLabelProperties(base_label);
```

Merging Label Properties

One layer can have a number of label themes associated with it, all contributing to the final look of the layer. It is possible that each label theme could change some of the same properties as a label theme below it. To accommodate these situations and achieve the final label result, the label themes must merge.

Merging is the process of evaluating each setting and determining the label theme’s priority in the group. Each label theme has a priority number, based on the number of label themes in the group and its position in the group.

Merging takes two things into consideration: which settings take precedence, and whether a default setting means that it continues to use the setting from the underlying label theme, or revert back to the label theme’s original default setting.

You control which settings take precedence by setting a higher `IntraGroupPriority` number. Additionally, you can control whether the priority should be treated as an absolute or relative value by using the `isIntraGroupPriorityCumulative` method.

To determine what default means, consider this example. The topmost label theme is set to a default alignment, but the label theme below is set it to `HORIZ_ALIGN_LEFT`. What does default mean in this case? If the property is set to `NULL`, then the last underlying property (`HORIZ_ALIGN_LEFT`) is accepted. If it’s not null, then the original default value, depending on the geometry type, is used.
Label Code Example

This section provides a code example for changing the label style. The code can also be found in the /codesamples directory of MapXtreme Java.

```java
// set property to display labels
layer.setAutoLabel(true);

// Retrieve the LabelProperties from the layer and then assign the
// Rendition to our rend instance
LabelProperties labelProp = layer.getLabelProperties();
Rendition rend = labelProp.getRendition();

// Set the new rendition values
rend.setValue(Rendition.SYMBOL_FOREGROUND, Color.green);
rend.setValue(Rendition.SYMBOL_BACKGROUND, Color.blue);
rend.setValue(Rendition.FILTER_EFFECTS, Rendition.FilterEffects.HALO);
labelProp.setRendition(rend);
BaseLabelProperties base_label = new BaseLabelProperties(labelProps);
layer.setLabelProperties(base_label);

// Create an ImageRequestComposer
ImageRequestComposer imageRC = ImageRequestComposer.create(
    myMap, 256, Color.blue, "image/gif");

// Create a MapXtremeImageRenderer
MapXtremeImageRenderer renderer = new
MapXtremeImageRenderer(mapxtremeServletURL);

// where mapxtremeServletUrl = URL to MapXtremeServlet,
// such as http://stockholm:8080/mapxtreme47/mapxtreme

// Render the map
renderer.render(imageRC);

// Render the map to the file:
renderer.toFile("comp.gif");
```

**Code Sample: OverrideLabelTheme**

This code sample illustrates how to override the labels for a layer. It is included in the Javadocs under OverrideLabelTheme.

```java
// obtain a reference to the target layer
FeatureLayer layer = null;

// name of the target layer whose labels' appearance will be altered by
// the OverrideLabelTheme
String TARGET_LAYER_NAME = "STATES";
if ((layer = (FeatureLayer)mapj.getLayers().getLayer(
            TARGET_LAYER_NAME)) != null)
{
    /* obtain the target layer's LabelThemeList, and add a new
    OverrideLabelTheme theme to it */
    LabelThemeList labelThemeList = null;
```
if ((labelThemeList = layer.getLabelThemeList()) != null)
{
    /* first, create a new LabelProperties object, which will hold
the settings that define the characteristics of the new
OverrideLabelTheme theme.
*/
    LabelProperties labelProperties = new LabelProperties();

    /* next, add a "LabelExpression" format string to our
LabelProperties object, which describes how the text of the
labels are to be fabricated. Specifically, the below format
string forces label creation such that each label will contain
field values from both the "state_name" and "state" columns of
the underlying table.
*/
    String labelTemplate =
        "\"NAME:\"+["+state_name+"]\\nABBREV:\"+["+state+"]\"";
    labelProperties.setMultiLineTextMode(
        LabelProperties.MULTILINE_TEXT_ON);
    labelProperties.setLabelExpression(labelTemplate);

    /* add a new rendition object to our LabelProperties object,
which will specify a new font to use when rendering all label
text.
*/
    Rendition rend = new RenditionImpl();
    rend.setValue(Rendition.FONT_FAMILY,"Arial");
    rend.setValue(Rendition.FONT_WEIGHT,2);
    // NOTE: 2 == bold
    labelProperties.setRendition(rend);

    /* create a new OverrideLabelTheme object and add it to the
target layer's LabelThemeList. Create a simple description for
the OverrideLabelTheme */
    String THEME_DESCRIPTION_NAME =
        "test_for_override_label_theme";
    OverrideLabelTheme overrideLabelTheme = new
        OverrideLabelTheme(labelProperties,THEME_DESCRIPTION_NAME);
    labelThemeList.add(overrideLabelTheme);
}

Code Sample: Ranged LabelTheme

The following demonstrates how to create a Ranged Label Theme. This sample is included in the
Javadocs under RangedLabelTheme.

// obtain a reference to the target layer
FeatureLayer layer = null;

// name of the target layer whose labels' appearance will
// be altered by the RangedLabelTheme
String TARGET_LAYER_NAME = "STATES";
if ((layer = (FeatureLayer)mapj.getLayers().getLayer(
    TARGET_LAYER_NAME)) != null)
// obtain the target layer's LabelThemeList, and add a
// new RangedLabelTheme theme to it
LabelThemeList labelThemeList = null;
if ((labelThemeList = layer.getLabelThemeList()) != null)
{

    /* generate a column statistics object for a specific column within
    our target table name of a column in the target layer's underlying
    table, which the created RangedLabelTheme will be based upon. */
    String TARGET_COLUMN_NAME = "POP_1990";
    ColumnStatistics columnStatistics =
        layer.fetchColumnStatistics(TARGET_COLUMN_NAME);

    /*
    generate a List of breakpoints, such that each breakpoints
    represents a specific numeric range into which the records of our
    table will be logically grouped.
    */

    //number of breakpoint ranges to create for the RangedLabelTheme
    int NUMBER_OF_BREAKPOINTS = 10;
    List breakPointSeries = (List) Bucketer.computeDistribution
        (NUMBER_OF_BREAKPOINTS, columnStatistics,
         Bucketer.DISTRIBUTION_TYPE_EQUAL_COUNT);

    /* create two(2) rendition objects that will act as the end-points
    for a range of renditions object that are to be created */
    Rendition rendStart = new RenditionImpl();
    rendStart.setValue(Rendition.FONT_FAMILY,"Arial");
    rendStart.setValue(Rendition.FONT_WEIGHT,2); // NOTE: 2 == bold
    rendStart.setValue(Rendition.FONT_SIZE,12);
    rendStart.setValue(Rendition.SYMBOL_FOREGROUND,Color.black);
    Rendition rendEnd = new RenditionImpl();
    rendEnd.setValue(Rendition.FONT_FAMILY,"Arial");
    rendEnd.setValue(Rendition.FONT_WEIGHT,2); // NOTE: 2 == bold
    rendEnd.setValue(Rendition.FONT_SIZE,24);
    rendEnd.setValue(Rendition.SYMBOL_FOREGROUND,Color.red);

    /* using the two(2) rendition objects, create a series of rendition
    objects that represent a gradation from the start rendition to the
    end rendition */
    List renditionSeries=(List) LinearRenditionSpreader.spread(
        NUMBER_OF_BREAKPOINTS, rendStart, rendEnd);

    /* create an ArrayList of LabelProperties objects, such that each
    LabelProperties object within this vector is assigned a rendition
    object from our above created series of rendition objects */
    ArrayList labelPropertiesSeries = new ArrayList();
    LabelProperties labelProperties = null;
    for (int i = 0; i < NUMBER_OF_BREAKPOINTS; i++)
    {
        labelProperties = new LabelProperties();

labelProperties.setRendition((Rendition) renditionSeries.get(i));
labelProperties.setLabelColumn(TARGET_COLUMN_NAME);
labelPropertiesSeries.add(labelProperties);

/* create a new RangedLabelTheme object and add it to our concerned layer's LabelThemeList object. Create a simple description for the RangedLabelTheme. */
String THEME_DESCRIPTION_NAME = "test_for_ranged_label_theme";
RangedLabelTheme rangedLabelTheme = new RangedLabelTheme(
    TARGET_COLUMN_NAME, // java.lang.String
    breakPointSeries, // java.util.List
    labelPropertiesSeries, // java.util.List
    THEME_DESCRIPTION_NAME // java.lang.String
);
labelThemeList.add(rangedLabelTheme);

Rendition Overview

A Rendition object is a sparse collection of display properties, each of which controls one aspect of how a map feature or label will be displayed on the map. It is a sparse collection in that you (the user) only need to set those display properties that matter. The rest will come from combining or merging with other renditions. This is how Themes work in MapXtreme Java. A Theme contains a Rendition that changes just one or two aspects of a map feature (such as the region fill color), which leaves all of the other properties (like the region's edge color, width, etc.) alone.

MapXtreme Java supports a wide variety of display properties to take better advantage of all of the rendering capabilities of the Java2D API. These include symbol paint for lines and regions, dashed and parallel lines, vector symbols, and more.

Additionally, RDBMS data sources (Oracle Spatial, Informix, SQL Server, etc.) have been enhanced to allow for per-feature (record) renditions. This allows for finer control over the display of features coming out of RDBM data sources, thus allowing them to look more like the maps that MapInfo users have come to expect from TAB files.

This section covers per-feature renditions and rendition properties.

Rendition Properties

Rendition properties are used to describe how to display a map feature. The Rendition API supports three categories of properties: fill, stroke, and symbol.
The fill properties control how a region is filled. The stroke properties control how a line (either a line geometry or the edge of a region) will be drawn. The symbol properties control how symbols are drawn for either point geometries, line markers, or symbol fills.

**Fill Properties**

Regions can be filled with either a solid color or a symbol. To set a color, you specify a Java Color object that represents the color you want. For example:

```java
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.FILL, Color.red);
```

Fills can also be defined by using a symbol for the fill paint. Symbols can be any of the supported symbol types (font, image, or vector). Symbol paint can fill a region or a wide line (think of wide lines as a polygon STROKE_WIDTH units in diameter, filled with the STROKE paint). The Symbol is used to create a 'tile' that is repeatedly drawn to fill the region.

Think of tile filling this way. Imagine a ceramic tile floor where each tile has a pattern on it that matches the symbol rendition specified for the region fill. The region is like a hole cut through a sheet of paper that is laid onto the tile floor. The tile pattern will show through the hole. If multiple regions use the same symbol paint, the symbol tile pattern will match or 'line up' such that it will appear as though all of the features were drawn at the same time.

An example of symbol paint would be in specifying a swampy region. A 'swamp' symbol is created (e.g., a GIF file that represents swamp grass). This 'swamp grass' symbol will be used to tile fill the region, giving the appearance that the region is filled with swamp grass.

```java
Rendition rendSymbol = new RenditionImpl();
rendSymbol.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.IMAGE);
rendSymbol.setValue(Rendition.SYMBOL_URL, "http://www.myhost.com/image/swamp.gif");
Rendition rendFill = new RenditionImpl();
rendFill.setValue(Rendition.FILL, rendSymbol);
```

With both solid and symbol paints you can also control the opacity of the paint using the Rendition.FILL_OPACITY property. Values range from 0.0 for a completely transparent fill to 1.0 for a completely opaque fill.

```java
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.FILL, Color.blue);
rend.setValue(Rendition.FILL_OPACITY, 0.5f);
```

**Gradients**

MapXtreme Java now allows you to fill polygons with a linear or a radial gradient fill.

**Linear Gradients**

A linear gradient is a transition through a series of colors along a straight line. Color stops are used to specify the colors desired as specific locations. The straight line that the linear gradient travels is called the transition line. The default transition line goes from left to right. This behavior may be overridden by constructing a LinearGradient with specific start and end points for the transition line. All coordinates for the transition line or specified as a percentage of the length and width of the object to be filled with numbers between 0 and 1. 0 meaning 0% and 1 meaning 100%. This
means to specify a transition line from right to left the user would specify a starting point of (1.0, 0.0) and an ending point of (0.0, 0.0). Other possibilities exist including diagonal transition lines, and transition lines that start and end entirely within the object to be stroked or filled. This behavior is modeled after linear gradients in Scalable Vector Graphics (SVG). When a point to be colored lies outside of the transition line the spread method specifies a rule to use to color the point.

The following example creates a LinearGradient that starts on the left of the region and stops on the right of the region. The color transitions themselves do not start until a quarter of the way down the vector and stops three-quarters of the way down the vector.

```java
// create two color stops: one a quarter of the way "down" the vector and the other three-quarters of the way "down" the vector.
List colorStops = new ArrayList();
colorStops.add(new ColorStop(Color.RED.brighter(), 0.25));
colorStops.add(new ColorStop(Color.RED.darker(), 0.75));

// the gradient vector travels from left to right through the middle of the region
DoublePoint startPoint = new DoublePoint(0.0, 0.5);
DoublePoint endPoint = new DoublePoint(1, 0.5);

// create LinearGradient
LinearGradient linearGradient = new LinearGradient(
    colorStops, SpreadMethod.PAD, startPoint, endPoint);

// set the fill to be the LinearGradient
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.FILL, linearGradient);
```

**Radial Gradients**

A radial gradient is a gradient where the color transition occurs along a circular path. If the bounding box of the object being filled or stroked is not square, that transition path will become elliptical to match the aspect ratio of the bounding box. A radial gradient's limits are determined by a circle; the center is the 0% stop point, and the outer circumference is the 100% stop point. The outer circle is specified with a center and a radius. The center and radius is specified as a percentage of the width and height of the bounding box to be stroked or filled with values between 0 and 1. The default values for the center and radius is (0.5, 0.5) and 0.5, respectively. This places the circle in the middle with a radius half the width and height of the object. If the bounding box of the object to be filled is not square then the circle will become an ellipse with two radii. One radius will be a percentage of the width and the other radius will be a percentage of the height. Color stops are used to specify the colors desired as specific locations. When a point to be colored lies outside of the circle the spread method specifies a rule to use to color the point.

The following example creates a RadialGradient that starts in the center of the object and transitions to a radius half that of the object being filled.

```java
// create two color stops: one a quarter of the way "down" the vector and the other three-quarters of the way "down" the vector.
List colorStops = new ArrayList();
colorStops.add(new ColorStop(Color.RED.brighter(), 0.25));
colorStops.add(new ColorStop(Color.RED.darker(), 0.75));
```
// the gradient vector travels from left to right through the middle of
// the region
DoublePoint center = new DoublePoint(0.5, 0.5);
double radius = 0.5;

// create RadialGradient
RadialGradient radialGradient = new RadialGradient(
    colorStops, SpreadMethod.PAD, center, radius);

// set the fill to be the RadialGradient
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.FILL, radialGradient);

## Stroke Properties

Stroke properties control how a line or region edge display. These properties control the stroke paint, line width, line segment joins and caps, dashing patterns, and more.

The Rendition.STROKE property controls the paint used when drawing the line or edge. The property value can be either a solid color, a Rendition used to specify the symbol paint or a gradient. This is very similar to the Rendition.FILL property described above. When the value is a Rendition, like the FILL property, the symbol properties from that Rendition are used to create a symbol that will be used to tile fill the line. Generally, symbol fills on lines only make sense when the STROKE_WIDTH > 1.

The Rendition.STROKE_WIDTH controls the width of the line (in points) and Rendition.STROKE_OPACITY, the opacity (0.0 for transparent to 1.0 for opaque).

```java
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.STROKE, Color.red);
rend.setValue(Rendition.STROKE_WIDTH, 3.2f);
rend.setValue(Rendition.STROKE_OPACITY, 0.3f);
```

## Parallel Lines

The Rendering engine supports the ability to render one or more lines parallel to the base line. The Rendition.STROKE_PARALLELARRAY property can contain an array of one or more Rendition.ParallelLine objects. Each Rendition.ParallelLine object contains the offset and rendition of a line that is to be drawn parallel to the base line.

The offset tells the rendering engine how many units to draw the line from the base line. Currently all units are specified in printer's points (1/72 inch). The offset can be any number where 0 means draw the parallel line on top of the base line (no offset), +N means draw the parallel line N units to the right of the base line, and -N means draw the parallel line N units to the left. Right and left are determined based on the direction of the base line. If the first point of a line starts on the left side of the screen and the next point is to the right of that point, then an +N offset would have the parallel line below or to the right of the base line's direction of travel.

Each parallel line has a separate Rendition to specify how it should be drawn.

Parallel lines are always drawn after the base line is rendered.

The classic example of parallel lines is that of railroad tracks. Here you have a transparent base line with two parallel lines, one on each side the same distance apart.
Rendition rendParallel = new RenditionImpl();
rendParallel.setValue(Rendition.STROKE, Color.black);
Rendition.ParallelLine parallel1 = new Rendition.ParallelLine(
    3.0f, rendParallel);
Rendition.ParallelLine parallel2 = new Rendition.ParallelLine(
    -3.0f, rendParallel);
Rendition.ParallelLine[] parallelArray = {parallel1, parallel2};
Rendition rendBaseLine = new RenditionImpl();
rendBaseLine.setValue(Rendition.STROKE_OPACITY, 0f);
rendBaseLine.setValue(Rendition.STROKE_PARALLELARRAY, parallelArray);

Dashed Lines

Dashed lines are defined by an array of numbers (float[]), specified in pairs. Each pair specifies the
length of the dash and length of space till the next dash. For example a dashed line with values
[5,3] will have 5 units of line and 3 units of space in between. The STROKE_DASHARRAY
property can be used with any type of line or edge.

The STROKE_DASHOFFSET property controls how many units into the dash array to start the
dashing pattern. For example, assume a STROKE_DASHARRAY property value of [5, 3]. If the
STROKE_DASHOFFSET is not set or set to 0, than the dashing pattern will start with 5 units of
dash followed by 3 units of space. If the STROKE_DASHOFFSET is set to 3, then the line would
have 2 units of line followed by 3 units of space and then 5 units of line followed by 3 units of
space, etc.

An example of dashed lines is to show roads under construction or proposed underground cables.

// Create a line that draws two points, skips four points, and so on
dashArray = new float[] {
    2, 4
};
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.STROKE_DASHARRAY, dashArray);

// Create a dashed line that draws two points, skips four,
// ... and so on except start drawing 4 points into the
// line. This has the effect of starting the line with two
// empty points, two drawn, four empty, two drawn, etc.
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.STROKE_DASHARRAY, dashArray);
rend.setValue(Rendition.STROKE_DASHOFFSET, 4);

Line Markers

Stroke markers are similar to Symbol paint, except that the symbol is rotated to match the angle of
the line. Use line markers to mark the path of a line with repeating symbols.

For example, use Rendition.STROKE_MARKERARRAY and an image of a car to render the effect
of rush hour traffic with one car following another down a road. In this case you would set the
Rendition.SYMBOL_MODE to Rendition.SymbolMode.IMAGE and point the
Rendition.SYMBOL_URL to the URL of a raster file of a transparent car image. (Symbols are
explained below). Then the Rendition.STROKE_MARKERARRAY of a line geometry is set to the
symbol rendition. When rendered, the line geometry will be drawn based on its other STROKE
properties and then the car image will be rotated and repeatedly drawn along the path of each line
segment (a line segment is the path between any two points of a line geometry).
Rendition rendSymbol = new RenditionImpl();
rendSymbol.setValue(Rendition.SYMBOL_MODE,
    Rendition.SymbolMode.IMAGE);
rendSymbol.setValue(Rendition.SYMBOL_URL,
    "http://www.myhost.com/image/car.gif");
Rendition.Marker marker = new Rendition.Marker(rendSymbol);
Rendition rendLine = new RenditionImpl();
rendLine.setValue(Rendition.STROKE_MARKERARRAY,
    new Rendition.Marker[]{marker});

Line Caps, Joins

The Rendition API provides a variety of ways to finish the ends of lines and to join lines together.

Use Rendition.LineCap with a STROKE_LINECAP property to complete lines with round or square 
endcap decorations, or no decoration (use round, square, or butt properties, respectively).

Similarly, the Rendition.STROKE_LINEJOIN property has three ways to connect line segments:
connect outer corners (bevel), extend outer edges to connect (miter), and round off the corner (round).

Gradient Strokes

MapXtreme Java now allows you to use linear or radial gradient strokes. For a complete 
description of linear and radial gradients, see Gradients on page 221.

The following example creates a LinearGradient that starts on the left of the region and stops on 
the right of the region. The color transitions themselves do not start until a quarter of the way down 
the vector and stops three-quarters of the way down the vector.

    // create two color stops: one a quarter of the way "down"
    // the vector and the other three-quarters of the way "down" the 
    //vector.
    List colorStops = new ArrayList();
    colorStops.add(new ColorStop(Color.RED.brighter(), 0.25));
    colorStops.add(new ColorStop(Color.RED.darker(), 0.75));

    // the gradient vector travels from left to right through the middle of
    // the region
    DoublePoint startPoint = new DoublePoint(0.0, 0.5);
    DoublePoint endPoint = new DoublePoint(1, 0.5);

    // create LinearGradient
    LinearGradient linearGradient = new LinearGradient(
        colorStops, SpreadMethod.PAD, startPoint, endPoint);

    // set the fill to be the LinearGradient
    Rendition rend = new RenditionImpl();
    rend.setValue(Rendition.STROKE, linearGradient);

The following example create a LinearGradient that starts on the left of the region and stops on the 
right of the region. The color transitions themselves do not start until a quarter of the way down the 
vector and stops three-quarters of the way down the vector.

    // create two color stops: one a quarter of the way "down"
    // the vector and the other three-quarters of the way "down" the 
    //vector.
```java
List colorStops = new ArrayList();
colorStops.add(new ColorStop(Color.RED.brighter(), 0.25));
colorStops.add(new ColorStop(Color.RED.darker(), 0.75));

// the gradient vector travels from left to right through the middle of
// the region
DoublePoint startPoint = new DoublePoint(0.0, 0.5);
DoublePoint endPoint = new DoublePoint(1, 0.5);

// create LinearGradient
LinearGradient linearGradient = new LinearGradient(
    colorStops, SpreadMethod.PAD, startPoint, endPoint);

// set the fill to be the LinearGradient
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.STROKE, linearGradient);
```

Symbol Properties: Font, Image and Vector

Symbols in MapXtreme Java can do a lot more than just mark a point location. As mentioned above, symbols can be used as the Rendition to fill regions, wide lines, or line markers. Symbols are divided into three types: font, image, and vector.

Font Symbols

Any font that is supported by the Java2 platform, such as Type 1 or TrueType, can be used as a symbol. MapXtreme provides a number of TrueType symbol sets that are typically used in mapping, including:

- MapInfo Cartographic
- MapInfo Transportation
- MapInfo Real Estate
- MapInfo Miscellaneous
- MapInfo Oil & Gas
- MapInfo Weather
- MapInfo Arrows
- MapInfo Shields
- MapInfo Symbols
- Map Symbols

These fonts are located in `/<OPERATING_SYSTEM>/fonts` directory after installation. On platforms other than Windows, you must register these fonts with your operating system in order to use them in MapXtreme Java Edition. To access the fonts programmatically use the common name (for example, MapInfo Arrows, MapInfo Cartographic, and MapInfo Symbols). For a preview of the fonts, see MapXtreme Java TrueType Symbols on page 318.

To view the fonts, use your operating system’s font viewer tool, such as the Unicode Character Map for Windows.
When the Rendition.SYMBOL_MODE property is set to Rendition.SymbolMode.FONT, the font properties (for example, Rendition.FONT_FAMILY) are used with the Rendition.SYMBOL_STRING property to specify a font symbol. When using a font symbol, you can choose the font family, size, background and foreground color, and creative effects such as bold, italic, underline, halo, box, and outline.

Use FONT_SIZE to specify a size using a number representing point size or a distance object representing real world height. This example displays a 12-point symbol.

```java
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.FONT);
rend.setValue(Rendition.FONT_FAMILY, "MapInfo Cartographic");
rend.setValue(Rendition.FONT_SIZE, 12);
rend.setValue(Rendition.SYMBOL_STRING, String.valueOf((char)33));
```

To specify a font symbol to be 12 miles in height such that the symbol scales appropriately at different zoom levels, follow this example:

```java
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.FONT);
rend.setValue(Rendition.FONT_FAMILY, "MapInfo Cartographic");
Distance fontsize = new Distance(12, LinearUnit.mile);
rend.setValue(Rendition.FONT_SIZE, fontsize);
rend.setValue(Rendition.SYMBOL_STRING, String.valueOf((char)33));
```

### Image Symbols

A symbol can also be represented by an image (GIF, JPEG, PNG, etc.), such as a company logo. When the Rendition.SYMBOL_MODE property is set to Rendition.SymbolMode.IMAGE, the Rendition.SYMBOL_URL property is used to retrieve an image from the specified URL. The Rendition.SYMBOL_URL property contains a URL (e.g., http://myhost.com/image/truck.gif).

For example, refer again to the example above of the repeating transparent car image. That example used Rendition.STROKE_MARKERARRAY to specify that a symbol of a car is repeated along the line.

```java
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.IMAGE);
rend.setValue(Rendition.SYMBOL_URL, "http://myhost.com/image/car.gif");
```

The SYMBOL_URL can be any URL including a data: URL in the form specified in RFC2397 ([http://www.faqs.org/rfcs/rfc2397.html](http://www.faqs.org/rfcs/rfc2397.html)). Data URLs allow data to be part of the URL itself. This allows MapXtreme Java map data to be more self-contained and not rely on a server or specific directory structure to exist.

MapXtreme Java provides a set of custom symbol GIF images that you can use to mark your map. They can be found in the /server/symbols/custom directory after installation. See Appendix F: Custom Symbols for symbol thumbnails.

**OVERLAY_IMAGE for Animated Images**

An image symbol can also be represented as an animated image. An animated image in MapXtreme Java terms is called an overlay image. An overlay image is a property of the Rendition object. When the Rendition.SymbolMode property is set to
Rendition.SymbolMode.OVERLAY_IMAGE, the Rendition.SYMBOL_URL property is used to retrieve an image from the specified URL. This is the same behavior for producing any type of image symbol. This property is available for point features only.

Overlay images require a special renderer called the EncodedImageRenderer which takes a new MIME type known as application/encodedimage+xml;image/xxx where xxx can be gif, jpg, png, etc. To signal that animated image information is desired, the MapImageRequest in the Enterprise XML protocol returns an image (image/gif, image/jpeg, etc.) and a MapImageResponse. The MapImageResponse is in the form of an XML document containing a base map and a list of point overlays. Each point overlay element contains information that describes its style and position relative to the base map.

For more information on EncodedImageRenderer, see Chapter 11: Rendering Considerations. For more on MapImageRequest, see Chapter 16: XML Protocol.

### Vector Symbols

If fonts and image symbols do not provide you with the symbology you need, you can draw your own. MapXtreme Java supports vector symbols, in any shape that can be specified by the Java2D Shape interface.

When the Rendition.SYMBOL_MODE property is set to Rendition.SymbolMode.SHAPE, the Rendition.SYMBOL_SHAPE property will be used to create the symbol. The Rendition.SYMBOL_SHAPE property contains a Rendition.SymbolShape object. This object consists of a Rendition object and an object that implements the java.awt.Shape interface.

For example, many of the objects in the java.awt.geom package (like Rectangle2D, Polygon, etc.) implement the Shape interface. You could also use the java.awt.geom.GeneralPath object to specify more complex geometries using commands like moveTo, lineTo, etc. The Rendition.SymbolShape’s Rendition object is used when displaying the Shape. That is, the Rendition might specify the FILL paint to use for a region shape.

```java
Rendition rendShape = new RenditionImpl();
rendShape.setValue(Rendition.FILL, Color.red);
Rectangle2D rect = new Rectangle2D.Float(0, 0, 10, 10);
Rendition.SymbolShape symbolShape = new Rendition.SymbolShape(
    shape, rend);
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.SHAPE);
rend.setValue(Rendition.SYMBOL_SHAPE, symbolShape);
```

### Symbol Sizing

MapXtreme Java supports absolute sizing of symbols, in which real-world distance units can be specified. You may wish to set an absolute size for symbols so that the symbols scale properly when the map is redrawn at a different zoom. Previously, only relative sizing was available in which symbols were scaled to twice their height using an Affine Transformation (SYMBOL_TRANSFORM). This wasn’t always appropriate when the zoom changed.

Symbol sizing is accomplished through a new property of the Rendition class that allows you to specify the real-world size of the symbol in height and width, such as 2 mi x 4 mi. The SYMBOL_SIZE property pertains to symbols of the following types: image symbols, overlay image symbols, and shape (vector) symbols. Font symbols continue to use the FONT_SIZE for scaling.
The SYMBOL_SIZE property will take a Size object that defines the width and height of the symbol as well as the distance units of the dimensions.

It is also possible to combine both relative and absolute sizing for symbols. In that case SYMBOL_SIZE is combined with SYMBOL_TRANSFORM. For example, if SYMBOL_SIZE is set to 2 x 3 miles and SYMBOL_TRANSFORM is set to scale by 2, the symbol will be drawn with a dimension of 4 x 6 miles.

Drawing with a SYMBOL_SIZE set will be slightly slower than when drawing symbols without it. The width and height in real-world dimensions need to be converted to screen pixels; however, the effect on drawing speed is negligible.

For more information, see the Rendition and Size classes in the Javadocs.

**Code Example: Symbol Sizing**

```java
// The symbol size can be in more than one linear unit
Distance symbolWidth = new Distance(50, LinearUnit.kilometer);
Distance symbolHeight = new Distance(75, LinearUnit.mile);
Size symbolSize = new Size(symbolWidth, symbolHeight);
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.IMAGE);
rend.setValue(Rendition.SYMBOL_URL,
    "http://host:8080/locationOfImage/image.gif");
rend.setValue(Rendition.SYMBOL_SIZE, symbolSize);

// Transforms are applied after scaling the image the specified size. So this scales the symbol to 100 kilometers by 225 miles.
rend.setValue(Rendition.SYMBOL_TRANSFORM,
    AffineTransform.getScaleInstance(2, 3));
// now add the rendition to a theme, save to a database, etc....
```

**Code Sample: Creating Vector Symbols for a Line Marker**

This sample code creates a marker array that is composed of an X and a diamond. This means that on top of this line there will be a string of markers: an X followed by a diamond, and so on until the end of the line is reached.

This code is also found in the Javadocs.

```java
// create a shape to represent the first marker (an X)
java.awt.geom.GeneralPath path = new java.awt.geom.GeneralPath();
path.moveTo(-3, -3);
path.lineTo(3, 3);
path.moveTo(3, -3);
path.lineTo(-3, 3);

// each marker is actually a symbol
Rendition symbol1Rend = new RenditionImpl();
symbol1Rend.setValue(Rendition.STROKE, Color.green);
Rendition marker1Rend = new RenditionImpl();
symbolShape = new Rendition.SymbolShape(path, marker1Rend);

// each marker gets its own rendition
Rendition marker1Rend = new RenditionImpl();
marker1Rend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.SHAPE);
```
marker1Rend.setValue(Rendition.SYMBOL_SHAPE, symbolShape);

// create a shape to represent the second marker (an diamond)
GeneralPath path = new GeneralPath();
path.moveTo(3, 0);
path.lineTo(6, -3);
path.moveTo(9, 0);
path.lineTo(6, 3);
path.closePath();

// each marker is actually a symbol
Rendition symbolRend = new RenditionImpl();
symbolRend.setValue(Rendition.STROKE, Color.green);
symbolShape = new Rendition.SymbolShape(path, markerRend);

// each marker gets its own rendition
marker2Rend = new RenditionImpl();
marker2Rend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.SHAPE);
marker2Rend.setValue(Rendition.SYMBOL_SHAPE, symbolShape);
Rendition.Marker[] markerArray = {
    new Rendition.Marker(marker1Rend),
    new Rendition.Marker(marker2Rend)
};

// the rendition with which to draw
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.STROKE_MARKERARRAY, markerArray);

**Code Sample: Font Properties**

The following code sample demonstrates setting a rendition’s FONT_XXX properties

    // Use the MapInfo Miscellaneous font
    rend.setValue(Rendition.FONT_FAMILY, "MapInfo Miscellaneous");

    // Make the font 14 pts
    rend.setValue(Rendition.FONT_SIZE, 14);

    // The weight should be greater than 0. 1.0 = normal, 2.0 = bold, etc.
    rend.setValue(Rendition.FONT_WEIGHT, new Float(2));

    // Make it a leaning garbage can
    rend.setValue(Rendition.FONT_STYLE, Rendition.FontStyle.ITALIC);

    // indicate that we are using a font for the symbol
    rend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.FONT);

    // the number 4 is the garbage can
    rend.setValue(Rendition.SYMBOL_STRING, "4");

    // make the symbol yellow on a red background
    rend.setValue(Rendition.SYMBOL_FOREGROUND, Color.yellow);
    rend.setValue(Rendition.SYMBOL_BACKGROUND, Color.red);

**Code Sample: Fill Properties**

    /* Create a rendition that is filled blue but allows 60 percent of the
    background to be visible (40 % transparent). */
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.FILL, Color.blue);
rend.setValue(Rendition.FILL_OPACITY, new Float(0.40));

Code Sample: Drawing a Line with a Symbol

// define a red line 7.5 pts wide
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.STROKE, Color.red);
rend.setValue(Rendition.STROKE_WIDTH, new Float(7.5f));

// demonstrate creating a rendition to draw a line with a symbol
Rendition symbolRend = new RenditionImpl();
symbolRend.setValue(Rendition.SYMBOL_MODE, Rendition.SymbolMode.FONT);
symbolRend.setValue(Rendition.FONT_FAMILY, "MapInfo Symbols");
symbolRend.setValue(Rendition.FONT_SIZE, 8);
symbolRend.setValue(Rendition.SYMBOL_STRING, "+");

// create rendition to draw lines using symbols
Rendition rend = new RenditionImpl();
rend.setValue(Rendition.STROKE, symbolRend);
rend.setValue(Rendition.STROKE_WIDTH, new Float(5));

Code Sample: Dashed and Parallel Lines

This example demonstrates STROKE, STROKE_DASHARRAY, STROKE_PARALLEL ARRAY, Layers, and Layer. The resulting line has a dashed central line in red and white and a purple line 12 points away and parallel to the central line.

// rendition for the first parallel line
Rendition rend1 = new RenditionImpl();
rend1.setValue(Rendition.STROKE, new Color(1.0f, 0.0f, 1.0f));

// purple line rendition for second parallel line
Rendition rend2 = new RenditionImpl();
rend2.setValue(Rendition.STROKE, Color.white);
rend2.setValue(Rendition.STROKE_DASHARRAY, new float[]{4, 4});

// Create array of parallel lines for the central line. The red and 
// white dashed line is created by specifying a 0.0 for an offset.
Rendition.ParallelLine[] parallelLines = new Rendition.ParallelLine[] {
    new Rendition.ParallelLine(12.0f, rend1);
    new Rendition.ParallelLine(0.0f, rend2);
};

// create rendition for Australia's borders.
Rendition borderRend = new RenditionImpl();
borderRend.setValue(Rendition.STROKE, Color.red);
borderRend.setValue(Rendition.STROKE_PARALLELARRAY, parallelLines);
Named Renditions

Named renditions are a type of named resource in which you give a unique name to the style it represents. NamedRendition is located in the com.mapinfo.graphics package.

Named renditions have the same benefits as any type of named resource in MapXtreme Java:

- The resource is known by its name and not by its properties.
- The resource is located in one spot but can be referenced from many locations.
- To change the look or behavior of applications or data, the resource only need be changed, not each application or data file.

Named renditions, like all our other named resources, use the Java Naming and Directory Interface (JNDI) application programming Interface (API). The NamedRendition object handles all of the interaction with JNDI transparently.

Creating a Named Rendition

Creating a named rendition is easy:

```java
NamedRendition namedRendition = new NamedRendition(providerUrl, name);
```

or

```java
NamedRendition namedRendition = new NamedRendition(jndiContext, name);
```

where:

- `providerUrl (String)` is an URL to the object that stores the named resources (the repository in JNDI terms)
- `name (String)` is a name to associate with the rendition
- `jndiContext (Context)` is a Context object previously created directly via the JNDI API

Named renditions are not resolved until they are accessed.

Named renditions will have their names persisted unless a property's value has been changed, in which case the named rendition will be persisted as just a rendition.

Named renditions implement the Rendition interface so they can be used anywhere any other rendition can be used.

MapXtreme Java ships with a collection of pre-defined pen, brush, and symbol styles that are also considered to be "named renditions" in that they have names, such as brush_008, pen_057. These are located in the mapxtreme47/resources/mistyles directory. These renditions can be used as the starting point for your named renditions, where you can customize them and save them with more descriptive names. The MapXtreme Java Manager’s Named Resources panel provides a GUI for managing these styles. See Chapter 5: Managing MapXtreme Java.

Storing Named Renditions with NamedResource

Any rendition can be saved as a named rendition via the JNDI API. The JNDI Context (javax.naming.Context) uses two methods to save named resources into the named resources repository. They are:
• `bind(String name, Object obj)`
• `rebind(String name, Object obj)`

Use `bind()` to save a brand new resource (does not yet exist) into the repository. Use `rebind()` to update a pre-existing resource in the repository.

Obviously, the first thing you need is a JNDI Context. This can be either an `InitialContext (javax.naming.InitialContext)`, or a sub-context of the `InitialContext` (obtained by doing a `lookup()` via the `InitialContext`). For more on Contexts and InitialContexts, see the Javadocs for the JNDI API.

To create an initial Context, you need to know the provider URL, which is (most likely) the URL of `NamedResourceServlet`. Then invoke the `createInitialContext(providerURL)` factory method of the `NamedResourceContextFactory` class, as shown below:

```java
```

Now you just need to decide where, relative to the root of your named resources repository, you want to store your named rendition. Remember that `NamedResourceServlet` knows where the root of your named resources repository is— this is specified via a "resourcesdir" init-param to `NamedResourceServlet`.

Let's say you've already created a directory beneath the root of your repository called "my renditions", and you want to save a particular rendition in that sub-directory as "blue star". You would do so like this:

```java
// blueStar Rendition was previously initialized with the
// desired symbol properties
// create a named resource out of the blueStar Rendition
NamedResource resource = new NamedResource(blueStar);

// Now save it via the container we obtained above
initCtx.bind("my renditions/blue star", resource);
```

In the above example, "myrenditions/blue star" represents a compound name. When specifying compound names, each component of the name must be separated with a "/" (forward slash).

**Note:** You must always store named resources (renditions) in a sub-directory of the root of the repository. They should never be stored directly in the root.

### Retrieving a Named Rendition

To retrieve a named rendition from the named resources repository, use the `com.mapinfo.graphics.NamedRendition` class. To retrieve the named rendition "blue star", which is stored in a "my renditions" sub-directory of the root of your named resources repository, you simply create a new `NamedRendition`, giving the provider URL and resource name (relative to the root of the repository) like this:

```java
NamedRendition rendition = new NamedRendition("http://torpedo:8080/mapxtreme47/namedresource", "my renditions/blue star");
```

`NamedRendition` implements the `Rendition` interface, so it can be used anywhere a rendition is allowed.
Per-Feature Renditions

MapXtreme Java allows you to specify a table level rendition (returned by the FeatureSet's getRendition method) in the MAPINFO_MPCATALOG. The table level rendition is used as the base set of display properties for all features from that table.

An additional column can be specified within your spatial table that will be used as a per-feature rendition (returned by the Feature's getRendition method). This per-feature rendition is merged (overrides) with the table (FeatureSet) rendition to determine the set of properties used to display that feature. If no per-feature rendition exists, the table level rendition is used.

The rendition column is determined either as a parameter at layer creation time (either programatically or via the LayerControl Bean) or from the MAPINFO_MPCATALOG. This rendition column within the spatial table can either be Null, a MapBasic style string, or a MapXtreme Java rendition. For further information on the MPCATALOG, see Appendix C: MAPINFO.MAPINFO_MPCATALOG.

Code Sample: Per-feature Renditions

/* Programatically create the rendition column at the layer creation time. * Constructs the objects given the table name and information * describing the Rendition to use for each Feature */
Parameters:
* tableName - The file name of the table.
* renditionColumn - The column containing Rendition information
  for each Feature.
* renditionType - Object identifying the type of rendition data
  found in the rendition column.
*/

//create the TABTableDescHelper
TABTableDescHelper tabTDHelper = new TABTableDescHelper( new File("States_Rend.tab").getName(),
  "FeatureRendition", RenditionType.mapxtreme );

//create the TABDataProviderHelper
TABDataProviderHelper tabDPHelper = new TABDataProviderHelper("D:\maps");

//create the LocalDataProviderRef
LocalDataProviderRef localhelper = new LocalDataProviderRef(tabDPHelper);

//add it to the layer
m_map.getLayers().addLayer(localhelper,tabTDHelper, "tabLayer");
This chapter discusses thematic mapping within MapXtreme Java. Thematic mapping is a powerful way to analyze and visualize your data.

In this chapter:

- Thematic Mapping Overview ................................................................. 236
- General Objects for Themes ................................................................. 237
- OverrideTheme ................................................................. 238
- RangedTheme ................................................................. 238
- SelectionTheme ................................................................. 242
- IndividualValueTheme ................................................................. 243
- Theme Legends ................................................................. 244
- Cartographic Legends ................................................................. 245
- Working with Analysis Layers ................................................................. 246
Thematic Mapping Overview

Thematic mapping is the process of shading your map according to a particular theme. The theme is usually some piece or pieces of your data. You thematically shade a map using data from a layer. The most commonly known example of a thematic map is a weather map. When you see red, you know it is hot (high number of degrees); where you see blue, it is cold (low number of degrees).

Themes represent your data with shades of color, fill patterns, or symbols. There are many uses for thematic maps to display your data. You create different thematic maps by assigning these colors, patterns, or symbols to map objects according to specific values in your data.

Thematic mapping allows you to visualize and highlight trends in data that would be difficult to see through tabular data. Using the methods in the ThemeList and the Theme Interface, you can create and define your own thematic shading.

Theme Types

MapXtreme Java Edition offers four types of thematic maps:

- **OverrideTheme** – changes the rendition of an entire layer
- **RangedTheme** – groups data into ranges and shading based on range value
- **IndividualValueTheme** – shades groups of features which share a specific attribute value
- **SelectionTheme** – applies a rendition to a user-defined list of selected features

All layers that have thematics use the ThemeList collection, the Theme interface, and the Rendition object. Each specific theme type may use additional objects. They are explained in detail later in this chapter.

Themes can be created programmatically or, in the case of RangedTheme and IndividualValueThemes, through the AddTheme Wizard Bean. Themes can be displayed and controlled through the Layer Control.
Theme Legends

MapXtreme Java can generate legends for Ranged and IndividualValue themes. The information in the legend is tied directly to the theme. When the theme changes, the legend is updated.

AddTheme Wizard

MapXtreme Java provides a wizard to assist you when creating a RangedTheme or IndividualValueTheme. This wizard is a JavaBean that you can add to the MapToolbar for easy access. See the Working with Analysis Layers on page 246.

General Objects for Themes

The general objects that are available for themes include ThemeList, Theme, and Rendition. Each of these objects are described in the following sections.

ThemeList

The ThemeList collection is accessible from the Layer object and contains Theme objects. The ThemeList collection has methods used to perform operations such as adding, removing, and reordering Theme object(s) from the collection.

ThemeList allows several thematic shades to exist for one layer. It is important to keep the themes ordered correctly so that they display on the map. Themes are rendered in reverse order, like layers, from the bottom up. For example, two themes both changed the fill color of an object, one would obscure the other when the map is rendered. The theme at the top of the ThemeList takes precedence. This ordering is valid within the context of one feature, not within the context of an entire layer.

The ThemeList works well for certain situations. For example, if you wanted to shade world countries by population and also by literacy rate, the ThemeList allows this. You could first create a theme to shade by population using a different fill color for the regions and add that to the ThemeList. You could then create a theme to shade by literacy using a hatch pattern for the regions and add that to the ThemeList. ThemeList objects are used even if you only have one theme.

Theme

Theme is an interface. The methods of this interface allow you to retrieve the column on which the theme is based and the theme's descriptive name. If either the column or the name do not exist for a particular theme, a null value will be returned.
**Rendition**

The Rendition object is used throughout MapXtreme. It gives all of the style characteristics to features. When creating thematic maps, you will use Renditions to specify the appearance of the objects.

**OverrideTheme**

An OverrideTheme can be used to change the rendition of an entire layer. For example, if you wanted the world table to display with a red fill pattern and green line color, you would use an OverrideTheme.

To make an OverrideTheme for a layer, you only need to pass a Rendition object in its constructor.

```java
// Assume myLayer is a Layer object.
// Assume myRend is a Rendition object.

OverrideTheme myOTheme = new OverrideTheme(myRend,"My Theme");
myLayer.getThemeList.add(myOTheme);
```

**RangedTheme**

A RangedTheme is a more complex type of thematic map. When you create a ranged thematic map, all features are grouped into ranges and each assigned a rendition for its corresponding range. For example, you have a table of weather stations for your television viewing area, and you want to shade the locations according to their reported snowfall amounts.

With the Ranged map feature, MapXtreme groups the snowfall amounts into ranges. For instance, all weather stations that received between zero and five inches of snowfall in the past month are grouped into one range. Stations receiving between five and 10 inches are in a separate range. Sites that received between 10 and 15 inches are in a third range, while those stations reporting greater than 15 inch snowfall amounts are in a fourth range. Each range is referred to as a Bin. Each Bin has an upper-bound cut-off value.

All records in the layer are assigned to a range and then assigned a rendition based on that range. For instance the weather stations reporting 15 plus inches of snow are shaded red. The other ranges are shaded in lighter shades of red with the last range in gray (default colors). When you display the map, the colors make it readily apparent which locations received the most and least snow accumulation.

**Note:** If your client VM runs out of memory when doing using ranged themes on large tables, you may need to increase your max heap size to 64m. The command line syntax for this is `Java -Xmx64m <classname>`.

MapXtreme includes several utility objects that help create a RangedTheme.
ColumnStatistics

A ColumnStatistics object is returned when you use the fetchColumnStatistics method of the Layer object. The ColumnStatistics object contains information on the minimum, maximum, mean, and standard deviation of the values in a column. When you use the fetchColumnStatistics method, you pass the column on which you want the map shaded. You will not need to use the methods of the ColumnStatistics object directly to create a RangedTheme. Once the object has been retrieved, it is used in the Bucketer object to create a vector of breakpoints.

Bucketer

The Bucketer class is responsible for calculating the breakpoints for the Bins in a RangedTheme. Continuing the snowfall example above, you have four ranges that represent weather stations receiving 0-5 inches, 5-10 inches, 10-15 inches, and 15 inches or more of annual snowfall. Each of these ranges is a Bin.

The Bucketer calculates the breakpoints of these Bins using the computeDistribution methods. These methods all return a vector of breakpoints. Each value in the vector is an Attribute object. All of the computeDistribution methods pass the number of ranges and a ColumnStatistics object. You may also pass a Distribution Type and a RoundOff object.

Distribution Types

DISTRIBUTION_TYPE_EQUAL_COUNT has approximately the same number of records in each range. If you want the Bucketer to group 100 records into 4 ranges using equal count, it computes the ranges so that approximately 25 records fall into each range, depending on the rounding factor you set.

When using equal count (or any other range method), it’s important to watch out for any extreme data values that might affect your thematic map (in statistics, these values are referred to as outliers). For example, if you shade according to equal count with this database:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>5000</td>
</tr>
<tr>
<td>Penny</td>
<td>6000</td>
</tr>
<tr>
<td>Miguel</td>
<td>4500</td>
</tr>
<tr>
<td>Ben</td>
<td>100</td>
</tr>
<tr>
<td>Andrea</td>
<td>7000</td>
</tr>
<tr>
<td>Kyle</td>
<td>5500</td>
</tr>
<tr>
<td>Angela</td>
<td>7500</td>
</tr>
<tr>
<td>Mark</td>
<td>7000</td>
</tr>
</tbody>
</table>

Ben and Miguel are grouped in the same range (since they have the two lowest values). This may not produce the results you want since the value for Ben is so much lower than any of the other values.

DISTRIBUTION_TYPE_EQUAL_RANGES divides records across ranges of equal size. For example, you have a field in your table with data values ranging from 1 to 100. You want to create a thematic map with four equal size ranges. The Bucketer produces ranges 1–25, 26–50, 51–75, and 76–100.
Keep in mind that the Bucketer may create ranges with no data records, depending on the distribution of your data. For example, if you shade the following database according to Equal Ranges:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>100</td>
</tr>
<tr>
<td>Penny</td>
<td>6</td>
</tr>
<tr>
<td>Miguel</td>
<td>4</td>
</tr>
<tr>
<td>Linda</td>
<td>95</td>
</tr>
<tr>
<td>Ben</td>
<td>10</td>
</tr>
<tr>
<td>Andrea</td>
<td>90</td>
</tr>
<tr>
<td>Kyle</td>
<td>1</td>
</tr>
<tr>
<td>Angela</td>
<td>92</td>
</tr>
<tr>
<td>Elroy</td>
<td>89</td>
</tr>
<tr>
<td>Mark</td>
<td>10</td>
</tr>
</tbody>
</table>

The Bucketer creates four ranges (1–25, 26–50, 51–75, and 76–100). Notice, however, that only two of those ranges (1–25 and 76–100) actually contain records.

**DISTRIBUTION_TYPE_STANDARD_DEVIATION** breaks at the middle range of the mean of your values, and the ranges above and below the middle range are one standard deviation above or below the mean.

**DISTRIBUTION_TYPE_NATURAL_BREAK** creates ranges according to an algorithm that uses the average of each range to distribute the data more evenly across the ranges. It distributes the values so that the average of each range is as close as possible to each of the range values in that range. This ensures that the ranges are well-represented by their averages, and that data values within each of the ranges are fairly close together.

**RoundOff**

The RoundOff object is used to create clean breakpoints for ranges. For example, if you were shading a map with values that ranged from 101 to 397, the range breaks would be cleaner if the range was 100 to 400. RoundOff can round down the lower end of your range, and round up the higher end of your range.

**LinearRenditionSpreader**

An important part of creating a useful thematic map is to represent the values with renditions that gradually go from one value to another. The example in the introduction to RangedTheme discussed shading snowfall amounts. One end of the values was represented with red, and the next range was a lighter red, and so forth. The spread method of the LinearRenditionSpreader will return a vector of Renditions that spread the style from one given rendition to another for the number of elements given. The number of elements should match the number of ranges passed to the Bucketer object. For example, if you passed a rendition that was a red fill, a rendition that was a white fill, and the number five, the LinearRenditionSpreader would create a vector of five renditions with the red fill at the beginning, the white fill at the end, and an even spread of fill types in between.

**Creating a RangedTheme**

The following example demonstrates the code for creating a RangedTheme in which the table of U.S. states is shaded from yellow to red in five ranges, based on population.
FeatureLayer lyr=null;
Rendition yellow=new RenditionImpl();
Rendition red=new RenditionImpl();
lyr = (FeatureLayer)m_map.getLayers().getLayer("States.tab");
String colName = "Pop_1990";
ColumnStatistics colStats = lyr.fetchColumnStatistics(colName);

// Set number of breaks for data
int numBreaks=5;

// Compute the distribution of data with 5 breaks and
// Equal Ranges
List rBreaks = (List) Bucketer.computeDistribution
    (numBreaks,colStats,Bucketer.DISTRIBUTION_TYPE_EQUAL_RANGES);

// Set up a red and a yellow rendition and then
// spread the colors
yellow.setValue(Rendition.FILL, Color.yellow);
yellow.setValue(Rendition.STROKE_WIDTH, 2);
red.setValue(Rendition.FILL, Color.red);
red.setValue(Rendition.STROKE_WIDTH, 4);
List rends = (List) LinearRenditionSpreader.spread
    (numBreaks, yellow, red);

// Create Theme object
RangedTheme rTheme = new RangedTheme
    (colName, rBreaks, rends, "States by Pop_1990");

// Get ThemeList class object
ThemeList tList=lyr.getThemeList();

// Add theme to Layers themeList
tList.add(rTheme);

A Ranged theme can also be constructed through the AddTheme Wizard. Associated theme
legends can be created, as well.

Specifying Your Own Ranges
You can also create a RangedTheme by specifying your own custom ranges. To do so, follow this
procedure:

1. Create a rendition for each range color and add to a vector.
2. Create a List of attributes that represents the range upper bounds (breakpoints).
   
   Note: The attributes must be in numerical/alphabetical order.

3. Instantiate the RangedTheme Object, specifying as parameters: theme column,
   breakpoints List, renditions List, and descriptor name.
4. Add the theme to the ThemeList.
5. Render the map.

The code sample below creates three ranges:

// Set up the ranges
List rBreaks = new ArrayList();
rBreaks.add(new Attribute(1));
rBreaks.add(new Attribute(5));
rBreaks.add(new Attribute(7));

    // Set up the renditions
    List rends= new ArrayList();
    rends.add(redRendition);
    rends.add(grayRendition);
    rends.add(greenRendition);

    // Create a ranged theme object
    RangedTheme rTheme = new RangedTheme(
                        themeCol, rBreaks, rends, "States by Pop_1994");

    // Assign theme to layers as element 1
    myMap.getLayers().get(1).getThemeList().add(rTheme);

    // Draw the map
    // Create an ImageRequestComposer
    ImageRequestComposer imageRC =  ImageRequestComposer.create(
                        myMap, 256, Color.blue, "image/gif");

    // Create a MapXtremeImageRenderer
    MapXtremeImageRenderer renderer = new MapXtremeImageRenderer("http://stockholm:8080//mapxtreme47/servelet/mapxtreme");

    // Render the map
    Renderer.render(imageRC);

    // Render the map to the file
    Renderer.toFile("comp.gif");

### SelectionTheme

A SelectionTheme applies a rendition to all features referenced in a selection object. This type of theme is commonly used to store features returned by a search method on a layer using add(FeatureSet fs). SelectionTheme replaces the deprecated IDSelectionTheme.

For example, given a FeatureLayer object and X and Y coordinates, the code below demonstrates the selection of the layer's feature(s) at the specified location, and the creation of a SelectionTheme to display the selected features in red.

    // Assume layer as a FeatureLayer object
    ArrayList columns = new ArrayList();
    DoublePoint dp = new DoublePoint(x, y);
    FeatureSet fs = null;

    // Select the features at the specified location
    fs = layer.searchAtPoint(columns, dp, null);

    // Create a SelectionTheme
    SelectionTheme selTheme = new SelectionTheme("PointSelection");

    // Create a Selection object, and add the selected features
    Selection sel = new Selection();
    sel.add(fs);
IndividualValueTheme

This type of theme allows the thematic shading of features based on specific attribute values for a specified column. For example, use Individual Value theme to modify the renditions of region features in a "Coverage" table based on the values in the "Territories" column. Territories with type values of "SouthWest" could be red, while values of "SouthEast" could be shaded blue.

To create an IndividualValueTheme, follow this code sample:

```java
// get a reference to the layer we will be applying theme to
FeatureLayer lyr = (FeatureLayer)lyrs.addLayer(dpr, ttdh, "Territories");

// create a new theme object
IndividualValueTheme iValThm = new IndividualValueTheme("CoverageTerritory", "Sales Coverage Breakdown");

// create a rendition
Rendition rend = new RenditionImpl();

// assign color to rendition add attribute to theme with previously set rendition
rend.setValue(Rendition.FILL, Color.red);
iValThm.add(new Attribute("SouthWest"), rend);

// assign color to rendition add attribute to theme with previously set rendition
rend.setValue(Rendition.FILL, Color.blue);
iValThm.add(new Attribute("SouthEast"), rend);

// assign color to rendition add attribute to theme with previously set rendition
rend.setValue(Rendition.FILL, Color.green);
iValThm.add(new Attribute("Central"), rend);

// Add the theme to layers theme list
lyr.getThemeList().add(iValThm);

// Store column name and type in hashtable
Hashtable ht = new Hashtable();
ht.put("geomtype", IndividualValueThemeLegend.REGION_GEOMETRY);
ht.put("lableorder", IndividualValueThemeLegend.ORDER_ASCENDING);
```
An IndividualValueTheme can also be constructed through the AddTheme Wizard. Associated theme legends can be created, as well.

**Theme Legends**

You can create a legend for your Ranged or Individual Value theme based on the data. You have a lot of control over how the legend will look. You can change the title, fonts, and insets, as well as modify the descriptive text and colors.

Legends can be used on the client or server to export the legend as an image (e.g., GIF, JPEG). Theme legends are Swing components and can be used with MapXtreme’s JavaBeans.

To enable theme legends, the Theme interface has been enhanced to support a theme’s Legend object. Every Theme will have an associated ThemeLegend object, which will initially be null. Only RangedThemes and IndividualValueThemes have legends that contain meaningful information. (The other theme types, OverrideThemes and SelectionThemes, return empty legends.) To set the legend associated with a Theme, the Theme object’s `setLegend(ThemeLegend)` method must be invoked. A ThemeLegend can be obtained by creating one explicitly, or via the Theme’s `createDefaultLegend()` method.

The following example demonstrates the code for creating a RangedThemeLegend:

```java
// Create Theme object
// Assume rends as a List of Rendition
// Assume colName as a attributeName(String)
// Assume rBreaks as a List of breakPoints
RangedTheme rTheme = new RangedTheme(
    colName, rBreaks, rends, "States by Pop_1990");

// Create a default legend
RangedThemeLegend rThmLeg = rTheme.createDefaultLegend(null);

// OR, Create a theme legend instance using theme and setting hashtable
// Add theme settings to hashtable
Hashtable ht = new Hashtable();
ht.put("geomtype", RangedThemeLegend.REGION_GEOMETRY);
ht.put("lableorder", RangedThemeLegend.ORDER_ASCENDING);
RangedThemeLegend rThmLeg = new RangedThemeLegend(rTheme, ht);

// Set legend title
rThmLeg.setTitle("Ranged Theme legend");

// send legend to image file
rThmLeg.toFile("c:\temp\rangeLeg.gif", "image/gif");
```
The LegendContainerBean manages how legends are laid out and displayed. It can be dropped in along-side a VisualMapJ object and display any legends that are added to the VisualMapJ maps. VisualMapJ notifies the LegendContainerBean when a theme changes and the LegendContainerBean updates its display accordingly.

The LegendContainerBean is demonstrated in the sample applet, SimpleMap. See MapXtreme JavaBeans in Chapter 7 on page 92 for more information.

Cartographic Legends

While Theme Legends offer a "key" for the styles (renditions) contained in a Theme, Cartographic Legends offer a "key" for the renditions of any arbitrary FeatureSet or set of Renditions. You use the com.mapinfo.legend.CartographicLegend class to create a cartographic legend. You can either populate a CartographicLegend with an entire FeatureSet (addFeatureSet(FeatureSet fs, String descriptionCol) method) at once, or one Feature (Rendition) at a time (via the addRow(Rendition rend, String description) method).

The following code demonstrates how to create a cartographic legend for a the "landmarks" layer in a map.

```java
// Assume mapJ is already initialized
FeatureLayer landmarks =
    (FeatureLayer)mapJ.getLayers().getLayer("landmarks");

// create the legend with title "State Landmarks"
CartographicLegend legend = new CartographicLegend("State Landmarks");

/* Perform a searchAll() on the layer to get its entire FeatureSet - need to make sure we fetch the column that */
/* we want to label with in the legend ("Name")
ArrayList columns = new ArrayList();
columns.add("Name");
FeatureSet fs = landmarks.searchAll(columns, null);

/* add the FeatureSet to the legend, and specify that we want to label each feature's rendition with the value */
in the "Name" column
legend.addFeatureSet(fs, "Name");

// always dispose of the FeatureSet once you're done with it!
fs.dispose();
```
Working with Analysis Layers

MapXtreme Java provides you with the ability to include pie charts, side-by-side bar charts, and stacked bar charts. The bars in a side-by-side chart display vertically, while the bars in a stacked bar chart display horizontally. These charts are represented by the AnalysisLayer class. For more information refer to the HTML Javadocs.

Bar Chart Maps

Unlike one variable thematic maps such as ranges of values or graduated symbols, a bar chart map allows you to examine more than one variable per row at a time. A bar chart is built for every map object (feature) at the centroid of the object, enabling you to analyze the thematic variables in a particular chart by comparing the height of the bars. You can also examine the same variable across all the charts in your map.

For example, you have a table of U.S. state boundaries containing female and male population. Using bar charts, you can create a thematic map that displays a two–bar chart for each state: one bar representing female, and the other representing male population. You can compare the population differences of each state, or you can examine several states and compare one state’s population or population differences to another’s.

Pie Chart Maps

Mapping analysis using pie charts also enables you to examine more than one variable per row at a time. Like comparing the height of the bars in bar charts, in pie charts you compare the wedges in a single pie, or examine a particular pie wedge across all of the pies. Pie charts also enable you to compare parts of a whole.

Both pie and bar charts are particularly useful for analyzing demographic data. For example, you have a dataset of demographic information for the United States. Your dataset shows the populations of several major demographic groups. Using pie charts, you can show the population of each demographic group, and see what fraction of the pie it makes up in each pie. This enables you to see the distribution of demographic groups on a per state basis, or across the entire United States. You can also look at one demographic group and see how the population of the group varies in different states.

Use the following code sample as a guide for creating pie chart maps.

```java
// add a pie chart analysis layer based on the world layer

// get a reference to the previously added world layer
FeatureLayer world = (FeatureLayer)mapj.getLayers().get("world");

// build a List of columns in world to analyze
List cols = new ArrayList(3);
cols.add("Pop_0_14");
cols.add("Pop_15_64");
cols.add("Pop_65Plus");

// create our AnalysisTableDescHelper - use world's
// TableDescHelper
```
AnalysisTableDescHelper atdh = new AnalysisTableDescHelper(
    world.getTableDescHelper(), cols,
    AnalysisLayerType.PIE_CHART);

// insert the pie chart layer as the top-most layer in the
// map - use world's DataProviderRef
AnalysisLayer analysis =
    (AnalysisLayer)mapj.getLayers().insertLayer(
        world.getDataProviderRef(), atdh, 0, "World Population
        Analysis");

// modify some of the PieChartProperties
PieChartProperties pcp =
    PieChartProperties)analysis.getAnalysisProperties();

// define a list of custom fill styles for each wedge
List wedgeStyles = new ArrayList(3);
Rendition red = new RenditionImpl();
red.setValue(Rendition.FILL, Color.red);
wedgeStyles.add(red);
Rendition yellow = new RenditionImpl();
yellow.setValue(Rendition.FILL, Color.yellow);
wedgeStyles.add(yellow);
Rendition orange = new RenditionImpl();
orange.setValue(Rendition.FILL, Color.orange);
wedgeStyles.add(orange);
pcp.setSectionFillStyles(wedgeStyles);

// use a 2-pixel wide line style for the pie chart
// border and wedge dividers
Rendition lineStyle = new RenditionImpl();
lineStyle.setValue(Rendition.STROKE_WIDTH, 2);
pcp.setLineStyle(lineStyle);

// start drawing the first wedge at 0°, and work
// counter-clockwise from there
pcp.setStartAngle(0);
pcp.setDrawnClockwise(false);
Part C: Advanced Topics

Part C of this Developer Guide provides information for developer’s with advanced MapXtreme Java experience.

Topics:

- **Chapter 16: XML Protocol**
  This chapter covers the Mapinfo Enterprise XML Protocol, document type definitions (DTD), MapImageRequests, and MapVectorRequests.

- **Chapter 17: Customizing the AddLayer Wizard**
  This chapter provides instructions for adding new data sources with the Add Layer Wizard.

- **Chapter 18: Creating Your Own Data Provider**
  For advanced mapping application development, this section covers how to create your own data provider and make it available in the Add Layer Wizard.

- **Chapter 19: Uploading TAB Data to Remote Databases**
  This chapter describes the latest version of EasyLoader, a tool for uploading MapInfo .tab files to remote databases.

- **Chapter 20: Web Map Service**
  This chapter provides you with instructions for working with WMS.
This chapter presents the MapInfo Enterprise XML Protocol.

In this chapter:

- MapInfo Enterprise XML Protocol ................................................................. 250
- MapImageRequest ................................................................. 251
- MapVectorRequest ................................................................. 254
MapInfo Enterprise XML Protocol

MapXtreme Java continues to embrace diverse enterprise applications with its support for the MapInfo Enterprise XML Protocol. This protocol, based on the Extensible Markup Language (XML), defines how requests and responses for map information and data are handled by MapInfo enterprise products, such as MapXtreme Java, MapMarker J Server, and Routing J Server.

The MapInfo Enterprise XML Protocol now offers two supported components:

- **MapImageRequests**
- **MapVectorRequests**

A MapImageRequest is an XML document submitted to MapXtremeServlet to request the rendering of a multi-layered map. It is also used to request a base map and a list of point overlays for producing animated images. See [MapImageRequest on page 251](#) for more information.

The MapVectorRequest protocol allows both clients created with MapXtreme Java classes and non-MTXJ clients to create a MapVectorRequest as part of a layer search and receive the raw data for the layer in Geography Markup Language (GML) format. MapVectorRequests are further discussed on [MapVectorRequest on page 254](#).

GML is an XML encoding developed by the Open GIS Consortium (OGC) and is used to describe the properties and geometry of geographic features. Like XML that serves to separate Web content from presentation, GML describes the geometry of features separate from the graphic interpretation and visualization of those features.

**Document TypeDefinitions (DTD)**

Included in MapXtreme Java are the document type definitions (DTDs) that specify a set of rules for the structure of an XML request and response documents. The DTD defines the syntax of each valid element. They are located in the mxjdtds.jar in the MapXtreme-4.7.0/lib directory after installation. Be sure that the jar file is in your classpath.

HTML documentation that defines the elements in the DTDs is provided with MapXtreme Java in /docs/xmlprot/index.htm.

**Note:** Versions 4.0 and 4.5 of the DTDs are supported as well.

<table>
<thead>
<tr>
<th>Supported DTDs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MI/XML_Protocol_MapImageRequest_4_7.dtd</td>
<td>Defines requests and responses between client and server for map images.</td>
</tr>
<tr>
<td>MI/XML_Protocol_MapVectorRequest_4_7.dtd</td>
<td>Defines requests and responses between client and server for vector data in Geography Markup Language (GML) format.</td>
</tr>
</tbody>
</table>

The following DTDs are supported to the extent that their contents are used in a MapImageRequest or MapVectorRequest.
Product Independent DTD

- MI_XML_Protocol_CommonElements_1_0.dtd:
  Defines measurements for time, distance, angle, and velocity. Also defines an element for internalization. This is product independent in support of routing, geocoding, and mapping.

MapXtreme Java DTDs

- MI_XML_Protocol_MapCommonElements_4_7.dtd:
  Defines map elements common to all MapXtreme requests and responses. Describes new GMLFeature and GMLFeatureSetElements in support of MapVectorResponses.

- MI_XML_Protocol_MapFeatureStyle_4_7.dtd:
  Defines elements that describe styles for a feature or theme.

- MI_XML_Protocol_MapStyle_4_7.dtd:
  Defines style elements such as fill, stroke, font, and markers.

- MI_XML_Protocol_MapTextLabels_4_7.dtd:
  Defines the elements that describe labels, including base properties, style, themes, placement, and constraints (such as whether label duplication or overlap is allowed).

OGC-Specific DTDs

These DTDs are outlined in the specifications by the Open GIS Consortium (OGC). Exact specifications are referenced in the DTD.

- MI_XML_Protocol_OGC_CRS_1_1.dtd:
  OGC Coordinate Reference System definition.

- MI_XML_Protocol_OGC_GML_1_0.dtd:
  Defines simple GML (Geographic Markup Language) geometry elements for points, polygons, polylines, and collections, as defined by the OGC.

- MI_XML_Protocol_OGC_Identification_1_0.dtd:
  Defines OGC identification elements.

- MI_XML_Protocol_OGC_Units_1_1.dtd:
  Defines OGC units of measurement for time, distance, angle, pixel spacing.

MapImageRequest

A MapImageRequest is an XML document that requests the MapXtreme Java server to render an image of a multi-layered map. The MapImageRequest contains all the information needed by the server to access the data in each map layer, the definition of thematic shading to be applied to the geometry of each layer, the definition of labels to be generated for each layer, and the definition of the properties of the image to be returned.
The DTD file MI_XML_Protocol_MapImageRequest_4_7.dtd defines the syntax of the MapXtreme 4.5 MapImageRequest, MapImageResponse and MapImageFaultResponse. This document uses the external !ENTITY construct to "include" nine other DTD files whose element definitions are shared among other XML Protocol requests. All the DTD files are found in the file midtds.jar distributed with the MapXtreme Java 4.7 release.

A MapImageRequest can be sent to the MapXtreme Java server by any client application, written in any language, and running on any platform, as long as it follows the protocol. It is not necessary for the client to be implemented using the MapXtreme Java client classes (MapJ and associated classes).

Creating and sending MapImageRequests is important in applications that write their own client and directly handle requests/responses to/from the MapXtreme Java server.

Creating a MapImageRequest

Client applications that do not use any of the MapXtreme Java client classes (MapJ and associated classes) must create the MapImageRequest by their own means and in accordance with the DTD syntax and other policies that govern the content of the XML document as described in the documentation.

Client applications that wish to use the MapXtreme Java client classes can create a MapJ object and use the class ImageRequestComposer to create the MapImageRequest from the MapJ object. The document generated by the ImageRequestComposer requires no change before being sent to the server.

Sending an MapImageRequest to the Server

The MapImageRequest is sent to the server using an HTTP 1.1 POST request.

HTTP 1.1
Request-Line: POST
Content-Language: en_us
Content-Type: text/xml; charset=iso-8859-4
Accept-Language: en_us
Accept-Charset: iso-8859-4

MI_XMLProtocolRequest: MapImageRequest
MI_XMLProtocolVersion: MI_XML_Protocol_MapImageRequest_4_7
MI_XMLProtocolTransactionId: client_defined_id

<RequestEnvelope>
  <MapImageRequest>
    //request content
  </MapImageRequest>
</RequestEnvelope>

The protocol includes three custom HTTP headers. Two are required and the third is optional.

- **MI_XMLProtocolRequest** – REQUIRED.
  Identifies the protocol request type. In this case it is the string "MapImageRequest".
• **MI_XMLProtocolVersion** – REQUIRED.
  Identifies the protocol version of the request. More specifically, this is the version of the DTD that defines the syntax of the request. This version string is identified in the header comments of the DTD file itself. For example, the file MI_XML_Protocol_MapImageRequest_4_7.dtd contains the header comment shown below that identifies this version.

```xml
<!-- The MI_XMLProtocolVersion of this DTD is: MI_XML_Protocol_MapImageRequest_4_7 -->
```

• **MI_XMLProtocolTransactionId** – OPTIONAL.
  Defined by the client application. Used to associate the response with the request.

### Receiving a Response from a MapImageRequest

The response from a MapImageRequest will be either: an image, a base64 encoded image included in an XML MapImageResponse, or an XML MapImageFaultResponse.

**Success case returning an image:**

```plaintext
HTTP 1.1
Content-Type: image/gif
MI_XMLProtocolTransactionId: client_defined_id
```

In the success case returning an image, the HTTP response is not an XML document but simply the image.

The header MI_XMLProtocolTransactionId is optional and only included if present in the original request.

**Success case returning a base64 encoded image:**

```plaintext
HTTP 1.1
Content-Language: en_us
Content-Type: text/xml; charset=iso-8859-4
MI_XMLProtocolResponse: MapImageResponse
MI_XMLProtocolVersion: MI_XML_Protocol_MapImageRequest_4_7
MI_XMLProtocolTransactionId: client_defined_id
```
In the success case returning an encoded image, the HTTP response is an XML document containing the image represented as encoded text. The headers MI_XMLProtocolResponse and MI_XMLProtocolVersion are required and the header MI_XMLProtocolTransactionId is optional and only included if present in the original request.

All failure cases:

HTTP 1.1
Content-Language: en_us
Content-Type: text/xml; charset=iso-8859-4
MI_XMLProtocolResponse: MapImageFaultResponse
MI_XMLProtocolVersion: MI_XML_Protocol_MapImageRequest_4_7
MI_XMLProtocolTransactionId: client_defined_id

All failure cases return an XML MapImageFaultResponse. The headers MI_XMLProtocolResponse and MI_XMLProtocolVersion are required and the header MI_XMLProtocolTransactionId is optional and only included if present in the original request.

MapVectorRequest

A MapVectorRequest is an XML document that requests the MapXtreme Java server to perform a search on a data source and return the results as a set of features containing attribute, style, and geometry data. The term 'vector' refers to the fact that geometry data is represented as sets of points. The MapVectorRequest identifies the data source, the search conditions and the particular data items to be returned.

The DTD file MI_XML_Protocol_MapVectorRequest_4_7.dtd defines the syntax of the MapXtreme 4.7 MapVectorRequest, MapVectorResponse, and MapVectorFaultResponse. This document uses the external !ENTITY construct to "include" nine other DTD files whose element definitions are shared among other XML Protocol requests. All the DTD files are found in the file midtds.jar distributed with the MapXtreme Java 4.7 release.
A MapVectorRequest can be sent to the MapXtreme Java server by any client application whatsoever, written in any language and running on any platform, as long as it follows the protocol. It is not necessary for the client to be implemented using the MapXtreme Java client classes (MapJ and associated classes).

Creating and sending MapVectorRequests is of interest to applications that need to write their own client and directly handle requests/responses to/from the MapXtreme Java server.

Creating a MapVectorRequest

Client applications that do not use any of the MapXtreme Java client classes (MapJ and associated classes) must create the MapVectorRequest by their own means and in accordance with the DTD syntax and other policies that govern the content of the XML document as described in the documentation.

Client applications that wish to use the MapXtreme Java client classes can create a MapJ object and use the class LayerVectorRequestComposer to create the MapVectorRequest from a Layer object. The document generated by the LayerVectorRequestComposer requires no change before being sent to the server.

Sending a MapVectorRequest to the Server

The MapVectorRequest is sent to the server using an HTTP 1.1 POST request.

```
HTTP 1.1
Request-Line: POST
Content-Language: en_us
Content-Type: text/xml; charset=iso-8859-4
Accept-Language: en_us
Accept-Charset: iso-8859-4

MI_XMLProtocolRequest: MapVectorRequest
MI_XMLProtocolVersion: Mi_XML_Protocol_MapVectorRequest_4_7
MI_XMLProtocolTransactionId: client_defined_id

<RequestEnvelope>
  <MapVectorRequest>
    //request content
  </MapVectorRequest>
</RequestEnvelope>
```

The protocol includes three custom HTTP headers. Two are required and the third is optional.

- **MI_XMLProtocolRequest** – REQUIRED. Identifies the protocol request type. In this case it is the string "MapVectorRequest".
- **MI_XMLProtocolVersion** – REQUIRED. Identifies the protocol version of the request. More specifically, this is the version of the DTD that defines the syntax of the request. This version string is identified in the header comments of the DTD file itself. For example, the file MI_XML_Protocol_MapVectorRequest_4_7.dtd contains the header comment shown below that identifies this version.

```
<!-- *********************************************** -->
```
Receiving a Response from a MapVectorRequest

The response from a MapVectorRequest will be either: an XML MapVectorResponse, or an XML MapVectorFaultResponse.

Success case returning an image:

HTTP 1.1
Request-Line: POST
Content-Language: en_us
Content-Type: text/xml; charset=iso-8859-4
Accept-Language: en_us
Accept-Charset: iso-8859-4

MI_XMLProtocolRequest: MapVectorResponse
MI_XMLProtocolVersion: MI_XML_Protocol_MapVectorResponse_4_7
MI_XMLProtocolTransactionId: client_defined_id

<ResponseEnvelope>
  <MapVectorResponse>
    <MIFeatureSet>
      . . .
    </MIFeatureSet>
  </MapVectorResponse>
</ResponseEnvelope>

In the success case, the HTTP response is an XML document containing an MIFeatureSet. The headers MI_XMLProtocolRequest and MI_XMLProtocolVersion are required and the header MI_XMLProtocolTransactionId is optional and only included if present in the original request. It should be noted that XML response documents may be very large depending on the number and type of geometry objects present in the data source.

Failure cases return an XML MapVectorFaultResponse. The headers MI_XMLProtocolRequest and MI_XMLProtocolVersion are required. The header MI_XMLProtocolTransactionId is optional and only included if present in the original request.

HTTP 1.1
Content-Language: en_us
Content-Type: text/xml; charset=iso-8859-4
MI_XMLProtocolResponse: MapVectorFaultResponse
MI_XMLProtocolVersion: MI_XML_Protocol_MapVectorResponse_4_7
MI_XMLProtocolTransactionId: client_defined_id

<ResponseEnvelope>
  <MapVectorFaultResponse>
    //response content
</MapVectorFaultResponse>
</ResponseEnvelope>
This chapter provides you with instructions to customize the AddLayer Wizard.

In this chapter:

- Overview .............................................................. 259
- Changing the List of Data Sources ................................. 259
- Specifying Named Database Connections ....................... 261
- Location of addlayerwizard.properties ......................... 262
- Specifying Default Values ........................................ 263
- Saving Passwords ................................................... 264
Overview

The Add Layer Wizard is a guided tool that assists users with adding layers of maps. It displays when you click on the Add button in the Layer Control dialog. For an example of the Add Layer Wizard behavior, run the MapXtreme Java Manager and click the Layer Control button. From the Layer Control dialog, click the Add button to display the Add Layer Wizard.

The Add Layer Wizard initializes itself based on values for data sources stored in the addlayerwizard.properties file. This file is a text file, installed by the MapXtreme Java installer in the lib/client directory beneath where you installed MapXtreme Java. Most lines in the file contain a key and a value to associate with that key, separated by a '='. You can modify the values in this file to change the configuration of the AddLayer Wizard, including:

- The list of data sources which should appear in the initial list of data sources.
- The list of "Named Connections" which appear when the "Use Named Database Connection" data source is selected from the initial list (this data source does not appear by default).
- Default and "most-recently-used" values for the data source information steps of the wizard.
- Default and/or "most-recently-used" values for the initial ("Select a Data Source") and final ("Specify How the Data Will Be Accessed") steps of the wizard.
- Whether passwords are saved in the properties file (default is to not save them).

Changing the List of Data Sources

The addlayerwizard.properties file installed by MapXtreme Java is configured to offer the following list of data sources in the initial step of the wizard:

- MapInfo TAB file
- Oracle with Spatial Option
- SQL Server with SpatialWare
- IUS with SpatialWare
- Any with X/Y Column Data
- GeoTIFF Raster
- ESRI Shape
- Data Binding
- Named Layer
- Use Named Database Connection (only available if named connections are added by hand).

If you open the addlayerwizard.properties file in your favorite text file viewer, you will see the following line near the top of the file:

```
DataSource1=MapInfo TAB file
```

The "1" indicates that the "MapInfo TAB File" data source should be the first data source in the list. This is followed (in numerical order) by each of the other data sources in the list above.
Removing Data Sources

If you know that you (or the users of your applet/application) will never want to add a layer from one of the data sources in the above list, you can remove it from the list in the addlayerwizard.properties file. If you remove a data source in the middle of the list, however, you must then adjust the numbers of any data sources that follow it in the list. For example, you remove the following group of entries because you don't need to add layers from a SQL Server data source:

```
DataSource3=SQL Server with SpatialWare
DataSource3_DPHelper_Page=DataSource3_Page2
DataSource3_DPHelper_Class=com.mapinfo.dp.jdbc.sqlserversw.SQLServerSpwDataProviderHelper
DataSource3_Page_Count=3
DataSource3_Page1=com.mapinfo.beans.addlayer.PageSQLSRVp1
DataSource3_Page1_Description=Specify SQL Server Data Source Information
DataSource3_Page2=com.mapinfo.beans.addlayer.PageSQLSRVp2
DataSource3_Page2_Description=Specify SQL Server Table or Query Information
DataSource3_Page3=com.mapinfo.beans.addlayer.PageSQLSRVp3
DataSource3_Page3_Description=Specify Other SQL Server Table or Query Information
```

You must then "move up" all of the data sources that follow it (numerically) in the properties file. So you would change the DataSource4 group of entries to be DataSource3 entries, the DataSource5 group of entries to be DataSource4 entries, etc.

We recommend that rather than removing a group of entries from the properties file, that you comment out that group of entries, by placing a '#' character before each line in the group of entries. For example, to comment out the SQL Server entries, follow the entry below.

```
#DataSource3=SQL Server with SpatialWare
#DataSource3_DPHelper_Page=DataSource3_Page2
#DataSource3_DPHelper_Class=com.mapinfo.dp.jdbc.sqlserversw.SQLServerSpwDataProviderHelper
#DataSource3_Page_Count=3
#DataSource3_Page1=com.mapinfo.beans.addlayer.PageSQLSRVp1
#DataSource3_Page1_Description=Specify SQL Server Data Source Information
#DataSource3_Page2=com.mapinfo.beans.addlayer.PageSQLSRVp2
#DataSource3_Page2_Description=Specify SQL Server Table or Query Information
#DataSource3_Page3=com.mapinfo.beans.addlayer.PageSQLSRVp3
#DataSource3_Page3_Description=Specify Other SQL Server Table or Query Information
```

By commenting out the lines, you can add the data source back into the list by just removing all of the '#'s.

Also note that if you remove all but one of the data sources, the initial step of the AddLayerWizard, which displays the list of data sources, will not display, since there is only one data source from which to choose.
Re-ordering the Data Sources

The number associated with each group of data source entries determines its place in the list of data sources that will appear in the initial step of the Add Layer Wizard.

It is generally a good idea to have frequently used data sources towards the top of this list. For example, if you know that the users of your applet/application will most often need to add layers from an IUS data source to their map, you might choose to make that data source the first in the list. The default addlayerwizard.properties file indicates the IUS data source will appear as the fourth data source in the list. To make it the first you would just change its group of entries as follows:

```
DataSource1=IUS with SpatialWare
DataSource1_DPHelper_Page=DataSource5_Page2
DataSource1_DPHelper_Class=com.mapinfo.dp.jdbc.iussw.IusSwDataProviderHelper
DataSource1_Page_Count=3
DataSource1_Page1=com.mapinfo.beans.addlayer.PageIUSp1
DataSource1_Page1_Description=Specify IUS Data Source Information
DataSource1_Page2=com.mapinfo.beans.addlayer.PageIUSp2
DataSource1_Page2_Description=Specify IUS Table or Query Information
DataSource1_Page3=com.mapinfo.beans.addlayer.PageIUSp3
DataSource1_Page3_Description=Specify Other IUS Table or Query Information
```

Of course, by doing this you now have two DataSource1 entries in your properties file. To remedy this, you can increment the numbers associated with the data sources that should now follow your new number 1 entry.

Specifying Named Database Connections

If you've already looked at the contents of the addlayerwizard.properties file installed by MapXtreme Java, you may have noticed that it includes an eleventh data source, Use Named Database Connection. You may also have noticed that this data source does not appear in the list of data sources to choose from in the initial step of the wizard. This is because the addlayerwizard.properties file, as installed, does not have any named connection entries. These named connection entries are the same named connections that are managed by the Connections Manager panel (see Chapter 12: Accessing Remote Data) and stored in the miconnections.properties file. Named connections provide an easy way to refer to a particular database connection (or more accurately, a set of database connection properties).

To give the users of your applet/application the ability to add a layer based on one of these named connections without specifying any connection information, you add the named connection to the list of named connections in the addlayerwizard.properties file.

For example, if you had previously defined a named connection called "Enigma" that defines a connection to an Oracle Spatial data source, you would add a pair of entries to the addlayerwizard.properties file, to make the Add Layer Wizard aware of the named connection/resource, as illustrated by the bold text below.
# Named Connections
# Add any named Connections that appear in
# miconnections.properties.
# Each named resource entry should include the name of the
# resource as well
# as the data source which is appropriate for that named
# resource.
# Sample named resource:
# NamedConnection1=snoopy
# NamedConnection1_DataSource=DataSource2
# In this sample, DataSource2 represents the data source which
# is appropriate for snoopy.
NamedConnection1=Enigma
NamedConnection1_DataSource=DataSource2

Add the entries right after the Named Connection comment section in the file, so you can manage all of your named connections in one place. The comment section also describes the convention to use when defining a named resource. The only thing which may not be obvious is this line:

NamedConnection1_DataSource=DataSource2

This line indicates that DataSource2 (which by default is the Oracle Spatial data source) is the data source that corresponds to the named Enigma connection.

You can add as many of these named resources as you like, provided that they coincide with named connections in your miconnections.properties file. Once there is at least one named resource in this list, the "Named Resource" data source will appear in the list of data sources in the initial step of the wizard. See Chapter 12: Accessing Remote Data for more information on connections and named resources.

**Location of addlayerwizard.properties**

There are two copies of addlayerwizard.properties. One copy is in lib/client; which is used when you run stand-alone applications such as the MapXtreme Java Manager application.

Another copy of addlayerwizard.properties is stored in mjmappletsup.jar in webapps/mapxtreme47/client; this copy is used when you run the MapXtreme Java Manager as an applet. When run as an applet MapXtreme Java Manager uses slightly different security settings, so it needs its own copy of the properties file.

If you need to edit the addlayerwizard.properties file, make sure you edit the appropriate copy of the file for the application you plan to use.
Specifying Default Values

It is possible to seed the various Add Layer Wizard controls with default values that will appear pre-selected or specified when you encounter them. This is useful, for example, when it is likely that the users of your applet/application will not know the connection information for a remote data source, and you would like to have it filled in automatically for them.

To specify a default value for a control, add a line to addlayerwizard.properties that indicates the page and control to be set (these are combined to form the “key”) and the default value. For example, if you wanted to set the data source that would appear pre-selected on the initial DataSourcePage page of the wizard, add a line like this:

```
DataSourcePage_default_datasource=Oracle with Spatial Option
```

**Note:** The file, addlayerwizard.properties is installed with this line already included, and MapInfo TAB file is specified as the default data source.

The convention to use for specifying a default value in addlayerwizard.properties is:

```
<page>_default_<control name>=<default value>
```

**Note:** Do not include double quotes around the default values.

Setting Defaults for Initial and Final Page Controls

The initial page (screen) of the wizard allows you to select the data source from which you want to add a layer. The final page allows you to specify how the data will be accessed and to set some other miscellaneous values.

To specify default values for the controls in either of these pages, add lines to the properties file using the convention described above to specify the default values.

<table>
<thead>
<tr>
<th>Page</th>
<th>Control Name</th>
<th>Control Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSourcePage</td>
<td>datasource</td>
<td>The data source to pre-select from the &quot;Available Data Sources&quot; list (e.g., &quot;MapInfo TAB file&quot;)</td>
</tr>
<tr>
<td>DataSourcePage</td>
<td>useprevious</td>
<td>True to pre-select the &quot;Use previous settings as the defaults&quot; option or False to pre-select the &quot;Use default property values as the defaults&quot; option</td>
</tr>
<tr>
<td>DataAc-</td>
<td>servlet</td>
<td>True to pre-select the &quot;Access Data Via Remote MapXtremeServlet&quot; or False to pre-select the &quot;Access Data Locally&quot; option</td>
</tr>
<tr>
<td>cessPage</td>
<td>url</td>
<td>URL — the URL of the MapXtremeServlet</td>
</tr>
</tbody>
</table>
Setting Defaults for Other DataSource Page Controls

Each data source in the Add Layer Wizard has one or more "pages" associated with it, which are used to query the user for the information necessary to add a specific layer from that type of data source. If you look at the contents of addlayerwizard.properties, you will notice the various DataSourceX_PageY entries within it. The default values you specify will be exclusive to one of these pages in the wizard.

The Add Layer Wizard can retain the set of values that were most recently used (MRU) from session to session. If the Add Layer Wizard has write access to the addlayerwizard.properties file, it will write out various MRU value lines each time you click the Finish button. These values will be pre-set in the initial page of the wizard if you select the "Use previous settings as the defaults" option. This feature allows you to avoid repetitive typing of the same or similar settings.

Note: The Add Layer Wizard is not able to write to this file if deployed within an applet.

The easiest way to determine which default values can be set is to look at the MRU values which are saved when you click Finish to add a layer. Copy an existing "MRU" line and create another (default) version of it, substituting the word "default" for the word "mru".

Saving Passwords

By default, the Add Layer Wizard is configured so that it does not save most recently used passwords to the properties file. Passwords are required as part of the connection information when adding a layer contained in a remote database. For obvious security reasons, these are not saved in the properties file by default. If, however, you would like to have your MRU passwords saved in the properties file, you need only change one line in addlayerwizard.properties. Change the very first line in the properties file so that it looks like:

\[\text{Save\_Passwords=true}\]
Creating Your Own Data Provider

This chapter provides you with instructions for implementing your own data provider.

In this chapter:

- Introduction .................................................. 266
- Overview of Principal Data Provider Interfaces and Files ......................... 266
- Steps for Implementing a DataProvider for Vector Data .......................... 268
- Steps for Implementing a DataProvider for Raster Data .......................... 270
- Adding Custom DataProviders to the AddLayerWizard ......................... 270
Introduction

Before MapXtreme Java 4.0, you could only retrieve data using the data providers supplied for the set of data sources supported by MapXtreme. TheQueryBuilder capability added in release 3.0 provided a mechanism for customizing the behavior of the MapXtreme data providers, but still limited accessibility to the supported data sources. In release 4.x, all of the interfaces and classes needed to create a data provider are fully exposed, thus allowing users to access data of any kind in any manner.

Implementing a DataProvider can be a difficult undertaking requiring a major commitment of development and testing resources. Nearly all users will find the MapXtreme DataProviders completely sufficient. Only those users that require specific capabilities not found in MapXtreme and have the wherewithal to commit to this effort should attempt to use this capability.

What Is a DataProvider?

MapXtreme applications are based on a MapJ object composed of one or more Layer objects. The data in each Layer is fetched from some data source (e.g. a TAB file, RDBMS or raster file). The Layer itself is oblivious to the specific source, type and access mechanism of the data that populates it. Using objects provided to its constructor, the Layer object creates an instance of the DataProvider interface that will do whatever is necessary to return the data to the Layer. As an example, if the data source is an RDBMS and the request is to return all Features contained within a bounding rectangle, the DataProvider would need to do the following:

1. Create an SQL query string that contained the proper geometric search constraint and returned the necessary data.
2. Establish a connection to the RDBMS.
3. Execute the query.
4. Fetch the result set and transform the data returned into the format required by MapXtreme.

In all, the DataProvider interface requires the implementation of eight data queries using different constraints and two metadata queries. The data returned from a DataProvider is expected to be either Feature data (vector geometry and attributes) or image data (raster image). The implementation of the DataProvider and related interfaces for these two types of data differs somewhat. The table below identifies the required interfaces for each.

Overview of Principal Data Provider Interfaces and Files

<table>
<thead>
<tr>
<th>Interface</th>
<th>Package</th>
<th>Vector</th>
<th>Raster</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataProviderHelper</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>yes</td>
<td>Identifies metadata needed to access the data source.</td>
</tr>
<tr>
<td>Interface</td>
<td>Package</td>
<td>Vector</td>
<td>Raster</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
<td>--------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TableDescHelper</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>yes</td>
<td>Identifies the data to be accessed at the data source.</td>
</tr>
<tr>
<td>DataProvider</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>yes</td>
<td>Queries the data source and retrieves data and metadata.</td>
</tr>
<tr>
<td>UpdateableDataProvider</td>
<td>com.mapinfo.dp</td>
<td>no</td>
<td>no</td>
<td>Extends DataProvider to allow editing of data in the data source.</td>
</tr>
<tr>
<td>FeatureSet</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>yes</td>
<td>Represents the result of all data queries by the DataProvider; a collection of records implemented as Features.</td>
</tr>
<tr>
<td>Feature</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>yes</td>
<td>A single record in a FeatureSet.</td>
</tr>
<tr>
<td>Geometry</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>no</td>
<td>Identifies general geometric information of a Feature.</td>
</tr>
<tr>
<td>PointGeometry</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>no</td>
<td>Extends Geometry to be specific to point data.</td>
</tr>
<tr>
<td>VectorGeometry</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>no</td>
<td>Extends Geometry to be specific to line or region data.</td>
</tr>
<tr>
<td>PointList</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>no</td>
<td>Provides access to VectorGeometry coordinate data.</td>
</tr>
<tr>
<td>MIRaster</td>
<td>com.mapinfo.dp</td>
<td>no</td>
<td>yes</td>
<td>Draws raster data to Java Graphics object.</td>
</tr>
<tr>
<td>TableInfo</td>
<td>com.mapinfo.dp</td>
<td>yes</td>
<td>yes</td>
<td>Metadata about data identified by the TableDescHelper (&quot;Layer&quot; metadata) or data returned from a DataProvider query (&quot;FeatureSet&quot; metadata).</td>
</tr>
<tr>
<td>ColumnStatistics</td>
<td>com.mapinfo.dp</td>
<td>no</td>
<td>no</td>
<td>Statistical information about the data in a single column in a data source. Only used with RangedTheme or RangedLabelTheme.</td>
</tr>
</tbody>
</table>
Steps for Implementing a DataProvider for Vector Data

1. **Implement TableDescHelper**
   Implement the interface methods and add all others specific to the data in your data source. Interface methods that are not needed by your DataProvider implementation can be no-ops (will not do anything). The TableDescHelper is stored in the Layer and passed to all DataProvider interface methods as well as LayerXMLHandler.createQueryElement() interface method so it must contain the data required by these methods.

2. **Implement DataProviderHelper**
   This interface has no methods so you must add all that are needed for your data source. The DataProviderHelper is stored in the Layer and passed to LayerXMLHandler.createConnectionElement() interface method so it must contain the data required by this method.

3. **Implement TableInfo and DataProvider.getTableInfo()**
   This will establish the ability to connect to the data source and understand the data types it holds. An implementation of TableInfo will be needed for both DataProvider.getTableInfo() and FeatureSet.getTableInfo() methods.

4. **Implement DataProvider.queryInRect()**
   This will establish the ability to do a conditional geometric search of the data in the data source. MapXtreme also requires this method for rendering.

5. **Implement FeatureSet, Feature, Geometry, VectorGeometry, PointGeometry and PointList**
   These objects enable MapXtreme to use the results of the queryInRect() implementation. Requests to the DataProvider indicate the coordinate system required for the returned geometry data in the FeatureSet. Therefore, it may be necessary to transform the
coordinates of the geometry fetched from the data source before it is returned to MapXtreme (see com.mapinfo.coordsys.CoordSys and com.mapinfo.coordsys.CoordTransform).

6. **Implement LayerXMLHandler**
   This interface implements a mapping between Java classes and XML elements (JDOM Element objects). The XML must conform to the document definitions contained in the DTDs distributed with MapXtreme (see midtds40.jar).

   ***Requirement on <Url> element under <Connection> element***
   The string in the <Url> element must begin with the data provider protocol tag that you have chosen to identify your DataProvider delimited by a ':' (colon character). For example, the MapXtreme Oracle and Tab data providers use the tags "oraso" and "tab" and sample XML Url elements look like:
   
   `<Url>jdbc:oracle:thin:@your_server:1521:your_servicename</Url>`
   `<Url>tab:d:\Program Files\maps</Url>`

   If your implementation was MyDataProvider, and you choose "mydp" as your protocol tag, your Url might look something like:
   
   `<Url>mydp:any_url_string@blah.blah.blah.dotcom?hello</Url>`

   You may choose whatever tag you wish. This same tag is also identified in the configuration file layerxmlhandlers.xml (described below). MapXtreme looks for this tag at the beginning of the <Url> element and maps it to the appropriate LayerXMLHandler to instantiate to read the parts of the XML document generated by that handler. The remainder of the Url can be formatted in any manner whatever.

7. **Add entry to layerxmlhandlers.xml**
   This file provides MapXtreme with the association between your data provider protocol tag and the class names of your DataProviderHelper and LayerXMLHandler implementations. This file must reside in the classpath of the MapXtreme application. A template of this file is provided with the MapXtreme installation.

   Below is an example entry for MyDataProvider. The fully qualified class names are required. The element <DataProviderProtocol> is the tag described above that must be prepended to the connection URL.
   
   `<LayerXMLHandlerMapping>
     <DataProviderHelperClass>
       com.mycompany.myapp.MyDataProviderHelper
     </DataProviderHelperClass>
     <LayerXMLHandlerClass>
       com.mycompany.myapp.MyLayerXMLHandler
     </LayerXMLHandlerClass>
     <DataProviderProtocol>
       mydp
     </DataProviderProtocol>
   </LayerXMLHandlerMapping>

8. **Render a layer that uses your DataProvider**
   When this is working, the major hurdles of the DataProvider implementation are over.

9. **Implement the remaining DataProvider interface methods**
   Implement ColumnStatistics.

10. **Implement UpdateableDataProvider (optional)**
If you want to allow your data source to be edited, as opposed to being read only, change your data provider to implement the UpdateableDataProvider interface and implement the additional methods on that interface.

**Steps for Implementing a DataProvider for Raster Data**

1. Same as step 1 above for vector data.
2. Same as step 2 for vector data.
3. Same as step 3 for vector data. Since raster data may have very little metadata, TableInfo interface methods that are not needed by your DataProvider implementation may be "no-ops".
5. Same as step 6 for vector data.
6. Same as step 7 for vector data.
7. (Optional) Override com.mapinfo.dp.util.AbstractRasterDataProvider.getValue(TableDescHelper, DoublePoint). This method only needs to be overridden when implementing a data provider that reads grid rasters (e.g. MIG, GRD, etc).

**Note:** When extending the AbstractRasterDataProvider object the TableDescHelper passed into a method will be a RasterTableDescHelper or subclass thereof. Also, the TableInfo should be a RasterTableInfo or a subclass as well.

**Adding Custom DataProviders to the AddLayerWizard**

Support for any DataProvider can easily be added to the AddLayerWizard. The wizard initializes itself based on settings in the addlayerwizard.properties file. This file defines:

- The list of DataProviders from which a layer may be added.
- For each DataProvider, a list of one or more GUI "pages" that will prompt the user for the necessary information to add a layer for that DataProvider.
- Default values for some or all of the controls in the GUI pages (optional).
- Most recently used values for some or all of the controls in the GUI pages (optional).

**The AddLayerPage Class**

The first step is to determine the set of GUI "pages" that are necessary to query the necessary information to allow you to build the required TableDescHelper and DataProviderHelper that may be used to add a layer for that DataProvider. Each of these pages will extend the com.mapinfo.beans.addlayer.AddLayerPage class. Each AddLayerPage is essentially a javax.swing.JPanel that contains all of the GUI controls for that page.

Your page class should invoke the default (no-argument) AddLayerPage constructor, and then do the necessary work to add in the GUI controls to its layout.
There are a few methods on AddLayerPage that you may additionally implement. They are:

- boolean `validatePage(AddLayerWizard wizard)`: This method will be invoked when you click "Next" or "Finish" in the wizard. This is your chance to verify that you have provided all of the required information for the page. Return true if everything checks out and it is ok to proceed, or false if there is a problem. (The default implementation of this method, if not over-ridden, returns true.)

- void `finishInitializing(AddLayerWizard wizard)`: This method will be invoked immediately before this page becomes visible (upon clicking "Next" in the page before this page in the wizard). This is your chance to perform any last-minute initialization. (The default implementation of this method, if not over-ridden, does nothing.)

- void `setPageLinks(AddLayerWizard wizard)`: This method will be invoked once the page has been successfully "validated" (i.e., `validatePage()` returns true). This provides a mechanism for dynamically changing the "next" or "previous" pages in the wizard. (The default implementation of this method, if not over-ridden, does nothing.)

See the Javadocs for the `com.mapinfo.beans.addlayer.AddLayerPage` class for more details.

### The AddLayerWizard Class

The `com.mapinfo.beans.addlayer.AddLayerWizard` class has some methods which you will need to leverage in your page to do things like set the current `TableDescHelper` and `DataProviderHelper`.

**Note:** You are provided an `AddLayerWizard` in the three `AddLayerPage` call-back methods discussed above.

These methods are:

- `setDataProviderHelper(DataProviderHelper dpHelper)`: Sets the current `DataProviderHelper`.
- `setTableDescHelper(TableDescHelper tdHelper)`: Sets the current `TableDescHelper`.
- `setLayerNameRequired(boolean bNameRequired)`: Sets whether a layer name is required when adding a layer for the current `DataProvider`. If this method is invoked with a value of true, then the user will be forced to specify a name for the layer in the final step of the wizard.

You will most likely invoke these methods from the `validatePage(AddLayerWizard)` method of the `AddLayerPage(s)` that obtain the necessary information to build the `DataProviderHelper` and or `TableDescHelper`.

See the Javadocs for the `com.mapinfo.beans.addlayer.AddLayerWizard` class for more details.

### addlayerwizard.properties

Once you have implemented the necessary pages for your `DataProvider`, you will need to add an entry for your `DataProvider` to `addlayerwizard.properties`. To do this, you will add a new `DataSourceI` entry to the existing set of `DataSource` entries in `addlayerwizard.properties`. See `addlayerwizard.properties` for an example.
The following table describes the key-value pairs that are supported:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSourceI</td>
<td>The name for the data source that will appear in the list of data source names in the first step of the wizard.</td>
<td>Yes</td>
</tr>
<tr>
<td>DataSourceI_TDHelper_Class</td>
<td>The full class specification of the TableDescHelper class for this data source.</td>
<td>Yes</td>
</tr>
<tr>
<td>DataSourceI_Allowed_In_Applet</td>
<td>True or false denoting whether it is permissible to add a layer from this data source in an applet context (default to true if not specified).</td>
<td>No</td>
</tr>
<tr>
<td>DataSourceI_Page_Count</td>
<td>The number of pages that are necessary for this data source.</td>
<td>Yes</td>
</tr>
<tr>
<td>DataSourceI_PageJ</td>
<td>The Java class that represents this page.</td>
<td>Yes</td>
</tr>
<tr>
<td>DataSourceI_PageJ_Description</td>
<td>A description of what this page will prompt the user for.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

It is also possible to specify, via addlayerwizard.properties, default values for some or all of the controls in your GUI pages. The trick is to actually name the control(s) in your page via the setName(String) method of java.awt.Component. Then, use that same name in your key to define a default value for your control.

The wizard will also automatically write the most recently used (MRU) values for named controls to addlayerwizard.properties. Again, the trick is to simply give names to the control(s) for which you want to store MRU values.
Uploading TAB Data to Remote Databases

This chapter explains how to work with EasyLoader to upload TAB data to remote databases.

In this chapter:

- **Introduction to EasyLoader** .......................................................... 274
- **Running EasyLoader** ................................................................. 274
- **EasyLoader Options** ................................................................. 276
- **Loading Oracle Spatial Data** ...................................................... 278
- **Command Line Flags** ............................................................... 279
- **Mixing Command Line Flags with GUI** ........................................ 282
Introduction to EasyLoader

EasyLoader v 7.0 is a Windows-only utility available from MapInfo Corporation that allows you to upload MapInfo .tab files to a remote database, such as MapInfo Professional for Oracle, Informix, or SQL Server. EasyLoader is installed into the Tools directory during the MapInfo Professional installation process. It is also available as a free download from the MapInfo Corporation website.

The following databases are supported:

- Oracle Spatial
- Informix Dynamic Server
- SQL Server

For spatial database support, the DBMS needs to have the ability to handle spatial geometry either by itself, like Oracle Spatial, or via additional SpatialWare Datablade/Extensions, like MapInfo SpatialWare for IUS, and SQL Server. To support the spatial database, the IUS driver must have a version of 2.8 or greater. If an above DBMS does not have spatial object type support, the table can only be uploaded as latitude and longitude data (XY or XY with MapInfo Key (MICODE)). Only one server connection may be open at any one time.

Running EasyLoader

To upload MapInfo .TAB files using EasyLoader:

1. Run the EasyLoader from the MapInfo Professional Tools menu. If necessary, run Tool Manager to load and/or autoload EasyLoader onto the Tools menu. The main EasyLoader dialog displays.
2. Under Connection Information, click the appropriate button (ODBC or Oracle Spatial) to connect your database. Provide the necessary connection information (for example, data source name or User ID, password and server name). Click **OK** to return to the EasyLoader dialog box.

3. Click the **SOURCE TABLES** button to display a list of MapInfo tables from a single directory. When tables are selected for uploading, the names will display in the MapInfo Tables list box.

4. Choose the tables and select the appropriate Server Table processing task (Create new table, Append to existing table, Replace existing Table). These and additional options are explained in **EasyLoader Options on page 276**.

   **Note:** The Upload button is not available until table(s) are chosen.

5. To create local TAB files, provide a directory or browse to its location. By default, EasyLoader will not generate these files. The file naming convention for these tables is $yourServerTableName$ _srv.tab.

6. To set options for the upload process, click the **OPTIONS** button. The Options dialog box will display. See **EasyLoader Options on page 276** for an explanation of available options.
7. Click the **Upload** button to start the upload process. Close EasyLoader once the upload process is finished.

If you haven’t already created the Spatial Index during the upload process, do so now by either executing a create index statement or re-uploading the table, making sure this time to create the Spatial Index and replace the table (see steps 1–3).

### EasyLoader Options

#### Table Processing Options (Main Dialog)

This section describes the main dialog for table processing options.

**Create New Table**

A server table will be created with the name that you specify. If this option is chosen and a table with the same name already exists on the server, an error message will display, making you aware of this problem. You will need to use a different name or choose the option: Replace Existing Table, in order to upload the table.

**Replace Existing Table**

If a server table of the same name already exists, it will be dropped and a new table will be created to match the MapInfo table being uploaded when this option is selected.

**Append to Existing Table**

The MapInfo table will be appended to the server table if the server table exists and the structure of the two tables match. Otherwise, you will get an error and the table will not be uploaded. The tables must have the same table structure and be in the same projection for Oracle Spatial.
Append All to One Table

All MapInfo tables listed are uploaded to a single server table. The server table name is the one visible in the Server Table box. This feature is meant to be used to upload tables with the same structure and projection to one table. For example, instead of creating a new table for each street layer, check the Append All to One box, and only one table will be created. All of the tables will then be appended to this table.

Spatial Object Type

Choose from MapInfo SpatialWare or Oracle Spatial (depending on the type of connection), MICODE (XY with Key), and XY. The default for loading spatial data is MapInfo SpatialWare or Oracle Spatial.

MapInfo SpatialWare/Oracle Spatial

To select this option, the server must be Oracle or have MapInfo SpatialWare installed. Tables will be uploaded as spatial data.

XY and XY with MapInfo key (MICODE)

Use this option if the server is not Oracle Spatial or does not have MapInfo SpatialWare installed. The data will be stored as xy coordinates on the server. Therefore, the server table will be created as a point table. If the MapInfo table to be uploaded is not a point table and this option is chosen, the centroid will be abstracted and stored on the server table if you instruct it to do so. The difference between XY and MICODE is that the MICODE will provide a MapInfo key as the spatial index, hence its performance is superior to XY.

Table Processing Options (Options Dialog)

This section describes the Options dialog for table processing options.

Grant Public Access to Table

PUBLIC is granted all access to the server table.

Exclusive Use of Table

You can speed up load time on large tables significantly if you know that you will be the only one attempting to update/upload the table. If you do not select this option, the loader will verify after each commit that no other updates are made to the table while it is being uploaded. Selecting this option will prevent this check from occurring, which can speed the runtime of large tables significantly.

Create Unique Index

A unique index is created on the column `sw_member` for SpatialWare, `mi_prinx` for Oracle, or `mi_sql_rec_num` for XY and MICODE. These columns are sequential numbers that are generated by the loader. These columns are always created, but do not have to be indexed.
Create Spatial Index

For SpatialWare tables the index is created on the geometry column called sw_geometry. A spatial index is created and Update Statistics is executed after an r-tree index is created. You may also build your own spatial index to suit your specific needs. If you choose to do this, clear this check box to save time in loading.

For Oracle Spatial tables the spatial index is created on the geometry column (by default GEOLOC) and is called <table_name>_SX. After the index is built the ANALYZE table function is run on the index table.

Create IDENTITY Column (SQL Server only)

Select this check box if you wish to create the key column (sw_member) with IDENTITY properties. When this feature is in use, the unique key column values will be generated automatically by SQL Server. You do not need to fill in the key manually when a new row is inserted.

In version 7.0, the key column (sw_member) is created with IDENTITY property by default, which is the opposite behavior from the previous version (6.8). To upload the table without IDENTITY, select the Options button and uncheck Create IDENTITY Column.

Style Column

Specify whether per-row symbology is to be loaded with the data. The symbology is loaded as a text string in the column specified. The name of the column to be used may be specified in the edit text box. The name is initialized to the default column name, which is MI_STYLE.

Note: To load per-row symbology, the MAPINFO_MAPCATALOG for the database must contain the following columns: RENDITIONTYPE, RENDITIONCOLUMN, RENDITIONTABLE, and NUMBER_ROWS. See Appendix C: MAPINFO.MAPINFO_MAPCATALOG for more information.

Loading Oracle Spatial Data

The following section explains how to load Oracle spatial data.

Loading Lat/Long Tables into Oracle 9i

When loading tables that use the Latitude/Longitude coordinate system(Geodetic Data) to Oracle 9i, it’s important to verify that all geometry coordinates are between (-180,180) longitude and (-90, 90) latitude. Geodetic data coordinates beyond that range are not supported in Oracle Spatial and may cause problems. You may check your data using MapInfo Professional before loading, or by using the Oracle Spatial SDO_GEOM.VALIDATE_LAYER( ) function on the table after loading it to Oracle Spatial.

Note: EasyLoader assigns the default datum World Geodetic System 1984/WGS84 to the datumless Lat/Long coordinate system when loading Oracle 9i.
Validating Oracle Data

There are two functions that allow you to validate data on Oracle:

- `SDO_GEOM.VALIDATE_GEOMETRY( )`
- `SDO_GEOM.VALIDATE_LAYER( )`

These functions may result in validation errors simply due to the tolerance level set by EasyLoader, such as: “ORA-13356 adjacent points in a geometry are redundant”, or “ORA-13022 polygon crosses itself”. The tolerance within the `USER_SDO-GEOM_METADATA` may be adjusted downwards (by a factor of 10) and the validation re-run. However, if you adjust the tolerance, you must re-create the spatial indexes because they use the tolerance when they are created.

Command Line Flags

Easyloader supports the following command lines:

- `/A Append all tables to one
  This flag allows multiple tables to be uploaded to a single table as long as the table structures are the same.
  SYNTAX: /A`
- `/D Server Create Table
  Generates TAB files to access remote DBMSs when you provide the TAB file directory. By default, EasyLoader will not generate these files. The newly generated TAB file is the Server Table Name plus _srv.TAB.
  The directory must be a valid directory to upload a table. Empty is treated as valid.
  Command line option is /D Pathname
  SYNTAX: /D "C:\tabfiles"
- `/E Exclusive Use of Table
  You can speed up load time on large tables significantly if you know that you will be the only one attempting to update the table. Specifying this flag does not guarantee that the loader will obtain exclusive use; you must guarantee that to the loader.
  The loader checks on the current maximum value of the primary key column after each commit to ensure that it detects any other entries that may have been made by other processes. This flag will prevent that check from occurring, which can speed the runtime of large tables significantly.
  This flag may be placed within a shortcut, allowing the interactive use of the EasyLoader interface for other functions.
  SYNTAX: /E`
- `/F Log File name`
The log file is always produced, this flag lets the user specify a log file name and location. EasyLoader.log is the default name and it will be created in the user’s temp directory temp directory. If the log file flag is used, but no path has been specified, the resulting log will be generated in the same directory as the EasyLoader.exe file.

SYNTAX: The first example shows specifying just the name of the log file, which will be written to the directory that EasyLoader.exe is in; the second specifies the full path of the log file.

/F myLogFile.txt
/F c:\temp\myLogFile.txt

• /G Grant all
This flag will grant all rights to PUBLIC. This flag is turned OFF by default.
SYNTAX: /G

• /I DO NOT Create a Spatial Index
By default, this flag is turned off. When the flag is turned off, a spatial index will be created. When this flag is turned on, it prevents the creation of a spatial index on the table.

For Informix Dynamic Server a Spatial Index will be created then the 'update statistics medium' statement will be issued. See the /U flag, which controls the unique index.

For Oracle Spatial tables the spatial index is created on the geometry column and is called <table_name>_SX.

For SpatialWare tables the index is created on the column geometry column and is called hg<table_name>_ind.

SYNTAX: /I

• /K Create Automated Key Column for SQL Server Only
The key column (sw_member) can be created with the IDENTITY property as an option. When this feature is in use, the unique key column values will be generated automatically by SQL Server and the user does not need to fill the key manually when a new row is inserted.

In EasyLoader version 7.0, the key column(sw_member) is created with the IDENTITY property by default, which is opposite from the previous version (6.8). Therefore, by not specifying the K option in the command line will have the same action as specifying it as /K, i.e., create the key column with IDENTITY property. If you want to turn off the property, you need to provide a keyword “NO_IDENTITY” following K.

Example: /K NO_IDENTITY

SYNTAX: /K

• /L List of MapInfo tables
This flag allows the user to specify a text file that contains a list of tables to be uploaded. The format of each line is the same as the /T flag.

SYNTAX: /LListOTables.txt
• /M MICODE/XY ?
  This flag allows the user to specify the object type to be used if it is SpatialWare. If the /M flag is used, the user has to provide MICODE (for XY with MapInfo key) or XY (for XY) after /M. Any words other than MICODE or XY after /M are treated as an error. If the /M flag is not used, SpatialWare will be used as the default if the selected database has SpatialWare installed.

SYNTAX: /M micode
  /M xy

• /O Connection String
  This flag allows a connection string for Oracle Spatial to be passed to the program. See /S flag for ODBC connections.

SYNTAX: /O user_name/password@server_name

• /P ACR
  This flag specifies what to do with the table(s) being loaded to the server. You can specify one of three options: 'A', 'C', or 'R'. Choose 'A' to append an existing server table; choose 'C' to create a new server table; choose 'R' to replace an existing table. You cannot specify more than one option. The create table flag will fail if a table by the same name exists on the server.

SYNTAX: /P A

Note: This flag is only persisted for backward compatibility.

• /Q Quit
  This flag forces the loader to exit when done.

SYNTAX: /Q

• /R Replace the server table
  This flag will cause the server table to be dropped. The new table will be created and uploaded. By default a will be created on the server table.

SYNTAX: /R/

• /T MapInfo Table Name;Server Table Name;Range
  This flag allows a single table to be uploaded as a different name, and restrict the amount of records uploaded. The separator between the MapInfo table name, the server name and the range is the semi-colon character. The range is in the format: starting record<COMMA> ending record. The server table name and the range are optional.

SYNTAX: /T c:\data\us_cnty.tab;counties;1,500

• /U DO NOT Create a Unique Index
  This flag is turned off by default. When the flag is turned off, a unique index will be created on the table. When this flag is turned on, it prevents the creation of a unique index for the table. See the /I flag, which controls the spatial index. For Oracle Spatial tables the unique index is created on the column MI_PRINX and is called <table_name>_IDX, for SpatialWare tables the index is created on the column SW_MEMBER and is called <table_name>_i.

SYNTAX: /U

• /V Oracle Version
This flag allows you to load tables on an Oracle 8.1.6 server with the 8.1.5 format. This not generally recommended, but it is available if you have a special need to do this.

If you want to use this flag with the dialog, see the Mixing Command Line Flags with GUI section on the next page.

SYNTAX: /V

• /X Commit interval
  This flag allows the user to specify a commit interval. Easyloader will commit the inserted records when the commit interval is reached. The default commit interval is 1000. This same interval applies to the creation of the spatial index for Oracle Spatial. If the commit interval is set to 0 (zero), the whole range of records will be inserted as a single transaction, before a commit is issued.
  SYNTAX: /X 500

• /Y Symbology Column Name
  This flag allows you to specify that per row symbology is to be loaded with the data. You can also specify the name of the column to be used. If none is provided the default column name will be used, which is MI_STYLE.
  SYNTAX: /Y StyleColumnName

Mixing Command Line Flags with GUI

Command line flags may be mixed with the EasyLoader interface by using a windows shortcut. This makes it easier to set flags as your default while being able to override them from the UI.

There are some flags only available from the command line.

• Create a shortcut to EasyLoader.
• Right click on the shortcut and choose Properties.
• Under the “Shortcut” tab, within the “Target” edit box, add on the command line flags at the end of the line.

When EasyLoader is run from that shortcut, the specified flags are in effect.
MapXtreme Java provides you with the ability to use a Web Map Service (WMS). WMS in MapXtreme Java is 1.1.1 compliant in accordance with the 1.1.1 OpenGIS® Web Map Service Implementation Specification. This document can be found at [http://www.opengis.org/techno/implementation.htm](http://www.opengis.org/techno/implementation.htm).

**In this chapter:**

- WMS Server Overview ................................................................. 284
- GetCapabilities ........................................................................... 284
- GetMap ....................................................................................... 287
- GetFeatureInfo ........................................................................... 289
- WMS Raster Data Provider ......................................................... 291
- WMS Client ............................................................................... 292
WMS Server Overview

In accordance with Open GIS Consortium’s Web Map Service Implementation Specification, WMS in MapXtreme Java allows you to use the following operations:

- **GetCapabilities** – returns service-level metadata. This metadata is a description of the information content and acceptable request parameters of the service.
- **GetMap** – returns a map image with well-defined geospatial and dimensional parameters.
- **GetFeatureInfo** - returns information about features shown on a map. This operation is optional.

**Note:** All of these request types are case-sensitive.

GetCapabilities

GetCapabilities returns service-level metadata. This metadata is a description of the information content and acceptable request parameters of the service.

Request Parameters

The following table lists the possible request parameters.

<table>
<thead>
<tr>
<th>Request Parameter</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION=version</td>
<td>No</td>
<td>Request version. MapXtreme Java only supports version 1.1.1.</td>
</tr>
<tr>
<td>SERVICE=WMS</td>
<td>Yes</td>
<td>Service type. The SERVICE attribute will always be set to WMS because that is all MapXtreme Java currently supports.</td>
</tr>
<tr>
<td>REQUEST=GetCapabilities</td>
<td>Yes</td>
<td>Request name</td>
</tr>
</tbody>
</table>

Sample Request

The following is a sample GetCapabilities request:

```
http://hostname:portnumber/wmsserver111/servlet/wms?SERVICE=WMS &REQUEST=GetCapabilities
```

Sample Response

The following is a sample GetCapabilities response:

```
- <WMT_MS_Capabilities version="1.1.1">
  - <Service>
    <Name>OGC:WMS</Name>
    <Title>MapInfo Corporation Map Server</Title>
```
</Service>
- <Capability>
  - <Request>
    - <GetCapabilities>
      <Format>application/vnd.ogc.wms_xml</Format>
      - <DCPType>
        - <HTTP>
          - <Get>
          </Get>
          - <Post>
          </Post>
        </HTTP>
      </DCPType>
    </GetCapabilities>
    - <GetMap>
      <Format>image/gif</Format>
      <Format>image/jpeg</Format>
      <Format>image/jpg</Format>
      <Format>image/png</Format>
      - <DCPType>
        - <HTTP>
          - <Get>
          </Get>
          - <Post>
          </Post>
        </HTTP>
      </DCPType>
    </GetMap>
  </Request>
  - <GetFeatureInfo>
    <Format>text/xml</Format>
    - <DCPType>
      - <HTTP>
        - <Get>
        </Get>
        - <Post>
        </Post>
      </HTTP>
    </DCPType>
  </GetFeatureInfo>
</Capability>
<Request>
  - <Exception>
    <Format>application/vnd.ogc.se_xml</Format>
    <Format>application/vnd.ogc.se_inimage</Format>
    <Format>application/vnd.ogc.se_blank</Format>
  </Exception>
  - <Layer queryable="0" opaque="0" noSubsets="0">
    <Title>MapInfo Corporation Map Server</Title>
    <SRS>EPSG:4201</SRS>
    <SRS>EPSG:4205</SRS>
    [...]
    <LatLonBoundingBox maxx="180" miny="-90" minx="-180" maxy="90" />
    - <Style>
      <Name>mistyles/brushes/brush_001</Name>
      <Title>brush_001</Title>
    </Style>
    [...]
  </Layer>
  - <Layer queryable="1" opaque="0" noSubsets="0">
    <Name>Layers/World/Capitals</Name>
    <Title>World Capitals</Title>
    <SRS />
    <LatLonBoundingBox maxx="0.0038672423472374423" miny="0.001097654483307838" minx="-9.318052647195836E-4" maxy="0.0021815925889656857" />
  </Layer>
  - <Layer queryable="1" opaque="0" noSubsets="0">
    <Name>Layers/World/Countries</Name>
    <Title>World Countries</Title>
    <SRS />
    <LatLonBoundingBox maxx="0.00497406046898038" miny="7.405105032985079E-4" minx="-0.001959711056303122" maxy="0.002361994556506867" />
  </Layer>
  - <Layer queryable="1" opaque="0" noSubsets="0">
    <Name>Layers/World/Grid</Name>
    <Title>Grid</Title>
    <SRS />
    <LatLonBoundingBox maxx="0.005025098523972836" miny="7.405105008635173E-4" minx="-0.002134326671474454" maxy="0.0024215249446383038" />
  </Layer>
</Capability>
Storing Service Element Information

You can supply a service.xml file that stores the Service element information as defined by OGC. This service.xml file needs to conform to the OGC DTD and it needs to be located in the wmsserver111 context in the WEB-INF folder. The following is a sample service.xml file. The root element of the XML file needs to be <Service> or the XML file will be considered invalid and the getCapabilitiesRequest will fail.

```
- <Service>
  <Name>OGC:WMS</Name>
  <Title>Custom Service title</Title>
  <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
</Service>
```

Caching GetCapabilities Requests

MapXtreme Java creates a capabilities.xml file at initialization time, that is used as a cache when MapXtreme Java responds to GetCapabilities requests.

You can modify this file to provide further customized information about your data. There is a new init param controlling the "lifetime" of this file. Here is a sample from its web.xml:

```
<!--
   A value of -1 means that the capabilities.xml file never expires.
   A positive value is interpreted as the maximum allowed age of the
   file in hours. When the server is initialized, it will check the
   age of the file, and if it has become outdated, it will regenerate
   a new version of the file.
-->
<init-param>
  <param-name>maxAgeOfCapabilitiesXML</param-name>
  <param-value>-1</param-value>
</init-param>
```

GetMap

GetMap returns a map image with well-defined geospatial and dimensional parameters. When invoking GetMap a WMS Client can specify:

- Layers
- Styles
- A bounding box
- Coordinate reference system
- Output format
- Output size
- Background transparency and color
Request Parameters

The following table lists the possible request parameters.

<table>
<thead>
<tr>
<th>Request Parameter</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION=version</td>
<td>Yes</td>
<td>Request version. MapXtreme Java only supports version 1.1.1.</td>
</tr>
<tr>
<td>REQUEST=GetMap</td>
<td>Yes</td>
<td>Request name.</td>
</tr>
<tr>
<td>LAYERS=layer_list</td>
<td>Yes</td>
<td>Comma-separated list of one or more map layers.</td>
</tr>
<tr>
<td>STYLES=style_list</td>
<td>Yes</td>
<td>Comma-separated list of one rendering style per requested layer.</td>
</tr>
<tr>
<td>SRS=namespace:identifier</td>
<td>Yes</td>
<td>Spatial Reference System.</td>
</tr>
<tr>
<td>BBOX=minx,miny,maxx,maxy</td>
<td>Yes</td>
<td>Bounding box corners (lower left, upper right) in SRS units.</td>
</tr>
<tr>
<td>WIDTH=output_width</td>
<td>Yes</td>
<td>Width in pixels of map picture.</td>
</tr>
<tr>
<td>HEIGHT=output_height</td>
<td>Yes</td>
<td>Height in pixels of map picture.</td>
</tr>
<tr>
<td>FORMAT=output_format</td>
<td>Yes</td>
<td>Output format of map.</td>
</tr>
<tr>
<td>TRANSPARENT=TRUE</td>
<td>FALSE</td>
<td>No</td>
</tr>
<tr>
<td>BGCOLOR=color_value</td>
<td>No</td>
<td>Hexadecimal red-green-blue color value for the background color (default=0xFFFFFF).</td>
</tr>
<tr>
<td>EXCEPTIONS=exception_format</td>
<td>No</td>
<td>The format in which exceptions are to be reported by the WMS (default=SE_XML).</td>
</tr>
<tr>
<td>WFS=web_feature_service_URL</td>
<td>No</td>
<td>URL of Web Feature Service providing features to be symbolized using Styled Layer Descriptor.</td>
</tr>
</tbody>
</table>

Sample Request

The following is a sample GetMap request:

Sample Response

The following is a sample GetMap response:

![Map Image]

GetFeatureInfo

GetFeatureInfo returns information about features shown on a map. This operation is optional. If GetFeatureInfo is employed, it must be issued after the GetMap request.

Request Parameters

The following table lists the possible request parameters.

<table>
<thead>
<tr>
<th>Request Parameter</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERSION=version</td>
<td>Yes</td>
<td>Request version. MapXtreme Java only supports version 1.1.1.</td>
</tr>
<tr>
<td>REQUEST=GetFeatureInfo</td>
<td>Yes</td>
<td>Request name.</td>
</tr>
<tr>
<td>&lt;map_request_copy&gt;</td>
<td>Yes</td>
<td>Partial copy of the Map request parameters that generated the map for which information is desired.</td>
</tr>
<tr>
<td>QUERY_LAYERS=layer_list</td>
<td>Yes</td>
<td>Comma-separated list of one or more layers to be queried.</td>
</tr>
<tr>
<td>INFO_FORMAT=output_format</td>
<td>No</td>
<td>Return format of feature information (MIME type).</td>
</tr>
<tr>
<td>FEATURE_COUNT=number</td>
<td>No</td>
<td>Number of features about which to return information (default=1).</td>
</tr>
<tr>
<td>X=pixel_column</td>
<td>Yes</td>
<td>X coordinate in pixels of feature (measured from upper left corner=0)</td>
</tr>
<tr>
<td>Y=pixel_row</td>
<td>Yes</td>
<td>Y coordinate in pixels of feature (measured from upper left corner=0)</td>
</tr>
</tbody>
</table>
Sample Request

The following is a sample GetFeatureInfo request:

http://hostname:portnumber/wmsserver111/servlet/wms?VERSION=1.1.1
&REQUEST=GetFeatureInfo&SRS=epsg:4326&LAYERS=Layers/World/Countries
&STYLES=&BBOX=-180,-180,180,180&WIDTH=800&HEIGHT=600&QUERY_LAYERS=Layers/World/
Countries&X=54&Y=54

Sample Response

The following is a sample GetFeatureInfo response:

- <AnnFeatureSetList>
  - <AnnFeatureSet>
    <name>world</name>
    - <boundedBy>
      - <Box srsName="mapinfo:coordsys122">
        - <coord>
          <X>-1.6194966287191512E7</X>
          <Y>-8621185.324024437</Y>
        </coord>
        - <coord>
          <X>1.6789976633244906E7</X>
          <Y>8326222.646170927</Y>
        </coord>
      </Box>
    </boundedBy>
    - <CoordinateReferenceSystem>
      - <Identifier>
        <code>coordsys122</code>
      </Identifier>
    </CoordinateReferenceSystem>
    - <AttNameTypedTuple>
      <AttNameTyped Type="string">Country</AttNameTyped>
      <AttNameTyped Type="string">Capital</AttNameTyped>
      <AttNameTyped Type="decimal">Pop_1994</AttNameTyped>
      <AttNameTyped Type="decimal">Pop_Grw_Rt</AttNameTyped>
      <AttNameTyped Type="decimal">Pop_Male</AttNameTyped>
      <AttNameTyped Type="decimal">Pop_Fem</AttNameTyped>
      <AttNameTyped Type="decimal">Pop_0_14</AttNameTyped>
      <AttNameTyped Type="decimal">Pop_15_64</AttNameTyped>
      <AttNameTyped Type="decimal">Pop_65Plus</AttNameTyped>
      <AttNameTyped Type="decimal">Male_0_14</AttNameTyped>
      <AttNameTyped Type="decimal">Male_15_64</AttNameTyped>
      <AttNameTyped Type="decimal">Male_65Plus</AttNameTyped>
      <AttNameTyped Type="decimal">Fem_0_14</AttNameTyped>
      <AttNameTyped Type="decimal">Fem_15_64</AttNameTyped>
      <AttNameTyped Type="decimal">Fem_65Plus</AttNameTyped>
      <AttNameTyped Type="decimal">Pop_Urban</AttNameTyped>
    </AttNameTypedTuple>
  </AnnFeatureSet>
</AnnFeatureSetList>
<AttNameTyped Type="decimal">Pop_Rural</AttNameTyped>
<AttNameTyped Type="decimal">Pop_Urb_Male</AttNameTyped>
<AttNameTyped Type="decimal">Pop_Urb_Fem</AttNameTyped>
<AttNameTyped Type="decimal">Pop_Rur_Male</AttNameTyped>
<AttNameTyped Type="decimal">Pop_Rur_Fem</AttNameTyped>
<AttNameTyped Type="decimal">Arable_Pct</AttNameTyped>
<AttNameTyped Type="decimal">Literacy</AttNameTyped>
<AttNameTyped Type="decimal">Inflat_Rate</AttNameTyped>
<AttNameTyped Type="decimal">Unempl_Rate</AttNameTyped>
<AttNameTyped Type="decimal">Indust_Growth</AttNameTyped>
<AttNameTyped Type="string">Continent</AttNameTyped>
<AttNameTyped Type="int">MapInfo_ID</AttNameTyped>
"/></AttNameTypedTuple>
- <KeyColumns>
  <AttName>MapInfo_ID</AttName>
</KeyColumns>
</AnnFeatureSet>
</AnnFeatureSetList>

WMS Raster Data Provider

Mapxtreme Java provides the classes WMSRasterDataProviderHelper and WMSRasterTableDescHelper that allow you to use any WMS v1.1.1 Web service to provide a layer to MapXtreme Java to be rendered as part of a map. The WMS Raster Data Provider only allows you to render a layer. Any queries not related to rendering will either return an empty FeatureSet or meaningless results.

Note: The only queries that will return valid results will be queries needed for rendering. Valid results will be returned from queryInRectangle.

The class, WMSRasterDataProviderHelper describes the location of the WMS 1.1.1 server to use. Use in conjunction with a WMSRasterTableDescHelper to add a layer to a MapJ object based on a layer located on a WMS 1.1.1 server. MapXtreme Java currently does not allow remote access to raster layers. As this data provider provides raster information, it cannot be used for remote data access using the MapXtremeDataProviderRef.

The class, WMSRasterTableDescHelper is used in conjunction with a WMSRasterDataProviderHelper to describe a layer in a MapJ object using a WMS 1.1.1 data source.

For more information, refer to these classes in the Javadocs.
MapXtreme Java Edition provides the JSP WMS Custom Tags sample application to display sample functionality of a WMS client. You can access the JSP WMS Custom Tags by opening a browser and typing the URL to the sample, such as:

http://localhost:8080/samples47/htmlmap

WMS Client Interface Description

This section describes the various items in the graphical user interface of the WMS client.

Server List
The following items are available in the Server List portion of the interface.

Add – adds a new server to the list and make its layers available to the add layer controls. Invalid WMS URLs are ignored.

Remove – removes the server from the list and removes any of its layers from the map.

Map Tools
The following items are available in the Map Tools portion of the interface.

ZoomIn/ZoomOut/Recenter – basic map tools for working with the map.
**ZoomToAllLayers** – change zoom to encompass all the visible layers on the map

**Layer Control**
The following items are available in the Layer Control portion of the interface.

**Visibility (On/Off)** – changes the visibility of a layer of the map.

**Style Chooser** – allows you to choose a style from a list of supported styles for the layer.

**Move Layer (Up/Down)** – changes the ordering of the layers of the map.

**Remove Layer** – adds or removes a layer that has been supplied from any of the available servers.

**Add Layer** – adds a layer that has been supplied from any of the available servers.

**Refresh Map Data** – redraws the map taking into account any layer visibility changes.

**Using the WMS Client**
To use the WMS client:

1. From the Server List portion of the interface, click **Add**. The Add Server dialog box displays.
2. Enter the URL for the WMS 1.1.1 server that you want to use and click **OK**.
3. From the WMS Layer Control portion of the interface, click **Add**. The Add Layer dialog box displays.
4. Choose a server, layer and image type from the appropriate drop-down boxes and click **OK**.
5. Use the controls described in **WMS Client Interface Description on page 292** to work with the map.
Part D: Appendices

Part D of this Developer Guide provides a number of references to help you get the most out of working with MapXtreme Java.

Topics:

- **Appendix A: Custom JSP Tag Library**
  Defines all the JSP tags that ship with MapXtreme Java in the custom library.

- **Appendix B: Understanding MapBasic Style Strings**
  A summary of MapBasic style pens, brushes and symbols.

- **Appendix C: MAPINFO.MAPINFO_MAPCATALOG**
  A summary of the MAPCATALOG, the registry table for databases that stores metadata about geometry tables in the database.

- **Appendix D: System Properties**
  This appendix covers the system properties that are supported by MapXtreme Java.

- **Appendix E: System Logging**
  Information on logging a system messages on the server and client.

- **Appendix F: Custom Symbols**
  Thumbnails of a collection of symbols that ship with MapXtreme Java.
Custom JSP Tag Library

This appendix contains a listing of custom JSP tags and instructions on how to create your own custom tags and expose them in the Web Applications builder in MapXtreme Java Manager.

In this appendix:

- Custom JSP Tags ................................................................. 296
- Creating a Custom JSP Tag .................................................. 305
- Creating an Add TAB Layer Tag ........................................... 307
- Adding Custom Tags to the Web Application Wizard ............ 309
Custom JSP Tags

The following are custom JSP tags that are available for use in MapXtreme Java.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Parent</th>
<th>JSP Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapapp</td>
<td>Root tag for all of MapInfo’s JSP custom tags. Defines the application name and mime type for all images generated.</td>
<td>mapapp</td>
<td><code>&lt;mapinfo:mapapp [name=&quot;app-name&quot;] [mimeType=&quot;mime&quot;]&gt;…&lt;/mapinfo:mapapp&gt;</code></td>
</tr>
<tr>
<td>toolbar</td>
<td>Root tag for toolbar tools.</td>
<td>mapapp</td>
<td><code>&lt;mapinfo:toolbar&gt;…&lt;/mapinfo:toolbar&gt;</code></td>
</tr>
<tr>
<td>tool</td>
<td>Generic tool for the toolbar.</td>
<td>toolbar</td>
<td><code>&lt;mapinfo:tool page=&quot;relativeURL&quot; name=&quot;imgName&quot; img=&quot;relativeURL&quot; [options=&quot;JavaScript window Options&quot;] [alt=&quot;text&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;]</code></td>
</tr>
<tr>
<td>colors</td>
<td>Displays an HTML select box containing colors to choose. The value of each element is an int value of the color.</td>
<td>mapapp</td>
<td><code>&lt;mapinfo:colors name=&quot;form-Name&quot; [selected=&quot;selectedColor&quot;]</code></td>
</tr>
<tr>
<td>cancel</td>
<td>Displays an HTML button to close the current browser window. If a src is provided, the button will be printed as an image.</td>
<td>mapapp</td>
<td><code>&lt;mapinfo:cancelButton [value=&quot;buttonCaption&quot;] [src=&quot;relativeURL&quot;]</code></td>
</tr>
<tr>
<td>map</td>
<td>Displays the map.</td>
<td>mapapp</td>
<td><code>&lt;mapxtreme:map /&gt;</code></td>
</tr>
<tr>
<td>resizableMap</td>
<td>Displays a map that can be resizable. In order for a map to be resizable, this tag cannot be nested within a table. See MapTag in the Java-docs for more information.</td>
<td>mapapp</td>
<td>`&lt;mapxtreme:resizableMap resizable=&quot;true</td>
</tr>
<tr>
<td>printPreviewMap</td>
<td>Displays a map whose size is determined by tag attributes. This map can only be used for displaying. Panning, zooming, etc. will not work properly with this tag. See printpreview.jsp for an example.</td>
<td>mapapp</td>
<td><code>&lt;mapxtreme:printPreviewMap width=&quot;size&quot; height=&quot;size&quot; /&gt;</code></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Parent</td>
<td>JSP Syntax</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>layercontrol</td>
<td>Root tag for the layer control. Defines whether the layer control is on the same page as the map. See layercontroldialog.jsp for an example.</td>
<td>mapapp</td>
<td>&lt;mapxtreme:layercontrol [dialog=&quot;true</td>
</tr>
<tr>
<td>layerlist</td>
<td>Loops through each layer and ThemeList in MapJ. Defines what themes to print in the layer control.</td>
<td>layercontrol</td>
<td>&lt;mapxtreme:layerlist [showRangedThemes=&quot;true</td>
</tr>
<tr>
<td>visible</td>
<td>Prints an HTML checkbox to change the visibility of the current layer.</td>
<td>layerlist</td>
<td>&lt;mapxtreme:visible /&gt;</td>
</tr>
<tr>
<td>label</td>
<td>Prints an HTML checkbox to change the auto labeling of the current layer.</td>
<td>layerlist</td>
<td>&lt;mapxtreme:label /&gt;</td>
</tr>
<tr>
<td>select</td>
<td>Prints an HTML checkbox to change whether the current layer is selectable or not.</td>
<td>layerlist</td>
<td>&lt;mapxtreme:visible /&gt;</td>
</tr>
<tr>
<td>layerIndex</td>
<td>Prints the index of the current layer, or nothing if the current layer is a theme.</td>
<td>layerlist</td>
<td>&lt;mapxtreme:layerIndex /&gt;</td>
</tr>
<tr>
<td>layername</td>
<td>Prints the layer name or theme name.</td>
<td>layerlist</td>
<td>&lt;mapxtreme:layername [tableInfo=&quot;true</td>
</tr>
<tr>
<td>displayOptionsTool</td>
<td>Prints a tool to open a dialog to change the display options for the current layer. If value is included, it will print an HTML button, otherwise it will print an image.</td>
<td>layerlist</td>
<td>&lt;mapxtreme;displayOptionsTool [value=&quot;text&quot;] [layer=&quot;index&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Window Options&quot;] [page=&quot;relativeURL&quot;] &gt;</td>
</tr>
<tr>
<td>labelOptionsTool</td>
<td>Prints a tool to open a dialog to change the label options for the current layer. If value is included, it will print an HTML button, otherwise it will print an image.</td>
<td>layerlist</td>
<td>&lt;mapxtreme:labelOptionsTool [value=&quot;text&quot;] [layer=&quot;index&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Window Options&quot;] [page=&quot;relativeURL&quot;] &gt;</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Parent</td>
<td>JSP Syntax</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>fontOptionsTool</code></td>
<td>Prints a tool to open a dialog to change the font options for the current layer. If value is included, it will print an HTML button, otherwise it will print an image.</td>
<td>layerlist</td>
<td><code>&lt;mapxtreme:fontOptionsTool [value=&quot;text&quot;] [layer=&quot;index&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Window Options&quot;] [page=&quot;relativeURL&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>removeLayer</code></td>
<td>Prints a button to remove the current layer or theme.</td>
<td>layerlist</td>
<td><code>&lt;mapxtreme:removeLayer [src=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>layercontroltool</code></td>
<td>Prints a tool for the toolbar to open a layer control dialog.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:layercontroltool [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>zoomin</code></td>
<td>Prints a toolbar tool to zoom in on the map.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:zoomin [alt=&quot;text&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>zoomout</code></td>
<td>Prints a toolbar tool to zoom out on the map.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:zoomout [alt=&quot;text&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>recenter</code></td>
<td>Prints a toolbar tool to recenter the map.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:recenter [alt=&quot;text&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>infotool</code></td>
<td>Prints a toolbar to perform a search at point and display the information about each layer at that point.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:infotool [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>info</code></td>
<td>Tag that loops through each layer and sets the Feature-Set for each. See infodialog.jsp for an example.</td>
<td>mapapp</td>
<td><code>&lt;mapxtreme:info [searchMode=&quot;SearchMode constant&quot;] [...]</code></td>
</tr>
<tr>
<td><code>selectiontool</code></td>
<td>Prints a toolbar tool to select objects on the map.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:selectiontool [searchMode=&quot;SearchMode constant&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>unselecttool</code></td>
<td>Prints a toolbar tool to unselect all object on the map.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:unselecttool [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>selectioninfotool</code></td>
<td>Prints a toolbar tool to open a dialog to display attribute data for selected features.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:selectioninfotool [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] /&gt;</code></td>
</tr>
<tr>
<td><code>selectionInfo</code></td>
<td>Tag that loops through each layer and sets the Feature-Set for each layer that is selectable. See selectioninfodialog.jsp for an example.</td>
<td>mapapp</td>
<td><code>&lt;mapxtreme:selectionInfo&gt;...&lt;/mapxtreme:selectionInfo&gt;</code></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Parent</td>
<td>JSP Syntax</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| opentool      | Prints a toolbar tool to open a dialog to choose a MDF or geoset to open.   | toolbar  | `<mapxtreme:opentool [page="relativeURL"] [alt="text"] [img="relativeURL"] [width="size"] [height="size"] [options="JavaScript Options"/>
| open          | Prints an HTML select box listing the MDFs and geosets is the given directory. See opendialog.jsp for an example. | mapapp   | `<mapxtreme:open [directory="path"]/>
| savetool      | Prints a toolbar tool to open a dialog to save an MDF.                      | toolbar  | `<mapxtreme:savetool [page="relativeURL"] [alt="text"] [img="relativeURL"] [width="size"] [height="size"] [options="JavaScript Options"/>
| save          | Prints an HTML text box to input the filename of the MDF to save. See savedialog.jsp for an example.             | mapapp   | `<mapxtreme:save [dialog="true|false"] [directory="path"] [size="textbox size"]/>
| displayOptions| Root tag for the display options tags. Defines whether this tag is in a dialog. See displayoptionsdialog.jsp for an example. | mapapp   | `<mapxtreme:displayOptions [layer="layerIndex"] [dialog="true|false"]>/
| displayZoomRange | Prints an HTML checkbox to change whether the current layer is displayed within a zoom range. | display-Options | `<mapxtreme:displayZoomRange />
| displayMinZoom | Prints an HTML text box to change the minimum zoom range for the current layer. | display-Options | `<mapxtreme:displayMinZoom [size="size"/>
| displayMaxZoom | Prints an HTML text box to change the maximum zoom range for the current layer. | display-Options | `<mapxtreme:displayMaxZoom [size="size"/>
| labelOptions  | Root tag for the label options tags. Defines whether this tag is in a dialog. See labeloptionsdialog.jsp for an example. | mapapp   | `<mapxtreme:labelOptions [layer="layerIndex"] [dialog="true|false"]>/
| labelZoomRange | Prints an HTML checkbox to change whether labels for the current layer are displayed within a zoom range. | labelOptions | `<mapxtreme:labelZoomRange />
| labelMinZoom  | Prints an HTML text box to change the minimum zoom range for labels.        | labelOptions | `<mapxtreme:labelMinZoom [size="size"/>

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Parent</th>
<th>JSP Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>labelMaxZoom</td>
<td>Prints an HTML text box to change the maximum zoom range for labels.</td>
<td>labelOptions</td>
<td><code>&lt;mapxtreme:labelMaxZoom [size=&quot;size&quot;] /&gt;</code></td>
</tr>
<tr>
<td>labelColumns</td>
<td>Prints a select box to choose which column should be used for labeling.</td>
<td>labelOptions</td>
<td><code>&lt;mapxtreme:labelColumns /&gt;</code></td>
</tr>
<tr>
<td>labelDuplicates</td>
<td>Prints a checkbox to change whether duplicate labels will be shown.</td>
<td>labelOptions</td>
<td><code>&lt;mapxtreme:labelDuplicates /&gt;</code></td>
</tr>
<tr>
<td>labelOverlapping</td>
<td>Prints a checkbox to change whether overlapping labels will be printed.</td>
<td>labelOptions</td>
<td><code>&lt;mapxtreme:labelOverlapping /&gt;</code></td>
</tr>
<tr>
<td>fontOptions</td>
<td>Root tag for the font options tags. Defines whether this tag is in a dialog. See fontoptionsdialog.jsp for an example.</td>
<td>mapapp</td>
<td>`&lt;mapxtreme:fontOptions [layer=&quot;layerIndex&quot;] [dialog=&quot;true</td>
</tr>
<tr>
<td>fontList</td>
<td>Prints a select box with all of the available fonts on the system.</td>
<td>fontOptions</td>
<td><code>&lt;mapxtreme:fontList /&gt;</code></td>
</tr>
<tr>
<td>fontSize</td>
<td>Prints an HTML text box to change the font size of labels.</td>
<td>fontOptions</td>
<td><code>&lt;mapxtreme:fontSize [size=&quot;size&quot;] /&gt;</code></td>
</tr>
<tr>
<td>fontColor</td>
<td>Prints a select box to choose the color of the labels.</td>
<td>fontOptions</td>
<td><code>&lt;mapxtreme:fontColor /&gt;</code></td>
</tr>
<tr>
<td>fontHaloColor</td>
<td>Prints a select box to choose the halo color.</td>
<td>fontOptions</td>
<td><code>&lt;mapxtreme:fontHaloColor /&gt;</code></td>
</tr>
<tr>
<td>fontBold</td>
<td>Prints a checkbox to change whether the labels are bold.</td>
<td>fontOptions</td>
<td><code>&lt;mapxtreme:fontBold /&gt;</code></td>
</tr>
<tr>
<td>fontItalic</td>
<td>Prints a checkbox to change whether the labels are italic.</td>
<td>fontOptions</td>
<td><code>&lt;mapxtreme:fontItalic /&gt;</code></td>
</tr>
<tr>
<td>fontUnderline</td>
<td>Prints a checkbox to change whether the labels are underlined.</td>
<td>fontOptions</td>
<td><code>&lt;mapxtreme:fontUnderline /&gt;</code></td>
</tr>
<tr>
<td>thematic</td>
<td>Root tag for the thematic wizard. Determines whether this tag is in a dialog. If layer is supplied, then the user will not be prompted to choose a layer. If layer and column are supplied, the user will only be able to change the ranged theme options. See thematicdialog.jsp for an example.</td>
<td>mapapp</td>
<td>`&lt;mapxtreme:thematic [dialog=&quot;true</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Parent</td>
<td>JSP Syntax</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>themelayer</td>
<td>If a layer has not been chosen, the contents of this tag will be evaluated</td>
<td>thematic</td>
<td><code>&lt;mapxtreme:themelayer /&gt;</code></td>
</tr>
<tr>
<td>themecolumn</td>
<td>If a column has not been chosen, the contents of this tag will be evaluated.</td>
<td>thematic</td>
<td><code>&lt;mapxtreme:themecolumn /&gt;</code></td>
</tr>
<tr>
<td>themeoptions</td>
<td>If a layer and column have been chosen and the column is a number, the contents of this tag will be evaluated.</td>
<td>thematic</td>
<td><code>&lt;mapxtreme:themeoptions /&gt;</code></td>
</tr>
<tr>
<td>themeStartColor</td>
<td>Prints a select box to choose the starting color.</td>
<td>themeoptions</td>
<td><code>&lt;mapxtreme:themeStartColor [selected=&quot;color&quot;] /&gt;</code></td>
</tr>
<tr>
<td>themeEndColor</td>
<td>Prints a select box to choose the ending color.</td>
<td>themeoptions</td>
<td><code>&lt;mapxtreme:themeEndColor [selected=&quot;color&quot;] /&gt;</code></td>
</tr>
<tr>
<td>themeDistributionType</td>
<td>Prints a select box to choose the distribution method for ranged themes.</td>
<td>themeoptions</td>
<td><code>&lt;mapxtreme:themeDistributionType /&gt;</code></td>
</tr>
<tr>
<td>themeBreaks</td>
<td>Prints a text box to choose how many break points there will be for a ranged theme.</td>
<td>themeoptions</td>
<td><code>&lt;mapxtreme:themeBreaks /&gt;</code></td>
</tr>
<tr>
<td>themeLayerList</td>
<td>Prints a select box to choose the layer to perform a theme on.</td>
<td>themelayer</td>
<td><code>&lt;mapxtreme:themeLayerList /&gt;</code></td>
</tr>
<tr>
<td>themeColumnList</td>
<td>Prints a select box to choose the column to perform a theme on.</td>
<td>themecolumn</td>
<td><code>&lt;mapxtreme:themeColumnList /&gt;</code></td>
</tr>
<tr>
<td>themetool</td>
<td>Prints a toolbar tool to open a thematic dialog.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:themetool [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Options&quot;] /&gt;</code></td>
</tr>
<tr>
<td>legendtool</td>
<td>Prints a toolbar tool to open a legend dialog.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:legendtool [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Options&quot;] /&gt;</code></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Parent</td>
<td>JSP Syntax</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>legendlist</td>
<td>Loops through each layer and theme and sets scripting variables for the legend element to print the legend. See legenddialog.jsp for an example.</td>
<td>mapapp</td>
<td>`&lt;mapxtreme:legendlist [showRangedThemes=&quot;true</td>
</tr>
<tr>
<td>legendelement</td>
<td>Prints the legend.</td>
<td>legendlist</td>
<td><code>&lt;mapxtreme:legendelement [titleFont=&quot;font&quot;] [titleFontSize=&quot;size&quot;] [font=&quot;font&quot;] [fontSize=&quot;size&quot;]&gt;</code></td>
</tr>
<tr>
<td>legend</td>
<td>Prints the legend for the given layer and theme. You can specify either the index or name for the layer and theme.</td>
<td>mapapp</td>
<td><code>&lt;mapxtreme:legend layer=&quot;layerIndex&quot; theme=&quot;themIndex&quot; layerName=&quot;layerName&quot; themeName=&quot;themeName&quot; [title=&quot;title&quot;] [titleFont=&quot;font&quot;] [titleFontSize=&quot;size&quot;] [font=&quot;font&quot;] [fontSize=&quot;size&quot;]]</code></td>
</tr>
<tr>
<td>printpreviewtool</td>
<td>Prints a toolbar tool to open a print preview dialog.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:printpreviewtool [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Options&quot;]&gt;</code></td>
</tr>
<tr>
<td>tableinfo</td>
<td>Loops through all of the records for the given layer. If layer is not specified, it will look for it in the incoming request. Rows specifies how many records to show at a time.</td>
<td>mapapp</td>
<td><code>&lt;mapxtreme:tableinfo [layer=&quot;layerIndex&quot;] [rows=&quot;numOfRows&quot;]&gt;</code>…&lt;/mapxtreme:tableinfo&gt;</td>
</tr>
<tr>
<td>rowNum</td>
<td>Prints the current row number.</td>
<td>tableinfo</td>
<td><code>&lt;mapxtreme:rowNum </code>/&gt;</td>
</tr>
<tr>
<td>tableColumns</td>
<td>Loops through the columns for the given layer. If layer is omitted, it will look for the layer in the tableinfo tag.</td>
<td>mapapp or tableinfo</td>
<td><code>&lt;mapxtreme:tableColumns [layer=&quot;layerIndex&quot;]&gt;</code>…&lt;/mapxtreme:tableColumns&gt;</td>
</tr>
<tr>
<td>columnValue</td>
<td>Prints the value for the current record and column.</td>
<td>tableColumns</td>
<td><code>&lt;mapxtreme:columnValue [layer=&quot;layerIndex&quot;] [column=&quot;columnIndex&quot;]&gt;</code>&gt;</td>
</tr>
<tr>
<td>columnName</td>
<td>Prints the name of the current column.</td>
<td>tableColumns</td>
<td><code>&lt;mapxtreme:columnName [layer=&quot;layerIndex&quot;] [column=&quot;columnIndex&quot;]&gt;</code>&gt;</td>
</tr>
<tr>
<td>nextTableInfo</td>
<td>Prints a button to go to the next group of records.</td>
<td>mapapp</td>
<td><code>&lt;mapxtreme:nextTableInfo </code>/&gt;</td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Parent</td>
<td>JSP Syntax</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>zoomNumeric</td>
<td>Root tag for the map display tags. Determines whether this tag is in a dialog. See mapdisplaydialog.jsp for an example.</td>
<td>mapapp</td>
<td>`&lt;mapxtreme:zoomNumeric [dialog=&quot;true</td>
</tr>
<tr>
<td>zoomRange</td>
<td>Prints the current zoom range of the map. If readOnly is set to false, this will be printed in a text box. Set round to a factor of 10 to determine how to round the value. Omit to not round the number.</td>
<td>zoomNumeric</td>
<td>`&lt;mapxtreme:zoomRange [size=&quot;size&quot;] [readOnly=&quot;true</td>
</tr>
<tr>
<td>distanceUnits</td>
<td>Prints the distance units for the current map.</td>
<td>zoomNumeric</td>
<td><code>&lt;mapxtreme:distanceUnits /&gt;</code></td>
</tr>
<tr>
<td>centerX</td>
<td>Prints the x coordinate of the center of the map. If readOnly is set to false, this will be printed in a text box. Set round to a factor of 10 to determine how to round the value. Omit to not round the number.</td>
<td>zoomNumeric</td>
<td>`&lt;mapxtreme:centerX [size=&quot;size&quot;] [readOnly=&quot;true</td>
</tr>
<tr>
<td>centerY</td>
<td>Prints the y coordinate of the center of the map. If readOnly is set to false, this will be printed in a text box. Set round to a factor of 10 to determine how to round the value. Omit to not round the number.</td>
<td>zoomNumeric</td>
<td>`&lt;mapxtreme:centerY [size=&quot;size&quot;] [readOnly=&quot;true</td>
</tr>
<tr>
<td>zoomnumerictoolbar</td>
<td>Prints a toolbar tool to open a zoom numeric dialog.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:zoomnumerictoolbar [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Options&quot;] /&gt;</code></td>
</tr>
<tr>
<td>viewEntirelayer</td>
<td>Root tag for view entire layer tags. Determines whether this tag is in a dialog. See viewentirelayerdialog.jsp for an example.</td>
<td>mapapp</td>
<td>`&lt;mapxtreme:viewEntireLayer [dialog=&quot;true</td>
</tr>
<tr>
<td>viewEntireLayerList</td>
<td>Prints a select box to choose the layer to zoom to.</td>
<td>viewEntireLayer</td>
<td><code>&lt;mapxtreme:viewEntireLayerList /&gt;</code></td>
</tr>
<tr>
<td>viewentirelayertoolbar</td>
<td>Prints a toolbar tool to open the view entire layer dialog.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:viewentirelayertoolbar [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Options&quot;] /&gt;</code></td>
</tr>
<tr>
<td>Element</td>
<td>Description</td>
<td>Parent</td>
<td>JSP Syntax</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>overviewmap</td>
<td>Prints an overview of the current map.</td>
<td>mapapp</td>
<td>`&lt;mapxtreme:overviewMap width=&quot;size&quot; height=&quot;size&quot; [dialog=&quot;true</td>
</tr>
<tr>
<td>projection</td>
<td>Root tag for projection tags. Determines whether this tag is in a dialog. See projectiondialog.jsp for an example.</td>
<td>mapapp</td>
<td>`&lt;mapxtreme:projection [dialog=&quot;true</td>
</tr>
<tr>
<td>projectionCategories</td>
<td>Prints a select box listing all of the projection categories.</td>
<td>projection</td>
<td><code>&lt;mapxtreme:projectionCategories [size=&quot;1&quot;] /&gt;</code></td>
</tr>
<tr>
<td>projectionMembers</td>
<td>Prints a select box listing all of the projections for the current category.</td>
<td>projection</td>
<td><code>&lt;mapxtreme:projectionMembers [size=&quot;5&quot;] /&gt;</code></td>
</tr>
<tr>
<td>changeProjection</td>
<td>Changes the projection to the selected projection.</td>
<td>projection</td>
<td><code>&lt;mapxtreme:changeProjection [value=&quot;text&quot;] [src=&quot;relativeURL&quot;] /&gt;</code></td>
</tr>
<tr>
<td>projectiontool</td>
<td>Prints a toolbar tool to open the projection dialog.</td>
<td>toolbar</td>
<td><code>&lt;mapxtreme:projectiontool [page=&quot;relativeURL&quot;] [alt=&quot;text&quot;] [img=&quot;relativeURL&quot;] [width=&quot;size&quot;] [height=&quot;size&quot;] [options=&quot;JavaScript Options&quot;] /&gt;</code></td>
</tr>
<tr>
<td>featureSet</td>
<td>Loops through all of the features of the given FeatureSet.</td>
<td>mapapp</td>
<td><code>&lt;mapxtreme:featureSet featureSet=&quot;FeatureSet&quot;&gt;...&lt;/mapxtreme:featureSet&gt;</code></td>
</tr>
<tr>
<td>feature</td>
<td>Loops through all of the attributes for the given Feature.</td>
<td>feature or mapapp</td>
<td><code>&lt;mapxtreme:feature [feature=&quot;Feature&quot;] [featureSet=&quot;FeatureSet&quot;] &gt;...&lt;mapxtreme:feature&gt;</code></td>
</tr>
<tr>
<td>featureName</td>
<td>Prints the name of the current attribute.</td>
<td>feature</td>
<td><code>&lt;mapxtreme:featureName /&gt;</code></td>
</tr>
<tr>
<td>featureValue</td>
<td>Prints the value of the current attribute.</td>
<td>feature</td>
<td><code>&lt;mapxtreme:featureValue /&gt;</code></td>
</tr>
<tr>
<td>svgmap</td>
<td>Renders an SVG document. (The SVGMapBean class must be in the list for the Requesthandler.)</td>
<td>mapapp</td>
<td><code>&lt;misvg:svgmap [width=&quot;size&quot;] [height=&quot;size&quot;] /&gt;</code></td>
</tr>
</tbody>
</table>
Creating a Custom JSP Tag

MapXtreme Java utilizes Java Server Pages technology to assist you in the rapid development and deployment of web applications. To that end, we are providing a library of custom JSP tags that you can insert into a .JSP file using a text editor or IDE.

These tags display as widgets in the MapXtreme Java Manager's Web Application Builder, where you can simply select the element you need and add it to a layout frame that will be saved as a .JSP file. At run time, the .JSP communicates with a servlet that carries out the business logic of the application. If changes to the display of the application are necessary, the widgets can easily be rearranged, added, or deleted in the Web Application Builder to create a new .JSP without affecting the content generation operations of the servlet.

These custom tags have been designed for use in an MVC (Model/View/Controller) JSP-servlet architecture. The resulting .JSP (View) contains forms, and the forms submit to a generic servlet (Controller). The controller servlet redirects to the appropriate Java bean (Model) that carries out the necessary business logic, such as creating themes, performing radius searches, etc. and will forward the request back to the .JSP file, which displays the updated map.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Parent</th>
<th>JSP Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>Removes the need to make a server request if a layer is loaded, but the layer is off.</td>
<td>layername</td>
<td>&lt;misvg:visible /&gt;</td>
</tr>
<tr>
<td>highlight</td>
<td>Calls the onmouseover event to change the color and add a drop shadow to the item.</td>
<td>layername</td>
<td>&lt;misvg:highlight /&gt;</td>
</tr>
</tbody>
</table>

View

There are several base classes provided depending on the desired functionality of the tag. The root class to all of these classes is MITag. This abstract class implements the javax.servlet.jsp.tagext.Tag interface defined in the JSP specification. It also provides utility methods to get a ServerProperties and MappingSession object as well as perform some common tasks.

Model

Model classes perform the desired business logic based upon the user's input from the custom tags. The focus of each model class should be fairly specific. For example, MapXtreme provides a bean class to zoom in (com.mapinfo.jsptags.mapxtreme.ZoomInToolBean) and yet another to zoom out (com.mapinfo.jsptags.mapxtreme.ZoomOutToolBean). Although it is possible to combine both into a single zoom bean, we choose not to in order to promote a good modular design.
To create a model class, simply implement the `com.mapinfo.jsptags.TagBean` interface. There are three methods to implement. First, `getParameterKey` should simply return a unique string representation of the class. This will allow the RequestHandler servlet to correctly determine the TagBean class to use for a given request. Secondly, `setServerProperties`. This method allows you to get a reference to the various server settings for the given applications. This method is guaranteed to be called prior to any client requests, however it may be called many times (for example, if the server settings have changed during the course of the application running). Finally, implement process. This method should perform the necessary business logic. A MappingSession, HttpServletRequest and HttpServletResponse are provided as calling parameters. The return value of the method will indicate how the RequestHandler servlet should proceed once the process method returns. Returning true will cause the RequestHandler servlet to perform any cleanup and return. A return value of false will cause the RequestHandler servlet to redirect the user to a JSP or HTML page.

Once complete, you must register your TagBean class with the RequestHandler servlet. Simply add your fully qualified class name to the "Beans" init parameter.

**Controller**

As a custom tag author, you are not required to write any servlets. The RequestHandler servlet will accept incoming requests, create an appropriate MappingSession object, pass control to a TagBean class to process and optionally redirect the client to a new view. In order to perform these tasks, the RequestHandler servlet will look for certain HTML parameters. These are often passed in as hidden form fields but can be added to the URL. The following table describes these parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appname</td>
<td>Yes</td>
<td>This is a unique name for the application. It can be overridden as an attribute to the mapapp tag.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The name is stored as a constant on the MappingSession class called APP_NAME_PARAMETER.</td>
</tr>
<tr>
<td>mapinfobean</td>
<td>Yes</td>
<td>The name of the TagBean class that should process the incoming request. The value should match the return value of the getParameterKey method for the desired class.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> The name is stored as a constant on the TagBean class called PARAMETER_KEY_NAME.</td>
</tr>
<tr>
<td>redirect</td>
<td>No</td>
<td>Sets the return JSP or HTML page overriding any default values. If redirect is not specified, then the user will be redirected to MappingSession.getDisplayURL().</td>
</tr>
<tr>
<td>debug</td>
<td>No</td>
<td>Prints debug information about the RequestHandler. This is provided for debugging purposes and shouldn't be used in applications.</td>
</tr>
</tbody>
</table>

There are utility methods to help construct the necessary syntax to pass the parameter values between the view and model. For TagBeans that redirect control themselves (return true from process), use one of the getEncodedURL methods on the RequestHandler servlet. Custom tags that extend one of the MITag classes have several methods to get the required parameters. Use
getRequiredFieldsAsHTML method for HTML forms. This method will return a string containing
hidden HTML form fields that contain the necessary parameters. For situations where you wish to
process a request from a URL (for example, an href tag), use the getRequiredFieldsAsQueryString
method.

Creating an Add TAB Layer Tag

This section describes the steps necessary for extending the JSP library with a custom tag. In this
example, an Add Tab Layer tag is created. This will allow the user to add a tab layer to MapJ. The
source code for this example is provided with the MapXtreme Java install.

For this feature, we wish to provide the user with a list of tab files to choose from to add to MapJ as
well as the new layer’s position. The JSP author will determine the directory. This will require three
tags.

View

First, let's create a tag that generates an HTML combo box. This class will extend MITag and
implement doStartTag(). This method will get a list of files in a given directory and print an <option>
tag for each file that has a "tab" file extension. In addition, it will implement a method called
setDirectory(String dir). This will allow us to define an attribute for the tag, allowing the JSP author
to determine what directory to use.

Next, we need a tag to allow the user enter a position to add the layer to MapJ. This will simply
create an HTML text box. We will also implement setPosition and setSize to allow the JSP author
to define the default position and the size of the text box as attributes.

Lastly, we’ll create a tag that will generate an HTML form tag as well as the required parameters.
The previous two tags will be contained within this one. The resulting JSP would look this:

```xml
<addtablayer:addTabLayer>
  <p>File: <addtablayer:tabFileChooser directory="c:/maps" />
  <br>Position: <addtablayer:tabPosition size="5" />
  <br><input type=submit value="Add Layer" />
</addtablayer:addTabLayer>
```

In order to use these tags in a JSP page, we need to create a tag library definition (TLD). The TLD
for these tags is provided in addtablayer_4_7.tld.

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
taglib
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <servername>addtablayer</servername>
  <uri></uri>
<info>Custom tags to add a tab layer.</info>
tag
  <name>addTabLayer</name>
```
<tagclass>
com.mapinfo.jsptags.mapxtreme.AddTabLayerTag
</tagclass>
<bodycontent>JSP</bodycontent>
	<tag>
		<name>tabFileChooser</name>
	<tagclass>
com.mapinfo.jsptags.mapxtreme.AddTabLayerFileChooserTag
</tagclass>
<bodycontent>JSP</bodycontent>
	<attribute>
		<name>directory</name>
	<required>true</required>
	<rtexprvalue>false</rtexprvalue>
	</attribute>
	</tag>
	<tag>
		<name>tabPosition</name>
	<tagclass>
com.mapinfo.jsptags.mapxtreme.AddTabLayerPositionTag
</tagclass>
<bodycontent>JSP</bodycontent>
	<attribute>
		<name:size</name>
	<required>false</required>
	<rtexprvalue>false</rtexprvalue>
	</attribute>
	<attribute>
		<name>position</name>
	<required>false</required>
	<rtexprvalue>false</rtexprvalue>
	</attribute>
	</tag>
	</taglib>

Within the JSP page, add the following taglib directives:

```jsp
<%@ taglib uri="/addtablayer" prefix="addtablayer" %>
<%@ taglib uri="/zoomtofeature" prefix="zoomtofeature" %>
```

**Model**

To process the user’s request, we need to get the file name and position and create the DataProviderHelper, TableDescHelper, and DataProviderRef to add it to the MapJ. We can get a reference to the application’s MapJ from the MappingSession. The source code is provided with the MapXtreme Java install.

**Controller**

The only required step for the controller is to register our TagBean class with the RequestHandler. Simply add the fully qualified class name (com.mapinfo.jsptags.mapxtreme.AddTabLayerBean) to the “Beans” init parameter list.
Adding Custom Tags to the Web Application Wizard

Once you have completed a custom tag, you might want to integrate your tag into the Web Application Wizard. That way, anyone who is building a JSP application through the Web Application Wizard will be able to use your tag by pointing and clicking.

To integrate your custom tag(s) into the Web Application Wizard:

1. Create a .jar file containing all the .class files of your custom tags. You can do this using the jar utility that is provided with the Java SDK. Some third-party software utilities, such as WinZip, may also be used to build jar files; however, you must be careful to select the appropriate options in the utility software, so that your class files’ package (path) is preserved inside the .jar file. (For example, a WinZip user may need to select the "Save Extra Folder Info" check box when adding .class files to the archive.)

2. In your servlet container installation, create a new directory inside this directory: webapps/mapxtreme47/client. For example, a Windows user running Tomcat might create this directory for deploying the Add Tab tags:

D:\mapxtreme\tomcat\webapps\mapxtreme47\client\addtab

3. Copy your .jar file and your addtablayer_4_0.tld file to the directory you just created.
4. Copy your .jar file to the lib directory of the mapxtreme47 context:

webapps/mapxtreme47/WEB-INF/lib

5. Shut down your servlet container.
6. Edit the file webapps/mapxtreme47/WEB-INF/manager.xml, and add a taglib block. This taglib block will contain one tag block for each custom tag that you want to appear in the Web Application Wizard.

You may want to make a backup copy of manager.xml, in case you make any mistakes while editing it.

The following excerpt shows a portion of the manager.xml file. This example assumes that the directory you created in step 2 is called "addtab":

```xml
<panel class="com.mapinfo.mjm.client.mapdefs.MapDefPanel"
    required-services="1" />
<panel class="com.mapinfo.mjm.client.appbuilder.AppBuilderPanel" />
<panel class="com.mapinfo.mjm.client.namedres.NamedResourcesPanel"
    required-services="9" />
<!-- 3rd-party JSP custom tag libraries -->
taglib uri="/client/addtab/addtabluer_4_0.tld">
tag name="addTabLayer" required_services="1"
    allowed_context="1" />
tag name="tabFileChooser" required_services="1"
    allowed_context="1" />
tag name="tabPosition" required_services="1"
    allowed_context="1" />
</taglib>
<!-- Mapping preferences specified in the MJM Client -->
<preferences maps_dir="D:/data/maps" mime_type="image/gif"
    is_remote="true" />
<!-- Recent maps loaded in MJM Client -->
<recent-maps>
The text you are adding to manager.xml contains references to your tag names (which are defined in your .tld file) and a reference to your .tld filename. If you rename your .tld file, or if you rename your tags, you will need to edit manager.xml to make the names match.

7. Save your edits to manager.xml, and restart your servlet container.
8. Run the MapXtreme Java Manager, and launch the Web Application Wizard. Each of the tags that you added to manager.xml will appear in the list of available tags.

Custom Tag Attributes
For the <taglib> entry, you may specify the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uri</td>
<td>Yes</td>
<td>The path to your .tld file, relative to the root of the mapxtreme47 context.</td>
</tr>
</tbody>
</table>

For each <tag> entry, you may specify the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Yes</td>
<td>The name of the custom tag (the same as it appears in the .tld file).</td>
</tr>
</tbody>
</table>
Adding Your Own Custom GUI Editor for Your Custom Tag

When you "Edit" the properties of a custom tag widget in the Web App Builder Wizard, you are generally just providing different values for the "attributes" that are exposed by that custom tag. In general, the default GUI editor provided by the Web App Builder Wizard should suffice for editing these attributes. If, however, your tag is more complex, and the default editor does not suffice, you will have to supply your own.

The first step is to create your own implementation of the com.mapinfo.mjm.util.CustomTagEditor class (which is an extension of javax.swing.JPanel). This panel will contain the GUI controls necessary to edit the properties of your custom tag. See the Javadocs of the CustomTagEditor class for information on the methods which you must implement to make this work.

Once you have your CustomTagEditor class, you simply specify that class as the editor attribute of your <tag> entry in manager.xml (see above).
Understanding MapBasic Style Strings

One of the ways that MapXtreme Java can read a per-feature rendition or per-feature label rendition is in the form of a MapBasic string. (The other way is as an XML document). This appendix describes how to create a MapBasic style string for five style types: pen, brush, symbol, font, and color. This appendix concludes with instructions on how to determine the MapBasic string for a style using MapInfo Professional's MapBasic window.

In this appendix:

- Pen Styles ................................................................. 313
- Brush Styles ............................................................. 314
- Symbol Styles ......................................................... 315
- Font Styles .............................................................. 321
- Colors ................................................................... 322
- Using MapInfo Professional to Determine MapBasic Styles ........................................... 323
Pen Styles

The Pen style, also known as a Line style, in MapBasic specifies the width, pattern, and color of a linear object, such as a line, polyline arc, or the border of a region. The MapBasic pen clause has the following syntax:

```mapbasic
PEN (width, pattern, color)
```

**Width** can be represented by either screen pixels or points. The values between 1 and 7 represent the width in screen pixels.

Values between 11 and 2,047 represent the number of tenths of a point (1/10 pt) if 10 is subtracted from the value. MapBasic provides functions for converting: PointsToPenWidth() and PenWidthToPoints().

0 is only valid when the pen pattern is 1 for invisible lines.

**Pattern** is an integer from 1 to 118; pattern number 1 is invisible. The pattern number corresponds to a pen number in the pen file. An illustration of the pen styles follows.

**Note:** Certain pen styles are not supported in MapXtreme Java at this time. They include the pens that have markers at one or both ends of the segment. Specifically they are numbers 59, 60, 61, 62, 78, 79, 80, 86, 87, 88, 94, 95, 96, 102, 103, 104.

**Color** is an integer, representing a 24-bit RGB color value. For more information on Color, see Colors on page 322.

An example of a MapBasic string for a black railroad track, 2 pixels wide, would be:

```mapbasic
Pen (2, 118, 0)
```

The following table lists the available line styles by default:
Brush Styles

The Brush style specifies the pattern, foreground color, and background color of a filled object, such as a circle or region. The Brush clause has the following syntax:

```
Brush (pattern, forecolor [, backcolor ])
```

**Pattern** is a number from 1 to 71. Pattern number 1 is “no fill,” and pattern number 2 is a solid fill. Pattern numbers 9–11 are reserved. The following table illustrates the available styles.
The `forecolor` and `backcolor` arguments are both integers, representing 24-bit RGB color values. For more information on Color, see Colors on page 322.

To specify a transparent fill style, use pattern number three or larger, and omit the backcolor argument. For example: `Brush(5, 255)`

### Symbol Styles

The Symbol clause specifies the appearance of a Point object. There are three different forms of the Symbol clause:

- **Symbol Clause – MapInfo 3.0 Compatible Syntax**
- **Symbol Clause – TrueType Font Syntax**
- **Symbol clause – Custom Image File Syntax**
Symbol Clause – MapInfo 3.0 Compatible Syntax

To specify a symbol style using “Old MapInfo Symbols” (the symbols that were used in earlier versions of MapInfo), use the following syntax:

```
SYMBOL (shape, color, size)
```

The `shape` argument is an integer value, 31 or larger; 31 represents a blank symbol (i.e. the object will not be visible). The standard set of symbols includes symbols 32 through 67, inclusive. You can add, edit, or delete symbols from this symbol set by using the Symbol Maker tool in MapInfo Professional.

The `color` argument is an integer representing a 24-bit RGB color value. For more information on Color, see Colors on page 322.

The `size` argument is an integer from 1 to 48, representing a point size.

The following table lists the default symbols provided with MapInfo 3.0 Compatible Symbol set:

<table>
<thead>
<tr>
<th>Shape</th>
<th>Color</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>41</td>
<td>51</td>
</tr>
<tr>
<td>32</td>
<td>42</td>
<td>52</td>
</tr>
<tr>
<td>33</td>
<td>43</td>
<td>53</td>
</tr>
<tr>
<td>34</td>
<td>44</td>
<td>54</td>
</tr>
<tr>
<td>35</td>
<td>45</td>
<td>55</td>
</tr>
<tr>
<td>36</td>
<td>46</td>
<td>56</td>
</tr>
<tr>
<td>37</td>
<td>47</td>
<td>57</td>
</tr>
<tr>
<td>38</td>
<td>48</td>
<td>58</td>
</tr>
<tr>
<td>39</td>
<td>49</td>
<td>59</td>
</tr>
<tr>
<td>48</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>50</td>
<td>60</td>
<td>61</td>
</tr>
<tr>
<td>61</td>
<td>50</td>
<td>62</td>
</tr>
<tr>
<td>62</td>
<td>51</td>
<td>63</td>
</tr>
<tr>
<td>63</td>
<td>52</td>
<td>64</td>
</tr>
<tr>
<td>64</td>
<td>53</td>
<td>65</td>
</tr>
<tr>
<td>65</td>
<td>54</td>
<td>66</td>
</tr>
<tr>
<td>66</td>
<td>55</td>
<td>67</td>
</tr>
<tr>
<td>67</td>
<td>56</td>
<td></td>
</tr>
</tbody>
</table>

© 2003 MapInfo Corporation. All rights reserved.
Symbol Clause – TrueType Font Syntax

To specify a symbol style based on a character from a TrueType font, use the following syntax:

\[
\text{SYMBOL (shape, color, size, fontname, fontstyle, rotation)}
\]

**Shape** is the ASCII value of the character in the font set. To determine the shape value for a TrueType symbol, view the font set in your operating system’s font viewer tool, such as the Unicode Character Map for Windows. Note the character code for your chosen symbol.

The **color** argument is an integer representing a 24-bit RGB color value. For more information on Color, see Colors on page 322.

The **size** argument is an integer from 1 to 48, representing a point size.

The **fontname** argument is a text string that identifies the name of a font (For example, “Wingdings”).

The **fontstyle** argument is an integer that controls settings such as **bold**. The following table lists the values you can use as fontstyle.

<table>
<thead>
<tr>
<th>fontstyle value</th>
<th>Effect on Symbol style</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Plain text</td>
</tr>
<tr>
<td>1</td>
<td>Bold text</td>
</tr>
<tr>
<td>16</td>
<td>Black border around symbol</td>
</tr>
<tr>
<td>32</td>
<td>Drop shadow</td>
</tr>
<tr>
<td>256</td>
<td>White border around symbol</td>
</tr>
</tbody>
</table>

To specify two or more style attributes, add the values from the left column. For example, to specify Bold and Drop Shadow, use 33.

The **rotation** argument is a floating-point number, representing a rotation angle, in degrees.
# MapXtreme Java TrueType Symbols

## Map Arrows font

sample ID: "mistyles/fonts/Map Arrows/33"

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="sample33.png" alt="Symbol 33-40" /></td>
<td>33 - 40</td>
</tr>
<tr>
<td><img src="sample41.png" alt="Symbol 41-42" /></td>
<td>41 - 42</td>
</tr>
</tbody>
</table>

## Map Cartographic font

sample ID: "mistyles/fonts/Map Cartographic/33"

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="sample33.png" alt="Symbol 33-40" /></td>
<td>33 - 40</td>
</tr>
<tr>
<td><img src="sample41.png" alt="Symbol 41-50" /></td>
<td>41 - 50</td>
</tr>
<tr>
<td><img src="sample51.png" alt="Symbol 51-60" /></td>
<td>51 - 60</td>
</tr>
<tr>
<td><img src="sample61.png" alt="Symbol 61-70" /></td>
<td>61 - 70</td>
</tr>
<tr>
<td><img src="sample71.png" alt="Symbol 71-80" /></td>
<td>71 - 80</td>
</tr>
<tr>
<td><img src="sample81.png" alt="Symbol 81-90" /></td>
<td>81 - 90</td>
</tr>
<tr>
<td><img src="sample91.png" alt="Symbol 91-100" /></td>
<td>91 - 100</td>
</tr>
<tr>
<td><img src="sample101.png" alt="Symbol 101-110" /></td>
<td>101 - 110</td>
</tr>
<tr>
<td><img src="sample111.png" alt="Symbol 111-113" /></td>
<td>111-113</td>
</tr>
</tbody>
</table>

## Map Miscellaneous font

sample ID: "mistyles/fonts/Map Miscellaneous/33"

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="sample33.png" alt="Symbol 33-40" /></td>
<td>33 - 40</td>
</tr>
<tr>
<td><img src="sample41.png" alt="Symbol 41-50" /></td>
<td>41 - 50</td>
</tr>
<tr>
<td><img src="sample51.png" alt="Symbol 51-60" /></td>
<td>51 - 60</td>
</tr>
<tr>
<td><img src="sample61.png" alt="Symbol 61-70" /></td>
<td>61 - 70</td>
</tr>
<tr>
<td><img src="sample71.png" alt="Symbol 71-80" /></td>
<td>71 - 80</td>
</tr>
<tr>
<td><img src="sample81.png" alt="Symbol 81-90" /></td>
<td>81 - 90</td>
</tr>
<tr>
<td><img src="sample91.png" alt="Symbol 91-100" /></td>
<td>91 - 100</td>
</tr>
<tr>
<td><img src="sample101.png" alt="Symbol 101-110" /></td>
<td>101 - 110</td>
</tr>
<tr>
<td><img src="sample111.png" alt="Symbol 111-120" /></td>
<td>111 - 120</td>
</tr>
</tbody>
</table>
Map Oil&Gas font
sample ID: "mistyles/fonts/Map Oil&Gas/33"

<table>
<thead>
<tr>
<th>Sample ID</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 - 40</td>
<td></td>
</tr>
<tr>
<td>41 - 50</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

Map Real Estate font
sample ID: "mistyles/fonts/Map Real Estate/33"

<table>
<thead>
<tr>
<th>Sample ID</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 - 40</td>
<td></td>
</tr>
<tr>
<td>41 - 50</td>
<td></td>
</tr>
<tr>
<td>51 - 60</td>
<td></td>
</tr>
<tr>
<td>61 - 70</td>
<td></td>
</tr>
<tr>
<td>71 - 80</td>
<td></td>
</tr>
<tr>
<td>81 - 85</td>
<td></td>
</tr>
</tbody>
</table>

Map Symbols
sample ID: "mistyles/fonts/Map Symbols/33"

<table>
<thead>
<tr>
<th>Sample ID</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 - 40</td>
<td></td>
</tr>
<tr>
<td>41 - 50</td>
<td></td>
</tr>
<tr>
<td>51 - 60</td>
<td></td>
</tr>
<tr>
<td>61 - 68</td>
<td></td>
</tr>
</tbody>
</table>

MapInfo Symbols
sample ID: "mistyles/fonts/MapInfo Symbols/33"

<table>
<thead>
<tr>
<th>Sample ID</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>33 - 40</td>
<td></td>
</tr>
<tr>
<td>41 - 50</td>
<td></td>
</tr>
<tr>
<td>51 - 60</td>
<td></td>
</tr>
</tbody>
</table>
Map Transportation font
sample ID: "mistyles/fonts/Map Transportation/33"

Map Weather font
sample ID: "mistyles/fonts/Map Weather/33"

Symbol clause – Custom Image File Syntax

To specify a symbol style based on a custom image file, such as a .gif or .bmp, use the following syntax:

SYMBOL (filename, color, size, customstyle)

The filename argument is a text string that identifies a bitmap file (e.g. “Arrow.BMP”) in the CustSymb directory.

The color argument is an integer representing a 24-bit RGB color value. For more information on Color, see Colors on page 322.

The size argument is not supported in MapXtreme Java. Custom images will display in their actual size.
The **customstyle** argument is an integer that controls whether color and background attributes are used. The following table lists the values you can use as customstyle:

<table>
<thead>
<tr>
<th>customstyle value</th>
<th>Effect on Symbol style</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Both the Show Background setting and the Apply Color setting are off; symbol appears in default state. White pixels in the bitmap appear transparent, allowing whatever is behind the symbol to show through.</td>
</tr>
<tr>
<td>1</td>
<td>The Show Background setting is on; white pixels in the image are opaque.</td>
</tr>
<tr>
<td>2</td>
<td>The Apply Color setting is on; non-white colors in the image are replaced with the Symbol’s color value.</td>
</tr>
<tr>
<td>3</td>
<td>Both Show Background and Apply Color settings are on.</td>
</tr>
</tbody>
</table>

For a summary of custom .gif symbols that ship with MapXtreme Java, see [Appendix D: System Properties](#).

**Font Styles**

The MapBasic Font clause specifies the appearance (typeface, color, etc.) of labels in MapXtreme Java. The Font clause has the following syntax:

```
FONT ("fontname", style, size, forecolor [, backcolor] )
```

*Fontname* in double quotation marks is the typeface to be displayed.

*Style* is the attribute applied to the typeface as shown in the following table.

*Size* must be 0 in a MapBasic string, because each label on a Map is attached to the map itself (thus the text size changes as you zoom in or out).

*Forecolor* is an integer representing a 24-bit RGB color. For more information on Color, see [Colors on page 322](#).

The **background** color is optional; if you include it, the area behind the label will be filled with the color you specify. For more information on Color, see [Colors on page 322](#).
To specify two or more style attributes, add the values from the left column. For example, to specify Bold and All Caps, use 513.

<table>
<thead>
<tr>
<th>style value</th>
<th>Effect on Font Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Plain</td>
</tr>
<tr>
<td>1</td>
<td>Bold</td>
</tr>
<tr>
<td>2</td>
<td>Italic</td>
</tr>
<tr>
<td>4</td>
<td>Underline</td>
</tr>
<tr>
<td>16</td>
<td>Outline (only supported on the Macintosh)</td>
</tr>
<tr>
<td>32</td>
<td>Shadow</td>
</tr>
<tr>
<td>256</td>
<td>Halo</td>
</tr>
<tr>
<td>512</td>
<td>All Caps</td>
</tr>
<tr>
<td>1024</td>
<td>Expanded</td>
</tr>
</tbody>
</table>

Colors

Colors are often defined in relative concentrations of red, green, and blue. Each color is a number from 0 to 255, inclusive; the RGB value of a color is calculated by the following formula:

\[(\text{red} \times 65536) + (\text{green} \times 256) + \text{blue}\]

These are some often used colors and their values:

- Red: 16711680
- Bright Green: 65280
- Dark Green: 32768
- Blue: 255
- Cyan: 65535
- Magenta: 16711935
- Yellow: 16776960
- Black: 0
- White: 16777215
- Dark Gray: 8421504
- Light Gray: 12632256
Using MapInfo Professional to Determine MapBasic Styles

If you have access to MapInfo Professional, you can determine the MapBasic string syntax for any supported pen, brush or symbol style for use in MapXtreme Java’s support of per-feature renditions and per-feature label renditions.

Use MapInfo Professional with the MapBasic window displayed. From Layer Control, set a style override for a region, line or point object. View the MapBasic syntax of the style in the MapBasic window.

For example, the illustration below shows the syntax necessary to display a house symbol from the MapInfo Real Estate TrueType font set in 18 point, red, with a white halo, drop shadow and no rotation.:

```mapbasic
set map redraw off
Set Map Layer 3 Display Global Global Symbol (35,16711680,18,"MapInfo Real Estate",268,0)
set map redraw on
```
This appendix describes the MAPINFO.MAPINFO_MAPCATALOG.

In this appendix:

- Overview .................................................. 325
- MapXtreme Java SQL Scripts .................................. 326
- MAPINFO.MAPINFO_MAPCATALOG Format .................. 327
Overview

The MAPINFO.MAPINFO_MAPCATALOG is a database table that stores metadata about geometry tables in the database. Using the tablename and ownername as the key, the MAPINFO.MAPINFO_MAPCATALOG identifies the geometry column, geometry type, coordinate system, minimum bounding rectangle, and table and feature level rendition information.

The MAPINFO.MAPINFO_MAPCATALOG is not specific to MapXtreme Java; it is used by several MapInfo products that access map data from databases.

If MapInfo TAB files are uploaded to the database using the EasyLoader utility, the MapCatalog is automatically updated for the table. EasyLoader is described in the previous appendix. If a table is created by some other means, the MapCatalog could be updated by hand or by a custom application.

While many MapInfo products require the MAPINFO.MAPINFO_MAPCATALOG to be present, MapXtreme does not require it if all the data that it would otherwise obtain from the MAPINFO.MAPINFO_MAPCATALOG is provided to it in the TableDescHelper constructors for table defined layers. Specifically, it needs to know the geometry column name, coordinate system, and rendition column information. The absence of any of this data will cause MapXtreme to query the MAPINFO.MAPINFO_MAPCATALOG and to throw an exception if the data is unavailable.
MapXtreme Java SQL Scripts

If your database does not have a MAPCATALOG and you do not have access to MapInfo Professional, we provide SQL scripts that create the MAPCATALOG. The following files are located in /MapXtreme-4.7.0/examples/server/scripts after installation:

- oracle_mapcat.sql – for Oracle
- informix_mapcat.sql – for Informix Universal Server

These scripts create a user named MAPINFO, then create a MAPCATALOG that contains three string columns: RENDITIONCOLUMN, RENDITIONTYPE and RENDITIONTABLE. These columns support MapXtreme Java’s ability to store rendition information on a per-feature basis.

RENDITIONCOLUMN contains the name of the column in the table that holds the style information for the feature, or null if there is no style column.

RENDITIONTYPE column contains an enumerated value to indicate how the RENDITIONCOLUMN is to be interpreted by MapXtreme Java. These types are supported:

- No Per Feature Column Present: 0
- MapBasic style rendition: 1
- MapXtreme Java XML style rendition: 2

The RENDITIONTABLE column is reserved for future use.
The MAPINFO.MAPINFO_MAPCATALOG has the following table structure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPATIALTYPE</td>
<td>FLOAT</td>
</tr>
<tr>
<td>TABLENAME</td>
<td>CHAR(32)</td>
</tr>
<tr>
<td>OWNERNAME</td>
<td>CHAR(32)</td>
</tr>
<tr>
<td>SPATIALCOLUMN</td>
<td>CHAR(32)</td>
</tr>
<tr>
<td>DB_X_LL</td>
<td>FLOAT</td>
</tr>
<tr>
<td>DB_Y_LL</td>
<td>FLOAT</td>
</tr>
<tr>
<td>DB_X_UR</td>
<td>FLOAT</td>
</tr>
<tr>
<td>DB_Y_UR</td>
<td>FLOAT</td>
</tr>
<tr>
<td>COORDINATESYSTEM</td>
<td>CHAR(254)</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>CHAR(254)</td>
</tr>
<tr>
<td>XCOLUMNNAME</td>
<td>CHAR(32)</td>
</tr>
<tr>
<td>YCOLUMNNAME</td>
<td>CHAR(32)</td>
</tr>
<tr>
<td>RENDITIONTYPE</td>
<td>INTEGER</td>
</tr>
<tr>
<td>RENDITIONCOLUMN</td>
<td>CHAR(32)</td>
</tr>
<tr>
<td>RENDITIONTABLE</td>
<td>CHAR(32)</td>
</tr>
<tr>
<td>NUMBER_ROWS</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>
System Properties

This appendix lists the system properties that are supported by MapXtreme Java.

Properties

The following tables list the properties associated with each topic.

### JDBC Connection Pooling

<table>
<thead>
<tr>
<th>Key</th>
<th>com.mapinfo.connpool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Specifies a Java class which provides custom JDBC connection pooling.</td>
</tr>
<tr>
<td>Value</td>
<td>The full class specification of a class that implements the com.mapinfo.dp.conn.ConnectionPool interface.</td>
</tr>
<tr>
<td>Default</td>
<td>If not specified, a default connection pool is provided by MapXtreme Java.</td>
</tr>
</tbody>
</table>

### Font Rendering

<table>
<thead>
<tr>
<th>Key</th>
<th>com.mapinfo.renderer.quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Specifies whether fonts are rendered using higher-quality (but more computationally expensive) methods.</td>
</tr>
<tr>
<td>Value</td>
<td>true for high-quality fonts, or false for normal quality</td>
</tr>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>

### Data Binding

<table>
<thead>
<tr>
<th>Key</th>
<th>com.mapinfo.max.where.items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Specifies the maximum number of WHERE conditions that may exist in a single data binding SQL query.</td>
</tr>
<tr>
<td>Value</td>
<td>A positive integer value. Note that Oracle Spatial requires a value less than 1000.</td>
</tr>
</tbody>
</table>
Data Binding

Key com.mapinfo.max.in.items
Description Specifies the maximum number of elements that may exist in the IN list of a WHERE clause in a single data binding SQL query.
Value A positive integer value.
Note: Oracle Spatial requires a value less than 30000.
Default If not specified, an unlimited number of elements are allowed.

Byte Buffering

Key com.mapinfo.mibytebuffer
Description Method to do all read and writes when using the TAB library implementation.
Value com.mapinfo.util.io.MemMappedByteBuffer – This will use java.nio.MappedByteBuffer to do all reads and writes when using the TAB library implementation. There is a limitation in Java that the maximum size of the sum of all mapped files cannot exceed 2GB.
com.mapinfo.util.io.NIOByteBuffer – This will use the java.nio.FileChannel to do all read and writes when using the TAB library implementation. This method doesn't have the size limitation present when using the java.nio.MappedByteBuffer.
Note: In general, com.mapinfo.util.io.MemMappedByteBuffer should perform faster but results may vary.
Default MemMappedByteBuffer

TAB Reading

Key com.mapinfo.tab.read.classname
Description Implementation to use when performing reads.
Value com.mapinfo.tab.dp.TABDataProvider – This will use the TAB library implementation when performing reads (e.g. searchAll, searchWithinRectangle, etc.).
com.mapinfo.dp.tab.ReadOnlyTABDataProvider – This uses the read only implementation when performing reads (e.g. searchAll, searchWithinRectangle, etc). This is the same implementation present in MapXtreme Java 4.5 and prior.
Note: All of the update methods (e.g. updateFeature, replaceFeature and addFeature) use the new TAB library.
Default ReadOnlyTABDataProvider
System Logging

This appendix describes the logging capability available in MapXtreme Java.

In this appendix:

- Logging Overview .......................................................... 331
- Logging on the Server ...................................................... 331
- Logging on the Client ...................................................... 331
- Logging Levels .............................................................. 331
Logging Overview

MapXtreme Java can log various types of messages, anything from informational messages to fatal errors. The level (importance) of the information that is logged, as well as where, is fully configurable.

MapXtreme Java uses the Logging API from Jakarta Commons ([http://jakarta.apache.org/commons/logging.html](http://jakarta.apache.org/commons/logging.html)). The Commons Logging API provides an abstraction that allows logging engines to be changed as needed without having to change application code.

MapXtreme Java uses Log4J as its logging engine. You may change the logger at will as long as there is an adaptor that interfaces between the chosen logging engine and the Commons Logging API. For more information on how to configure Log4J visit the Apache Jarkarta Log4J site at [http://jakarta.apache.org/log4j/docs/index.html](http://jakarta.apache.org/log4j/docs/index.html).

Logging on the Server

If you are using Log4J as your logging engine, you can modify the log4j.properties file that ships with MapXtreme Java as you need.

Logging on the Client

If you are using Log4J as your logging engine, you can modify the log4j.properties file that ships with MapXtreme Java as you need.

Logging Levels

MapXtreme Java uses the following logging levels.

<table>
<thead>
<tr>
<th>Level</th>
<th>Description/Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>FATAL</td>
<td>An unexpected error has occurred, and has left the system in an unstable/ unusable state. Program execution cannot continue and the application must be restarted.</td>
<td>Servlet initialization failed because a configuration file could not be accessed.</td>
</tr>
<tr>
<td>ERROR</td>
<td>An unexpected error has occurred which prevents us from completely fulfilling the user's request. Program execution can continue.</td>
<td>Render failed because a named style was invalid, or not present.</td>
</tr>
<tr>
<td>Level</td>
<td>Description/Purpose</td>
<td>Example</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------</td>
<td>---------</td>
</tr>
<tr>
<td>WARN</td>
<td>An &quot;expected&quot; error has occurred from which the system can recover and complete the user's request.</td>
<td>An invalid value was specified – fall back to an acceptable default.</td>
</tr>
<tr>
<td>INFO</td>
<td>A means of conveying meaningful information to the user.</td>
<td>Let you know that certain system properties are being used.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Shows trace information that can help prove that something is working as it should, or help decipher a problem.</td>
<td>Display how many points are in each pointlist of a region during rendering.</td>
</tr>
</tbody>
</table>
This appendix displays custom symbol GIF images.

## Symbols

These custom symbol GIF images are located in MapXtreme-4.7.0\support\symbols\Custom or MapXtreme-4.7.0\lib\client\previewgifs.jar. For information on how to use these symbols in an application, see Chapter 14: Labeling and Renditions.

<table>
<thead>
<tr>
<th>AMBU1-32.gif</th>
<th>BADG1-32.gif</th>
<th>BADG2-32.gif</th>
<th>BANK1-32.gif</th>
<th>BANK2-32.gif</th>
<th>BOOK1-32.gif</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAMP1-32.gif</td>
<td>CAR1-32.gif</td>
<td>CAUT1-32.gif</td>
<td>CHUR1-32.gif</td>
<td>COMP1-32.gif</td>
<td>DB-CON.gif</td>
</tr>
<tr>
<td>FARM1-32.gif</td>
<td>FAST1-32.gif</td>
<td>FIRE1-32.gif</td>
<td>GLOB1-32.gif</td>
<td>GOLF1-32.gif</td>
<td>HOSP1-32.gif</td>
</tr>
<tr>
<td>HOUS1-32.gif</td>
<td>HOUS2-32.gif</td>
<td>HOUS3-32.gif</td>
<td>HYDR1-32.gif</td>
<td>INTE1-32.gif</td>
<td>LITE1-32.gif</td>
</tr>
<tr>
<td>LITE2-32.gif</td>
<td>MAIL1-32.gif</td>
<td>MBOX1-32.gif</td>
<td>MBOX2-32.gif</td>
<td>MOSQ1-32.gif</td>
<td>ONEW1-32.gif</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONEW2-32.gif</td>
<td>ONE Way Traffic Sign</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PENC1-32.gif</td>
<td>Pen Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN1-32.gif</td>
<td>Pin Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN2-32.gif</td>
<td>Pin Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN3-32.gif</td>
<td>Pin Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN4-32.gif</td>
<td>Pin Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN5-32.gif</td>
<td>Pin Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN6-32.gif</td>
<td>Pin Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POL1-32.gif</td>
<td>Police Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAIL1-32.gif</td>
<td>Railroad Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAIL2-32.gif</td>
<td>Railroad Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAIL3-32.gif</td>
<td>Railroad Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REST1-32.gif</td>
<td>Rest Area Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAT1-32.gif</td>
<td>Status Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOP1-32.gif</td>
<td>Stop Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYNA1-32.gif</td>
<td>Synonym Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TARG1-32.gif</td>
<td>Target Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TAXI1-32.gif</td>
<td>Taxi Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEM1-32.gif</td>
<td>Temp Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOWE1-32.gif</td>
<td>Tow Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOWE2-32.gif</td>
<td>Tow Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRAF1-32.gif</td>
<td>Traffic Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUC1-32.gif</td>
<td>Truck Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUC2-32.gif</td>
<td>Truck Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YIEL1-32.gif</td>
<td>Yield Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YIEL2-32.gif</td>
<td>Yield Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Index

A

Accessing data
  connection pooling 183–185
  Data Providers 43–44
  LocalDataProviderRef 145
  MapJ object 43
  MapXtremeDataProviderRef 145

Add Layer wizard
  adding custom data providers 270–272
  adding layers 71–72, 97
  adding named layers 60–61
  addlayerwizard.properties file 259
  configuring 259
  creating Analysis Layers 61
  creating Map Definitions 58–61
  most-recently-used (MRU) values 264
  named database connection 261–262
  page control defaults 263–264
  page controls 263–264
  removing data sources 260
  reordering data sources 261
  saving passwords 264

Add Theme wizard
  MapXtreme Java Manager 68–69

Adding individual map tools 105

Adding layers
  Annotation layer 153
  from Oracle8i 148
  GeoTIFF raster image 162
  Layer Control command 71–72
  named layers 154–155
  SQL Server data 149
  to collection 143, 147–150
  to Layers collection 135
  XY column data 149

AddTheme Wizard Bean
  adding themes 98–99

Alignment
  of labels 214

AnalysisLayer
  creating bar charts 61
  creating pie charts 61, 246

Animated images
  rendering 171–172

Annotation layers
  data source type 147
  editing 204
  Feature coordinate system 204

PrimaryKeys 204
  showing a search layer 152

Apache web server
  directory location 25

Applets
  MapXtreme Java Manager client as 51
  sample application 103–104

Attribute objects
  Feature objects 190
  searching 200, 201–202

Audience 2

Auto-increment columns
  IDBC layers 206

Autolabeling
  Layer Control command 70

B

BoundarySelectionMapTool
  MapTool Bean 95

Brightness style tag
  raster images 165

Brush styles
  MapBasic style strings 314–315

Bucketer object
  RangedTheme maps 239

C

Cartographic legends 245

Class files
  modifying servlet 124

Classpath
  adding files to at install 17
  miconnections.properties file 183

Click and drag actions
  custom map tools 110–111

Client-side design
  planning your application 44

Code examples
  creating a QueryParams object 198
  creating a RangedTheme 240–242
  creating a SelectionTheme 242–243
  creating an IndividualValueTheme 243
  creating AnnotationDataProvider 153
  creating map features 194–195
  creating OraSoDataProvider 148
  creating ranged label themes 218–220
  creating SQLServerSpwDataProvider 149
  creating TABDataProvider 147
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>creating XY Data Provider</td>
<td>149</td>
</tr>
<tr>
<td>dashed and parallel lines</td>
<td>231</td>
</tr>
<tr>
<td>dashed line stroke property</td>
<td>224</td>
</tr>
<tr>
<td>data binding</td>
<td>152</td>
</tr>
<tr>
<td>drawing lines with symbols</td>
<td>231</td>
</tr>
<tr>
<td>fill properties</td>
<td>230</td>
</tr>
<tr>
<td>font properties</td>
<td>230</td>
</tr>
<tr>
<td>GeoTIFFDataProvider, creating</td>
<td>162</td>
</tr>
<tr>
<td>getLayer method</td>
<td>156</td>
</tr>
<tr>
<td>getting layer bounds</td>
<td>156</td>
</tr>
<tr>
<td>getting layer bounds if null</td>
<td>158</td>
</tr>
<tr>
<td>image symbols</td>
<td>227</td>
</tr>
<tr>
<td>initializing MapJ object</td>
<td>130</td>
</tr>
<tr>
<td>inserting a layer</td>
<td>156</td>
</tr>
<tr>
<td>LabelProperties</td>
<td>209–210</td>
</tr>
<tr>
<td>line marker stroke property</td>
<td>224–225</td>
</tr>
<tr>
<td>loading map data</td>
<td>130–132</td>
</tr>
<tr>
<td>miconnections.properties file entry</td>
<td>184–185</td>
</tr>
<tr>
<td>moving a layer</td>
<td>156</td>
</tr>
<tr>
<td>obtaining layer names</td>
<td>155</td>
</tr>
<tr>
<td>OverrideTheme</td>
<td>238</td>
</tr>
<tr>
<td>overriding labels</td>
<td>217–218</td>
</tr>
<tr>
<td>parallel lines stroke property</td>
<td>223</td>
</tr>
<tr>
<td>per-feature rendtions</td>
<td>234</td>
</tr>
<tr>
<td>Pie charts</td>
<td>246</td>
</tr>
<tr>
<td>RangedThemeLegend</td>
<td>244</td>
</tr>
<tr>
<td>recentering a map</td>
<td>133–134</td>
</tr>
<tr>
<td>removing layers</td>
<td>156</td>
</tr>
<tr>
<td>rendering a map</td>
<td>132</td>
</tr>
<tr>
<td>rendering a named map with additional layers</td>
<td>169–171</td>
</tr>
<tr>
<td>servlet forwarding</td>
<td>180–181</td>
</tr>
<tr>
<td>setting coordinate system</td>
<td>134</td>
</tr>
<tr>
<td>setting distance units</td>
<td>135</td>
</tr>
<tr>
<td>setting label styles</td>
<td>217</td>
</tr>
<tr>
<td>setting label visibility</td>
<td>213</td>
</tr>
<tr>
<td>setting map bounds</td>
<td>134</td>
</tr>
<tr>
<td>setting rendered map size</td>
<td>132</td>
</tr>
<tr>
<td>setZoomAndCenter</td>
<td>134</td>
</tr>
<tr>
<td>sizing symbols</td>
<td>229</td>
</tr>
<tr>
<td>using font symbols</td>
<td>227</td>
</tr>
<tr>
<td>using OracleQueryBuilder</td>
<td>203</td>
</tr>
<tr>
<td>using Rendition stroke properties</td>
<td>223</td>
</tr>
<tr>
<td>using searchAll method</td>
<td>198</td>
</tr>
<tr>
<td>using searchAtPoint method</td>
<td>200</td>
</tr>
<tr>
<td>using searchByAttribute method</td>
<td>200</td>
</tr>
<tr>
<td>using searchByAttributes method</td>
<td>201–202</td>
</tr>
<tr>
<td>using searchWithinRadius method</td>
<td>199</td>
</tr>
<tr>
<td>using searchWithinRectangle method</td>
<td>200</td>
</tr>
<tr>
<td>using searchWithinRegion method</td>
<td>199</td>
</tr>
<tr>
<td>using symbols as fills</td>
<td>221</td>
</tr>
<tr>
<td>vector symbols</td>
<td>229–230</td>
</tr>
<tr>
<td>viewing entire layer</td>
<td>156</td>
</tr>
<tr>
<td>zoom layering</td>
<td>159</td>
</tr>
<tr>
<td>Colors</td>
<td></td>
</tr>
<tr>
<td>MapBasic style strings</td>
<td>322</td>
</tr>
<tr>
<td>Columns</td>
<td></td>
</tr>
<tr>
<td>auto-increment</td>
<td>206</td>
</tr>
<tr>
<td>labeling</td>
<td>211</td>
</tr>
<tr>
<td>ColumnStatistics object</td>
<td></td>
</tr>
<tr>
<td>RangedTheme maps</td>
<td>239</td>
</tr>
<tr>
<td>Compatibility</td>
<td></td>
</tr>
<tr>
<td>web environments</td>
<td>5</td>
</tr>
<tr>
<td>Composite Renderer</td>
<td></td>
</tr>
<tr>
<td>renderer type</td>
<td>43</td>
</tr>
<tr>
<td>specifying layers for redraw</td>
<td>177–179</td>
</tr>
<tr>
<td>Configuration</td>
<td></td>
</tr>
<tr>
<td>connection pooling</td>
<td>183–185</td>
</tr>
<tr>
<td>lax files</td>
<td>38</td>
</tr>
<tr>
<td>map tools</td>
<td>106–109</td>
</tr>
<tr>
<td>three-tier option</td>
<td>114–115</td>
</tr>
<tr>
<td>troubleshooting</td>
<td>36–38</td>
</tr>
<tr>
<td>two-tier option</td>
<td>116–118</td>
</tr>
<tr>
<td>Connection pooling</td>
<td></td>
</tr>
<tr>
<td>accessing pooled connections</td>
<td>186</td>
</tr>
<tr>
<td>configuration</td>
<td>183–185</td>
</tr>
<tr>
<td>Connections Manager</td>
<td></td>
</tr>
<tr>
<td>managing named connections</td>
<td>54, 185</td>
</tr>
<tr>
<td>Contrast style tag</td>
<td></td>
</tr>
<tr>
<td>raster images</td>
<td>165</td>
</tr>
<tr>
<td>Controller</td>
<td></td>
</tr>
<tr>
<td>JSP-servlet architecture component</td>
<td>306–307</td>
</tr>
<tr>
<td>Coordinate systems</td>
<td></td>
</tr>
<tr>
<td>Annotation layer Features</td>
<td>204</td>
</tr>
<tr>
<td>IDBC layer Features</td>
<td>205</td>
</tr>
<tr>
<td>micsys.txt file</td>
<td>68</td>
</tr>
<tr>
<td>setting</td>
<td>134</td>
</tr>
<tr>
<td>setting to raster image</td>
<td>163</td>
</tr>
<tr>
<td>Copy protection</td>
<td>38</td>
</tr>
<tr>
<td>Creating</td>
<td></td>
</tr>
<tr>
<td>annotation layers</td>
<td>153</td>
</tr>
<tr>
<td>custom JSP tags</td>
<td>305–307</td>
</tr>
<tr>
<td>data providers</td>
<td>265–272</td>
</tr>
<tr>
<td>GeoTIFFDataProvider</td>
<td>162</td>
</tr>
<tr>
<td>map definitions</td>
<td>58–61</td>
</tr>
<tr>
<td>map Feature objects</td>
<td>193–195</td>
</tr>
<tr>
<td>maps with MapJ API</td>
<td>130–133</td>
</tr>
<tr>
<td>named connections</td>
<td>184–186</td>
</tr>
<tr>
<td>named renditions</td>
<td>232</td>
</tr>
<tr>
<td>Oracle8i DataProvider</td>
<td>148</td>
</tr>
<tr>
<td>SQL Server DataProvider</td>
<td>149</td>
</tr>
<tr>
<td>XY DataProvider</td>
<td>149</td>
</tr>
<tr>
<td>Creating applications</td>
<td></td>
</tr>
<tr>
<td>using MapXtreme JavaBeans</td>
<td>101–102</td>
</tr>
<tr>
<td>Creating map tools</td>
<td></td>
</tr>
<tr>
<td>adding individual tools</td>
<td>105</td>
</tr>
<tr>
<td>associating with VisualMapJ object</td>
<td>105</td>
</tr>
<tr>
<td>MapToolBar</td>
<td>104</td>
</tr>
<tr>
<td>removing individual tools</td>
<td>106</td>
</tr>
<tr>
<td>Custom Image symbols</td>
<td></td>
</tr>
<tr>
<td>MapBasic style strings</td>
<td>320</td>
</tr>
</tbody>
</table>
Custom map tools
   click and drag actions 110–111
   creating 109–112
   drawing on the map 112
   keyboard input 111
   requirements 110
   SimpleRulerMap Tool example 109
Custom symbols
   installed components 32
   thumbnail views 333–334
Customizing Add Layer wizard 259–264

Data
   installing base map data 19–24
   loading 130–132
   managing with MapXtreme Java Manager 55–56
 Data aggregation
   in data binding 151–152
Data binding
   Add Layer wizard data source 259
   code example 152
   combining .TAB and JDBC data 150–152
   creating Map Definitions 61
   data source type 43, 147
   system properties 328
Data Providers
   accessing data sources 43–44, 49
   adding to Add Layer wizard 270–272
   creating 265–272
   creating OraSoDataProvider 148
   creating SQLServerSpwDataProvider 149
   creating TABDataProvider 147
   creating XY 149
   data source types 43
  DataProviderRef 144
   DataProviderRef 145
   defining layers 143–145
   for raster data 163, 270
   for vector data 268–270
   interfaces and files 266–268
   raster 162
   TableDescHelpers 144
Data sources
   changing in addlayerwizard.properties file 259–261
   DataProviderRef 145
   DataProviderRef 145
   defaults in addlayerwizard.properties file 263–264
   JDBC layers 146
   removing in addlayerwizard.properties file 260
   reordering in addlayerwizard.properties file 261
   supported databases 49
   supported types 147
DataProviderHelpers
   defining data source 144
   JDBC layer input parameters 186
DataProviderRef
   accessing data sources 145
Default values
   in addlayerwizard.properties file 263–264
Deployment
   Apache/Tomcat 25
   options 41
   speed 3
   Web Application Builder 90
   web-based mapping applications 40–41
Design considerations
   client side 44
   server side 44
Displaying map layers
   controlling in Layer Control 74
   features 48
   raster images 162
   renderers 43
   zoom layering 159–160
Displaying ToolTips
   in custom map tools 111
Distance units
   setting 135
Distribution types
   RangedTheme maps 239–240
Dithering method
   WBMP output 176
Document type definitions
   MapXtreme Java 251
   OGC-specific 251
   product independent 251
Documentation set
   additional resources 6
Drawing on the map
   custom map tools 112

EasyLoader
   command line flags 279–282
   loading Oracle Spatial Data 278–279
   options 276–278
   running 274–276
Editing
   Annotation layers 204
   Feature objects 204
   JDBC layers 205–206
   layers 72–73, 97
   TAB layers 207
   widgets 89
EditLayer Wizard
   editing layers 72–73, 97
   LayerControl Bean 97
EncodedImageRenderer
   animated images 171
Equal count ranged theme 239
Equal ranges theme 239–240
ESRI Shapefiles
   Add Layer wizard data source 259
Data Provider type 43
  data source type 147
Export format
  WBMP 175–176
Expressions
  labeling 211

F
Feature objects
  annotation layers 152
  Attribute objects 190
  editing 204
  geographic objects 189–192
  Geometry objects 190
  map objects 48
  methods 189–192
  MIRaster objects 191
  per-feature renditions 146
  record-level information 139
  Rendition objects 139, 190
  searching 197–202
  style override 75–77
  tabular and geometric data 139
  using themes 140
Feature themes 98–99
FeatureFactory object
  creating map Feature objects 193–195
  creating map features example 194–195
  methods 193–195
FeatureSet collection
  methods 196
File formats
  supported grid 166–167
  supported raster 162
  supported raster output 173
Fill properties
  code example 230
  Rendition properties 221–223
Font rendering
  system properties 328
Fonts
  code example 227, 230
  MapBasic style strings 321–322
  registering 32–33
  TrueType symbols 32, 226–227
Front-to-back ordering
  in Layer Control 71

G
Geography Markup Language (GML) 198, 250
Geometry objects
  Feature objects 139, 190
Geosets
  loading 130–132
  sample data 56
  saving as map definitions 47
  saving workspaces as 47
GeoTIFF files
  Add Layer wizard data source 259
  data source type 147
  raster format 162
getLayer method
  obtaining layer from collection 156
GIF files
  installed custom symbols 32, 333–334
  output recommendations 173
  provided custom symbols 227
  raster format 43, 162
Gradients
  linear 221
  radial 222
  Rendition stroke property 225
Grayscale style tag
  raster images 165
Grid images
  raster style tag 166
  supported types 166–167
H
HTML pages
  in thin client deployments 41
HTML parameters
  handling servlet requests 306–307
HTMLEmbeddedMapServlet
  sample servlet 123–126
HTMLThemeServlet
  sample servlet 126
I
Image IO
  Data Provider 163
Image symbols 227
IndividualValueTheme
  legends 237, 244–245
  thematic map type 243
Info tool
  viewing map object attributes 65
InfoMapTool
  MapTool Bean 95
  settings 107
Information resources 6
Informix Universal Server SpatialWare DataBlade
  Add Layer wizard data source 259
  data source type 43, 49, 147
Init parameters
  editing in Tomcat 125–126
Initializing a MapJ object
  code example 130
Input parameters
  JDBC layer DataProviderHelpers 186
Installation
  installed components 31–32
  MapXtreme Java Edition 12–19
  sample map data 19–24
  system requirements 12
IntraServlet Container Renderer
renderer type 43, 180–181

J
Jar files
HTMLEmbeddedMapServlet 123–126
HTMLThemeServlet 126
Java 2 Enterprise Edition
compliance 114
web deployment requirement 40
Java 2 virtual machine
system requirement 12
Java applets
in medium client deployments 41
in thick client deployments 41
Java SQL scripts
creating MAPINFO_MAPCATALOG 326–327
Java threading 4
Java2D Graphics2 objects
LocalRenderer 169
Java2DShape interface
using vector symbols 228
JavaBeans
creating an application 101–102
in two-tier configurations 116–118
JBuilder 6.0
creating an application in 101–102
JDBC compatible tables
data source type 43, 49, 147
JDBC connection pooling
accessing pooled connections 186
connection configuration 183–185
Connections Manager 54, 185
named resources 184
security benefits 183
system properties 328
JDBC Data Providers
creating XY Data Provider 149
JDBC drivers
lax.class.path 38
JDBC layers
auto-increment columns 206
coordinate system of Features 205
creating Map Definitions 60
DataProviderHelper input parameters 186
defining 146
editing 205–206
per-feature renditions 146
persistence of changes 205
saving Feature renditions 205
JNDI context
named renditions 232–233
Joins
Rendition stroke property 225
JPEG files
controlling quality 173
output recommendations 173
raster format 43, 162
JSP tag library
adding GUI editor to custom tags 311
creating an Add TAB Layer tab 307–308
creating custom tags 305–307
exposing tags 309–311
tag descriptions 295–305
Web Application Builder 90–91
K
Keyboard input
custom map tools 111
L
Label
text options 78
Label properties
merging 216
methods 211–216
zoom settings 210
Label themes 99
Labeling
alignment 81, 214
along a path 81, 214–215
code example 217
columns 211
controlling visibility 212–213
expressions 211
feature description 48
Geometry calculation mode 82
layers 160
modifying in Layer Control 77–82
multi-line text 212
offset distance 81
per-feature renditions 210
position 213–216
scale 212
styles 79, 211–212
thematic 209–210
visibility 79
Lax files
configuration information 38
Layer Control command
adding layers 71–72
autolabeling 70
editing layers 72–73
layer and theme visibility 70
layer display options 74–77
MapXtreme Java Manager 69–82
modifying labels 77–82
ordering map objects 71
removing layers 73
reordering layers 71
selectable layers 70
Layers
adding to collection 135, 143, 147–150
building maps 142
defining with Data Providers 143–145
draw order 143
labeling 160
LayerControl Bean 96–97
MapInfo tables 46
raster images 161
search methods 197–202
searching SQL query defined 202–203
using Layer control 69–82
using the Map tools 64–69
View Entire Layer button 68
zoom layering 159
Layers collection
adding layers 135, 143, 147–150
adding named layers 154–155
inserting a layer 156
map building blocks 142
map definitions 143
methods 155–156
moving a layer 156
obtaining a layer object 156
removing layers 156
LegendContainer Bean
legend display 245
managing legends 99
Legends
cartographic display 245
creating RangedTheme 244
themc 237, 244–245
Line caps
Rendition stroke property 225
Line markers
Rendition stroke property 224–225
Line objects
displaying 223–225
feature type 48
Linear Gradients 221
LinearRenditionSpreader object 240
Loading map data 130–132
Loading map definitions 57–58
LocalDataProviderRef
accessing data using 145
LocalRenderer
Java2D Graphics2D object 169
renderer type 43
Logging system messages 330–332
M
Map bounds 134
Map data
gesets 56
installing 19–24
loading 130–132
map definitions 57
Map definitions
creating 58–61
Layers collection 143
loading 57–58, 130–132
map data 57
saving to a database 62–64
saving to a file programmatically 135
storing in remote tables 135–136
with SimpleMap applet 104
Map Definitions panel
MapXtreme Java Manager 52
Map device bounds
rendered map size 132
Map distance units
setting 135
Map features
creating 193–195
display properties 220
map objects 48
Map layers
activating labels in Layer Control 77–82
adding 71–72
annotation layers 152
displaying MapInfo tables as 46
displaying raster images 162
editing 72–73
labeling 160
Layer Control display options 74–77
layer order 71, 143
removing 73
setting visibility 70
using Layer Control 69–82
using the Map tools 64–69
zoom layering 159–160
Map menu
controlling layer display 69–82
Map objects
Features 189–192
ordering in Layer Control 71
selectability within a layer 70
Map options tool
setting map distance units and coordinate system 67–68
Map state
using MapJ object 42
Map tools
adding individual 105
associating with VisualMapJ object 105
configuration 106–109
controlling layer display 64–69
creating custom 109–112
creating toolbar 104
general settings 106
including in applications 104–106
removing individual 106
zoom in/zoom out 106
Map view
controlling 133–135
MapBasic style strings
brush styles 314–315
custom image files 320
determining in MapInfo Professional 323
font styles 321–322
pen styles 313
symbols styles 315–321
MapDefContainer 136–137
MapImageRequests
  rendering multi-layered maps 251–254
MapInfo .TAB files
  creating Map Definitions 59
MapInfo 3.0 symbols
  MapBasic style strings 316
MapInfo Enterprise XML protocol
  map requests and responses 250–257
MapInfo geosets
  loading 130–132
  saving as map definitions 47
  saving workspaces as 47
MapInfo Grid
  data source type 147
MapInfo grid files 43, 167
MapInfo tables
  Add Remote Layer wizard data source 259
  creating TABDataProvider 147
  data source type 43, 147
  displaying as layers 46
  geosets 56
  table structure 46
MapInfo XML Protocol
  document type definitions 250–251
MAPINFO_MAPCATALOG
  creating with Java SQL scripts 326–327
  remote spatial data 325–327
MapJ API
  creating a map 130–133
  with MapXtremeServlet 4
MapJ methods
  initializing MapJ 130
  loading map data 130–132
  recentering a map 132–134
  rendering a map 132
  setting coordinate system 134
  setting distance units 135
  setting map bounds 134
  setting rendered map size 132
  setZoomAndCenter 134
MapJ object
  accessing data sources 43
  connection pooling 184
  in three-tier configurations 114–115
  in two-tier configurations 116–118
  initializing 130
  managing map state 42
  map-creation interface 130
MapMarker
  adding to MapXtreme Java Manager 54
MapMarker J Server
  with MapXtreme Java Manager 51
MapMouseListener interface 110–111
MapPainter interface
  drawing on the map 112
Mapping server
  MapXtremeServlet 42
Maps
  building with layers 142
  creating with MapJ API 130–133
  saving programmatically 135–137
  using in applications 6
MapTool Beans
  navigation and selection tools 94–95
MapToolBar Bean
  adding tools to SwingJToolbar 95
MapToolkit
  methods 127
  Servlet Utility Library 126–127
MapVectorRequests
  MapInfo XML Protocol 250
  retrieving features from data sources 254–257
MapViewer sample application
  JSP tag library 91
MapXtreme Java Edition
  Developer Guide 6
  information resources 6
  installation 12–19
  Layer Control command 69–82
  managing data 55–56
  Map Definitions panel 52
  Map tools 64–69
  named layers 84
  named maps 83–84
  named renditions 85
  named resources 53, 82–85
  starting 51–52
  Web Applications builder 53
MapXtreme Java Manager client
  as an applet 51
MapXtreme Java SQL Scripts
  creating MAPINFO_MAPCATALOG 326–327
MapXtremeJavaBeans
  AddTheme Wizard Bean 98–99
  creating an application 101–102
  creating mapping applets 93
  Layer Control Bean 96–97
  LegendContainer Bean 99
  MapTool Beans 94–95
  MapToolbar Bean 95
  SimpleMap applet sample application 103–104
  ViewEntireLayer Bean 100
  Visual MapJ 93
  ZoomPanel Bean 100
MapXtremeDataProviderRef
  accessing data using 145
MapXtremeImageRenderer
  remote rendering 169
renderer type 43
rendering a map 132

MapXtremeServlet
checking status of 36–37
connection pooling 183
developing client servlet 114
in three-tier configurations 114–115
MapJ object 4, 130
mapping server 42
supported raster formats 43
verifying URL 37–38

Medium client deployment
using Java applets 41

Methods
Feature object 189–192
FeatureFactory object 193–195
FeatureSet collection 196
Label properties class 211–216
Layers collection 155–156
MapJ API 130–135
searching a layer 197–202
Servlet Utility Library (MapToolkit) 127

miconnections.properties file
Connections Manager 54, 185
MapXtremeServlet classpath 183
sample entry 184–185

micsys.txt file
cordinate system data 134
supported coordinate systems 68

MIME format standard 173

MIRaster objects
Feature objects 191

Model
JSP-servlet architecture component 305

Multi-line label text 212
Multi-platform support 3
Multi-threading 4
MVC JSP-servlet architecture
custom JSP tags 305–307

N
Named database connections
Add Layer wizard data source 259
adding to addlayerwizard.properties file 261–262
connection pooling 183–185
creating 184–186

Named layers
Add Layer wizard data source 259
adding 60–61
adding to collection 154–155
creating 84
named resource type 153–155
saving 73
saving programmatically 154

Named maps
creating 83–84
retrieving programmatically 138–139
saving programmatically 138

Named renditions
creating 85, 232
retrieving 233
storing 232–233

Named resources
in MapXtreme Java Manager 53
JDBC connection pooling 184
managing 82–85

Natural break distribution type
Raster maps 240

Netscape
web server compatibility 5

New features
2

Northwood Grid
data source type 147

O

ObjectSelectionMapTool
Map Tool Bean 95

Open GIS Consortium (OGC)
document type definitions 251
Geography Markup Language (GML) 250

Oracle Spatial data
with EasyLoader 278–279

Oracle with Spatial Option
Add Remote Layer wizard data source 259
creating OraSoDataProvider 148
data source type 43, 49, 147
OraSoTableDescHelper 144

OracleQueryBuilder
code example 203

OraSoDataProvider
creating 148

Ordering overlapping objects 71

Output
JPEG quality 173
raster format recommendations 173
WBMP 175–176

OverrideTheme
thematic map type 238

P

Pan tool
MapXtreme Java Manager 65

PanMapTool
MapTool Bean 94

Parallel lines
code example 231
Rendition stroke property 223–224

Pass-through queries
QueryBuilder 202–203

Passwords
saving in addlayerwizard.properties file 264

Pen styles
MapBasic style strings 313

Per-feature renditions
labels 210
overriding table rendition 234
### Performance
- raster images [163, 169–171]

#### PNG files
- output recommendations [173]
- supported raster format [43]

### Point objects
- feature type [48]

### PolygonSelectionMapTool
- MapTool Bean [95]

### Pooled connections
- accessing [186]

### Positioning of labels [213–216]

### Preferences tool
- default settings [67]

### PrimaryKeys
- Annotation layer Features [204]
- Tab layer Features [207]

### Product description [2–3]

### Progressive rendering [179]

### Projection file, micsys.txt [68, 134]

### Properties
- Visual MapJ Bean [93]

### Prototype builder
- deploying applications [90]
- editing widgets [89]
- JSP tag library [90–91]
- templates [89]
- using [87–88]

### R

#### QueryBuilder
- OracleQueryBuilder [203]
- using pass-through queries [202–203]

#### QueryParams object
- searches [198]

### RangedTheme
- Bucketer object [239]
- ColumnStatistics object [239]
- creating [240–242]
- distribution types [239–240]
- legends [237, 244–245]
- thematic map type [238–242]

### Raster data
- Data Providers [270]

### Raster images
- adding to MapJ [162]
- Data Provider type [43]
- display coordinate system [163]
- format considerations [169–171]
- layers [161]
- MIRaster objects [191]
- performance [163]
- rendering [163]

### Rendition objects
- Feature objects [190]
- map feature display properties [220]
- themes [238]

### Rendition properties
- fill [221–223]
- stroke [223–226]
- symbol properties [226–228]

### Renditions
- displaying map features [48, 139]
- label styles [211–212]
- named [232–233]
- per-feature [146, 210, 234]
- saving JDBC layer Features [205]

### Reordering layers
- Layer Control command [71]

### Retrieving
- named maps [138–139]
RoundOff object 240
Routing J Server
adding to MapXtreme Java Manager 55
with MapXtreme Java Manager 51
Ruler tool
measuring distances 65
RulerMapTool
MapTool Bean 95
settings 107
Running EasyLoader 274–276

S
Sample data
installation 19–24
Samples
HTMLEmbeddedMapServlet 123–126
HTMLThemeServlet 126
location of 32
Saving
JDBC layer Feature renditions 205
map definitions to a database 62–64
maps programmatically 135–137
named layers 73, 154
named maps 138
passwords in addlayerwizard.properties file 264
Scalability
web systems 40
Scalable labels 212
Scalable Vector Graphics 173
Search layers
using annotation layers 152
searchAll method
layer searches 198
searchAtPoint method
features at specified point searches 200
searchByAttribute method
Attribute searches 200
searchByAttributes method
Attribute searches 201–202
Searching
attributes 200, 201–202
etire layer 198
Feature objects 197–202
features at specified point 200
Layer object methods 197–202
layers defined by SQL queries 202–203
Limiting the returned information 197–198
QueryBuilder 202–203
QueryParams object 198
within regions 199
within specified distance 199
within specified rectangle 200
searchWithinRadius method
searching within specified distance 199
searchWithinRectangle method
specified rectangle searches 200
searchWithinRegion method
region searches 199
Selection tools
MapXtreme Java Manager 66–67
Select 66
Select within boundary 67
Select within polygon 67
Select within radius 66
Select within rectangle 66
SelectionMapTool 107–109
SelectionTheme 242–243
Server-side
design considerations 44
Java component 4
Servlet architecture
MapXtremeServlet 2
Servlet container
Tomcat 25
Servlet forwarding 123
code example 180–181
Servlet Utility Library (MapToolkit)
using 126–127
Servlets
HTMLEmbeddedMapServlet 123–126
HTMLThemeServlet 126
MapXtreme mapping engine 114
web deployment requirement 40
setLabelColumn method
specifying label column 211
setLabelExpression method
specifying label expression 211
Setting map bounds 134
Shapefiles
Add Layer wizard data source 259
data source type 43
SimpleMap
loading map definitions 104
with MapXtreme JavaBeans 103–104
SimpleRulerMapTool
custom tool example 109
Software copy protection 38
Spatial data
MAPINFO_MAPCATALOG 325–327
SpatialWare Datablade 5
Splined labels 214–215
SQL queries
searching layers defined by 202–203
SQL Server Data Provider 149
SQL Server for SpatialWare
Add Remote Layer wizard data source 259
data source type 43, 49, 147
Standard deviation distribution type
RangedTheme maps 240
Stroke properties
Rendition properties 223–226
Style Override
layer display 75–77
Styles
labeling 211–212
overriding with custom 76
overriding with pre-defined 75
VisualMapJ Bean
  displaying maps 93
  with MapTool Beans 94
VisualMapJ object
  associating map tools 105

W
WarFile Generator utility 26–31
WBMP files
  dithering 176
  exporting to 175–176
  recommendations 173
  thresholding 175
Web Application Builder
  deploying applications 90
  editing widgets 89
  exposing JSP tags 309–311
  JSP tag library 90–91
  templates 89
  using 87–88
Web Applications panel
  MapXtreme Java Manager 53
Web environment compatibility 5
Web server
  system requirements of 12
Web-based deployment
  infrastructure requirements 40
  mapping applications 40–41
  options 41
Widgets
  editing 89
WMS JSP Sample Application 292
WMS Server
  GetCapabilities 284
  GetFeatureInfo 289
  GetMap 287
Workspaces
  saving as geosets 47

X
X/Y column data
  Add Layer wizard data source 259
XML protocol 250–257
XY Data Provider 149

Z
Zoom In / Zoom Out tools
  map tool setting 106
  MapXtreme Java Manager 64
Zoom layering
  labels 212–213
  Layer Control options 74
  map layers 159–160
Zoom settings
  for labels 79, 210
  for map 133
ZoomInMapTool
  MapTool Bean 95
ZoomOutMapTool
  MapTool Bean 95
ZoomPanel Bean
  managing the map view 100