Oracle® Fusion Middleware

User's Guide for Oracle MapViewer

11*g* Release 1 (11.1.1)

E10145-09

May 2013

Describes how to use Oracle MapViewer, a tool that renders maps showing different kinds of spatial data.

ORACLE®

Oracle Fusion Middleware User's Guide for Oracle MapViewer, 11*g* Release 1 (11.1.1)

E10145-09

Primary Author: Chuck Murray

Contributors: Joao Paiva, L.J. Qian, Jayant Sharma, Ji Yang, Honglei Zhu

Contents

2 MapViewer Concepts

9 Oracle Map Builder Tool

10 Oracle MapViewer Editor

C Creating and Registering a Custom Image Renderer

D Creating and Registering a Custom Spatial Data Provider

E OGC WMS Support in MapViewer

Index

List of Examples

List of Figures

List of Tables

Preface

Oracle Fusion Middleware User's Guide for Oracle MapViewer describes how to install and use Oracle MapViewer (MapViewer), a tool that renders maps showing different kinds of spatial data.

Audience

This document is intended primarily for programmers who develop applications that require maps to be drawn. You should understand Oracle database concepts and the major concepts associated with XML, including DTDs. You should also be familiar with Oracle Spatial or Oracle Locator concepts, or at least have access to *Oracle Spatial and Graph Developer's Guide*.

This document is not intended for end users of websites or client applications.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit
`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documentation

For more information, see the following documents in the Oracle Database documentation set:

- *Oracle Spatial and Graph Developer's Guide*
- *Oracle Spatial and Graph GeoRaster Developer's Guide*
- *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*
- *Oracle Database Concepts*
- *Oracle Database SQL Language Reference*

See also the following document in the Oracle Fusion Middleware documentation set:

- *Oracle Fusion Middleware High Availability Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

New and Changed Features

This section describes major features that are new or changed since the previous release of MapViewer, which was included in Oracle Application Server Release 10.1.3.1. This section groups the new features into "MapViewer Core" and "Oracle Maps" subsections.

In addition, the MapViewer JSP tag library and PL/SQL API are deprecated features. For more information, see the notes at the beginning of Chapter 5, "MapViewer JSP Tag Library" and Chapter 6, "MapViewer PL/SQL API".

MapViewer Core

This section describes features related to MapViewer generally, including the Map Builder Tool.

MapViewer Editor

Effective with Oracle Fusion Middleware Release 11.1.1.7, the MapViewer Editor is provided as an integrated component of MapViewer. The MapViewer Editor is a web-based spatial data editing tool. For more information, see Chapter 10, "Oracle MapViewer Editor".

GDAL-OGR and Teradata Data with Custom Geometry Themes

Effective with Oracle Fusion Middleware Release 11.1.1.7, for custom geometry themes you can now associate the theme with spatial data from a GDAL-OGR or Teradata provider. (Previously, custom geometry themes could be based only on shapefile data.) For more information, see Section 2.3.9, "Custom Geometry Themes".

WMTS Themes Support

Support is added for the rendering of data delivered using the Open GIS Consortium (OGC) Web Map Tile Service (WMTS) standard, specifically the WMTS 1.0.0 implementation standard. For more information, see Section 2.3, "Themes", especially Section 2.3.8, "WMTS Themes".

Secure Map Rendering

MapViewer now supports secure map rendering based on a web user's identity. Users with different roles or permissions will see different feature sets when viewing the same theme. For more information, see Section 1.8.

WFS Themes

Web Feature Service (WFS) features can now be viewed through MapViewer's WFS themes. These themes support the parsing and caching of WFS capabilities, and the use of feature conditions and queries. WFS theme support also works with Oracle Maps; for example, you can display a WFS theme as an interactive feature of interest (FOI) layer. For information about WFS themes, see Section 2.3.7.

Map Builder Enhancements

The Map Builder tool now supports the creation of WFS themes and Annotation Text-based themes. The base map panel also supports identifying features (and a list of rendered themes) on mouse clicks. For information about Map Builder, see Chapter 9.

Improved Nonspatial Data Provider Support

MapViewer now supports a default provider and format for applications to supply XML-based nonspatial data for thematic mapping. You can also specify columns from the nonspatial data set to be used in conjunction with an advanced style. For information about thematic mapping using nonspatial external attribute data, see Section 2.3.11.1.

Multiple Rendering Styles for a Single Feature

Previously, a feature could be rendered by only one rendering style. You can now use multiple rendering styles when rendering a theme's features. For example, you can shade a polygon with a color style while also plotting a pie chart on top of it, without defining two themes. This is done using stacked styles in a theme's definition. For an example of map request using stacked styles, see Section 3.1.13.

Automatic Reduction of Repetitive Labels

Previously, repetitive street labels or highway shields on linear features were displayed when such features consisted of many small segments. You can now use the Map Builder tool to specify the **No Repetitive Labels** option in the base map properties, to cause features (such as road segments) with same name to be labeled only once. For information about specific options in Map Builder, see the online help for that tool.

Scale Ranges for Theme Labeling

In the context of a base map, you can now assign scale limits to its themes' labels. These scale limits control when a theme's features will display their label texts. For more information and an example, see Section 2.4.1.

PDF Output

Full PDF map output support is provided. If you use `PDF_URL` or `PDF_STREAM` as the map format in your XML map request, MapViewer will generate vector PDF maps. For more information, see the explanation of the `format` attribute in Section 3.2.1.1.

Text Style Enhancements

The TEXT style has been improved to support customizable spacing between letters. It also supports additional (vertical) alignment options when labeling linear features.

Heat Map Support

MapViewer now supports heat maps, which are two-dimensional color maps of point data sets. Heat map styles are described in Section A.6.8.

Scalable Styles

MapViewer now supports scalable styles. A scalable style (such as a MARKER or LINE style) uses real-world units such as meter or mile to specify its size and other dimensional attributes; however, at runtime MapViewer automatically scales the style so that the features rendered by the style always show the correct size, regardless of the current map display scale. For information about using scalable styles, see Section 2.2.1.

Custom Tags for Theme and Base Map Definitions

The XML definition of a theme or base map now supports application-specific attribute tags. You can use the **Custom Tags** option in the theme definition in Map Builder to specify tags and their values, which can be interpreted by your application but are ignored by MapViewer itself.

Getting Style Names Referred to in a Predefined Theme

The new <list_theme_styles> element enables you to get the names of styles referred to in a predefined theme. This element is described in Section 7.6.

Simple URL Request to Get a Sample Image for a Style

You can now issue a simple URL request to the MapViewer server and get back a sample image of any style that you specified in the URL. This is useful if you want to build a custom map legend. For information about getting a sample image of any style, see Section 2.2.5.

Annotation Text

Support is provided for OpenGIS Consortium standard annotation text. Oracle Spatial in Oracle Database Release 11*g* supports storage of annotation text objects in the database, and MapViewer now supports displaying such annotation texts on a map. For information about annotation text themes, see Section 2.3.10.

Logging Mechanism Changes

A new logging mechanism based on Java logging is provided. You can also use the Oracle Application Server management console to customize how MapViewer logs things at runtime.

Custom (External) Spatial Data Providers

MapViewer now supports rendering of geospatial data stored in non-Oracle Spatial repositories. This is achieved through a Custom Spatial Data Provider API, where you can implement an Interface that feeds your own (proprietary) spatial data to MapViewer for rendering. Note that you will still need an Oracle Database to manage the mapping metadata, such as styles and themes definitions. For more information, see Section 1.5.2.9.

User-Specified JDBC Fetch Size for Predefined Themes

You can now specify a nondefault row fetch size on a theme, by setting the **Fetch Size** base map property with the Map Builder tool. MapViewer can use this value when fetching theme features from the database. Specifying an appropriate value can make performance tuning easier in certain situations.

New Array Types (MV_xxxLIST)

MapViewer uses the SQL array types MV_STRINGLIST, MV_NUMBERLIST, and MV_DATELIST, which support array-type binding variables that might exist in some

predefined themes. In some situations, you will need to create these types. For more information, see Section 1.4.3.3.

transparent_nodata Attribute for GeoRaster Themes

The optional `transparent_nodata` attribute can be specified for GeoRaster themes (described in Section 2.3.4). If `transparent_nodata` is `true`, any GeoRaster NODATA value is to be rendered as transparent. The default value is `false`.

Reprojection of GeoRaster Themes

Effective with Oracle Spatial GeoRaster for Release 11.2.0.1, GeoRaster objects can be reprojected into a different SRID. For more information, see Section 2.3.4.3.

Authentication with Predefined WMS Map Themes

You can specify the user and password in a predefined WMS map theme for a WMS server that requires authentication for access to the WMS data. For more information, see Section E.3.3.

max_connections Attribute No Longer Used

The `max_connections` attribute is obsolete: it no longer affects rendering and is ignored. The `number_of_mappers` attribute value affects the actual maximum number of database connections or sessions open for the data source at any given time. See the explanation of the `number_of_mappers` attribute in Section 1.5.2.14, "Defining Permanent Map Data Sources".

Oracle Maps

This section describes features for Oracle Maps, which is documented in Chapter 8.

Oracle Maps JavaScript API V2

Effective with Oracle Fusion Middleware Release 11.1.1.7, the Oracle Maps JavaScript API has a second API (Version 2, or V2) that is based on HTML5, as well as the original V1 API. The V2 API supports a richer set of features, and is introduced in Section 8.4.2, "JavaScript API V2".

Multi-Touch Mobile Device Support

Effective with Oracle Fusion Middleware Release 11.1.1.6, the Oracle Maps JavaScript API fully supports multi-touch gestures on popular iOS (iPhones and iPads) and late-version Android devices. Your Oracle Maps applications can take full advantage of this enhancement without any modification in your code.

Displaying Google Maps and Bing Maps Tiles as Built-in Map Tile Layer

Applications can now display Google Maps tiles or Microsoft Bing Maps tiles as a built-in map tile layer. Internally, the Oracle Maps client uses the official Google Maps or Bing Maps API to display the map that is directly served by the Google Maps server. For more information, see Section 8.6, "Using Google Maps and Bing Maps". (If you need to overlay your own spatial data on top of the Google Maps or Microsoft Bing Maps tile layer, see also Section 8.7, "Transforming Data to a Spherical Mercator Coordinate System".)

Effective with Oracle Fusion Middleware Release 11.1.1.6, Google Maps API Version 3 and Bing Maps Version 7 are the default APIs used for those technologies by Oracle Maps.

MVBaseMap Renamed to MVMapTileLayer

The class `MVBaseMap` in the Oracle Maps Javascript API is renamed to `MVMapTileLayer` to prevent possible confusion with the concept of MapViewer base map. For more information, see the JavaScript API documentation for `MVMapTileLayer`.

Web-Based User Interface for Map Tile Layer Management

A new web-based user interfaced has been added to the MapViewer web administration console for editing map tile layer definitions, as well as previewing and managing map tiles.

External Map Tile Support

The Oracle Maps JavaScript client can now display map tiles rendered directly by an external map tile server without caching the tiles with the MapViewer map tile server. For more information, see Section 8.8, "Dynamically Displaying an External Tile Layer" and the JavaScript API documentation for `MVCustomMapTileLayer`.

Improved Client-Side Support for Accessing Cross-Domain Map Tile Server and FOI Server

The Oracle Maps client can now communicate with cross-domain map cache tile and FOI servers without relying on a proxy server, which was previously required. For more information, see the JavaScript API documentation for `MVMapView.enableXMLHTTP`.

Dynamic Client Side Styles

More support is provided for rendering FOI data using dynamic client side styles for business intelligence (BI) applications. New classes are added to the Javascript API to support client side defined styles such as color style (`MVStyleColor`), marker style (`MVStyleMarker`), bar chart style (`MVBarChartStyle`), pie chart style (`MVPieChartStyle`), bucket style (`MVBucketStyle`), as well as any MapViewer supported style defined in XML (`MVXMLStyle`).

JDBC Theme-Based FOI

Oracle Maps now supports client side dynamically constructed JDBC theme-based FOI layers. For more information, see the JavaScript API documentation for `MVThemeBasedFOI`.

Simplified Dynamic BI Data Injection and Visualization

Business Intelligence applications can now visualize application generated nonspatial attribute data on the map through the combined use of a nonspatial data provider and theme-based FOIs. For more information, see the JavaScript API documentation for `MVNSDP`.

Improved Information Window

The positioning, styling, and sizing of the information window have been improved. Previously, the Oracle Maps client always displayed the information window at a fixed position relative to the specified map location. The Oracle Maps client now can place the information window at the optimal variable position relative to the specified map location. As the result, the map does not to be panned in order to make the information window visible inside the map. In addition, you can specify tabs for the information window.

For more information, see the JavaScript API documentation for `MVMapView.displayInfoWindow` and `MVMapView.displayTabbedInfoWindow`. The *Tabbed info window* demo on the Oracle Maps tutorial page shows how to display a tabbed information window.

Enhanced Map Decoration

The client now supports multiple collapsible map decoration components that can be positioned at arbitrary positions inside the map container. Map decoration can now be dragged inside the map container. For more information, see the JavaScript API documentation for `MVMapDecoration`.

Flexible Placement and Visibility for Navigation Panel and Scale Bar

The navigation panel and the scale bar can now be placed inside a map decoration component, which can be displayed or hidden and can be placed at a position of your choice inside the map container. For more information, see the JavaScript API documentation for `addNavigationPanel`.

Navigation Panel Informational Tips

Applications can now define mouseover informational tips or labels for map zoom levels. The informational tips are displayed when the user moves the mouse over the navigation panel. The user can then zoom to a selected zoom level by clicking on the corresponding info tip. For more information, see the JavaScript API documentation for `MVNavigationPanel.setZoomLevelInfoTips` and the Navigation Panel demo on the Oracle Maps tutorial page

Polygon Theme-Based FOI Layer Labeling

Applications can now choose whether to label the polygon features of a polygon theme-based FOI layer. For more information, see the JavaScript API documentation for `MVThemeBasedFOI`.

Image-Less Polygon Themes with FOI Layers

You can now base a feature of interest (FOI) layer on an image-less polygon theme, causing FOI images not to be rendered if the theme is already rendered as part of the base map. This feature can result in much faster performance with polygon layers. For more information, see the JavaScript API documentation for `MVThemeBasedFOI`.

FOI Layer Automatic Selection and Highlighting

You can now associate a filtering geometry with any predefined theme-based FOI layer so that only the features that fall inside the filtering geometry are rendered on the map. This feature is part of the new support for proximity and within-distance mapping. It can be used with the selection tools (circle, rectangle, or polygon) to implement theme feature highlighting. For more information, see the JavaScript API documentation for `MVThemeBasedFOI`.

Client-Side Construction of Geodetic Geometries Based on Earth Distance Parameters

You can now construct the following type of geometries in geodetic coordinate systems with parameters based on Earth distance: a circle polygon geometry specified by its center and radius, a rectangle with its height and width specified, and a point geometry at the specified distance and bearing from the start point. These geometries, especially the first two, can be used for implementing proximity and within-distance

type mapping. For more information, see the JavaScript API documentation for `MVSdoGeometry`.

Animated Loading Icon for Maps and Themes

The Oracle Maps client now displays an animated icon during the loading of a base map or a theme. This is especially useful for providing visual reassurance to users with maps and themes that take a long time to load.

User-Defined FOI Customizations

The JavaScript API now provides methods for applications to modify the geometry representation and rendering style of an already rendered user-defined FOI, as well as the custom marker image for a user-defined point FOI. For more information, see the JavaScript API documentation for `MVFOI`.

Prompt Mode for Marquee Zoom Tool

The new prompt mode provides enhanced user control over marquee zoom operations. Prompt mode zooms the map when the use clicks on the marquee zoom rectangle, which eliminates the possible problem of accidental zooming associated with "continuous" mode. For more information, see the JavaScript API documentation for `MVMapView.startMarqueeZoom`.

Mouse Cursor Customization

Applications can now customize the appearance of the mouse cursor when the cursor is over different map components, such as map tiles, FOIs, and map decorations.

Built-in Toolbar and Distance Measurement

Applications can now use a built-in distance measurement tool to measure distance on the map. The built-in toolbar provides an easy graphic user interface for accessing utilities such as the redline tool, rectangle tool, circle tool, distance measurement tool, and any user-defined capabilities. For more information, see the JavaScript API documentation for `MVToolBar` and the *Tool bar* demo on the Oracle Maps tutorial page.

Automatic Determination for Whole Image Theme Display

Displaying a theme-based FOI layer as a whole image may greatly improve the application performance, but it may be difficult for application developers to determine when to display a theme as a whole image theme. However, you can now choose to let MapViewer make the determination automatically. For more information, see the JavaScript API documentation for `MVThemeBasedFOI.enableAutoWholeImage`.

Automatic Long Tile Administrative Request Recovery

Long running tile admin requests that are interrupted due to Fusion Middleware or MapViewer shutdown will be able to resume automatically after MapViewer is restarted. (You do not need to do anything to enable this feature, other than creating the new database view USER_SDO_TILE_ADMIN_TASKS if it does not already exist. For more information, see Section 2.9.)

Wraparound Map Display

Applications can now display a map in the wrap-around manner. When the map is displayed in this manner, the map wraps around at the map coordinate system

boundary horizontally and therefore can be scrolled endlessly. For more information, see the JavaScript API documentation for `MVMapView.enableMapWrapAround`.

Individual Theme Feature Highlighting (Selection)

Applications can enable the user to select and highlight individual theme features (FOIs) by clicking the mouse on the features. For more information, see the JavaScript API documentation for `MVThemeBasedFOI.enbleHightlight` and the *Highlighting individual features of a theme based FOI layer* demo on the Oracle Maps tutorial page.

Enhanced Redline Tool

The redline line tool can now be used to create polyline, polygon, and point geometries. The redline line tool also supports an editing mode, in which you can move an existing redline point or line segment, remove a redline point or line segment, or add a redline point or line segment programmatically. For information about redlining using the Oracle Maps JavaScript API, see Section 8.4.

Error Reporting

Previously, all error messages thrown by the Oracle Maps client were displayed as browser alerts. Now applications can customize how the error messages are handled by using a custom error handler. For more information, see the JavaScript API documentation for `MVMapView.setErrorHandler`.

New Tutorials

Many new Oracle Maps tutorials illustrate the new features. To access the MapViewer demos and tutorials, go to:

`http://host:port/mapviewer/fsmc/tutorial/demos.html`

1

Introduction to MapViewer

Oracle Fusion Middleware Mapviewer (MapViewer) is a programmable tool for rendering maps using spatial data managed by Oracle Spatial and Graph or Oracle Locator (also referred to as Locator). MapViewer provides tools that hide the complexity of spatial data queries and cartographic rendering, while providing customizable options for more advanced users. These tools can be deployed in a platform-independent manner and are designed to integrate with map-rendering applications.

This chapter contains the following major sections:

- Section 1.1, "Overview of MapViewer"
- Section 1.2, "Getting Started with MapViewer"
- Section 1.3, "Prerequisite Software for MapViewer"
- Section 1.4, "Installing and Deploying MapViewer"
- Section 1.5, "Administering MapViewer"
- Section 1.6, "Oracle Real Application Clusters and MapViewer"
- Section 1.7, "High Availability and MapViewer" (for advanced users)
- Section 1.8, "Secure Map Rendering"
- Section 1.9, "MapViewer Demos and Tutorials"

1.1 Overview of MapViewer

MapViewer is shipped as part of Oracle Fusion Middleware. Its main deliverable is a Java EE (Java Platform, Enterprise Edition) application that can be deployed to a Java EE container, such as that for Oracle Fusion Middleware. MapViewer includes the following main components:

- A core rendering engine (Java library) named `SDOVIS` that performs cartographic rendering. A servlet is provided to expose the rendering functions to web applications.
- A suite of application programming interfaces (APIs) that allow programmable access to MapViewer features. These APIs include XML, Java, and an AJAX-based JavaScript API.
- A graphical Map builder tool that enables you to create map symbols, define spatial data rendering rules, and create and edit MapViewer objects.
- Oracle Map, which includes map cache and FOI (feature of interest) servers that facilitate the development of interactive geospatial web applications.

The core rendering engine connects to the Oracle database through Java Database Connectivity (JDBC). It also reads the map metadata (such as map definitions, styling rules, and symbology created through the Map Builder tool) from the database, and applies the metadata to the retrieved spatial data during rendering operations.

The XML API provides application developers with a versatile interface for submitting a map request to MapViewer and retrieving the map response. The JavaBean-based API provides access to MapViewer's rendering capabilities. The JavaScript API enables you to create highly interactive web applications that use the Oracle Maps feature of MapViewer.

The Map Builder tool simplifies the process of creating and managing map, theme, and symbology metadata in a spatial database. For information about this tool, see Chapter 9.

Oracle Maps, built on core MapViewer features, uses a map tile server that caches map image tiles, and a feature of interest (FOI) server that streams live data out of a database to be displayed as interactive features on a map. You can use the AJAX-based JavaScript API with Oracle Maps to provide sophisticated mapping solutions. Oracle Maps also allows for advanced customization and querying capabilities.

The primary benefit of MapViewer is its integration with Oracle Spatial and Graph, Oracle Locator, and Oracle Fusion Middleware. MapViewer supports two-dimensional vector geometries stored in Oracle Spatial and Graph, as well as GeoRaster data and data in the Oracle Spatial and Graph topology and network data models. Oracle MapViewer is also an Open Geospatial Consortium (OGC)-compliant Web Map Service (WMS) server.

1.1.1 Basic Flow of Action with MapViewer

With MapViewer, the basic flow of action follows a two-step request/response model, whether the client requests a map or some MapViewer administrative action.

For a map request:

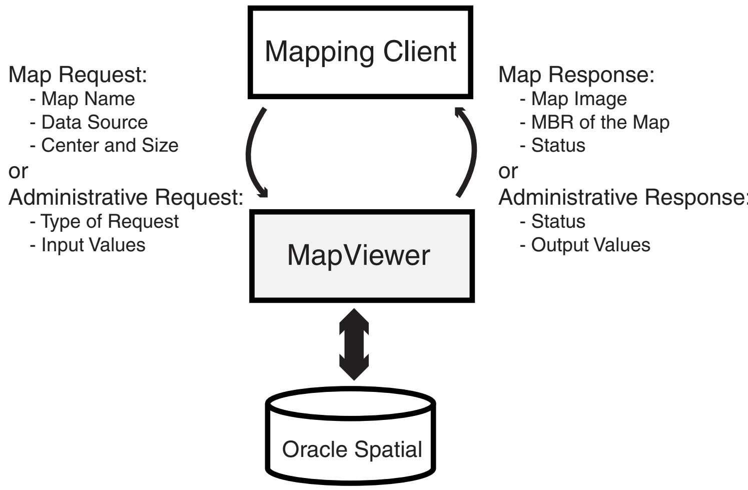
1. The client requests a map, passing in the map name, data source, center location, map size, and, optionally, other data to be plotted on top of a map.
2. The server returns the map image (or a URL for the image) and the minimum bounding rectangle (MBR) of the map, and the status of the request.

For a MapViewer administrative request:

1. The client requests a MapViewer administrative action, passing in the specific type of request and appropriate input values.
2. The server returns the status of the request and the requested information.

Figure 1–1 shows the basic flow of action with MapViewer.

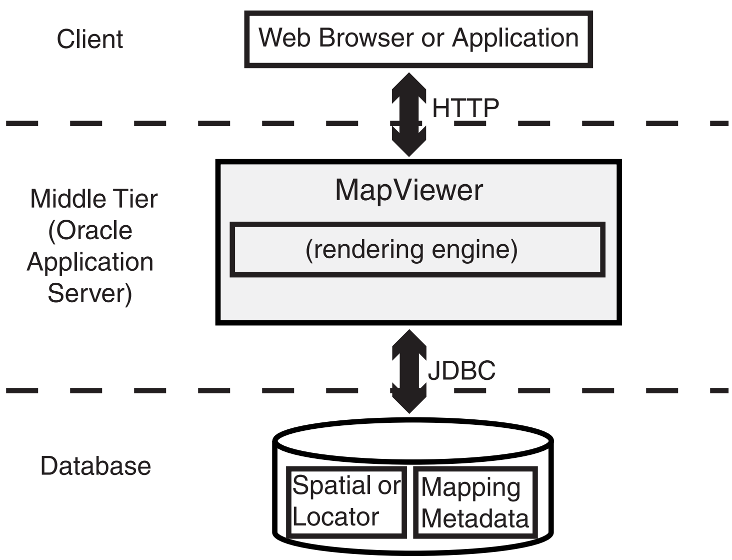
Figure 1–1 Basic Flow of Action with MapViewer



1.1.2 MapViewer Architecture

Figure 1–2 illustrates the architecture of MapViewer.

Figure 1–2 MapViewer Architecture



As shown in Figure 1–2:

- MapViewer is part of the Oracle Fusion Middleware middle tier.
- MapViewer includes a rendering engine.
- MapViewer can communicate with a client web browser or application using the HTTP protocol.
- MapViewer performs spatial data access (reading and writing Oracle Spatial and Graph or Oracle Locator data) through JDBC calls to the database.
- The database includes Oracle Spatial and Graph or Oracle Locator, as well as mapping metadata.

1.2 Getting Started with MapViewer

To get started using MapViewer, follow these steps:

1. Either before or after you install and deploy MapViewer, read Chapter 2 to be sure you understand important terms and concepts.
2. Ensure that you have the prerequisite software (see Section 1.3).
3. Install (if necessary) and deploy MapViewer (see Section 1.4).
4. Use MapViewer for some basic tasks. For example, create an Oracle Maps application (see Chapter 8.
5. Optionally, use the Map Builder tool (described in Chapter 9) to familiarize yourself with styles, themes, and maps, and the options for each, and optionally to preview spatial data.

1.3 Prerequisite Software for MapViewer

To use MapViewer, you must have the following software:

- A Java EE server supported by Oracle MapViewer (see `http://www.oracle.com/technetwork/middleware/mapviewer/j2ee-server-support-097757.html`)
- Oracle Database with Spatial and Graph option or Locator (Release 10*g* or later)
- Oracle Client (Release 10*g* or later), if you need to use JDBC Oracle Call Interface (OCI) features. In general, though, the JDBC thin driver is recommended for use with MapViewer, in which case Oracle Client is not required.
- Java SDK 1.6 or later

MapViewer also supports the headless AWT mechanism in J2SE SDK, which enables MapViewer to run on Linux or UNIX systems without setting any `X11 DISPLAY` variable. To enable AWT headless mode on Linux or UNIX systems, specify the following in the command line to start MapViewer:

```
-Djava.awt.headless=true
```

1.4 Installing and Deploying MapViewer

You can install (if necessary) and deploy MapViewer to run in the middle tier. MapViewer runs as a Java EE web application and listens for incoming map requests on the container's HTTP port.

To deploy MapViewer, choose one of the following procedures that applies to your needs:

- If you have already installed WebLogic Server 10 or later and you want deploy MapViewer to it, follow the instructions in Section 1.4.1.
- If you want to deploy MapViewer to a Java Platform, Enterprise Edition (Java EE) server from an archived (zipped) EAR file or from an exploded (unzipped) EAR folder, follow the instructions in Section 1.4.2.
- Download the latest MapViewer Quick Start kit from the MapViewer page on the Oracle Technology Network (OTN). This kit includes a standalone Glassfish server with MapViewer already deployed and configured. It takes only minutes to get MapViewer running, and is convenient for testing and basic development.

Regardless of where and how MapViewer is deployed, a home directory for MapViewer is created during deployment. For example, in the Quick Start kit this directory is located under the following directory:

$ORACLE_HOME/apps

$ORACLE_HOME is the top directory of the Glassfish installation. The MapViewer directory is typically named `mapviewer.war` or `mapviewer` (or the same as the context path under which MapViewer is deployed), and has many subdirectories. You may wish to familiarize yourself with some of the subdirectories in case you want to perform debugging, administration, or manual configuration.

The following are the main subdirectories of a MapViewer deployment:

```
/mapviewer
   admin/
   fsmc/
   jslib/
   WEB-INF/
            lib/
            conf/
            log/
            tilecache/
            classes/
            admin/
```

The `/mapviewer/fsmc` directory contains the Oracle Maps JavaScript V1 API library, and the `/mapviewer/jslib` directory contains the Oracle Maps JavaScript V2 API library. The `/mapviewer/WEB-INF` directory and its subdirectories contain libraries and MapViewer administration and configuration files.

If you want to use GeoRaster themes to view GeoRaster data, after successfully deploying MapViewer you may need to ensure that certain JAI (Java Advanced Imaging) library files are in the MapViewer Java classpath. The library files are `jai_core.jar`, `jai_codec.jar`, and `jai_imageio.jar`, and they can be found in a full Oracle Fusion Middleware or Oracle Database installation, usually under the directory for Oracle Multimedia (formerly called Oracle *inter*Media) files. You can copy them into the MapViewer `WEB-INF/lib` directory.

For annotation themes, MapViewer uses the JAXB 2.x libraries `jsr173_api.jar`, `jaxb-api.jar`, `jaxb-impl.jar`, and `activation.jar`.

1.4.1 Deploying MapViewer in a WebLogic Server Environment

This section explains how to deploy MapViewer to WebLogic Server Version 10 or 10.3. (Deployment to earlier WebLogic versions has not been tested.) For the deployment:

- MapViewer can be deployed from an exploded directory, or from the EAR archive directly. This section shows how to deploy MapViewer from an exploded directory, but the overall process is similar for unexploded EAR deployment or read-only exploded EAR deployment.

  For direct EAR file deployment, skip Section 1.4.1.1, "Unpacking the MapViewer EAR Archive", and later in WebLogic Server console select the EAR file instead of an open directory for deployment.

- The WebLogic console is used in this section, although you could also use the WLS command line instead.

- A new WebLogic domain is created to host MapViewer. This approach is recommend because MapViewer is a resource-intensive application, and it is better to run it in a separate environment such as its own domain. However, it is also possible (although not recommended) to deploy MapViewer to an existing WebLogic domain.

The main steps for deploying MapViewer to WebLogic Server are the following:

1. Unpack the MapViewer EAR Archive.
2. Configure WebLogic Server.
3. Deploy and Start MapViewer in WebLogic Server.
4. As needed, use the MapViewer Administration Page.

1.4.1.1 Unpacking the MapViewer EAR Archive

You can deploy MapViewer from an exploded directory, that is, a directory where `mapviewer.ear` has already been unpacked.

You can unpack the mapveiwer.ear archive to any directory on the server where WebLogic is running. This directory will become the working folder of your MapViewer installation, in that MapViewer will (by default) read the configuration file from this location, and will save generated map images to a folder under this directory. It is recommended that the directory be a permanent (not temporary) one. It can be a shared directory if you want the same MapViewer binaries to be deployed to multiple WebLogic servers running on multiple hosts.

In the following instructions, assume that you have created a directory named `/ul/mapviewer` as the top MapViewer directory. (If you create another directory, adapt the instructions accordingly.) Follow these steps:

1. Copy `mapviewer.ear` into `/ul/mapviewer`.
2. If `/ul/mapviewer` is not already your current directory, go there.
3. Rename `mapviewer.ear` to `mapviewer1.ear`.
4. Create a subdirectory named `mapviewer.ear`.
5. Unpack `mapviewer1.ear` into `mapviewer.ear` (that is, into `/ul/mapviewer/mapviewer.ear`).
6. Go to `mapviewer.ear`.
7. Rename `web.war` to `web1.war`.
8. Create a subdirectory named `web.war`.
9. Unzip `web1.war` into `web.war` (that is, into `/ul/mapviewer/mapviewer.ear/web.war`).
10. Modify the Mapviewer configuration file (`/ul/mapviewer/mapviewer.ear/web.war/WEB-INF/conf/mapViewerConfig.xml`) as needed, such as to change its logging level or to add permanent data source definitions. You can also modify this configuration file at any time later.

MapViewer is now unpacked and configured. You must next ensure that WebLogic Server is properly configured for MapViewer, so that you will be able to deploy and run MapViewer in WebLogic Server.

1.4.1.2 Configuring WebLogic Server

To configure WebLogic Server, follow these steps:

1. Create a new WebLogic domain to host MapViewer by running the following script:

   ```
   $BEA_HOME/wlserver_10.0/common/bin/config.sh
   ```

   This script starts a configuration wizard. It is suggested that you name the administration user `weblogic`; although if you use a different name, you can specify it when you configure MapViewer. You will use the administration user to log in to the MapViewer Administration page.

2. Start the domain by running the following script:

   ```
   $BEA_HOME/user_projects/domains/map-domain/startWebLogic.sh
   ```

   where *map-domain* is the name of the domain that you created in step 1.

1.4.1.3 Deploying and Starting MapViewer in WebLogic Server

After the new domain is running, you can log in to its console to start deploying MapViewer. Follow these steps.

1. Log in to the console, which is typically accessed at:

   ```
   http://<host>:7001/console
   ```

   where *<host>* is the host name or IP address of the system running WebLogic server.

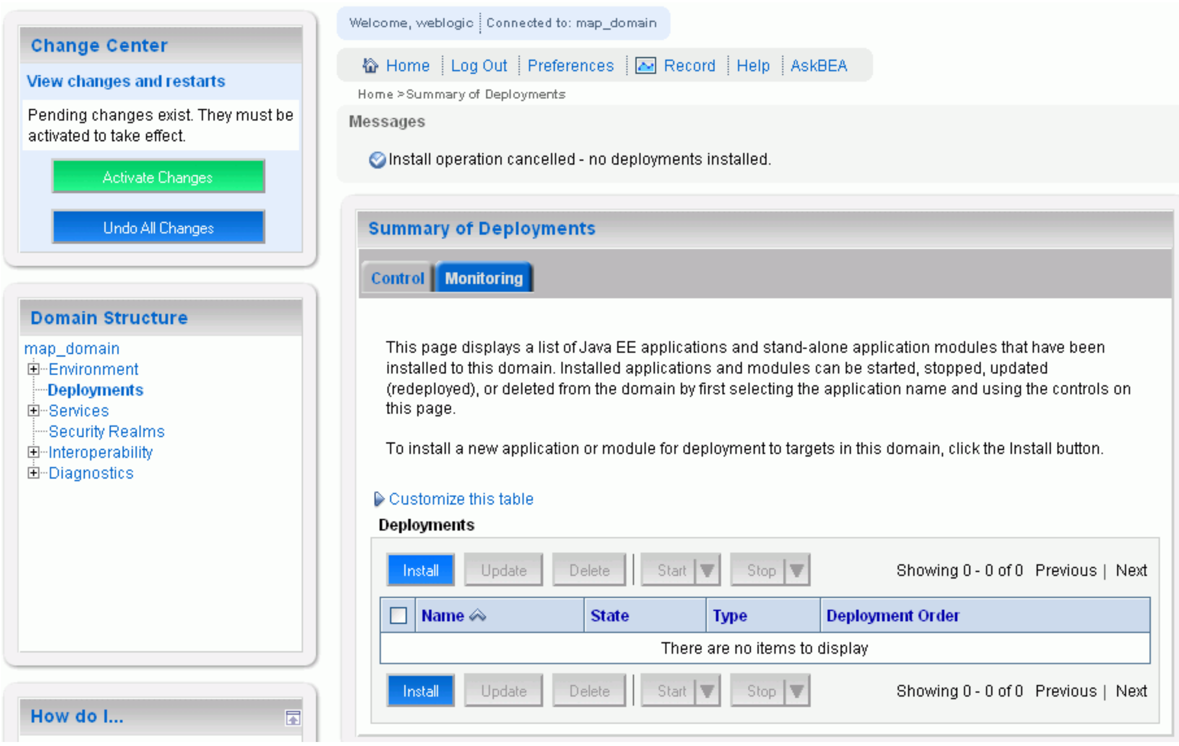
2. In the Change Center, if a **Lock & Edit** button is visible, click it.

   If a **Lock & Edit** button is not visible, go to the next step. If this button is not visible, it probably means that the WebLogic server has been configured with the Automatically Acquire Lock and Activate Changes option enabled.

3. Under Domain Structure, click **Deployments**.

   The administration console page will look similar to Figure 1–3.

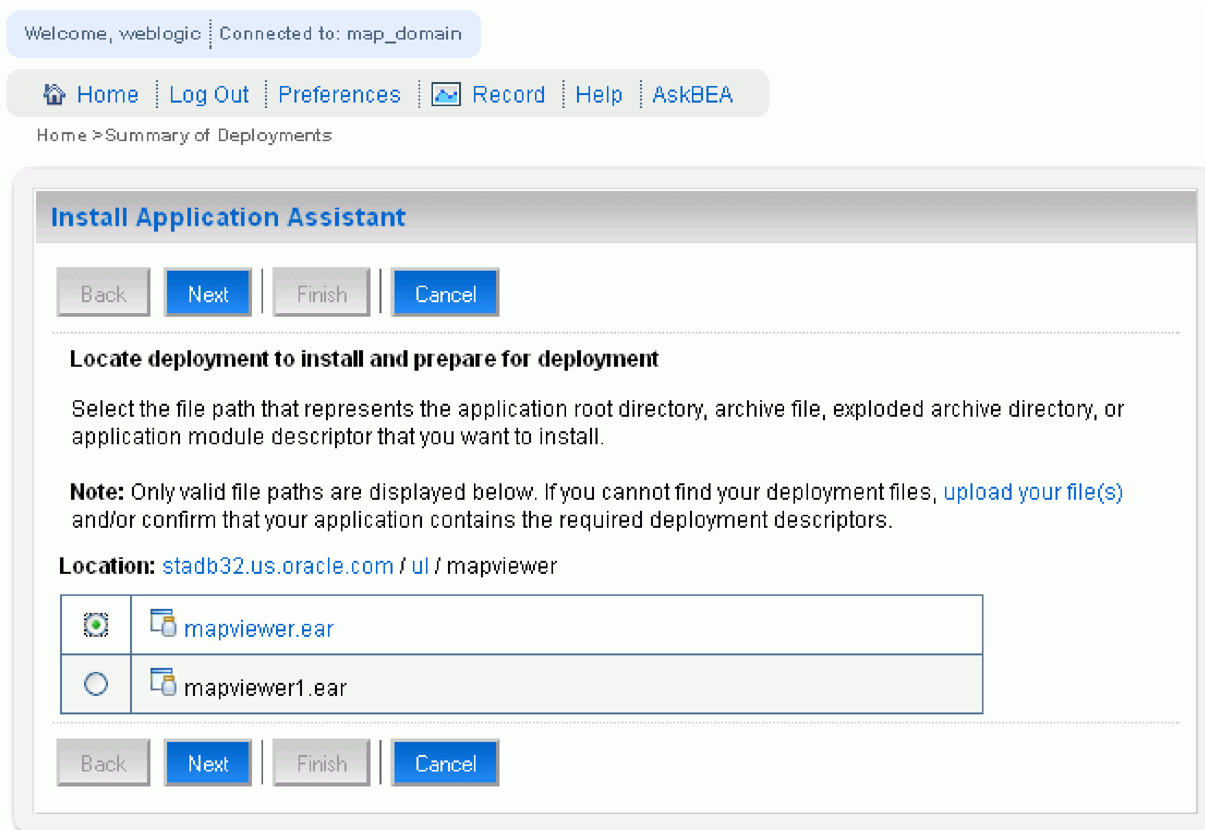
Figure 1–3 WebLogic Administration Console (Deployments)



4. Under Deployments, click **Install**.

   The next page is displayed, as shown in Figure 1–4. The location of the MapViewer directory (`/ul/mapviewer/mapviewer.ear` in this case) is the name of the directory, not the name of the `.ear` file.

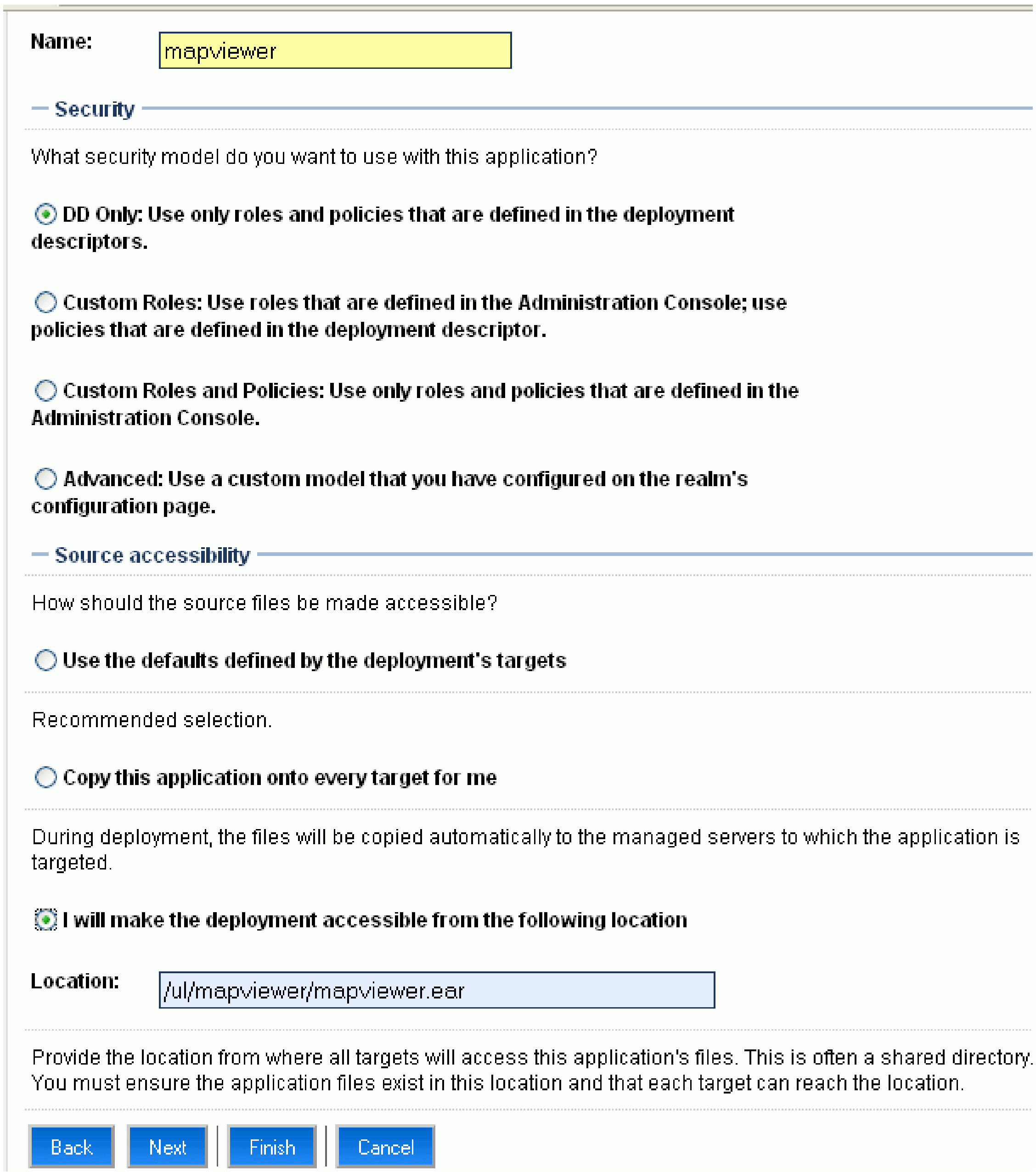
Figure 1–4 WebLogic Administration Console (Location)



5. Click **Next**.
6. Select **Install this deployment as an application**, and click **Next**.

   A page with the Source Accessibility section is displayed, as shown in Figure 1–5

Figure 1–5 WebLogic Administration Console (Source Accessibility)



7. In the Source Accessibility section, select **I will make the deployment accessible from the following location**.

   This option causes the unpacked MapViewer location to becomes the "working" directory of MapViewer. It also makes it easier if you want to upgrade MapViewer in the future, in which case you simply unpack the new `mapviewer.ear` file to this directory and restart WebLogic Server.

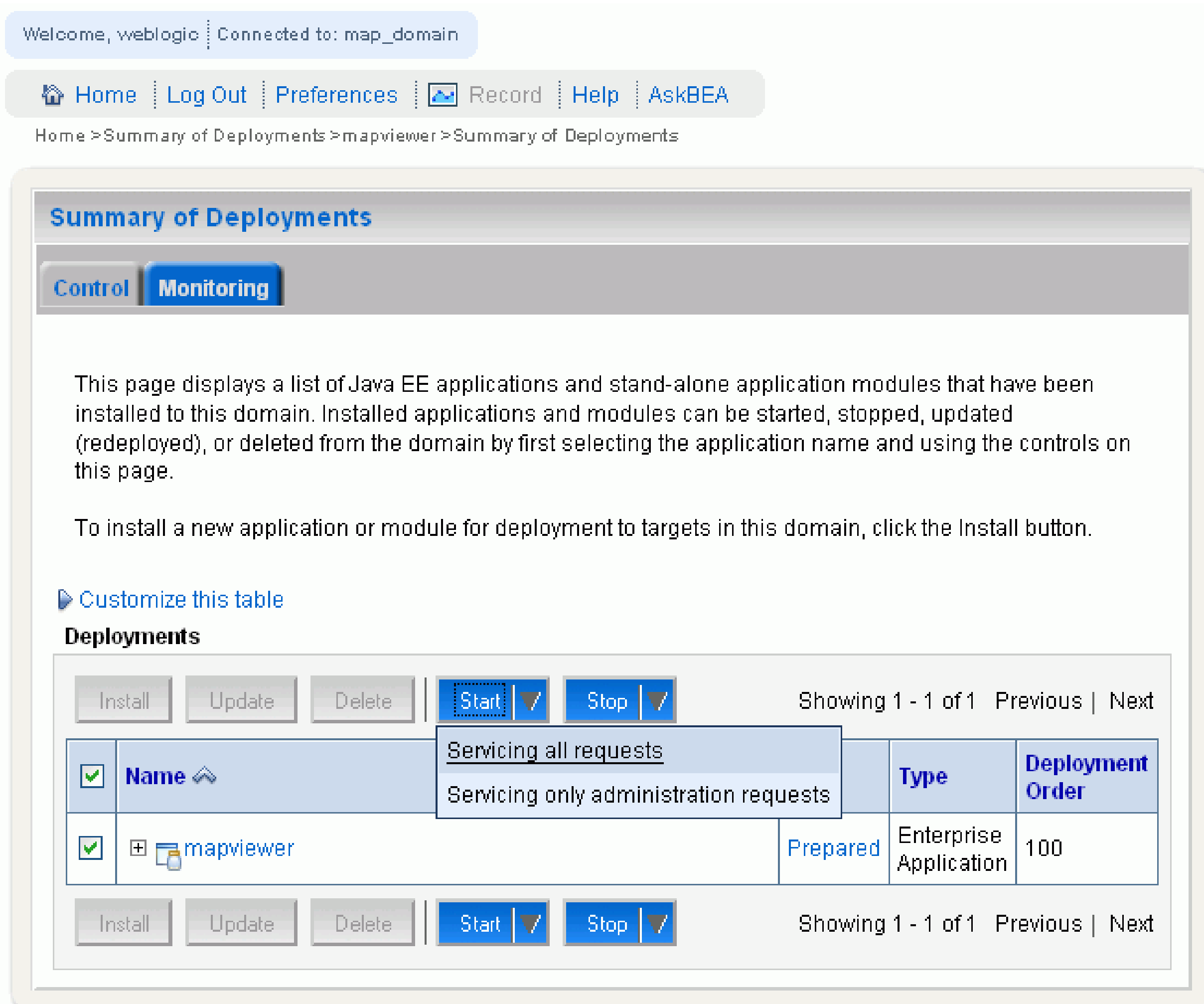
8. Click **Finish**, to start the deployment of MapViewer.

9. If the WebLogic server has been configured with the Automatically Acquire Lock and Activate Changes option enabled, skip the rest of this step and go to the next step when the deployment is finished.

If the WebLogic server has *not* been configured with the Automatically Acquire Lock and Activate Changes option enabled, when the deployment is finished, go to the Change Center, and click **Activate Changes** and then **Release Configuration** to complete the deployment process.

10. Start MapViewer by selecting **mapviewer** from Deployments, clicking **Start**, and selecting **Servicing all requests**, as shown in Figure 1–6

Figure 1–6 WebLogic Administration Console (Starting MapViewer)



11. Go to the following location to access MapViewer.

```
http://<host>:7001/mapviewer
```

where *<host>* is the host name or IP address of the system running WebLogic server.

1.4.1.4 Using the MapViewer Administration Page

When you first click the Admin button on the MapViewer home page, you are prompted for login information. You can use the default WebLogic administration account user name of `weblogic` to log in; however, if your WebLogic domain administration account uses a different user name, you must change the MapViewer

weblogic.xml file, located in $MAPVIEWER_HOME/mapviewer.ear/web.war/WEB-INF/.

To change the weblogic.xml file, open it is a text editor and replace the two occurrences of weblogic with the actual administration account name. The following except shows the lines with the name to be replaced:

```
<security-role-assignment>
    <role-name>map_admin_role</role-name>
    <principal-name>weblogic</principal-name>
</security-role-assignment>

<security-role-assignment>
    <role-name>secure_maps_role</role-name>
    <principal-name>weblogic</principal-name>
</security-role-assignment>
```

1.4.2 Non-Exploded and Read-Only Deployments of MapViewer

Starting with Release 11.1.1.7, MapViewer can be deployed to a Java Platform, Enterprise Edition (Java EE) server from an archived (zipped) EAR file or from an exploded (unzipped) EAR folder. When MapViewer is deployed from an exploded EAR folder, you can also make the folder read-only so that MapViewer does not create or change any files in the exploded EAR folder during run time. These types of deployments are supported in addition to the traditional deployment, which is based on an exploded EAR folder tree that is also writable to MapViewer at runtime.

To deploy and run MapViewer from an EAR file or a read-only exploded-EAR folder, you must specify an external mapViewerConfig.xml file, so that at run time MapViewer can find and load configuration parameters from this file instead of from the default file that is included in the MapViewer EAR archive's WEB-INF/conf/ directory.

There is no rule on where you should keep the external mapViewerConfig.xml file, but it is recommended that you designate a folder that serves as MapViewer's private folder, and store the mapViewerConfig.xml file in a subfolder named conf. This private folder should be outside any temporary deployment folders, and outside any other locations that might be overwritten during system install or upgrade. If the Java EE servers are running as a cluster, it is strongly recommended that the private folder for MapViewer be placed on a shared drive, so that all deployed MapViewer binaries can see and use it. This private folder should never be accessible to the public. Note that during runtime MapViewer also creates and modifies several folders and files inside the private folder, such as log files, temp files, and the folder for storing generated map tiles.

To tell MapViewer where to look for the external mapViewerConfig.xml file, you can choose one of the following methods:

- Method 1: Use a JVM Option
- Method 2: Use a <context-param> Element in web.xml
- Method 3: Use a Properties File in the Classpath

For all methods, you specify the absolute path to the file, and then MapViewer derives the private folder based on this path. As an example, assume that the external mapViewerConfig.xml file is stored as:

```
/scratch/_maps/conf/mapViewreConfig.xml
```

In this case, during MapViewer startup, it looks in the locations for the preceding methods *in the order listed*, and uses whichever `mapViewerConfig.xml` file it finds first. Using this example, when MapViewer finds the location, it uses `/scratch/_maps/` as the private folder.

If MapViewer still cannot find an external `mapViewerConfig.xml` file after trying locations for all of the preceding methods, it attempts to find the default `mapViewerConfig.xml` file in the EAR file's `WEB-INF/conf/` folder. Consequently, this will no longer be a read-only deployment, because MapViewer will be writing files into the exploded EAR folder just as in the traditional deployment.

For all methods in which an external `mapViewerConfig.xml` file is being used, you must also specify a public folder for MapViewer to save generated map images, as explained in Section 1.4.2.4, "Specifying a Public Folder for Generated Map Images".

1.4.2.1 Method 1: Use a JVM Option

This method specifies the location of the external `mapViewerConfig.xml` file using a JVM option, `oracle.maps.config`, which is typically added to the Java EE server startup script. For example, for WebLogic server you can add this option to the domain's `setDomainEvn.sh` script (right after the `-Djava.awt.headless=true` option):

```
EXTRA_JAVA_PROPERTIES=
"… -Doracle.maps.config=/scratch/_maps/conf/mapViewerConfig.xml … "
```

After you make the change, you must restart the server before attempting the MapViewer deployment.

1.4.2.2 Method 2: Use a <context-param> Element in web.xml

This method requires modifying the included `web.xml` file in the MapViewer EAR archive's `WEB-INF/` folder before deployment. In the `web.xml` file, add a new `<context-param>` tag right after the `<description>` tag to specify the location of the external config file. For example:

```
<context-param>
   <param-name>oracle.maps.config</param-name>
   <param-value>/scratch/_maps/conf/mapViewerConfig.xml</param-value>
</context-param>
```

1.4.2.3 Method 3: Use a Properties File in the Classpath

This method requires several steps, which you must perform before attempting the MapViewer deployment.

1. Create a text file named `config.properties` containing a single line referring to the location of the `mapViewerConfig.xml` file. For example:

   ```
   oracle.maps.config= /scratch/_maps/conf/mapViewerConfig.xml
   ```

2. Create an empty folder `oracle/` and a subfolder `maps/`, and save the just created file in the `maps/` subfolder. In other words, you should have a path like this:

   ```
   oracle/maps/config.properties
   ```

3. Create a JAR archive that contains this path and file, and run the `jar` command from inside the parent folder of `oracle`. For example:

   ```
   jar cvf maps_config.jar oracle
   ```

   The name of the JAR file can be anything you like.

4. Place this newly created jar file in the MapViewer classpath. For example, in WebLogic Server you can place this jar in the domain's `lib` folder.

1.4.2.4 Specifying a Public Folder for Generated Map Images

When an external `mapViewerConfig.xml` file is being used, you must also specify a public folder for MapViewer to save generated map images so that users and MapViewer client applications can access them over the web. To do this, specify an appropriate path attribute in the <save_images_at> element of the external `mapViewerConfig.xml` file.

Be sure that the <save_images_at> element is not commented out. (By contrast, in the traditional deployment you can often leave the <save_images_at> element commented out, in which case MapViewer uses the `images` folder of the exploded WAR file for saving generated map images.)

The specified path must not point to any location inside the MapViewer EAR archive or folder.

If you have multiple MapViewer instances running in a cluster, this public folder must be on a shared drive.

1.4.3 After Deploying MapViewer

After successfully deploying MapViewer to Oracle Fusion Middleware 10*g*R3, standalone OC4J, or WebLogic Server, you may want to verify whether it is actually working, as described in Section 1.4.3.1. It is also a good idea to become familiar with its web interface, particularly the administration pages.

You must also run at least one, and perhaps several, SQL scripts, as explained in Section 1.4.3.2.

1.4.3.1 Verifying That the Deployment Was Successful

To test if the MapViewer server has started correctly, point your browser to that OC4J instance. For example, if MapViewer is installed on a system named www.example.com and the HTTP port is 8888, enter the following URL to invoke the MapViewer server with a simple get-version request:

```
http://www.example.com:8888/mapviewer/omserver?getv=t
```

If MapViewer is running correctly, it should immediately send back a response text string indicating the version and build number, such as the following:

```
Ver11_1_1_7_B121108
```

The actual version and build number will reflect the version that you installed.

If the server has not been started and initialized correctly, there will be no response, or the message *500 internal server error* will be displayed.

If the response message includes wording like *MapServer is not ready. Please try again later*, it could mean that the MapViewer server is initializing, but the process will take some additional time (for example, because the system is slow or because multiple predefined data sources are specified in the configuration file and MapViewer is attempting to connect to these databases). In this case, you can wait at least a few seconds and try the preceding request again.

However, if you continue to get this response message, there may be a problem with the deployment. Check for any error messages, either in the OC4J console for a standalone OC4J deployment or in the redirected `output/errors` log file of the OC4J

instance for a full Oracle Fusion Middleware 10*g*R3 deployment. The following are common causes of this problem:

- On a UNIX or Linux operating system, the Java virtual machine (JVM) was not started with the `-Djava.awt.headless=true` option, and no `DISPLAY` environment variable is set. This causes the MapViewer server to fail because the server accesses the Java graphics library, which on UNIX and Linux systems relies on the X11 windowing system.
- You deployed the `mapviewer.ear` file to an incompatible version of Oracle WebLogic Server or Glassfish. Note that, unlike previous versions, MapViewer as of Release 11.1.1.7 will not work with standalone OC4J.

1.4.3.2 Running SQL Scripts

This section describes SQL scripts, one or more of which you must run while connected as the MDSYS user. For each script that you run, you must run it on each target Oracle database from which MapViewer will render spatial data.

MapViewer uses a set of system views to store necessary mapping metadata in a target database. A target database is a database with Oracle Spatial and Graph or Oracle Locator (Release 10*g* or later) installed and from which you want MapViewer to be able to render maps. MapViewer requires following system views:

- USER_SDO_MAPS
- USER_SDO_THEMES
- USER_SDO_STYLES
- USER_SDO_CACHED_MAPS

The USER_SDO_CACHED_MAPS view is used by the Oracle Maps feature. It stores definitions of map tile cache instances. You must create this view manually by running the following script while connected as the SYS user:

`$MV_HOME/WEB-INF/admin/mcsdefinition.sql`

If the target database is release 9.2 or later, the other three views (USER_SDO_MAPS, USER_SDO_THEMES, and USER_SDO_STYLES) are created and populated automatically. However, if the target database has a release number lower than 9.2, you must manually create and populate these views by running the following scripts while connected as the MDSYS user:

`$MV_HOME/WEB-INF/admin/mapdefinition.sql`

`$MV_HOME/WEB-INF/admin/defaultstyles.sql`

1.4.3.3 Creating MapViewer Array Types, if Necessary

For each database schema that it connects to, MapViewer checks for the existence of the following SQL array types that support array-type binding variables that might exist in some predefined themes:

- MV_STRINGLIST
- MV_NUMBERLIST
- MV_DATELIST

If these types do not exist, MapViewer attempts to create them in the database schema associated with the MapViewer data source. However, if the user associated with that schema does not have sufficient privileges to create new types, a privileged user must

create the types by connecting to the data source schema and entering the following statements:

```
CREATE or REPLACE type MV_STRINGLIST as TABLE of VARCHAR2(1000);
CREATE or REPLACE type MV_NUMBERLIST as TABLE of NUMBER;
CREATE or REPLACE type MV_DATELIST as TABLE of DATE;
```

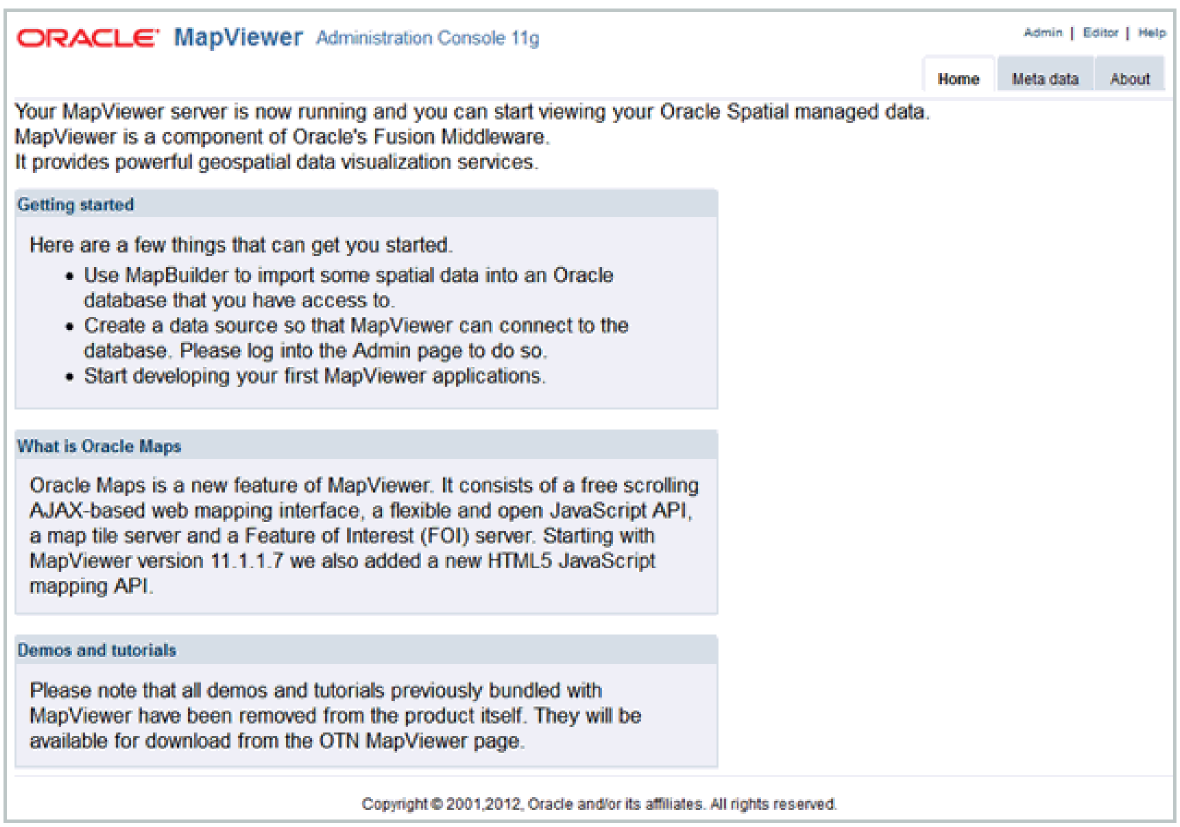
1.5 Administering MapViewer

This section introduces the MapViewer Administration page and some administrative and configuration tasks that you can perform, such as adding new data sources, managing map tile layers used by Oracle Maps, and setting logging levels.

1.5.1 Logging in to the MapViewer Administration Page

After you have verified that MapViewer is running properly, it is suggested that you log in to the MapViewer Administration page. To do this, go first to the MapViewer Welcome page, which is typically `http://`*`<host>:<port>`*`/mapviewer`, where *`<host>`* and *`<port>`* should be replaced by the correct value for your installation. Figure 1–7 shows the MapViewer Welcome page

Figure 1–7 MapViewer Welcome Page



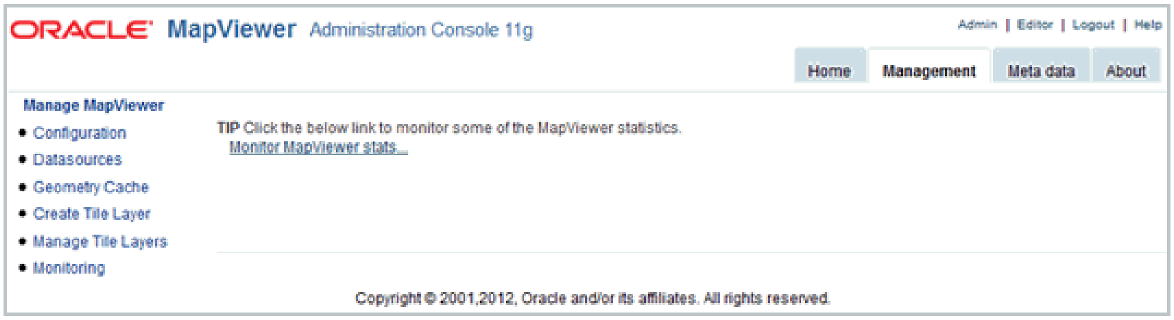
Click the **Admin** icon at the top right. A login prompt is displayed, asking for user name and password for the MapViewer administration page.

User Name: Enter `admin`.

Password: Enter the password that you use to log in to the Server Control page of the WebLogic Server of Glassfish installation.

After you log in, the MapViewer administration page is displayed (on the Management tab), as shown in Figure 1–8.

Figure 1–8 MapViewer Administration Page



You can use this page to perform administrative tasks, such as configuring MapViewer to your site's specific requirements, adding predefined data sources so that MapViewer will automatically connect to the specified target database whenever it is started, and managing map tile layers. For detailed about configuration tasks, see Section 1.5.2; for information about administrative tasks, see Section 1.5.3.

1.5.2 Configuring MapViewer

If the default configuration settings for running MapViewer are not adequate, you can configure MapViewer by editing the MapViewer configuration file, `mapViewerConfig.xml`, which is located in the *`$MAPVIEWER_HOME`*`/WEB-INF/conf` directory. To modify this file, you can use a text editor, or you can use the MapViewer Administration page.

After you modify this file, you must restart the container to have the changes take effect; however, you can instead use the MapViewer Administration page to restart only the MapViewer servlet (instead of the entire Java EE instance, which may have other applications deployed and running) if you installed MapViewer with a standalone Glassfish instance.

If you deployed MapViewer to a WebLogic Server instance with multiple processes (thus with multiple physical JVMs on the same host), or if you deployed to a WebLogic Server instance that is in a clustered island (with multiple WLS instances running on multiple hosts), you must restart the WebLogic Server instance itself for the changes to the MapViewer configuration file to take effect in all MapViewer servers. In the latter case (clustered WebLogic Server instances), you may also need to modify the MapViewer configuration file in the MapViewer directory hierarchy for each host's WebLogic Server instance in the cluster. For more information about repository-based middle-tier clustering, see *Oracle Fusion Middleware High Availability Guide*.

The MapViewer configuration file defines the following information in XML format:

- Logging information, defined either through container-controlled logging (recommended) or in the `<logging>` element (see Section 1.5.2.1)
- Map image file information, defined in the `<save_images_at>` element (see Section 1.5.2.2)
- Administrative request restrictions, defined in the `<ip_monitor>` element (see Section 1.5.2.3)

- Web proxy information for accessing external information across a firewall, defined in the <web_proxy> element (see Section 1.5.2.4)
- Global map "look and feel" configuration, defined in the <global_map_config> element (see Section 1.5.2.5)
- Internal spatial data cache settings, defined in the <spatial_data_cache> element (see Section 1.5.2.6)
- Custom image renderer registration, defined in the <custom_image_renderer> element (see Appendix C)
- Permanent map data sources, defined in the <map_data_source> element (see Section 1.5.2.14)
- Security configurations, defined in the <security_config> element
- WMS services configurations, defined in the <wms_config> element
- External attribute data provider registration, defined in <ns_data_provider> elements
- Map tile server configurations, defined in the <map_tile_server> element

All path names in the `mapViewerConfig.xml` file are relative to the directory in which the file is stored, unless otherwise specified.

Example 1–1 shows a sample `mapViewerConfig.xml` file.

Example 1–1 Sample MapViewer Configuration File

```
<?xml version="1.0" ?>
<!-- This is the configuration file for MapViewer. -->
<!-- Note: All paths are resolved relative to this directory (where
           this config file is located), unless specified as an absolute
           path name.
 -->

<MapperConfig>

  <!-- ****************************************************************** -->
  <!-- ************************ Logging Settings ************************ -->
  <!-- ****************************************************************** -->

  <!-- Uncomment the following to modify logging. Possible values are:
       log_level = "fatal"|"error"|"warn"|"info"|"debug"|"finest"
             default: info) ;
       log_thread_name = "true" | "false" ;
       log_time = "true" | "false" ;
       one or more log_output elements.
  -->
  <!--
    <logging log_level="info" log_thread_name="false"
             log_time="true">
       <log_output name="System.err" />
       <log_output name="../log/mapviewer.log" />
    </logging>
  -->

  <!-- ****************************************************************** -->
  <!-- ********************** Map Image Settings ************************ -->
  <!-- ****************************************************************** -->
```

```
<!-- Uncomment the following only if you want generated images to
     be stored in a different directory, or if you want to customize
     the life cycle of generated image files.

     By default, all maps are generated under
     $ORACLE_HOME/lbs/mapviewer/web/images.

     Images location-related attributes:
     file_prefix: image file prefix, default value is "omsmap"
     url:  the URL at which images can be accessed. It must match the 'path'
           attribute below. Its default value is "%HOST_URL%/mapviewer/images"
     path: the corresponding path in the server where the images are
           saved; default value is "%ORACLE_HOME%/lbs/mapviewer/web/images"

     Images life cycle-related attributes:
     life: the life period of generated images, specified in minutes.
           If not specified or if the value is 0, images saved on disk will
           never be deleted.
     recycle_interval:  this attribute specifies how often the recycling
           of generated map images will be performed. The unit is minute.
           The default interval (when not specified or if the value is 0)
           is 8*60, or 8 hours.

 -->
<!--
 <save_images_at  file_prefix="omsmap"
                 url="http://mypc.mycorp.com:8888/mapviewer/images"
                 path="../web/images"
 />
-->

<!-- ****************************************************************** -->
<!-- ******************** IP Monitoring Settings ******************** -->
<!-- ****************************************************************** -->

<!-- Uncomment the following to enable IP filtering for administrative
      requests.
  Note:
  - Use <ips> and <ip_range> to specify which IPs (and ranges) are allowed.
    Wildcard form such as 20.* is also accepted. Use a comma-delimited
    list in <ips>.

  - Use <ips_exclude> and <ip_range_exclude> for IPs and IP ranges
    prohibited from accessing eLocation.

  - If an IP falls into both "allowed" and "prohibited" categories, it is
    prohibited.

  - If you put  "*" in an <ips> element, then all IPs are allowed, except
    those specified in <ips_exclude> and <ip_range_exclude>.
    On the other hand, if you put "*" in an <ips_exclude> element, no one
    will be able to access MapViewer (regardless of whether an IP is in
    <ips> or <ip_range>).

  - You can have multiple <ips>, <ip_range>, <ips_exclude>, and
    <ip_range_exclude> elements under <ip_monitor>.

  - If no <ip_monitor> element is present in the XML configuration
    file, then no IP filtering will be performed (all allowed).
```

```
  - The way MapViewer determines if an IP is allowed is:

        if(IP filtering is not enabled) then allow;
        if(IP is in exclude-list) then not allow;
        else if(IP is in allow-list) then allow;
        else not allow;
 -->

<!--
   <ip_monitor>
        <ips> 138.1.17.9, 138.1.17.21, 138.3.*, 20.* </ips>
        <ip_range> 24.17.1.3 - 24.17.1.20 </ip_range>
        <ips_exclude> 138.3.29.* </ips_exclude>
        <ip_range_exclude>20.22.34.1 - 20.22.34.255</ip_range_exclude>
   </ip_monitor>
 -->

<!-- ****************************************************************** -->
<!-- ********************** Web Proxy Setting  ************************ -->
<!-- ****************************************************************** -->
<!-- Uncomment and modify the following to specify the web proxy setting.
     This is only needed for passing background image URLs to
     MapViewer in map requests or for setting a logo image URL, if
     such URLs cannot be accessed without the proxy.
 -->

<!--
  <web_proxy host="www-proxy.my_corp.com"  port="80" />
-->

<!-- ****************************************************************** -->
<!-- *********************** Security Configuration ******************* -->
<!-- ****************************************************************** -->
<!-- Here you can set various security related configurations of MapViewer.
-->

<security_config>
  <disable_direct_info_request> false </disable_direct_info_request>
</security_config>

<!-- ****************************************************************** -->
<!-- *********************** Global Map Configuration ***************** -->
<!-- ****************************************************************** -->
<!-- Uncomment and modify the following to specify systemwide parameters
     for generated maps. You can specify your copyright note, map title, and
     an image to be used as a custom logo shown on maps. The logo image must
     be accessible to this MapViewer and in either GIF or JPEG format.
     Notes:
       -  To disable a global note or title, specify an empty string ("") for
          the text attribute of <note> and <title> element.
       - position specifies a relative position on the map where the
               logo, note, or title  will be displayed. Possible values are
               NORTH, EAST, SOUTH, WEST, NORTH_EAST, SOUTH_EAST,
               SOUTH_WEST, NORTH_WEST, and CENTER.
       - image_path specifies a file path or a URL (starts with "http://")
                 for the image.

     <rendering> element attributes:
     - Local geodetic data adjustment: If allow_local_adjustment="true",
       MapViewer automatically performs local data
```

```
        "flattening" with geodetic data if the data window is less than
        3 decimal degrees. Specifically, MapViewer performs a simple
        mathematical transformation of the coordinates using a tangential
        plane at the current map request center.
        If allow_local_adjustment="false" (default), no adjustment is
        performed.
      - Automatically applies a globular map projection (geodetic data only):
        If use_globular_projection="true", MapViewer will
        apply a globular projection on the fly to geometries being displayed.
        If use_globular_projection="false" (the default), MapViewer does no map
        projection to geodetic geometries. This option has no effect on
        non-geodetic data.
 -->

<!--
  <global_map_config>
      <note text="Copyright 2009, Oracle Corporation"
            font="sans serif"
            position="SOUTH_EAST"/>
      <title  text="MapViewer Demo"
              font="Serif"
              position="NORTH" />
      <logo image_path="C:\\images\\a.gif"
            position="SOUTH_WEST" />

      <rendering allow_local_adjustment="false"
                 use_globular_projection="false" />
  </global_map_config>
-->

<!-- ****************************************************************** -->
<!-- ****************** Spatial Data Cache Setting  ******************* -->
<!-- ****************************************************************** -->
<!-- Uncomment and modify the following to customize the spatial data cache
     used by MapViewer. The default is 64 MB for in-memory cache.

     To disable the cache, set max_cache_size to 0.

     max_cache_size:  Maximum size of in-memory spatial cache of MapViewer.
                      Size must be specified in megabytes (MB).
     report_stats:    If you would like to see periodic output of cache
                      statistics, set this attribute to true. The default
                      is false.
 -->

<!--
  <spatial_data_cache   max_cache_size="64"
                        report_stats="false"
  />
-->

<!-- ****************************************************************** -->
<!-- ******************* Custom Image Renderers ********************* -->
<!-- ****************************************************************** -->
<!-- Uncomment and add as many custom image renderers as needed here,
     each in its own  <custom_image_renderer> element. The "image_format"
     attribute specifies the format of images that are to be custom
     rendered using the class with full name specified in "impl_class".
     You are responsible for placing the implementation classes in the
     MapViewer's classpath.
```

```
-->
<!--
<custom_image_renderer image_format="ECW"
                       impl_class="com.my_corp.image.ECWRenderer" />
-->

<!-- **************************************************************** -->
<!-- ***************** Custom WMS Capabilities Info ***************** -->
<!-- **************************************************************** -->
<!-- Uncomment and modify the following tag if you want MapViewer to
     use the following information in its getCapabilities response.
     Note: all attributes and elements of <wms_config> are optional.
-->
<!--
<wms_config host="www.my_corp.com" port="80">
  <title>
      WMS 1.1 interface for Oracle Mapviewer
  </title>
  <abstract>
      This WMS service is provided through MapViewer.
  </abstract>
  <keyword_list>
     <keyword>bird</keyword>
     <keyword>roadrunner</keyword>
     <keyword>ambush</keyword>
  </keyword_list>
  <sdo_epsg_mapfile>
    ../config/epsg_srids.properties
  </sdo_epsg_mapfile>
</wms_config>
-->

<!-- **************************************************************** -->
<!-- *************** Custom Non-Spatial Data Provider *************** -->
<!-- **************************************************************** -->
<!-- Uncomment and add as many custom non-spatial data provider as
     needed here, each in its own <ns_data_provider> element.
     You must provide the id and full class name here. Optionally you
     can also specify any number of global parameters, which MapViewer
     will pass to the data provider implementation during initialization.
     The name and value of each parameter is interpreted only by the
     implementation.
-->

<!-- this is the default data provider that comes with MapViewer; please
     refer to the MapViewer User's Guide for instructions on how to use it.

<ns_data_provider
  id="defaultNSDP"
  class="oracle.sdovis.NSDataProviderDefault"
/>
-->

<!-- this is a sample NS data provider with prameters:
<ns_data_provider
  id="myProvider1" class="com.mycorp.bi.NSDataProviderImpl" >

  <parameters>
    <parameter name="myparam1" value="value1" />
    <parameter name="p2"       value="v2"     />
```

```
    </parameters>

  </ns_data_provider>
  -->

  <!-- ****************************************************************** -->
  <!-- *******************  Map Tile Server Setting  ****************** -->
  <!-- ****************************************************************** -->
  <!-- Uncomment and modify the following to customize the map tile server.

       <tile_storage> specifies the default root directory under which the
       cached tile images are to be stored if the cache instance configuration
       does not specify the root directory for the cache instance. If the
       default root directory is not set or not valid, the default root
       direcotry will be set to be $MAPVIEWER_HOME/web/tilecache

          default_root_path:  The default root directory under which the cached
                              tile images are stored.
  -->

  <!--
     <map_tile_server>
       <tile_storage default_root_path="/scratch/tilecachetest/"/>
    </map_tile_server>
  -->

  <!-- ****************************************************************** -->
  <!-- ******************* Predefined Data Sources  ******************* -->
  <!-- ****************************************************************** -->
  <!-- Uncomment and modify the following to predefine one or more data
       sources.
       Note: You must precede the jdbc_password value with a '!'
             (exclamation point), so that when MapViewer starts the next
             time, it will encrypt and replace the clear text password.
  -->

  <!--
  <map_data_source name="mvdemo"
                   jdbc_host="elocation.us.oracle.com"
                   jdbc_sid="orcl"
                   jdbc_port="1521"
                   jdbc_user="scott"
                   jdbc_password="!password"
                   jdbc_mode="thin"
                   number_of_mappers="3"
   />
   -->

</MapperConfig>
```

1.5.2.1 Specifying Logging Information

MapViewer provides a flexible logging mechanism to record runtime information and events. You can configure the granularity, volume, format, and destination of the log output. You can also configure the maximum size of log files as well as automatic log file rotation.

There are two ways to configure MapViewer's logging, the container-controlled approach and legacy logging using the <logging> element in the configuration file:

- Container-controlled logging: Use Oracle Fusion Middleware 10*g*R3 Control if MapViewer is deployed to an Oracle Fusion Middleware 10*g*R3 instance, or directly edit the `$OC4J_HOME/j2ee/home/config/j2ee-logging.xml` file if MapViewer is deployed to a standalone OC4J instance. This approach takes full advantage of the Fusion Middleware 10*g*R3 diagnostic logging mechanisms and allows such advanced features such as maximum log file size and log file rotation.
- Legacy logging: Involves using the `<logging>` element in the `mapViewerConfig.xml` file. When MapViewer is deployed to WebLogic Server, legacy logging is the only supported way of configuring MapViewer logging behavior.

Container-Controlled Logging

> **Note:** For container-controlled logging to work, you must comment out or remove the `<logging>` element in the `mapViewerConfig.xml` file. By default that element is commented out (disabled), so that container-controlled logging settings will function properly. If you enable the `<logging>` element (even if you make no other changes to its attributes), then the container-controlled logging settings are ignored by MapViewer.

To configure MapViewer logging when it is deployed to an OC4J 11*g* standalone instance, edit the `$OC4J_HOME/j2ee/home/config/j2ee-logging.xml` file. For example, the following code in that file logs all messages from MapViewer at the FINEST level to the default OC4J log file (`j2ee/home/log/oc4j/diagnostic.log`):

```
<log_handler name='oc4j-handler'
class='oracle.core.ojdl.logging.ODLHandlerFactory'>
      <property name='path' value='../log/oc4j'/>
      <property name='maxFileSize' value='10485700'/>
      <property name='maxLogSize' value='1048576'/>
      <property name='encoding' value='UTF-8'/>
      <property name='supplementalAttributes' value='J2EE_APP.name,J2EE_
MODULE.name,WEBSERVICE.name,WEBSERVICE_PORT.name'/>
 </log_handler>
```

The preceding code defines the default OC4J log handler. It specifies where the log file will be saved, its maximum file size, and other information. A log handler like this can be associated with multiple actual loggers that are created by OC4J components and applications (such as MapViewer).

The following example associates a MapViewer logger, in this case one that is responsible for generating all internal log messages, with the preceding log handler:

```
<logger name="oracle.mapviewer.logger" level="FINEST" useParentHandlers='false'>
    <handler name='oc4j-handler'/>
</logger>
```

The preceding example tells OC4J that all log records produced by the logger named `oracle.mapviewer.logger` should be handled by the log handler named `oc4j-handler`. It sets the logging level to FINEST so that all messages generated by MapViewer will be visible in the log file. The possible logging levels supported here are the following standard Java logging levels: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, and FINEST.

The following loggers are used by MapViewer for container-controlled logging:

- `oracle.mapviewer.logger` is used by all server side components of MapViewer to generate diagnostic records.
- `oracle.mapviewer.access` is used by MapViewer for logging only user access records.

The preceding example associated an existing log handler named `oc4j-handler`, which is already defined in the `j2ee-logging.xml` file. You can also define your own log handler in the `j2ee-logging.xml` file and specify a different log file location and name, as well as the maximum file size and the file rotation. The following example creates a new log handler to store only MapViewer access records:

```
<log_handler name='mv-handler' class='oracle.core.ojdl.logging.ODLHandlerFactory'>
      <property name='path' value='../log/mapaccess/access.log'/>
      <property name='maxFileSize' value='600000'/>
      <property name='maxLogSize' value='10000'/>
      <property name='format' value='ODL-TEXT'/>
      <property name='encoding' value='UTF-8'/>
      <property name='supplementalAttributes' value='J2EE_APP.name'/>
</log_handler>
```

The following example associates this new log handler to the MapViewer access logger named `oracle.mapviewer.access`:

```
<logger name='oracle.mapviewer.access' level='FINEST' useParentHandlers='false'>
    <handler name='mv-handler'/>
</logger>
```

Note that the level must be FINEST or FINER in order for the access log messages to appear in the log file. Now, if you restart OC4J and make map requests, you should see a new log file (`access.log`) in the OC4J `log/mapaccess` directory that contains records of users accessing MapViewer.

For more information about logging configuration, specifically how to configure logging using Fusion Middleware 10*g*R3 Control, see *Oracle Containers for J2EE Configuration and Administration Guide*

Legacy Logging

If you do not use container-controlled logging, you can use the legacy approach, which is to uncomment-out and modify the `<logging>` element in the MapViewer configuration file.

You can specify the following information as attributes or subelements of the `<logging>` element:

- The `log_level` attribute controls the levels of information that are recorded in the log, which in turn affect the log output volume. Set the `log_level` attribute value to one of the following, listed from most restrictive logging to least restrictive logging: `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `FINEST`. The `FATAL` level outputs the least log information (only unrecoverable events are logged), and the other levels are progressively more inclusive, with the `FINEST` level causing the most information to be logged. For production work, a level of `WARN` or more restrictive (`ERROR` or `FATAL`) is recommended; however, for debugging you may want to set a less restrictive level.
- The `log_thread_name` attribute controls whether or not to include the name of the thread that encountered and logged the event.

- The log_time attribute controls whether or not the current time is included when a logging event occurs.
- The log_output subelement identifies output for the logging information. By default, log records are written to the system error console. You can change this to the system output console or to one or more files, or some combination. If you specify more than one device through multiple log_output subelements, the logging records are sent to all devices, using the same logging level and attributes.

1.5.2.2 Specifying Map File Storage and Life Cycle Information

Map image file information is specified in the <save_images_at> element. By default, images are stored in the *$ORACLE_HOME* /lbs/mapviewer/web/images directory. You do not need to modify the <save_images_at> element unless you want to specify a different directory for storing images.

A mapping client can request that MapViewer send back the URL for an image file instead of the actual map image data, by setting the format attribute of the <map_request> element (described in Section 3.2.1.1) to GIF_URL or PNG_URL. In this case, MapViewer saves the requested map image as a file on the host system where MapViewer is running and sends a response containing the URL of the image file back to the map client.

You can specify the following map image file information as attributes of the <save_images_at> element:

- The file_prefix attribute identifies the map image file prefix. A map image file name will be a fixed file prefix followed by a serial number and the image type suffix. For example, if the map image file prefix is omsmap, a possible GIF map image file could be omsmap1.gif.

  Default value: file_prefix=omsmap

- The url attribute identifies the map image base URL, which points to the directory under which all map image files are saved on the MapViewer host. The map image URL sent to the mapping client is the map image base URL plus the map image file name. For example, if the map image base URL is http://dev04.example.com:1521/mapviewer/images, the map image URL for omsmap1.gif will be http://dev04.example.com:1521/mapviewer/images/omsmap1.gif.

  Default value: url=*$HOST_URL*/mapviewer/images

- The path attribute identifies the path of the directory where all map image files are saved on the MapViewer host system. This directory must be accessible by HTTP and must match the map image URL. Map image files saved in the directory specified by the path attribute should be accessible from the URL specified by the url attribute.

  However, if you are deploying MapViewer to WebLogic Server, the default value for the path attribute (../web/images) is not correct. The path attribute value in this case should be ../../images, because the physical "images" directory is mapviewer.ear/web.war/images; so using relative path, the value should be ../../images for the path attribute to resolve to the physical directory.

- The life attribute specifies the number of minutes that a generated map image is guaranteed to stay on the file system before the image is deleted. If the life attribute is specified, the recycle_interval attribute controls how frequently MapViewer checks for possible files to delete.

  Default: MapViewer never deletes the generated map images.

- The `recycle_interval` attribute specifies the number of minutes between times when MapViewer checks to see if it can delete any image files that have been on the file system longer than the number of minutes for the `life` attribute value.

  Default value: `480` (8 hours)

1.5.2.3 Restricting Administrative (Non-Map) Requests

In addition to map requests, MapViewer accepts administrative (non-map) requests, such as requests to list all data sources and to add and delete data sources. (Chapter 7 describes the administrative requests.) By default, all MapViewer users are permitted to make administrative requests.

However, if you want to restrict the ability to submit administrative requests, you can edit the MapViewer configuration file to allow administrative requests only from users with specified IP addresses.

To restrict administrative requests to users at specified IP addresses, add the `<ip_monitor>` element to the MapViewer configuration file (or uncomment and modify an existing element, if one is commented out). Example 1–2 shows a sample `<ip_monitor>` element excerpt from a configuration file.

Example 1–2 Restricting Administrative Requests

```
<MapperConfig>
   . . .
   <ip_monitor>
      <ips> 138.1.17.9, 138.1.17.21, 138.3.*, 20.* </ips>
      <ip_range> 24.17.1.3 - 24.17.1.20 </ip_range>
      <ips_exclude> 138.3.29.* </ips_exclude>
      <ip_range_exclude>20.22.34.1 - 20.22.34.255</ip_range_exclude>
   </ip_monitor>
   . . .
</MapperConfig>
```

In Example 1–2:

- The following IP addresses are explicitly included as able to submit administrative requests (unless excluded by an `<ips_exclude>` element): 138.1.17.9, 138.1.17.21, all that start with 138.3., all that start with 20., and all in the range (inclusive) of 24.17.1.3 to 24.17.1.20.
- The following IP addresses are explicitly excluded from submitting administrative requests: all starting with 138.3.29., and all in the range (inclusive) of 20.22.34.1 to 20.22.34.255.
- All other IP addresses that are not explicitly included cannot submit administrative requests.

Syntax notes for the `<ip_monitor>` element:

- Use `<ips>` and `<ip_range>` elements to specify which IP addresses (and ranges) are allowed. Asterisk wildcards (such as `20.*`) are acceptable. Use a comma-delimited list for addresses.
- Use `<ips_exclude>` and `<ip_range_exclude>` elements to exclude IP addresses and address ranges from submitting administrative requests. If an address falls into both the included and excluded category, it is excluded.
- If you specify the asterisk wildcard in an `<ips>` element, all associated IP addresses are included except any specified in `<ips_exclude>` and `<ip_range_exclude>` elements.

1.5.2.4 Specifying a Web Proxy

Sometimes the MapViewer server needs to make HTTP connections to external web servers, such as to obtain a background image through a URL or to contact an external WMS server to fetch its map images. In such cases, if there is a firewall between the MapViewer server and the target web server, you may need to specify the HTTP proxy information to MapViewer so that it will not be blocked by the firewall. The following example specifies web proxy information:

```
<web_proxy host="www-proxy.mycorp.com" port="80" />
```

1.5.2.5 Specifying Global Map Configuration Options

You can specify the following global "look and feel" options for the display of each map generated by MapViewer:

- Title
- Note (such as a copyright statement or a footnote)
- Logo (custom symbol or corporate logo)
- Local geodetic data adjustment
- Splitting geometries along the 180 meridian

To specify any of these options, use the `<global_map_config>` element. For example:

```
<global_map_config>
    <note text="Copyright (c) 2009, Example Corporation"
              font="sans serif"
              position="SOUTH_EAST"/>
    <title  text="Map Courtesy of Example Corp."
              font="Serif"
              position="NORTH"/>
    <logo image_path="C:\\images\\a.gif"
              position="SOUTH_WEST"/>

    <rendering allow_local_adjustment="false"
               use_globular_projection="false"/>
</global_map_config>
```

Set the map title through the `<title>` element of the `<global_map_config>` element. You can also set the map title in an individual map request by specifying the `title` attribute with the `<map_request>` element, and in this case, the title in the map request is used instead of the global title in the MapViewer configuration file. Note the following information about the attributes of the `<title>` element:

- The `text` attribute specifies the title string.
- The `font` attribute specifies a font. The font must exist on the system where MapViewer is running.
- The `position` attribute provides a positioning hint to MapViewer when determining where the map title will be drawn on a map. Possible values are: `NORTH`, `EAST`, `SOUTH`, `WEST`, `NORTH_EAST`, `SOUTH_EAST`, `SOUTH_WEST`, `NORTH_WEST`, and `CENTER`.

  Default value: `NORTH`

Set the map note through the `<note>` element of the `<global_map_config>` element. Note the following information about the attributes of the `<note>` element:

- The `text` attribute specifies the note string.

- The font attribute specifies a font. The font must exist on the system where MapViewer is running.
- The position attribute provides a positioning hint to MapViewer when determining where the map note will be drawn on a map. Possible values are: NORTH, EAST, SOUTH, WEST, NORTH_EAST, SOUTH_EAST, SOUTH_WEST, NORTH_WEST, and CENTER.

  Default value: SOUTH_EAST

Set the map logo through the <logo> element of the <global_map_config> element. The map logo image must be in either JPEG or GIF format. The image can be stored in a local file system where the MapViewer instance will have access to it, or it can be obtained from the web by specifying its URL. To specify a map logo, uncomment the <map_logo> element in the MapViewer configuration file and edit its attributes as needed.

Note the following information about the attributes of the <logo> element:

- The image_path attribute must specify a valid file path name, or a URL starting with http://.
- The position attribute provides a positioning hint to MapViewer when determining where the map logo will be drawn on a map. Possible values are: NORTH, EAST, SOUTH, WEST, NORTH_EAST, SOUTH_EAST, SOUTH_WEST, NORTH_WEST, and CENTER.

  Default value: SOUTH_WEST

If the logo image is obtained through a URL that is outside your firewall, you may need to set the web proxy in order for MapViewer to retrieve the logo image. For information about specifying a web proxy, see Section 1.5.2.4.

If you also specify a map legend, be sure that its position is not the same as any position for a map title, note, or logo. (Map legends are explained in Section 2.4.2 and Section 3.2.11. The default position for a map legend is SOUTH_WEST.)

To have MapViewer automatically project geodetic data to a local non-geodetic coordinate system before displaying it if the map data window is less than 3 decimal degrees, specify allow_local_adjustment="true" in the <rendering> element.

To have MapViewer automatically apply a globular map projection (that is, a map projection suitable for viewing the world, and specifically the azimuthal equidistant projection for MapViewer), specify use_globular_projection="true" in the <rendering> element. This option applies to geodetic data only.

1.5.2.6 Customizing the Spatial Data Cache

You can customize the in-memory cache that MapViewer uses for spatial data by using the <spatial_data_cache> element. For example:

```
<spatial_data_cache   max_cache_size="64"
                      report_stats="true"
/>
```

You can specify the following information as attributes of the <spatial_data_cache> element:

- The max_cache_size attribute specifies the maximum number of megabytes (MB) of in-memory cache.

  Default value: 64

- The `report_stats` attribute, if set to `true`, instructs the MapViewer server to periodically (every 5 minutes) output cache statistics, such as the number of objects cached, the total size of cache objects, and data relating to the efficiency of the internal cache structure. The statistics are provided for each data source and for each predefined theme. They can help you to determine the optimal setting of the in-memory cache. For example, if you want to pin all geometry data for certain themes in the memory cache, you need to specify a `max_cache_size` value that is large enough to accommodate these themes.

  Default value: `false`

The spatial data cache is always enabled by default, even if the element is commented out in the configuration file. To completely disable the caching of spatial data, you must specify the `max_cache_size` attribute value as `0` (zero).

> **Note:** The disk-based spatial cache, which was supported in the previous release, is no longer supported, because performance tests have shown that disk-based spatial caching was often less efficient than fetching spatial objects directly from the database when needed (that is, in cases where the cached objects frequently did not need to be retrieved again after caching).

For detailed information about the caching of predefined themes, see Section 2.3.1.5.

1.5.2.7 Specifying the Security Configuration

You can use the `<security_config>` element to specify whether MapViewer should reject `<info_request>` elements in requests. An `<info_request>` element is a type of request from a client that asks MapViewer to execute a simple SQL statement and return the result rows in plain text or XML format. This request is often used by MapViewer applications to identify features displayed on a map, or to run simple spatial search queries.

However, if the MapViewer data source information is exposed, malicious attackers might be able to abuse this capability and obtain sensitive information. To prevent this from happening, you can make sure MapViewer always connects to a database schema that has very limited access rights and hosts only non-sensitive information, and you can also reject all `<info_request>` requests by specifying the `<security_config>` element as follows:

```
<security_config>
  <disable_direct_info_request> true </disable_direct_info_request>
</security_config>
```

Note, however, that this setting affects some Mapviewer features. For example, the `identify()` method of the MapViewer Java API will no longer work, and applications will need to implement their own `identify()` method through other means.

1.5.2.8 Registering a Custom Image Renderer

MapViewer can display images stored in a database BLOB through its image theme capability. When the image data stored in the BLOB is in a format unknown to MapViewer, such as ECW, you can register a custom image renderer so that MapViewer can use it to display such images. For information about creating and registering a custom image renderer, see Appendix C.

To specify a custom image renderer, use the <custom_image_renderer> element, as shown in the following example:

```
<custom_image_renderer image_format="ECW"
                       impl_class="com.my_corp.image.ECWRenderer" />
```

The image_format attribute specifies the image format name with which this custom image renderer should be associated.

The impl_class attribute specifies the name of the class that implements the custom image renderer.

1.5.2.9 Registering a Custom Spatial Provider

MapViewer can render spatial data that is in an external (non-Oracle Spatial and Graph) native format, such as shapefile, if there is a spatial provider implementation registered for the format. For information about implementing an external spatial data provider (in connection with custom geometry themes), see Section 2.3.9.

To register an external spatial data provider, use the <s_data_provider> element, as shown in the following example:

```
<s_data_provider
  id="shapefileSDP"
  class="oracle.sdovis.ShapefileDataProvider"
  >
  <parameters>
    <parameter name="datadir" value="/temp/data" />
  </parameters>
</s_data_provider>
```

The class attribute specifies the name of the class that implements the external spatial data provider.

The <parameters> element specifies a set of initialization parameters that are used by the data provider during its initialization process. In this example, the shapefile provider has a data directory ("datadir") parameter that points to directory where MapViewer can look for the data.

1.5.2.10 Registering Custom Nonspatial Data Providers

When generating thematic map layers, MapViewer can dynamically join nonspatial attribute data (such as sales for each region) that originates from an external source with the base geometries (boundaries of all the regions) that are stored in the database. For information about thematic mapping using external attribute data from nonspatial data providers, see Section 2.3.11.1.

To register a nonspatial data provider, use the <ns_data_provider> element, as shown in the following example:

```
<ns_data_provider id="testProvider"
                  class="com.mycorp.GetSalesData" >
  <parameters>
    <parameter name="bi_database" value="stadb32.mycorp.com" />
    <parameter name="sid" value="bidata"  />
  </parameters>
</ns_data_provider>
```

The id attribute uniquely identifies a nonspatial data provider. Use this id value in any map request that involves the provider.

The `class` attribute specifies the name of the class that implements the nonspatial data provider.

The `<parameters>` element specifies a set of initialization parameters that are used by the nonspatial data provider during its initialization process.

1.5.2.11 Customizing SRS Mapping

You can use the `<srs_mapping>` element to specify an SDO to EPSG SRID mapping file, which define mappings between Oracle Spatial and Graph SDO_SRID values and EPSG codes. As explained in Section E.1.3, each line in the specified mapping file must contain an SDO_SRID value and the corresponding EPSG code. The `<srs_mapping>` element can be used with WMS and WFS themes.

The following example uses the `<srs_mapping>` element to specify an SDO to EPSG SRID mapping file:

```
<srs_mapping>
  <sdo_epsg_mapfile>
    ../config/epsg_srids.properties
  </sdo_epsg_mapfile>
</srs_mapping>
```

1.5.2.12 Customizing WMS GetCapabilities Responses

MapViewer can be used as an Open Geospatial Consortium WMS (Web Map Server) 1.1.1 compliant server. As such, a WMS client can send MapViewer the `GetCapabilities` request. In response, MapViewer will send back the list of themes that it hosts and other important information, such as the data provider's name and a list of keywords, that might of interest to the requesting client.

You can use the `<wms_config>` element to customize the descriptive information sent back to the client as part of the `GetCapabilities` response, as shown in the following example:

```
<wms_config host="www.my_corp.com" port="80"
            protocol="http" default_datasource="dsrc1"
            public_datasources="dsrc1,dsrc2">
  <title>
    WMS 1.1 interface for Oracle Application Server 10g MapViewer
  </title>
  <abstract>
    This WMS service is provided through Oracle MapViewer.
  </abstract>
  <keyword_list>
    <keyword>bird</keyword>
    <keyword>roadrunner</keyword>
    <keyword>ambush</keyword>
  </keyword_list>
  <sdo_epsg_mapfile>
    ../config/epsg_srids.properties
  </sdo_epsg_mapfile>
</wms_config>
```

The `host` attribute specifies the host part of the service request URL that the client should use for future WMS requests made to this MapViewer server.

The `port` attribute specifies the port part of the service request URL that the client should use for future WMS requests made to this MapViewer server.

The `protocol` attribute specifies the protocol part of the service request URL that the client should use for future WMS requests made to this MapViewer server.

The `default_datasource` attribute specifies the base data source used to retrieve the capabilities response. If this attribute is not defined, the data source `WMS` is used, and that data source must exist in this MapViewer server.

The `public_datasources` attribute specifies which data source contents are to be listed in the GetCapabilities response. If this attribute is not defined, all data source contents will be listed.

The `<title>` element specifies the service title to be included as part of the response.

The `<abstract>` element specifies the abstract to be included as part of the response.

The `<keyword_list>` element specifies a list of keywords that best describe the types of layers served by this MapViewer server.

The `<sdo_epsg_mapfile>` element specifies a text file that defines mappings from Oracle Spatial and Graph (SDO) SRID values to the corresponding EPSG SRID values that are typically used in most WMS requests and responses. For information about this mapping file, see Section E.1.3.

1.5.2.13 Configuring the Map Tile Server for Oracle Maps

The Oracle Maps feature of MapViewer can pre-generate base map image tiles and cache them through the map tile server. You can use the <map_tile_server> element to provide configuration information to the map tile server, such as default location for map tile file storage, and logging information, as shown in the following example:

```
<map_tile_server>
   <tile_storage default_root_path="/scratch/tilecache/" />
   <logging log_level="finest" log_thread_name="false" log_time="true">
      <log_output name="System.err"/>
   </logging>
</map_tile_server>
```

The `<tile_storage>` element specifies the default root directory where all map image tiles generated by this MapViewer server will be stored.

The `<logging>` element specifies logging information specific to the map tile server.

1.5.2.14 Defining Permanent Map Data Sources

Every map request must have a data source attribute that specifies a map data source, which is a database user with geospatial data. You can predefine available map data sources by using the `<map_data_source>` element. For example:

```
<map_data_source name="mvdemo"
                 jdbc_host="mapsrus.us.oracle.com"
                 jdbc_sid="orcl"
                 jdbc_port="1521"
                 jdbc_user="scott"
                 jdbc_password="!password"
                 jdbc_mode="thin"
                 number_of_mappers="5"
                 allow_jdbc_theme_based_foi="true"
                 plsql_package="web_user_info"
/>
```

You can specify the following information as attributes of the `<map_data_source>` element:

- The `name` attribute specifies a unique data source name to MapViewer. You must specify the data source name in all map requests that identify a data source.

- You must specify all necessary connection information, or a container data source name, or a net service name (TNS name). That is, you must specify only one of the following, which are described in this section: `jdbc_host`, `jdbc_sid`, `jdbc_port`, and `jdbc_user`; or `container_ds`; or `jdbc_tns_name`.

  If the database on which you defined a data source on is restarted, and if the data source is created from `jdbc_host/jdbc_sid/jdbc_port` or `jdbc_tns_name` attributes, MapViewer will resume normal operation (for example responding to map requests with properly created maps) as soon as the database is back online.

- The `jdbc_host`, `jdbc_sid`, `jdbc_port`, and `jdbc_user` attributes specify the database connection information and the database user name. (As an alternative to specifying these attributes and the `jdbc_password` and `jdbc_mode` attributes, you can specify the `container_ds` attribute, described later in this section.)

- The `jdbc_password` attribute specifies the database user's login password. It must be prefixed with an exclamation point (!) when you specify the password for the first time. When MapViewer next restarts, it will automatically obfuscate and replace the clear text password.

  MapViewer does not change this password string in any way; no conversion to upper or lower case is performed. If the database uses case-sensitive passwords, the specified password must exactly match the password in the database.

- The `jdbc_mode` attribute tells MapViewer which Oracle JDBC driver to use when connecting to the database. The default is `thin` (for the "thin" driver). The other possible value is `oci8`, which requires that you also have the Oracle Database client installed on the same host on which MapViewer is running.

- The `container_ds` attribute lets you specify the Java EE container name (from the `ejb-location` attribute value) instead of specifying the `jdbc_host`, `jdbc_sid`, `jdbc_port`, `jdbc_user`, `jdbc_password`, and `jdbc_mode` attributes. For example, assume that the `<data_source>` element in the `data-source.xml` file for the standalone OC4J instance contains `ejb-location="jdbc/OracleDS"`. In this case, instead of using the example at the beginning of this section, you can define the permanent MapViewer data source as follows:

  ```
  <map_data_source name="mvdemo"
                   container_ds="jdbc/OracleDS"
                   number_of_mappers="5"
  />
  ```

  To use the `container_ds` attribute in the MapViewer configuration file, you must start the OC4J instance with the `-userThreads` option. MapViewer processes its configuration file in a separate user thread; if the `-userThreads` option is not specified, the container's context information is not available to user threads. However, if you are dynamically defining a data source through the MapViewer Administration page, you can use the `container_ds` attribute regardless of whether you started the OC4J instance with the `-userThreads` option.

  If you use the `container_ds` attribute, and if you want MapViewer to resume normal operation (for example responding to map requests with properly created maps) automatically after the database on which you defined a data source on is restarted, you must instruct the container data source to always validate a connection before it can be returned to the application. Check your middleware documentation for whether this option is supported and, if it is supported, how to enable it.

- The jdbc_tns_name attribute identifies a net service name that is defined in the tnsnames.ora file.
- The number_of_mappers attribute identifies the maximum number of map renderers available (and thus the maximum number of map requests that MapViewer can process in parallel for the data source) for this data source. Any unprocessed map requests are queued and eventually processed. For example, if the value is 3, MapViewer will be able to process at most three mapping requests concurrently. If a fourth map request comes while three requests are being processed, it will wait until MapViewer has finished processing one of the current requests.

  Specifying a large number_of_mappers value (such as 50 or 100) can improve the overall throughput, but it will also increase runtime memory and CPU usage at times of peak loads, since MapViewer will attempt to process more concurrent map requests. It will also increase the number of active database sessions. Therefore, be sure that you do not set too large a number for this attribute.

  > **Note:** The obsolete max_connections attribute no longer affects rendering and is ignored. The number_of_mappers attribute value affects the actual maximum number of database connections or sessions open for the data source at any given time.

- The allow_jdbc_theme_based_foi attribute lets you specify whether to allow JDBC theme-based FOI requests to be performed against this data source. A JDBC theme-based FOI request is based on a dynamic SQL query constructed by the JavaScript client application.

  By default, such FOI requests are not allowed unless you set this attribute to true. Due to the potential security threat, JDBC theme-based FOI requests should be used with caution. You should only allow JDBC theme-based FOI requests on database connections that are granted very low privilege and contain only data that you want to expose. See Section 8.3.1.3 for more information about JDBC theme-based FOI requests.
- The plsql_package attribute lets you specify a PL/SQL package to be used for secure map rendering, as explained in Section 1.8.
- The web_user_type attribute (not shown in the example in this section) lets you specify the source for the authenticated user's name. It is especially useful for getting the authenticated user's name from a cookie, in conjunction with specifying a PL/SQL package to be used for secure map rendering. For more information about the web_user_type attribute and an example of its use, see Section 1.8.2.

1.5.3 Performing MapViewer Administrative Tasks

Besides knowing how to configure MapViewer, you should also know how to perform other important administrative tasks using the MapViewer administration page. To log in to this page, see the instructions in Section 1.5.1.

The tasks you can do as a MapViewer administrator include the following:

- Editing the configuration file

  Click **Manage MapViewer**, then **Configuration**.
- Creating dynamic data sources

Click **Manage MapViewer**, then **Datasources**. Enter the appropriate parameters, then click **Submit**.

- Refreshing the list of data sources

  Click **Manage MapViewer**, then **Datasources**. Click **Refresh**.

- Clearing cached definitions of MapViewer styles, themes, and base maps

  Click **Manage MapViewer**, then **Datasources**. Select the data source, then click **Purge Cached Metadata**.

- Clearing cached geometry data for predefined themes

  Click **Manage MapViewer**, then **Geometry Cache**. Under **Purge Cached Geometries**, select the data source and theme, and click **Submit**.

- Creating map tile layers for Oracle Maps

  Click **Manage Map Caches**, then **Create**. Select **Internal** or **External** for the map source type, and click **Continue**.

  Internal map source: Enter the map cache name, then select the data source and base map. Also define parameters for cache storage (where tiles will be stored), zoom levels, minimum and maximum scale, spatial reference ID (SRID), data bounding box (MBR), and tile size and format. Click **Submit** to create the map tile layer. You can also define the map cache properties in XML by clicking XML.

  External map source: Enter the map cache name, then select the data source. To provide access to the external source, define parameters such as the map service URL, the request method (GET or POST), the proxy information (if needed), the java adapter class name and its location on the server, and additional adapter properties. Also define parameters for cache storage (where tiles will be stored), zoom levels, minimum and maximum scale, spatial reference ID (SRID), data bounding box (MBR), and tile size and format. Click **Submit** to create the map tile layer. You can also define the map cache properties in XML by clicking XML.

- Managing map tile layers for Oracle Maps

  Click **Manage Map Caches**, then **Manage**. Then do any of the following:

  To refresh map caches, click **Refresh**.

  To edit a map tile layer, under **Existing Map Tile Layers**, select the data source. At the cache level, you can delete the cache, view cache details, and place the cache offline or online. At the tile level, you can perform operations such as clearing, prefetching, and refreshing the tiles, specifying the zoom level, and specifying the bounding box.

  To check the status of a request, enter the request ID and click **Submit**.

1.6 Oracle Real Application Clusters and MapViewer

When the database is an Oracle Real Application Cluster (Oracle RAC), you cannot create MapViewer data sources that directly connect to it. Instead, MapViewer must connect to an Oracle RAC database through the data source of the Java EE container. To enable MapViewer to connect to an Oracle RAC database, you must do the following:

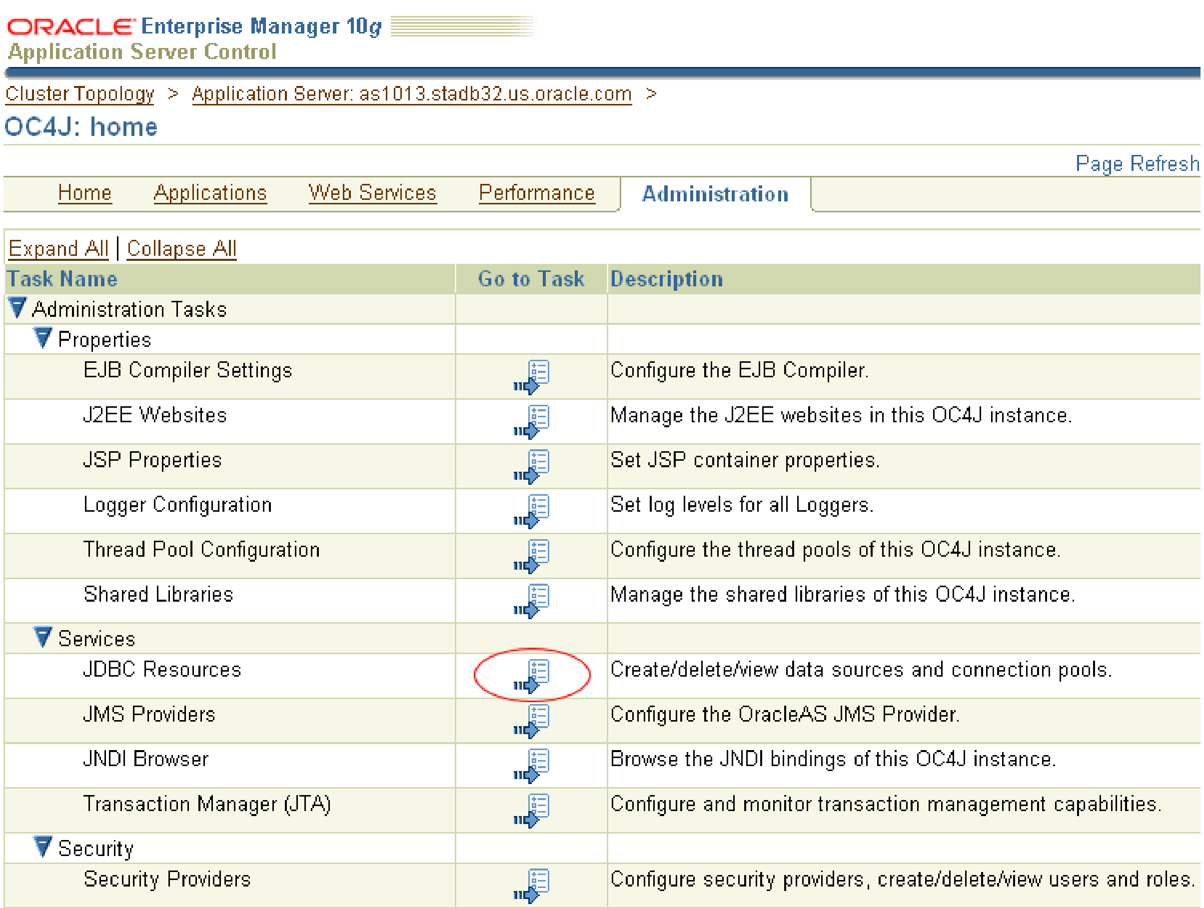
1. Create a JDBC data source that connects to the Oracle RAC database at the OC4J level, as explained in Section 1.6.1. The data source can then be used by applications such as MapViewer through JNDI lookup.

2. Configure the OC4J instance so that it publishes the JNDI location of the Oracle RAC data source so that MapViewer can access it, as explained in Section 1.6.2.
3. Define a MapViewer data source that reuses the container data source through the JNDI location in its configuration file, as explained in Section 1.6.3.
4. Restart MapViewer.

1.6.1 Creating a Container Oracle RAC Data Source

With either a full Oracle Fusion Middleware or standalone OC4J installation, use Oracle Enterprise Manager to create a data source that connects to the Oracle RAC database. For example, if using Oracle Application Server release 10.1.3 or later, you can log in to Enterprise Manager, navigate to the OC4J instance that contains the MapViewer server, click the **Administration** tab, and click the **JDBC Resources Go to Task** link to start creating a new data source, as shown in Figure 1–9.

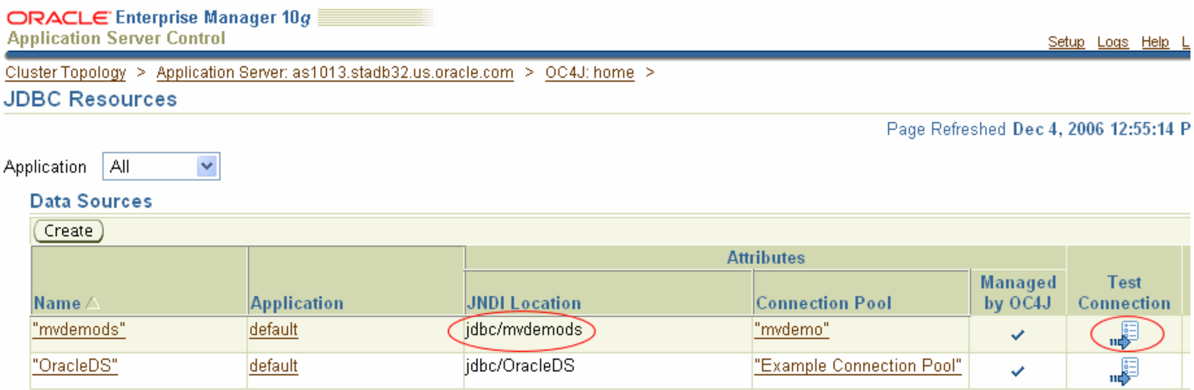
Figure 1–9 Administration Tab for Creating Oracle RAC Container Data Source



For more information about creating a data source to connect to an Oracle RAC database, see *Oracle Application Server Administrator's Guide*.

After creating the data source, you should test the connection using Enterprise Manager, by clicking the Test Connection icon for the connection, as shown in Figure 1–10.

Figure 1–10 Testing the Connection for the Data Source



Be sure to note the **JNDI Location** value (which is `jdbc/mvdemods` in Figure 1–10), because you will need this value when you create the MapViewer data source (explained in Section 1.6.3).

1.6.2 Adding the userThreads Option to the OC4J Container

You must specify the `userThreads` option to tell the OC4J instance to publish the JNDI locations, such as the one for the newly created data source, to all user threads. Without this option, MapViewer cannot access the JNDI location that references the data source, because by default OC4J makes such JNDI locations available only to the main thread within which OC4J itself is running. MapViewer, however, is started in a separate user thread.

The mechanism for specifying the `userThreads` option depends on whether you re using a standalone OC4J instance or a full Oracle Fusion Middleware installation.

1.6.2.1 Adding userThreads for a Standalone OC4J Instance

With a standalone OC4J instance, you must start the OC4J instance with the `-userThreads` option, as in the following example:

```
java -jar oc4j.jar -userThreads
```

1.6.2.2 Adding userThreads for a Full Oracle Fusion Middleware 10*g*R3 Installation

With a full Oracle Fusion Middleware 10*g*R3 installation, the Java startup parameters are defined in the `$OAS_HOME/opmn/conf/opmn.xml` configuration file. (`opmn` is the master process that starts and stops various Oracle Fusion Middleware 10*g*R3 components, such as OC4J instances.)

In this file you can specify Java JVM startup parameters for the OC4J instance running MapViewer. For example, if you deployed MapViewer to the `home` OC4J instance, add the text `-Doc4j.userThreads=true`, as shown in the following example:

```
          <ias-component id="OC4J">
             <process-type id="home" module-id="OC4J" status="enabled">
                <module-data>
                   <category id="start-parameters">
                      <data id="java-options" value="-server
 -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
 -Djava.awt.headless=true -Dhttp.webdir.enable=false
 -Doc4j.userThreads=true"/>
                   </category>
```

```
...
```

After editing and saving the `opmn.xml` file, you must restart the OC4J instance for the `userThreads` option to take effect; and if that does not work, restart Oracle Fusion Middleware 10*g*R3. For information about restarting the OC4J instance or Oracle Fusion Middleware 10*g*R3, see *Oracle Application Server Administrator's Guide*.

1.6.3 Creating a MapViewer Data Source

Create a new MapViewer data source that enables it to connect to the Oracle RAC database, by using the `container_ds` attribute of the MapViewer data source. Specifically, you must add an entry like the following in the `mapViewerConfig.xml` file:

```
<map_data_source  name="mvdemo"
                  container_ds="jdbc/mvdemods"
                  number_of_mappers="7" />
```

In the preceding example:

- The `name` attribute specifies the MapViewer data source name, which is needed for map requests.
- The value for the `container_ds` attribute must match the JNDI Location string that you noted when you created the container Oracle RAC data source (see Section 1.6.1).
- The `number_of_mappers` attribute specifies the maximum number of supported concurrent map requests that can target this data source.

For more information about the `name` and `number_of_mappers` attributes, see Section 1.5.2.14.

After adding the data source definition, you must restart MapViewer to have the new data source created. After you do this, whenever you request a map from this data source, MapViewer obtains the necessary database connections from the container before proceeding.

1.7 High Availability and MapViewer

> **Note:** This section is intended for advanced users who want to take full advantage of the high availability features of Oracle Fusion Middleware with MapViewer. You must have a strong understanding of high availability features, which are described in *Oracle Fusion Middleware High Availability Guide*.

MapViewer users can benefit from the high availability features of Oracle Database and Oracle Fusion Middleware.

1.7.1 Deploying MapViewer on a Multiprocess OC4J Instance

You can safely deploy MapViewer in an OC4J instance of Oracle Fusion Middleware that has multiple processes. Oracle Fusion Middleware lets you configure the number of actual processes (JVMs) that can be started for each OC4J instance. On a multiprocessor host, starting multiple processes for a single OC4J can better utilize the system resources. (Releases of MapViewer before 10*g* Release 2 (10.1.2) could not take advantage of this feature and thus could not be deployed on such OC4J instances.)

When MapViewer is deployed to an OC4J instance with multiple processes, each process has a MapViewer server running inside it. These MapViewer servers all reside on the same host but in different Java processes. Map requests sent to this OC4J instance are automatically dispatched to the individual MapViewer servers. Each MapViewer server generates map image files according to a unique naming scheme, with the names coordinated when the different MapViewer servers are first started (that is, when the containing OC4J instance is started). This avoids the possibility of two MapViewer servers generating map files in the same sequence with the same file names.

1.7.2 Deploying MapViewer on a Middle-Tier Cluster

OC4J instances in different Oracle Fusion Middleware 10*g*R3 installations can be clustered into an island. This provides a middle-tier fail-safe option. MapViewer can be deployed to an OC4J island. You must take care, however, about how the generated image files on each host are named and referenced through URLs by client applications.

Consider the following sample scenario. When a map request is sent to the front web server, it reaches the MapViewer server running on host A. MapViewer on host A then sends back the URL for the generated map image, and the client then sends a second request to fetch the actual image. This second request might be received by the OC4J container running on host B, which has no such image (or which will send back an incorrect image with the same name).

There is no single best solution for this problem in all environments. One option is to have the hosts share common networked storage, so that the map images are deposited in the same virtual (networked) file system by different MapViewer servers running on different hosts. You must configure the map file storage information (see Section 1.5.2.2) for each MapViewer instance so that the images are deposited in different subdirectories or so that they have different file prefixes. Otherwise, the image files generated by the multiple MapViewer servers might overwrite each other on the disk. By properly configuring the map file storage information, you ensure that each URL sent back to the client uniquely identifies the correct map on the network drive.

If you cannot use networked drives, consider using a load balancer. You may first need to configure the map file storage information for each MapViewer instance (as explained in the preceding paragraph), so that each MapViewer instance names its generated images using an appropriate scheme to ensure uniqueness. You can then specify rules in the load balancer to have it redirect image requests to a certain host if the URL matches a certain pattern, such as containing a specified map image file prefix.

1.8 Secure Map Rendering

This section describes how to implement secure map rendering based on a web user's identity. Users with different roles or permissions will see different feature sets when viewing the same theme. The basic idea is that MapViewer will always invoke a specified PL/SQL package to set the web user's identity in the database whenever accessing the database for any themes. This user information can be used by the database to enforce data access control.

Note: In this section, the terms *user* and *authenticated user* refer to the application or web user that logs into Oracle Fusion Middleware or Oracle Single Sign-On (SSO). It is *not* the same as the database user. MapViewer itself will connect directly to a database schema that stores all the geospatial data.

MapViewer will connect directly to a database schema that stores all the geospatial data. To enforce access control for MapViewer on the data in this schema, you must perform the following steps:

1. Create a PL/SQL package in the database schema. The package must have at least two named procedures: `set_user(username)` and `clear_user()`.
2. Create views, set access rights on database objects, and perform other tasks, based on the user identity stored in the PL/SQL package (which is set by MapViewer through the set_user procedure for each database session).
3. Create a MapViewer data source to the schema, providing the name of the PL/SQL package as part of the data source definition. This is considered a secured data source.
4. Create MapViewer themes that are based on the views created in step 2.
5. Establish web authentication for users accessing your MapViewer application page or pages, so that when a map request reaches the MapViewer servlet, the web session object should contain an authenticated user's identity.
6. Issue map and FOI (feature of interest) requests that view the themes defined in step 4, either directly or through the use of base maps and Oracle Maps.

   MapViewer will automatically pass the user identity to the database using the PL/SQL package before it executes any query for these themes. Only those rows that are visible to the identified user will be returned from the database and rendered by MapViewer.

Section 1.8.1 explains how secure map rendering works and provides implementation details and examples. Section 1.8.3 describes some options for authenticating users and refers to a supplied demo.

1.8.1 How Secure Map Rendering Works

MapViewer, as a Java EE application, can obtain the identity of a web user that has been authenticated to Oracle Fusion Middleware or Oracle Single Sign-On (SSO). This user information can then be preserved and propagated to the database, where secure access to map layers and tables can be set up based on the user identity. For example, a database administrator (DBA) can create a view of a base table that selects only those spatial features visible to a specific user.

To pass the web user identity from Oracle Fusion Middleware or Oracle Single Sign-On (SSO) to the database, use a secure PL/SQL package that sets the user identity in the database. This PL/SQL package is created by a DBA or application developer and installed in the data source schema. Such a package can have any number of procedures and functions, but it must contain at least the following two procedures:

- `set_user(username)`
- `clear_user()`

Whenever a theme is requested from a secured data source, MapViewer invokes the `set_user` procedure in the associated PL/SQL package before it executes any data

query for the theme, and it invokes the `clear_user` procedure when the querying process is complete for the theme.

Example 1–3 shows a PL/SQL package that you can use for secure map rendering. You can create this package in the example MVDEMO schema.

Example 1–3 PL/SQL Package for Secure Map Rendering

```
CREATE OR REPLACE PACKAGE web_user_info
AS
    PROCEDURE set_user (p_name IN VARCHAR2);
    PROCEDURE clear_user;
    FUNCTION  get_user
        RETURN VARCHAR2;
END;
CREATE OR REPLACE PACKAGE BODY web_user_info
AS
    w_name VARCHAR2 (32767);

  PROCEDURE set_user (p_name IN VARCHAR2)
  AS
  BEGIN
     w_name := LOWER (p_name);
  END;

  PROCEDURE clear_user
  AS
  BEGIN
      w_name := null;
  END;

  FUNCTION get_user
    RETURN VARCHAR2
  AS
  BEGIN
    RETURN w_name;
  END;
END;
/
```

In Example 1–3, set_user and clear_user are two required methods, and get_user is a convenience function that can be used in creating views or for other data access control purposes

After you create the package (which essentially contains the user identity for the current database session), you can set up an elaborate virtual private database that uses this user information (see *Oracle Database Security Guide* for information about using Oracle Virtual Private Database, or VPD). For simplicity, however, this section does not discuss VPD creation, but shows that you can create views that use this user information to enforce data access control.

For example, in the example MVDEMO schema you can add a column named ACCOUNT_MGR to the existing CUSTOMERS table, and assign an account manager to each customer stored in this table. You can then create a view that returns only customer rows for a specific account manager, as shown in Example 1–4.

Example 1–4 View for Secure Map Rendering

```
CREATE OR REPLACE VIEW customers_view
AS
```

```
SELECT * FROM customers
WHERE account_mgr = web_user_info.get_user;
```

You can now define a MapViewer theme based on this view, so that whenever account managers log in and want to view customer data on a map, each will only see his or her own customers.

After you have installed the PL/SQL package, you can pass the name of this package to MapViewer as part of the definition of a data source by using the `plsql_package` attribute, as shown in Example 1–5.

Example 1–5 Data Source Definition for Secure Map Rendering

```
<map_data_source name="mvdemo"
                 jdbc_host="stadb32.us.oracle.com"
                 jdbc_sid="mv"
                 jdbc_port="15214"
                 jdbc_user="mvdemo"
                 jdbc_password="password"
                 jdbc_mode="thin"
                 number_of_mappers="3"
                 allow_jdbc_theme_based_foi="true"
                 plsql_package="web_user_info"
   />
```

When you specify a PL/SQL package name in a data source definition, MapViewer flags the data source as a secure data source, and it automatically invokes the package's `set_user` and `clear_user` procedures whenever performing any theme queries on the data source.

1.8.2 Getting the User Name from a Cookie

Sometimes the authenticated user's name is not passed to MapViewer through a Java EE or OSSO session. such as when you integrate MapViewer within Application Express (APEX), where authentication is carried out by APEX and the user name is not available through a Java EE or OSSO session. To enable you to work around this issue, MapViewer also supports getting the user name from a cookie. It is your responsibility to set up the cookie within APEX to hold the authenticated user name.

To ensure that MapViewer picks up the user name from a named cookie, you must specify the `web_user_type` attribute in the data source definition (in addition to the mandatory `plsql_package` attribute). For example, if you want MapViewer to pick up the user name from a cookie named MON_USER, your secure data source definition should look like Example 1–6.

Example 1–6 Data Source Definition Specifying Cookie Name

```
<map_data_source name="mvdemo"
                  jdbc_host="stadb32.us.oracle.com"
                  jdbc_sid="mv"
                  jdbc_port="25650"
                  jdbc_user="mvdemo"
                  jdbc_password="LfCDQ6NH59nuV7zbeY5QY06sqN7XhiUQ"
                  jdbc_mode="thin"
                  number_of_mappers="3"
                  allow_jdbc_theme_based_foi="true"
                  plsql_package="web_user_info"
                  web_user_type="MON_USER"
  />
```

The possible values for the web_user_type attribute are:

- J2EE_USER: tells MapViewer to get the authenticated user name from a Java EE session
- OSSO_USER: tells MapViewer to get the authenticated user from an OSSO session.
- *<cookie-name>*: tells MapViewer to get the authenticated user from a cookie with the specified name. The cookie name is not case sensitive.

If web_user_type is not specified, MapViewer first looks for the user name in the Java EE session; and if none is found, it looks for the user name in the OSSO session (if present).

1.8.3 Authenticating Users: Options and Demo

How, when, and where users are authenticated depend on the requirements of your application and the setup of your installation. For example, your options include the following:

- Deploy MapViewer as part of an enterprise portal site, so that end users always first log onto the portal before performing any mapping functions through MapViewer.
- Deploy MapViewer on a separate system, and have users authenticate to a central Oracle SSO server.

As long as the HTTP requests reaching MapViewer contain the authenticated user information, MapViewer will be able to pass the requests on to the database, and the secure data access approach will work as expected.

The demo files supplied with MapViewer (see Section 1.9) include an explanation, plus related files, for restricting a single mapping page to be accessible only by authenticated users. This demo involves making simple changes to MapViewer's own deployment files. In this case, this protected page is the entry point that causes users to be authenticated, and the authentication is performed by the OC4J instance running MapViewer.

1.9 MapViewer Demos and Tutorials

Several MapViewer demos and tutorials are included in a separate application archive named mvdemo.ear, and deployed to the same Java EE container where mapviewer.ear is deployed.

Once deployed the demos and tutorials will be accessible from a URL in this format: http://*host:port*/mvdemo/

See also the resources available at:

http://www.oracle.com/technetwork/middleware/mapviewer/downloads/

2 MapViewer Concepts

This chapter explains concepts that you should be familiar with before using MapViewer.

Some fundamental concepts include *style*, *theme*, *base map*, *mapping metadata*, and *map*.

- Styles define rendering properties for features that are associated with styles. For example, a text style determines how such a feature is labeled on a map, while a line style determines the rendition of a linear feature such as a road.
- A theme is a collection of features (entities with spatial and nonspatial attributes) that are associated with styles through the use of styling rules.
- A base map consists of one or more themes.
- Mapping metadata consists of a repository of styles, themes, and base maps stored in a database.
- A map is one of the components that MapViewer creates in response to a map request. The map can be an image file, the object representation of an image file, or a URL referring to an image file.

This chapter contains the following major sections:

- Section 2.1, "Overview of MapViewer"
- Section 2.2, "Styles"
- Section 2.3, "Themes"
- Section 2.4, "Maps"
- Section 2.5, "Data Sources"
- Section 2.6, "How a Map Is Generated"
- Section 2.8, "Workspace Manager Support in MapViewer"
- Section 2.9, "MapViewer Metadata Views"

2.1 Overview of MapViewer

When an application uses MapViewer, it applies specific styles (such as colors and patterns) to specific themes (that is, collections of spatial features, such as cities, rivers, and highways) to render a map (such as a GIF image for display on a web page). For example, the application might display a map in which state parks appear in green and restaurants are marked by red stars. A map typically has several themes representing political or physical entities, or both. For example, a map might show national and state boundaries, cities, mountain ranges, rivers, and historic sites. When the map is rendered, each theme represents a layer in the complete image.

MapViewer lets you define styles, themes, and base maps, including the rules for applying one or more styles to each theme. These styles, themes, base maps, and associated rules are stored in the database in map definition tables under the MDSYS schema, and they are visible to you through metadata views. All styles in a database instance are shared by all users. The mapping metadata (the set of styles, themes, and base maps) that you can access is determined by the MapViewer metadata views described in Section 2.9 (for example, USER_SDO_STYLES, USER_SDO_THEMES, and USER_SDO_MAPS). The set of map definition objects that a given user can access is sometimes called that user's *mapping profile*. You can manage styles, themes, and base maps with the standalone Map Builder tool, described in Chapter 9.

2.2 Styles

A **style** is a visual attribute that can be used to represent a spatial feature. The basic map symbols and labels for representing point, line, and area features are defined and stored as individual styles. Each style has a unique name and defines one or more graphical elements in an XML syntax.

Each style is of one of the following types:

- **Color**: a color for the fill or the stroke (border), or both.
- **Marker**: a shape with a specified fill and stroke color, or an image. Markers are often icons for representing point features, such as airports, ski resorts, and historical attractions.

  When a marker style is specified for a line feature, the rendering engine selects a suitable point on the line and applies the marker style (for example, a shield marker for a U.S. interstate highway) to that point.
- **Line**: a line style (width, color, end style, join style) and optionally a center line, edges, and hash mark. Lines are often used for linear features such as highways, rivers, pipelines, and electrical transmission lines. You can also use cased line styles, which are useful for drawing streets and highways.
- **Area**: a color or texture, and optionally a stroke color. Areas are often used for polygonal features such as counties and census tracts.
- **Text**: a font specification (size and family) and optionally highlighting (bold, italic) and a foreground color. Text is often used for annotation and labeling (such as names of cities and rivers).
- **Advanced**: a composite used primarily for thematic mapping, which is described in Section 2.3.11. The key advanced style is `BucketStyle`, which defines the relationship between a set of simple rendering (and optionally labeling) styles and a set of buckets. For each feature to be plotted, a designated value or set of values from that feature is used to determine which bucket the feature falls into, and then the style associated with that bucket is used to plot the feature. Bucket styles are described in Section A.6.1.

  Two special types of bucket styles are also provided: color scheme (described in Section A.6.2) and variable (graduated) marker (described in Section A.6.3). Other advanced styles are dot density (described in Section A.6.4), bar chart (described in Section A.6.5), collection (described in Section A.6.6), variable pie chart (described in Section A.6.7), and heat map (described in Section A.6.8).

Table 2–1 lists the applicable geometry types for each type of style.

Table 2–1 Style Types and Applicable Geometry Types

Style Type	Applicable Geometry Types
Color	(any type)
Marker	point, line
Line	line
Area	polygon
Text	(any type)
Advanced	(any type)

All styles for a database user are stored in that user's USER_SDO_STYLES view, which is described in Section 2.9 and Section 2.9.3.

You can also create **dynamically defined styles** (that is, temporary styles) of any style type as part of a map request. The way to create them depends on which API you are using:

- With the native XML API, define the style using its XML elements within the <map_request> element.
- With the JavaBean API, add a dynamically defined style to a map request, as explained in Section 4.3.4.
- With the Oracle Maps JavaScript API, use classes and methods to create all types of styles dynamically.

In each case, what you are actually creating is the XML definition of the styles; it is the MapViewer server that actually creates such dynamically defined styles from the definitions when it processes the map request, and it discards the dynamically created styles when the request is completed.

For more detailed information about the types of styles, including information about the XML format for defining each type, see Appendix A.

This section contains the following major subsections:

- Section 2.2.1, "Scaling the Size of a Style (Scalable Styles)"
- Section 2.2.2, "Specifying a Label Style for a Bucket"
- Section 2.2.3, "Orienting Text Labels and Markers"
- Section 2.2.4, "Making a Text Style Sticky"
- Section 2.2.5, "Getting a Sample Image of Any Style"

2.2.1 Scaling the Size of a Style (Scalable Styles)

If you specify a unit other than the default of pixels (`px`) in a style definition, the style becomes scalable: that is, the size of features associated with the style is scaled as users zoom in or out on a map. For example, if you specify a marker style's width and height as `100m`, the marker is displayed as a square 100 meters on each side according to the map scale at the current zoom level.

The following are style types and the attributes that can have an associated size unit:

- Marker styles: marker size (height and width) and text attributes (font size, label offsets)

- Line styles: overall line width, center line width and dash pattern, wing line width and dash pattern, hash mark, and marker pattern (size, offset, interval)
- Text styles: font size, halo width
- Bar chart styles: bar width and height
- Dot density styles: dot width and height
- Pie chart styles: pit radius

Example 2–1 defines a star-shaped marker within a bounding box 15 kilometers (`15.0km`) on each size. This definition might be useful for identifying capital cities of states on a map showing all or a large part of a country; however, it would not be useful for a display zoomed in on a specific city and its local surrounding area.

Example 2–1 Scalable Marker Style

```
<style name="M.STAR_CAPITAL_CITY">
  <svg width="1in" height="1in">
    <desc/>
    <g class="marker"
style="stroke:#000000;fill:#FF0000;fill-opacity:0;width:15.0km;height:15.0km;font-
family:Dialog;font-size:12;font-fill:#FF0000">
        <polyline
points="138.0,123.0,161.0,198.0,100.0,152.0,38.0,198.0,61.0,123.0,0.0,76.0,76.0,76
.0,100.0,0.0,123.0,76.0,199.0,76.0"/>
    </g>
  </svg>
</style>
```

Example 2–2 defines a line style with an overall line width of 10 meters (`10.0m`) and a border line width of 1 meter (`1.0m`). This definition might be useful for identifying capital cities of primary highways.

Example 2–2 Scalable Line Style

```
<style name="L.PRIMARY_HIGHWAY">
  <svg width="1in" height="1in">
    <desc></desc>
    <g class="line" cased="true" style="fill:#33a9ff;stroke-width:10.0m">
      <line class="parallel" style="fill:#aa55cc;stroke-width:1.0m"/>
    </g>
  </svg>
</style>
```

When MapViewer renders or labels styles that have size units other than pixel, it first transforms the size units into screen pixels based on the current map area and display area, and it then renders the or labels the style. The size of a scalable style changes as users zoom in or out on a map. If zooming out results in an overall style size less than or equal to zero, the style is not rendered or labeled.

Size units can be used only with data associated with a known spatial reference system (SRS). If the data has no SRS or an unknown SRS, pixels are used for all size values. Note also that pixel values are used instead of any specified size unit in legends and in previews rendered by the Map Builder utility. (Legends are explained in Section 2.4.2.)

Scalable styles work with MapViewer Release 11*g* (11.1.1) or later; they cannot be used with earlier releases of MapViewer.

2.2.2 Specifying a Label Style for a Bucket

For collection-based bucket styles and individual range-based bucket styles (described in Section A.6.1.1 and Section A.6.1.2, respectively), you can specify a labeling style by using the `label_style` attribute in each bucket element. Example 2–3 creates an advanced style named `V.COUNTY_POP_DENSITY` in which each bucket is assigned a text label style (using the `label_style` attribute), with some styles being used for several buckets.

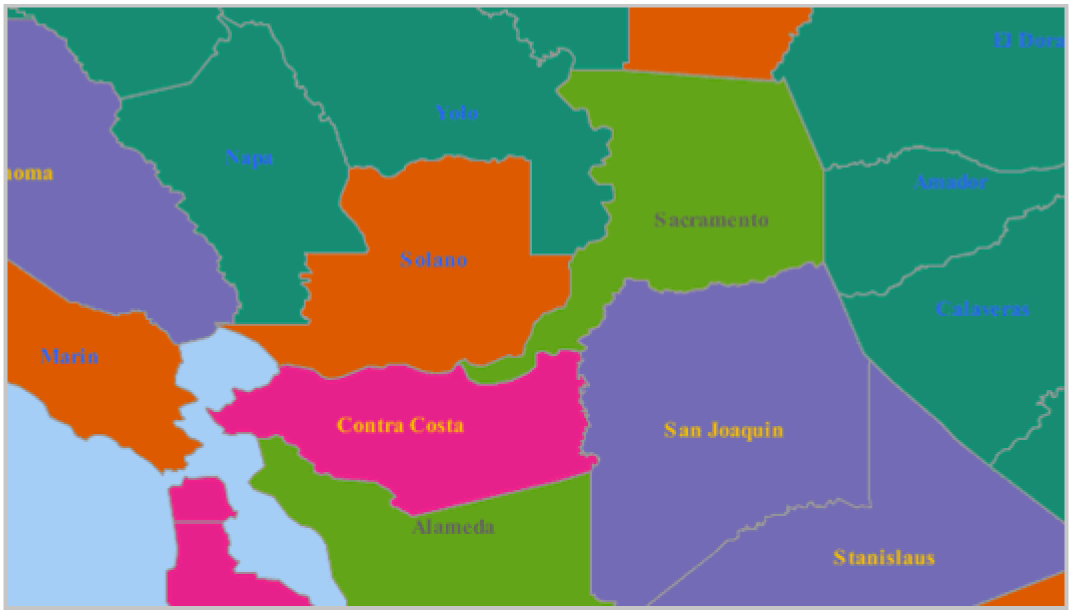
Example 2–3 Advanced Style with Text Label Style for Each Bucket

```
<?xml version="1.0" ?>
<AdvancedStyle>
     <BucketStyle>
        <Buckets>
            <RangedBucket seq="0" label="&lt;150k"
              low="-Infinity" high="150000"
              style="C.CB_QUAL_8_CLASS_DARK2_1"
              label_style="T.BLUE_SERIF_12"/>
            <RangedBucket seq="1" label="150k - 350k"
              low="150000" high="350000"
              style="C.CB_QUAL_8_CLASS_DARK2_2"
              label_style="T.BLUE_SERIF_12"/>
            <RangedBucket seq="2" label="350k - 600k"
              low="350000" high="600000"
              style="C.CB_QUAL_8_CLASS_DARK2_3"
              label_style="T.BROWN_SERIF_12"/>
            <RangedBucket seq="3" label="600k - 1000k"
              low="600000" high="1000000"
              style="C.CB_QUAL_8_CLASS_DARK2_4"
              label_style="T.BROWN_SERIF_12"/>
            <RangedBucket seq="4" label="1000k - 1500k"
              low="1000000" high="1500000"
              style="C.CB_QUAL_8_CLASS_DARK2_5"
              label_style="T.GREY_SERIF_12"/>
            <RangedBucket seq="5" label="1500k - 2500k"
              low="1500000" high="2500000"
              style="C.CB_QUAL_8_CLASS_DARK2_6"
              label_style="T.GREY_SERIF_12"/>
            <RangedBucket seq="6" label="2500k - 5000k"
              low="2500000" high="5000000"
              style="C.CB_QUAL_8_CLASS_DARK2_7"
              label_style="T.GREEN_SERIF_12"/>
            <RangedBucket seq="7" label="&gt;=5000k"
              low="5000000" high="Infinity"
              style="C.CB_QUAL_8_CLASS_DARK2_8"
              label_style="T.GREEN_SERIF_12"/>
    </Buckets>
   </BucketStyle>
</AdvancedStyle>
```

For individual range-based buckets, the lower-bound value is inclusive, while the upper-bound value is exclusive (except for the range that has values greater than any value in the other ranges; its upper-bound value is inclusive). No range is allowed to have a range of values that overlaps values in other ranges.

If the `V.COUNTY_POP_DENSITY` style in Example 2–3 is used in a map request, it displays a map that might look like the display in Figure 2–1, where the county names are shown with labels that reflect various text styles (in this case depending on the county's total population).

Figure 2–1 Varying Label Styles for Different Buckets



In Example 2–3, all buckets except the last one specify a label style. For any features that fall into a bucket that has no specified label style, the label style (if any) applied to the feature depends on the following:

- If the `<label>` element of the theme's styling rules specifies a label style other than the advanced style itself, the specified label style is used to label the feature. In the following example, because the `<label>` element's style specification (`T.STATE_NAME`) is different from the `<features>` element's style specification (`V.COUNTY_POP_DENSITY`), features that fall into a bucket with no specified label style are labeled using the `T.STATE_NAME` style:

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule column="TOTPOP">
    <features style="V.COUNTY_POP_DENSITY">
    </features>
    <label column="county" style="T.STATE NAME">
        1
    </label>
  </rule>
</styling_rules>
```

- If the `<label>` element of the theme's styling rules specifies the advanced style as its label style, the feature is *not* labeled. (This is why some counties in Figure 2–1 are not labeled.) In the following example, because the `<features>` and `<label>` elements both specify the advanced style `V.COUNTY_POP_DENSITY`, features that fall into a bucket with no specified label style are not labeled:

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule column="TOTPOP">
    <features style="V.COUNTY_POP_DENSITY">
    </features>
    <label column="county" style="V.COUNTY_POP_DENSITY">
        1
    </label>
  </rule>
```

```
</styling_rules>
```

2.2.3 Orienting Text Labels and Markers

You can control the orientation of text labels and markers on a map by using oriented points. The oriented point is a special type of point geometry in Oracle Spatial and Graph. In an oriented point, the coordinates represent both the location of the point and a virtual end point, to indicate an orientation vector. The text is aligned or the marker symbol is rotated according to the orientation vector, which is explained in Section 3.2.5 and illustrated in Figure 3–3 in that section. For more information about oriented points, see *Oracle Spatial and Graph Developer's Guide*.

2.2.3.1 Controlling Text Style Orientation

To orient the text label of a point in the direction of an orientation vector, you can specify the point as an Oracle Spatial and Graph oriented point in the map request. When MapViewer labels an oriented point, it automatically centers the text label on the point position, and aligns the label so that it points in the direction of the orientation vector.

For each feature to be so labeled, you must specify its location as an oriented point. You can group these oriented points in a single table and create a spatial index on the column containing the point geometries. You can then create a theme based on the table, specifying a desired text style as the labeling, and specifying transparent color style as the rendering style so that the points themselves are not displayed on the map.

Example 2–4 is a map request that labels a single oriented point with coordinates (12,14, 0.3,0.2), where (12,14) represents the X and Y coordinates of the point and (0.3,0.2) represents the orientation vector. It renders the point using a dynamically defined transparent color style (named `transparent_color`) to ensure that the text is displayed but the underlying point is not displayed.

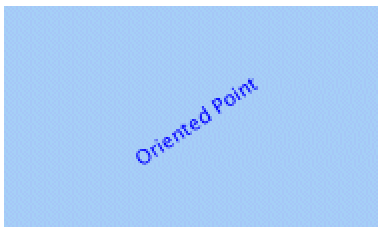
Example 2–4 Labeling an Oriented Point

```
<map_request
  title="Labeling Oriented Points"
  datasource="my_datasource"   width="400"  height="300"
  antialiase="true"
  format="PNG_STREAM">

  <themes>
    <theme name="theme1">
       <jdbc_query
         spatial_column="geom"  jdbc_srid="8265"
         render_style="transparent_color"
         label_column="label" label_style="t.street name"
         datasource="tilsmenv">
         SELECT SDO_GEOMETRY(2001, 8265, NULL,
                      SDO_ELEM_INFO_ARRAY(1, 1, 1, 3, 1, 0),
                      SDO_ORDINATE_ARRAY(12, 14, .3, .2))
         geom, 'Oriented Point' label FROM dual
       </jdbc_query>
     </theme>
  </themes>

  <styles>
    <style name="transparent_color">
      <svg width="1in" height="1in">
        <g class="color" style="stroke:#ff0000;stroke-opacity:0">
```

```
            <rect width="50" height="50"/>
          </g>
        </svg>
      </style>
    </styles>
</map_request>
```

Figure 2–2 shows part of the map generated by the request in Example 2–4. (The label is the phrase *Oriented Point*.)

Figure 2–2   Map Display of the Label for an Oriented Point

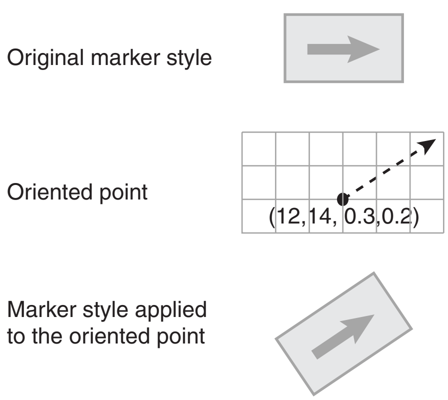


2.2.3.2 Controlling Marker Orientation

When a marker style is applied to an oriented point, MapViewer automatically rotates the marker style so that it points to the orientation vector. Any necessary rotation of the marker style is around the center of the marker.

Figure 2–3 shows how you can use an oriented point to control the orientation of marker styles. In this figure, the original marker style is first shown without any rotation. However, when the marker is applied to the same oriented point shown in Example 2–4 in Section 2.2.3.1, the marker style is rotated accordingly (in this case about 34 degrees counterclockwise) to reflect the orientation vector.

Figure 2–3   Oriented Marker



2.2.4 Making a Text Style Sticky

You can specify that a text style is "sticky," which means that any feature that uses it as a label style will always have its text label drawn on a map. Example 2–5 shows an XML definition of a style with the `sticky` attribute set to `true`.

Example 2–5 Text Style with Sticky Attribute

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<desc></desc>
 <g class="text" sticky="true" style =
"font-style:plain;font-family:Serif;font-size:11pt;font-weight:bold;fill:#000000">
    Hello World!
</g>
</svg>
```

2.2.5 Getting a Sample Image of Any Style

To get a sample image for any pre-defined style stored in a database, you can issue a simple HTTP request to the MapViewer server. This request can specify the size of the sample image, the background color, and the format of the returned image. Such requests are useful if you want to display a visual list of styles on a web page, to build a custom map legend, or just to see how various styles will appear.

The HTTP request has the following parameters, all of which are optional except for `sty`:

- `sty` (required) specifies the name of the style.
- `ds` specifies the data source where the style can be accessed. By default, the default MapViewer data source is used.
- `w` specifies the width of the sample image in pixels. The default value is 20.
- `h` specifies the height of the sample image in pixels. The default value is 20.
- `f` specifies the format of the sample image. Possible values are `png` for direct PNG image stream, `png_url` for the URL of a PNG image, `gif` for direct GIF image stream, or `gif_url` for the URL of a GIF image. The default value is `png`, which means the MapViewer server will directly stream the generated PNG image data back to the client without first saving it to the server disk.
- `bg` specifies the background color of the sample image. The format must be a hexadecimal string in the form of `0x`*`rrggbb`*, such as 0x808080 for a gray color. The default value is 0xffffff (white).

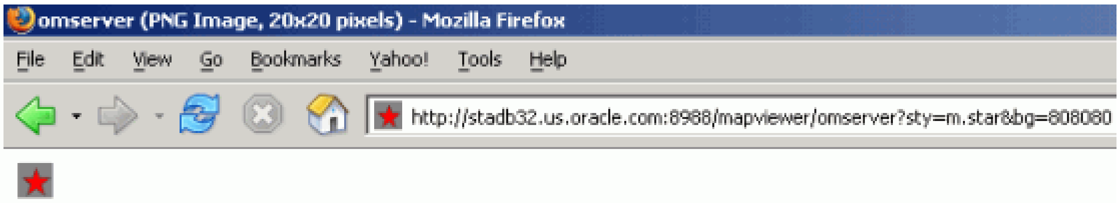
  For a transparent background, specify `bg` as an extended hexadecimal string to include the alpha values, in the format of `0x`*`aarrggbb`*. For example, 0x00ffffff will make the style image's background completely transparent, while 0x55ffffff is a white background with a transparency value of 0x55 (decimal value 80). The alpha value can range from 0x00 (completely transparent) to 0xff (completely opaque).
- `aa` specifies whether the sample image should be rendered in antialiasing mode. The default value is the string `true`. Specify the string `false` if you do not want to use antialiasing.

The following example generates an antialiased PNG image with a gray background with the default size of 20x20 pixels, displaying the marker style named `M.STAR` from the MapViewer default data source:

```
http://www.mycorp.com/mapviewer/omserver?sty=m.star&bg=808080
```

The preceding request generates a display similar to that in Figure 2–4.

Figure 2–4 Sample Image of a Specified Marker Style

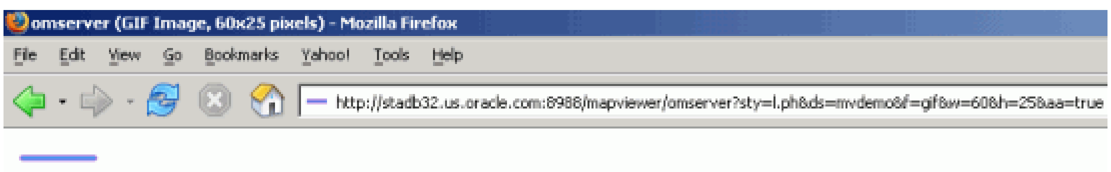


The following example generates an antialiased GIF image with the default white background, a width of 60 pixels, and a height of 25 pixels, displaying the line style named `L.PH` from the MapViewer data source named `mvdemo`:

```
http://www.mycorp.com/mapviewer/omserver?sty=l.ph&ds=mvdemo&f=gif&w=60&h=25&aa=true
```

The preceding request generates a display similar to that in Figure 2–5.

Figure 2–5 Sample Image of a Specified Line Style



2.3 Themes

Theme is perhaps the most important concept in MapViewer. A **theme** is a visual representation of a particular data layer. Conceptually, a theme is a collection of geographic features that share similar attributes, plus the rendering and labeling rules that tell MapViewer what styles to use to render and label the features. To be more exact, when you define a theme, you are actually providing MapViewer with the following information: where and how to get the data, and how to render and label the data.

Depending on how a theme is created, it can also be categorized as either a predefined theme or a dynamic (JDBC) theme. For a **predefined theme**, the theme's definition is created in the standalone Map Builder tool and stored in the database. For a **dynamic theme**, the theme's definition (XML) is created in real time by an application. Dynamic themes typically employee a custom SQL query constructed by the application to get its data.

Typically, the data for a theme comes from a spatially enabled table, that is, a database table or view with a column of type SDO_GEOMETRY. For example, a theme named `US_STATES` might be based on a STATES table that has a column named GEOMETRY, plus any other nonspatial attribute columns. This type of theme is often called a geometry theme. Besides geometric data, other types of database-managed geographic data can be associated with corresponding types of themes; for example:

- Georeferenced images stored in BLOBs (image themes)
- Oracle Spatial and Graph GeoRaster data (GeoRaster themes)
- Oracle Spatial and Graph network data model (network themes)
- Oracle Spatial and Graph topology data model (topology themes)

- Cartographic annotation text (annotation themes)

MapViewer themes can be used to render not only geographic data stored in a database, but also data originating from other sources, such as web services (WFS, WMS, and WMTS) or the local file system (through the custom spatial data provider interface).

Regardless of what type of data is associated with a theme (except for WMS and WMTS themes, which represent externally rendered map layers), the MapViewer styling rules still need to be defined for each theme, and the styles referenced by the styling rules must exist and be stored in the database as part of the mapping metadata.

This section contains the following major subsections:

- Section 2.3.1, "Predefined Themes"
- Section 2.3.2, "JDBC Themes"
- Section 2.3.3, "Image Themes"
- Section 2.3.4, "GeoRaster Themes"
- Section 2.3.5, "Network Themes"
- Section 2.3.6, "Topology Themes"
- Section 2.3.7, "WFS Themes"
- Section 2.3.8, "WMTS Themes"
- Section 2.3.9, "Custom Geometry Themes"
- Section 2.3.10, "Annotation Text Themes"
- Section 2.3.11, "Thematic Mapping"
- Section 2.3.12, "Attributes Affecting Theme Appearance"

2.3.1 Predefined Themes

A **predefined theme** is a theme whose definition is stored in a user's database schema. All predefined themes for a database user are stored in that user's USER_SDO_THEMES view (described in Section 2.9, especially Section 2.9.2). When you include a predefined theme in a map request, you need to specify only the theme name. MapViewer automatically finds the theme's definition, constructs a query based on it, retrieves the relevant spatial and attribute data, and renders the data according to the styling rules for the theme.

Each predefined theme must have an associated base table or view. If you base a theme on a view, you must insert a row in the view owner's USER_SDO_GEOM_METADATA view (described in *Oracle Spatial and Graph Developer's Guide*) specifying the view and its spatial column. If the view is a join view (that is, if it is based on multiple tables), you must specify the `key_column` attribute (described in Section A.7) in the theme's styling rules. The reason for this requirement is that MapViewer by default caches geometries for a predefined theme based on the rowid in the base table; however, for a join view there is no ROWID pseudocolumn, so you must specify a key column.

For most types of predefined themes (but not WMS themes), you can use the Map Builder tool to create and preview themes. For information about the Map Builder tool, see Chapter 9.

2.3.1.1 Styling Rules in Predefined Spatial Geometry Themes

Each predefined theme is always associated with one or more **styling rules**, specifications in XML format that control aspects of how the theme is displayed. This section describes styling rules for predefined spatial geometry themes, such as the airport theme shown in Example 2–6. Other types of themes, such as image, GeoRaster, network, and topology themes, have their own distinct styling rules requirements, and these are discussed in sections that explain these themes. However, the styling rules for all types of themes are grouped under the <styling_rules> element in an XML document, which is stored in the STYLING_RULES column for each predefined theme in the USER_SDO_THEMES view. (The <styling_rules> DTD is described in Section A.7.)

Note: The following naming conventions are used for prefixes in style names in the examples in this chapter: v. indicates variable (advanced style), m. indicates marker, c. indicates color, l. indicates line, and t. indicates text. (If the style is not under the current user's schema, you must specify the owner's schema name followed by a colon. For example: mdsys:c.red.)

In the content (character data) of an XML document, < and > must be used to represent < and >, respectively. Otherwise, < or >, such as in WHERE CATEGORY > 'B', will be interpreted by the XML parser as part of an XML tag.

Example 2–6 XML Definition of Styling Rules for an Airport Theme

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule>
    <features style="c.black gray">
    runway_number &gt; 1
    </features>
    <label column="name" style="t.airport name">
      1
    </label>
  </rule>
  <rule>
    <features style="m.airplane">
    runway_number = 1
    </features>
  </rule>
</styling_rules>
```

Each styling rule has a required <features> element and an optional <label> element. The <features> element specifies which row or rows (features) in the table or view will be selected based on the user-defined predicate and on the style to be used for the selected features. You can specify any valid SQL predicate as the value of this element. The <label> element specifies whether or not to annotate the selected features, and if so, which column in the table or view to use for text labels.

In Example 2–6, there are two styling rules associated with the Airport theme:

- The first rule specifies that only those rows that satisfy the condition runway_ number > 1 (that is, runway number greater than 1) will be selected, and these will be rendered using the style named c.black gray. If no value is supplied, no WHERE clause condition is applied. For example, assume that the

definition had been the following (that is, omitting the `runway_number > 1` condition):

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule>
    <features style="c.black gray"/>
    <label column="name" style="t.airport name">
      1
    </label>
  </rule>
</styling_rules>
```

In this case, all airport features would be selected and would be rendered using the color style named `c.black gray`.

The first rule also has a `<label>` element, which specifies that the NAME column in the table or view will be used to annotate each airport, using the text style `t.airport name`. The value of the `<label>` element, which can be any SQL expression that evaluates to a numeric value, is used to determine whether or not a feature will be annotated. If the numeric value is greater than zero, the feature will be annotated. In this case, because the value is the constant 1, all features specified by the `<features>` element will be annotated, using the values in the NAME column. If the value is less than or equal to zero for a feature, that feature will not be annotated.

- The second rule, which applies to those airports with only one runway, does not have a `<label>` element, thus preventing all such airports from being annotated. In addition, the features that satisfy the second rule will be rendered using a different style (`m.airplane`), as specified in its `<features>` element.

You can think of each styling rule as a filter into the base table or view of the theme, because it selects only a subset of the rows and applies the rendering and labeling styles of that rule. In fact, MapViewer formulates a complete SQL query for each styling rule. This query string follows a fixed format, as described in Section 2.3.1.2.

2.3.1.2 How MapViewer Formulates a SQL Query for a Styling Rule

To see how MapViewer formulates a SQL query for a styling rule, consider the first styling rule from the airport theme example (Example 2–6 in Section 2.3.1.1):

```
<styling_rules>
  <rule>
    <features style="c.black gray">
    runway_number &gt; 1
    </features>
    <label column="name" style="t.airport name">
      1
    </label>
  </rule>
  . . .
</styling_rules>
```

When MapViewer processes this theme, it formulates a query string for this styling rule that looks like this:

```
SELECT ROWID, GEOMETRY, 'C.BLACK GRAY', NAME, 'T.AIRPORT NAME', 1, 'rule#0'
  FROM AIRPORT_POINT
  WHERE MDSYS.SDO_FILTER(GEOMETRY,
    MDSYS.SDO_GEOMETRY(2003, 8265, NULL, MDSYS.SDO_ELEM_INFO_ARRAY(1, 1003, 3),
    MDSYS.SDO_ORDINATE_ARRAY(:mvqboxxl, :mvqboxyl, :mvqboxxh, :mvqboxyh)),
```

```
'querytype=WINDOW') = 'TRUE'
```

In the preceding query string:

- The base table name of the theme, AIRPORT_POINT, appears in the FROM clause
- The SELECT list includes ROWID as the first column. ROWID is the default key_column attribute of a predefined theme.
- The next column in the SELECT list is GEOMETRY. This is the geometry column of this theme.
- The next column in the SELECT list is the literal string 'C.BLACK GRAY', which is the rendering style name for this rule.
- The next column in the SELECT list is the column NAME, which will provide the label text. It is specified in the <label> element of this styling rule.
- The next column in the SELECT list is the literal string 'T.AIRPORT NAME', which is the labeling style name specified in the <label> element.
- The next column in the SELECT list is the literal value 1, which is the value of the <label> element itself.
- The next column in the SELECT list is the literal string 'rule#0'. This is used internally by MapViewer only.
- The large WHERE clause is essentially an Oracle Spatial and Graph filtering operator, SDO_FILTER. This WHERE clause is automatically added by MapViewer (and is *not* something you need to specify when defining a theme). It ensures that only those geographic features that are in contact with the current map viewing window will be fetched from the base table. The four binding variables, mvqboxxl, mvqboxyl, mvqboxxh and mvqboxyh, will be automatically filled in with the coordinates for the current map viewing window.

MapViewer always uses the preceding format when constructing SQL queries for the styling rules of a predefined geometry theme's styling rules. It uses different formats for the queries for other types of themes, such as a topology or GeoRaster theme. The formats for these other queries are not described here; however, if you are interested, you can set the logging level of your MapViewer instance to FINEST, submit a map request containing a particular type of theme, and check the MapViewer log file to see the exact query that MapViewer constructs.

Each row (or feature) in the query's result set now contains all the information MapViewer needs: the spatial data, the rendering and labeling style names, the label text, and the labeling conditions. MapViewer then constructs an in-memory feature object for each row and sends them to the rendering pipeline to be displayed on the map.

If two or more styling rules are specified for a theme, a UNION ALL operation is performed on the SQL queries for the rules (from first to last) to fetch the qualified features from the table or view.

If an advanced style is specified in a rule, the SELECT list of the query for that rule will include the additional attribute column or columns that are required by the advanced style.

2.3.1.3 Styling Rules with Binding Parameters

As explained in Section 2.3.1.2, the <features> element of a styling rule can define a query condition to select features from the base table or view. This query condition typically contains hard-coded SQL expressions, such as runway_num > 1 in the

airport theme. However, you can instead include binding variables in the query predicate. Such a theme is often called a *templated theme*, because it is essentially defining a template for how to display certain features, and the exact set of features is determined at runtime by providing a binding value to the query predicate.

The concept of templated theme allows you to define a single theme and to have the binding values change between map requests. For example, consider the following styling rule:

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule>
    <features style="C.RED"> (state_abrv=:1) </features>
    <label column="STATE" style="T.STATE NAME"> 1 </label>
  </rule>
</styling_rules>
```

The preceding styling rule defines a `<features>` element with a query condition based on the value of the `state_abrv` attribute, which the application must supply. In MapViewer requests, the binding parameter must be defined on the theme section, and each binding parameter is defined by a value and by a SQL type. In the following theme definition on a map request, the state abbreviation value is `ME` and the variable SQL type is `String`. The value `ME` will be used with the predefined theme styling rule.

```
<theme name="THEME_US_DYN_STATES" >
   <binding_parameters>
      <parameter value="ME" type="String"/>
   </binding_parameters>
</theme>
```

2.3.1.4 Applying Multiple Rendering Styles in a Single Styling Rule

The `<feature>` element of a styling rule allows you to specify only one rendering style using the `style` attribute. If you want to apply multiple rendering styles to a feature without using multiple themes, you cannot specify multiple styling rules, because each rule selects a different subset of features. To apply multiple rendering styles to a feature without using multiple themes, you must use the `<rendering>` element instead of the `style` attribute of the `<features>` element.

The `<rendering>` element has the format shown in the following example:

```
<rendering>
 <style name="V.POIVMK" value_columns="FEATURE_CODE">
  <substyle name="V.POIVBKT" value_columns="POINT_ID" changes="FILL_COLOR"/>
 </style>
</rendering>
```

In the `<rendering>` element, the `<style>` element specifies the name of the style to use when rendering features, and one or more value columns (comma-delimited) for use with advanced styles. In the preceding example, the style name is `V.POIMVK` and the value column is FEATURE_CODE.

In the `<style>` element, the `<substyle>` element enables rendering of a feature using a combination of two attribute values.,such as defining the feature shape by the `<style>` element and the feature color by the `<substyle>` element. This is useful for rendering point features once but based on two attribute values. You can specify one or more value columns (comma-delimited), and the change to be applied (only `FILL_COLOR` is currently supported).

You can specify multiple <style> elements with a <rendering> element, to achieve the following goals:

- To create an advanced style in which a base advanced style, associated with some attributes (columns), can have its rendering affected by some other attributes through the use of a substyle. For example, an advanced style can display markers of different sized based on one value column, while using a secondary color style to change the fill color of those markers based on another value column.
- To use multiple styles to render a feature (achieving the effect of stacked styles).

Example 2–7 shows a predefined theme styling rule that uses the <rendering> element. The <features> element is part of the rules and must be define, because it also specified the query condition, but no style attribute is specified. The <rendering> element defines how to render the features.

Note: The use of styling rules with the <rendering> element, as shown in Example 2–7, is not compatible with MapViewer release 10.1.3.1 and earlier releases.

Example 2–7 Styling Rules Using the <rendering> Element

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
 <rule>
   <features> </features>
   <label column="NAME" style="T.STREET2"> 1 </label>
   <rendering>
      <style name="V.POIVMK" value_columns="FEATURE_CODE">
        <substyle name="V.POIVBKT" value_columns="POINT_ID" changes="FILL_COLOR"/>
      </style>
   </rendering>
 </rule>
</styling_rules>
```

See also Section 3.1.12, which contains an example that uses the <rendering> element.

The <rendering> element can also be used with dynamic themes, geometry themes, and topology themes.

2.3.1.5 Caching of Predefined Themes

By default, MapViewer automatically caches the spatial data for a predefined theme when it is fetched from the database for processing by the MapViewer rendering engine. By contrast, data for dynamic (JDBC) themes is never cached in MapViewer. If you do not want any data for a predefined theme to be cached (such as for a theme whose underlying base table is constantly being updated), you can set the caching attribute to NONE in the <styling_rules> element for the theme. (The <styling_rules> element, including the caching attribute, is described in Section A.7.)

For frequently used themes whose base data is static or read-only, specify caching ALL for the best performance. This causes MapViewer, when it first accesses the theme definition, to fetch all the features (including spatial data, attribute data, and styling information associated with them) and cache them in the MapViewer memory, creating an in-memory R-tree for the theme's spatial data. All subsequent requests requiring that theme occur locally instead of going to the database.

If the caching attribute value is NORMAL (the default), each time a map involving that theme is requested, MapViewer queries the database to get the spatial data and any associated attribute data. However, if any of the spatial geometry data, as referenced by rowid or a user-specified key column, has already been cached, the unpickling process (the conversion from the raw database geometry format to a Java geometry object) is skipped. Still, if memory is not an issue and if a frequently used theme can completely fit in the cache, you should specify caching ALL, to eliminate virtually all database access for that theme after the initial loading.

Because the MapViewer spatial data cache is global, all predefined themes that are accessed by MapViewer compete for a global fixed-sized memory cache. The cache resides completely in memory, and you can specify the maximum size of the cache as explained in Section 1.5.2.6. When the cache limit is reached, older cached data is removed from the cache to make room for the most recently accessed data, except that data for themes specified with caching ALL is not removed from the cache, and MapViewer does not requery the database for these themes.

Caching is currently disabled for predefined annotation and custom geometry themes. For custom geometry themes, you can implement a caching mechanism in your provider implementation. However, for each request, a new instance of your provider is created; and if you implement a local caching mechanism, it will be lost.

2.3.1.6 Feature Labels and Internationalization

MapViewer includes support for translated theme labels. Typically with a predefined MapViewer theme, you can specify a label column that will provide all the text strings for labeling each feature of the theme. These text strings are string values stored in the database table column, in a specific language (such as English). However, you can also supply different translations of these stored string values by using a *resource bundle*. When such translated text strings are available, you can instruct MapViewer to label the features of a theme using a specific language or locale.

Note: Only predefined geometry themes support resource bundles at this time.

The steps for supplying translations and instructing MapViewer to label a theme using a specific user language are as follows:

1. Prepare the translations.

   A typical MapViewer predefined geometry theme gets all the underlying data from a table. You can specify one of the (string type) columns as the labeling column for this theme. This is called the label column. When a label column needs to be translated into different languages, you extract all the values from the table, and store them in a properties file, such as StringResources.properties. (Note that the file name StringResources.properties assumes that the extracted texts are all in English. If they are not, then the properties file name needs to follow a convention where the language code, and an optional region or country code, is a suffix in the file name. For example, StringResources_fr.properties will contain French translations only, while StringResources_zh_CN.properties is for simplified Chinese.)

   A properties file is a plain text file that follows a very simple format. For example, a simple StringResources.properties file might contain the following:

   ```
   # This is the English version of the strings.
   California = California
   Nevada = Nevada
   ```

```
Montana = Montana
```

The first line is a comment, and starts with the # character. Each subsequent line contains one pair of key (first string) and value (second string). The keys come directly from the label column, whereas the values are corresponding translations. Because this particular file contains the default English text strings, the key and the value (translation) are the same in each case. Note that the keys should always be in English.

From this default properties file, your translation specialists should create a set of property files, one file for each translation. Using the preceding simple example, the translated file for simplified Chinese (`StringResources_zh_CN.properties`) should look like the following, in which the value of each key has been replaced by the Chinese translation of the key, encoded as a Unicode string:

```
# This is the Chinese version of the strings.
California =  \u6CA1\u6709\u8981\u5448\u73B0\u7684\u4E3B\u9898\u3002
Nevada = \u65E0\u6CD5\u52A0\u8F7D\u4E3B\u9898\u3002
Montana = \u65E0\u6CD5\u52A0\u8F7D\u6837\u5F0F\u3002
```

The default properties file, `StringResouces.properties`, plus all the language specific files that share the same file name (except for the language and region suffixes) collectively form what is called a *resource bundle*. In this case the resource bundle is named `StringResources`. You can name your resource bundles with any name you like, but different bundles (containing different set of keys) should always use different base names.

For more information about Java resource bundles and properties files, see the Java language documentation.

2. Supply the translated text strings as a Java Resource Bundle, which can be based on either Java resource classes or plain properties files.

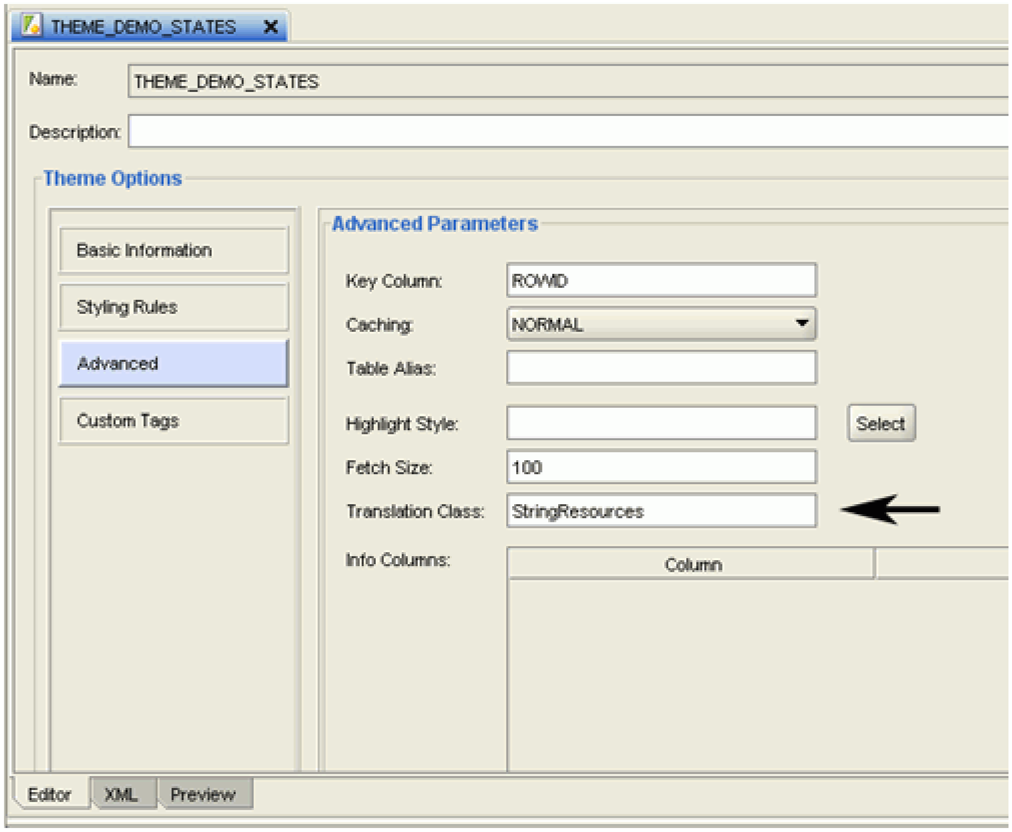
   After all the label text strings have been translated, you must place all the files (the resource bundle) in the MapViewer CLASSPATH so that MapViewer can find these files at runtime. Typically, you can use the MapViewer `WEB-INF/classes` folder: copy all the files including the base `StringResources.properties` and language-specific files (such as `StringResources_fr.properties` and `StringResources_zh_CN.properties`) into this folder.

   If you place all the files of a resource bundle into a subfolder under `WEB-INF/classes`, then the name of the resource bundle (as known to MapViewer) will need to be prefixed with this subfolder name. This is similar to how one places a Java class in a directory structure that follows the package names. For example, if you put all the `StringResources*.properties` files in `WEB-INF/classes/i18n/`, then later when you register the resource bundle with MapViewer, the actual name of your resource bundle should be `i18n.StringResources`.

3. Specify the name of the resource bundle in the theme definition by registering the resource bundle with MapViewer.

   For MapViewer to find your translated classes, you must specify the complete name of your resource bundle in the theme definition. The easiest way to do this is with the Map Builder utility, specifying the resource bundle name as the Translation Class in the Advanced Parameters pane of the theme editor. Figure 2–6 shows `StringResources` being specified for the Translation Class.

Figure 2–6 Specifying a Resource Bundle for a Theme



As mentioned in the preceding step, if your resource bundle files are located in a subfolder of , then the subfolder name must the base name of your resource bundle, separated by a period, as if the resource bundle files were Java classes in a package.

4. Specify a language parameter when requesting a map or theme.

Specify the preferred language for each map request the Oracle Maps JavaScript API (described in Section 8.4) or the XML map request API (described in Chapter 3).

- In JavaScript code, specify the label language code in the call to the MVThemeBasedFOI class. The following example causes the FOI theme to display its labels in simplified Chinese:

```
themebasedfoi = new MVThemeBasedFOI('themebasedfoi1','mvdemo.theme_demo_
states');
themebasedfoi.setLabelLanguageCode("zh-cn");
themebasedfoi.enableLabels(true);
```

With the `setLabelLanguageCode(lang_code)` method, you can specify a language code so that MapViewer labels the features using the text strings for the specified language, which must be a 2 letter language code (such as `zh`), followed optionally a hyphen (-) and a 2-letter country code (such as `zh-cn`). The language codes are defined by the ISO 639 standards and are listed at several websites, such as `http://www.loc.gov/standards/iso639-2/php/English_list.php`. If no translated text strings for the specified language code are

found, the English text strings (or whatever the default strings are for the theme) will be used for labeling.

- In an XML map request, specify the language in the `lang` attribute. The following example causes the labels to be displayed in simplified Chinese:

```
<map_request title="Oracle LBS MAP"
basemap="demo_map"
datasource = "mvdemo"
width="640" height="480"
lang="zh-CN"
format="PNG_STREAM">

<center size="5.15">
<geoFeature> <geometricProperty typeName="center">
  <Point> <coordinates>-122.2615, 37.5266</coordinates>
  </Point> </geometricProperty>
</geoFeature>
</center>
</map_request>
```

Only language codes and country codes specified by the ISO 639 standards can be used as possible `lang` values. If an optional country code is used, it must be connected to the language code by a hyphen (-). Country codes and language codes are not case sensitive.

If the `lang` attribute is specified as part of the XML map request, every theme rendered to the result map it checked to see if it has an associated resource bundle. If a theme does not have an associated resource bundle, or the translated text strings for the specified language cannot be found, the default values (those stored in the table column) are used.

If the `lang` attribute is not specified as part of the XML map request, the default text string values (those stored in the table column) are always used, regardless of which locale in effect for MapViewer itself (or rather, its containing JVM).

2.3.2 JDBC Themes

A **JDBC theme** is a theme that is dynamically defined with a map request. JDBC themes are not stored permanently in the database, as is done with predefined themes.

For a JDBC theme, you must specify a valid SQL query that retrieves all the necessary spatial data (geometries or other types of data, such as image, GeoRaster, network, or topology). If attribute data is needed, such as for thematic mapping or spatial data analysis, the query must also select it. In other words, you must provide a correct and complete query for a JDBC theme. In addition to the query, you can also specify the rendering and labeling styles to be used for the theme.

For a JDBC theme based on spatial geometries, MapViewer processed the columns specified in the query according to the following rules:

- The column of type SDO_GEOMETRY is treated as the spatial data column.
- Any column whose name or alias matches that specified in the JDBC theme's `label_column` attribute is treated as the labeling column, whose values are used as text for labels.
- Any other columns are treated as attribute data columns, which may or may not be used by MapViewer. For example, if the rendering style is an advanced style, any attribute columns are processed by that style in the order in which they

appear in the SELECT list in the query. Thus, if you are performing thematic mapping and using an advanced style, you must specify all attribute columns that are needed for the thematic mapping, in addition to the geometry column and optional labeling column. (A labeling column can also be an attribute column, in which case you do not need to specify that column in the SELECT list.)

Example 2–8 is a map request that includes a JDBC theme.

Example 2–8 JDBC Theme in a Map Request

```
<?xml version="1.0" standalone="yes"?>
<map_request title="My MAP" datasource = "mvdemo">

  <themes>
    <theme name="jdbc_theme_1">
       <jdbc_query
            datasource="mvdemo"
            jdbc_srid="41052"
            spatial_column="geometry"
            render_style="C.RED">
          SELECT geometry from states where name='MA'
       </jdbc_query>
     </theme>
  </themes>

</map_request>
```

The full query that MapViewer executes for the JDBC theme in Example 2–8 is:

```
SELECT geometry FROM states WHERE name='MA';
```

For this request, MapViewer generates a map that contains only the selected geometry as a result of executing this JDBC theme's query. In a more typical case, however, the map request will need to use several JDBC themes to plot additional dynamic data on top of the base map. Furthermore, the map request may have a query window associated with it; that is, the user may want to see only a portion of the area included in the whole base map. In this case, the SQL queries in the JDBC themes will be subjected to a spatial window query, to eliminate any unwanted results.

For more information about JDBC themes, see the information about the <jdbc_query> element in Section 3.2.9.

2.3.2.1 Defining a Point JDBC Theme Based on Two Columns

If a database table uses two columns (such as longitude and latitude) to represent a point coordinate, you can define a JDBC theme based on the two columns to render points. The table does not need to have a spatial geometry column, but it can have one; however, if the theme request defines the point columns and also the geometry column, MapViewer will try to render the points using the two columns, not the geometry column.

Example 2–9 is a JDBC theme that renders points from two columns, named LONG_LOC and LAT_LOC, of a table named POI. The x_column and y_column attributes specify the columns containing the point coordinate values. In this example, the points are rendered using the C.RED style, and the table values from the NAME column are rendered using the T.POI_NAME style.

Example 2–9 JDBC Theme Based on Columns

```
<map_request>
```

```
  . . .
  <center>
   . . .
  </center>
  <themes>
    <theme name="theme1" >
       <jdbc_query
         datasource="mvdemo"
         jdbc_srid="8265"
         x_column="long_loc"
         y_column="lat_loc"
         render_style="C.RED"
         label_column="name"
         label_style="T.POI_NAME"
         >SELECT long_loc, lat_loc,name FROM poi
       </jdbc_query>
     </theme>
  </themes>
</map_request>
```

If the request specifies a valid query window (that is, not the full extent), a WHERE expression based on the size of the request window is automatically added to the query.

If the table has a geometry column, you can specify SQL code to use the geometry column as a filter. Example 2–10 is similar to Example 2–9, but it adds the use of the SDO_FILTER operator to specify a query window based on the geometry in the column named GEOMETRY. In Example 2–10, the question mark (?) characters indicate that the lower-left and upper-right coordinates of the query window rectangle are taken from values supplied at runtime (not shown in this example).

Example 2–10 JDBC Theme Based on Columns, with Query Window

```
<map_request>
  . . .
  <center>
   . . .
  </center>
  <themes>
    <theme name="theme1" >
       <jdbc_query
         datasource="mvdemo"
         jdbc_srid="8265"
         x_column="long_loc"
         y_column="lat_loc"
         render_style="C.RED"
         label_column="name"
         label_style="T.POI_NAME"
         >SELECT long_loc, lat_loc FROM poi WHERE
           SDO_FILTER(geometry,MDSYS.SDO_GEOMETRY(2003, 8265, NULL,
           MDSYS.SDO_ELEM_INFO_ARRAY(1, 1003, 3),
           MDSYS.SDO_ORDINATE_ARRAY(?,?,?,?)),
           'querytype=WINDOW') = 'TRUE'
       </jdbc_query>
     </theme>
  </themes>
</map_request>
```

2.3.2.2 Storing Complex JDBC Themes in the Database

Sometimes the SQL query for a JDBC theme is so complex that you may want to save the query. In such cases, you can define a predefined theme (whose definition is stored in the database's USER_SDO_THEMES view), and then include the full SQL query as the content of the <features> element in the styling rules for that theme.

The feature style specified in the <features> element is then used to render the geometries retrieved using the full query. The base table as defined for such a theme is ignored because the full SQL query already includes a FROM clause. The geometry column defined in the USER_SDO_THEMES view is still needed, and it must be the same as the geometry column selected in the user-supplied SQL query. If you have a <label> element for a styling rule, the label style specified is used to label the geometries, as long as the query selects a column that contains label text.

Example 2–11 is a sample <styling_rules> element of a predefined theme with a complex SQL query.

Example 2–11 Complex Query in a Predefined Theme

```
<?xml version="1.0" standalone="yes"?>
 <styling_rules>
   <rule>
     <features style="L.POOR_ROADS" asis="true">
        select sdo_lrs.clip_geom_segment(geometry,start_measure,end_measure)
              geometry
        from (select /*+ no_merge use_hash(a b) */
                a.street_id, name, start_measure, end_measure, geometry
             from (select /*+ no_merge */ a.street_id, name, geometry
                   from philly_roads a
                   where sdo_filter(geometry,sdo_geometry(2002,41124,null,
                    sdo_elem_info_array(1,2,1),
                                       sdo_ordinate_array(?,?,?,?)),
                                 'querytype=window')='TRUE') a,
                   philly_road_conditions b
             where condition='POOR' and a.street_id = b.street_id)
     </features>
   </rule>
 </styling_rules>
```

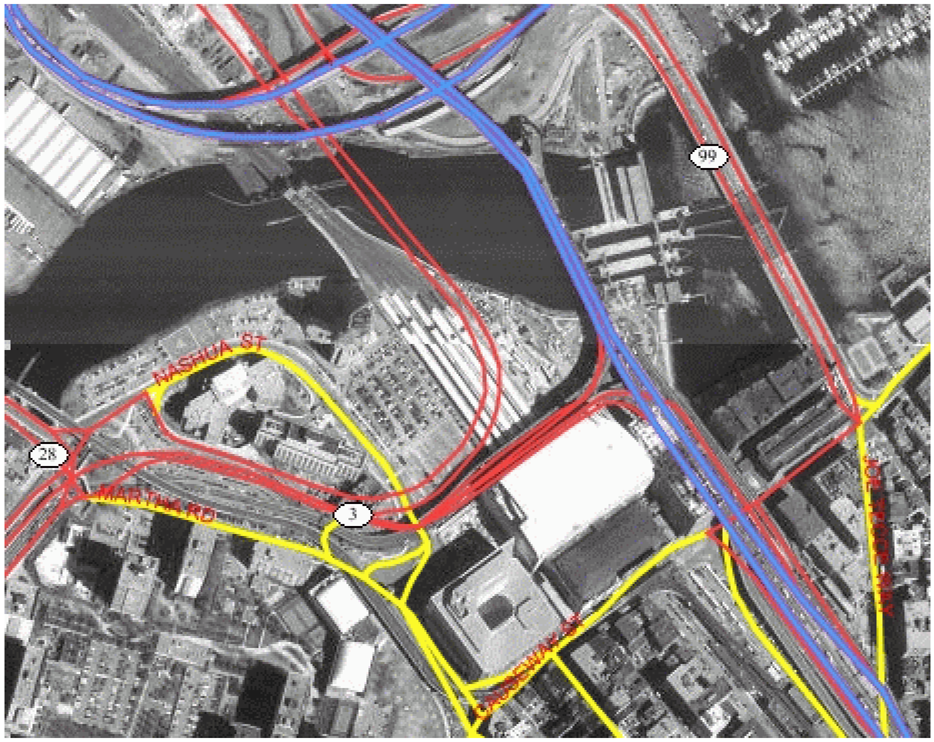
Even though Example 2–11 is defined as a predefined theme, MapViewer still treats it as a JDBC theme at runtime when a user requests a map that includes this theme. As with a normal JDBC theme, MapViewer by default imposes a window filtering process (if a query window was included in the map request) on top of the SQL query. To override this default behavior and have the supplied query string executed without any modification, specify `asis="true"` in the <features> element, as shown in Example 2–11. (For information about the `asis` attribute, see Section 3.2.9.)

2.3.3 Image Themes

An **image theme** is a special kind of MapViewer theme useful for visualizing geographically referenced imagery (raster) data, such as from remote sensing and aerial photography.

You can define an image theme dynamically or permanently (as a predefined theme) in the database. You can use image themes with vector (nonimage) themes in a map. Figure 2–7 shows a map in which an image theme (showing an aerial photograph of part of the city of Boston) is overlaid with themes showing several kinds of roadways in the city.

Figure 2–7 Image Theme and Other Themes Showing Boston Roadways



Before you can define an image theme, you must follow these rules in organizing your image data:

- Store image data in its original format (such as JPEG) in a BLOB column in a database table, or as an Oracle Multimedia object (ORDSYS.ORDImage) that points to the original image file. For information about creating an ORDSYS.ORDImage object, see *Oracle Multimedia User's Guide*.
- Add a geometry (SDO_GEOMETRY) column to the same table, and store the minimum bounding rectangle (MBR) for each image in that column.

  Each geometry in the MBR column contains the geographic bounds for an image, not its size in the pixel space. For example, if an orthophoto image is 2000 by 2000 pixels in size, but covers a ground rectangle starting at the corner of (936000, 248000) and having a width and height of 8000 meters, the MBR for the geometry column should be populated with (936000, 248000, 944000, 256000).
- Insert an entry for the geometry column in the USER_SDO_GEOM_METADATA view.
- Create a spatial index on the geometry column.

To predefine an image theme, follow the guidelines in Section 2.3.3.1. To define a dynamic image theme in a map request, follow the guidelines for defining a JDBC theme, as explained in Section 2.3.2 and Section 3.2.9, but note the following additional considerations with dynamic image themes:

- You must provide the original image resolution information when defining an image theme.

- MapViewer by default automatically scales the image data when generating a map with an image theme, so that it fits the current query window. To disable this automatic scaling, specify `imagescaling="false"` in the map request.

For any image theme definition, MapViewer supports only GIF, JPEG, PNG, and TIFF image formats. To enable MapViewer to visualize data in any other image format, you must implement a custom image renderer using the `oracle.sdovis.CustomImageRenderer` interface in Java, and then register your implementation class in the `mapViewerConfig.xml` file (to tell MapViewer which custom image renderer to use for image data in a specific format). For detailed information about implementing and registering a custom image renderer, see Appendix C.

For an example of a map request specifying an image theme, including an explanation of how MapViewer processes the request, see Example 3–6 in Section 3.1.6.

2.3.3.1 Creating Predefined Image Themes

To create a predefined image theme, you must store the definition of the image theme in the database by inserting a row into the USER_SDO_THEMES view (described in Section 2.9.2). Example 2–12 stores the definition of an image theme.

Example 2–12 Creating a Predefined Image Theme

```
INSERT INTO user_sdo_themes  VALUES (
   'IMAGE_LEVEL_2',
   'Orthophotos at pyramid level 2',
   'IMAGES',
   'IMAGE_MBR',
   '<?xml version="1.0" standalone="yes"?>
    <styling_rules theme_type="image" image_column="image"
                   image_format="JPEG" image_resolution="2"
                   image_unit="M">
      <rule >
        <features style="C.RED"> plevel=2 </features>
      </rule>
    </styling_rules>' );
```

Example 2–12 creates an image theme named `IMAGE_LEVEL_2`. The base table (where all image data and associated MBRs are stored) is named IMAGES, and the minimum bounding rectangles (MBRs) for the images are stored in the column named IMAGE_MBR. In the STYLING_RULES column of the USER_SDO_THEMES view, an XML document with one `<styling_rules>` element is inserted.

The `<styling_rules>` element for an image theme has the following attributes:

- `theme_type` must be `image` in order for this theme to be recognized as an image theme.
- `image_column` specifies the column in the base table or view that stores the actual image data.
- `image_format` is a string identifying the format of the image data. If you specify `GIF` or `JPEG`, MapViewer can always render the image data. If you specify any other value, such as `ECW`, you must have implemented a custom image renderer and registered it to MapViewer in order for the image to be rendered properly. For information about implementing a custom image renderer, see Appendix C.
- `image_resolution` is an optional attribute that identifies the original image resolution (number of `image_unit` units for each pixel).

- `image_unit` is an optional attribute, except it is required if you specify the `image_resolution` attribute. The `image_unit` attribute specifies the unit of the resolution, such as `M` for meter. The value for this attribute must be one of the values in the SDO_UNIT column of the MDSYS.SDO_DIST_UNITS table. In Example 2–12, the image resolution is 2 meters per pixel.

The DTD for the `<styling_rules>` element is presented in Section A.7.

2.3.4 GeoRaster Themes

A **GeoRaster theme** is a special kind of MapViewer theme useful for visualizing GeoRaster objects. GeoRaster is a feature of Oracle Spatial and Graph that lets you store, index, query, analyze, and deliver raster image and gridded data and its associated metadata. GeoRaster objects are defined using the SDO_GEORASTER data type. For detailed information about GeoRaster, see *Oracle Spatial and Graph GeoRaster Developer's Guide*.

Before you can use MapViewer with GeoRaster themes, you must ensure that the Java Advanced Imaging (JAI) library files (`jai_core.jar` and `jai_codec.jar`) are in the MapViewer library path, as explained in Section 1.4. You must also perform the following actions with the GeoRaster data:

1. Georeference the GeoRaster data to establish the relationship between cell coordinates of the GeoRaster data and real-world ground coordinates (or some other local coordinates).

   If you are using Oracle Database Release 10.1, you must also set the spatial resolution values.

2. Generate or define the spatial extent (footprint) associated with the raster data.
3. Optionally, generate pyramid levels that represent the raster image or data at different sizes and degrees of resolution.
4. Insert a row into the USER_SDO_GEOM_METADATA view that specifies the name of the GeoRaster table and the SPATIALEXTENT attribute of the GeoRaster column (that is, the column of type SDO_GEORASTER). The following example inserts a row for a table named GEOR_TABLE with a GeoRaster column named GEOR_COLUMN:

   ```
   INSERT INTO USER_SDO_GEOM_METADATA VALUES
   ( 'geor_table',
     'geor_column.spatialextent',
     SDO_DIM_ARRAY(
       SDO_DIM_ELEMENT('X', 496602.844, 695562.844, 0.000005),
       SDO_DIM_ELEMENT('Y',8788409.499,8973749.499, 0.000005)
     ),
     82279   -- SRID
   );
   ```

5. Create a spatial index on the spatial extent of the GeoRaster table. The following example creates a spatial index named GEOR_IDX on the spatial extent of the table named GEOR_TABLE:

   ```
   CREATE INDEX geor_idx ON geor_table(geor_column.spatialextent)
     INDEXTYPE IS MDSYS.SPATIAL_INDEX;
   ```

Example 2–16 in Section 2.3.4.1 prepares GeoRaster data for use and stores a GeoRaster theme in the database.

MapViewer supports two types of map requests with objects from a GeoRaster table:

- A request containing a SQL statement to select one or more GeoRaster objects
- A request specifying a single GeoRaster object by the combination of its raster data table name and its `rasterID` attribute value in the SDO_GEORASTER object. (The `rasterID` attribute value in the SDO_GEORASTER object is distinct from and unrelated to any primary key or ID column in the GeoRaster table.)

The following elements and attributes apply to the definition of a GeoRaster theme:

- `<jdbc_georaster_query>` element: Specifies that this is a dynamically defined GeoRaster theme. For a theme that uses a SQL statement to select one or more GeoRaster objects, this element contains the SQL query statement (without a terminating semicolon). The complete DTD for this element is included in the map request DTD in Section 3.2.
- `georaster_table` attribute: Specifies the name of the GeoRaster table.
- `georaster_column` attribute: Specifies the name of the column of type SDO_GEORASTER in the GeoRaster table.
- `polygon_mask` attribute (optional): Specifies a set of two-dimensional coordinates representing a polygon, to be used as a mask to make transparent the part of the GeoRaster image that is outside the polygon mask. The coordinates are defined as x1,y1,x2,y2, ... The mask coordinates must be in the data coordinate space.
- `raster_bands` attribute (optional): Specifies the band composition to be assigned to the red, green, and blue channels. If you specify only one value, the resulting image uses one band (gray levels for monochromatic images). If you specify two values, they are used for the red and green channels, and the default blue band stored in the GeoRaster metadata is used for the blue channel. If you do not specify this attribute, MapViewer uses the default values stored in the GeoRaster metadata.
- `raster_pyramid` attribute (optional): Specifies the pyramid level (level of resolution). If you do not specify this attribute, MapViewer calculates the best pyramid level for the current window query and device area.
- `raster_id` attribute (only if the definition does not include a SQL statement): Specifies the `rasterID` attribute value in the SDO_GEORASTER object definition of the single GeoRaster object for the map request.
- `raster_table` attribute (optional, and only if the definition does not include a SQL statement): Specifies the raster data table associated with the single GeoRaster object for the map request.
- `transparent_nodata` attribute (optional): Specifies if any GeoRaster NODATA value is to be rendered as transparent. The default value is "`false`".

Example 2–13 defines a GeoRaster theme that contains a SQL statement that selects a single GeoRaster object. The theme assigns band 1 to the red channel, band 2 to the green channel, and band 3 to the blue channel. Because the `raster_pyramid` attribute is not specified, MapViewer calculates the best pyramid level by using the spatial resolution values set during or after the georeferencing process. (In Example 2–13, `georid=1` in the WHERE clause refers to a column named GEORID in the GeoRaster table named PCI_IMAGE.)

Example 2–13 GeoRaster Theme Containing a SQL Statement

```
<theme name="georaster_theme">
  <jdbc_georaster_query
    georaster_table="pci_image"
```

```
    georaster_column="georaster"
    raster_bands="1,2,3"
    jdbc_srid="82301"
    datasource="mvdemo"
    asis="false"> SELECT georaster FROM pci_image WHERE georid =1
  </jdbc_georaster_query>
</theme>
```

Example 2–14 defines a GeoRaster theme that specifies the single GeoRaster object whose `rasterID` attribute value in the SDO_GEORASTER object is 1 (`raster_id="1"`) and associated with the raster data table named RDT_PCI. The theme specifies 2 as the pyramid level.

Example 2–14 GeoRaster Theme Specifying a Raster ID and Raster Data Table

```
<theme name="georaster_theme">
  <jdbc_georaster_query
    georaster_table="pci_image"
    georaster_column="georaster"
    raster_id="1"
    raster_table="rdt_pci"
    raster_pyramid="2"
    raster_bands="1,2,3"
    jdbc_srid="82301"
    datasource="mvdemo"
    asis="false">
  </jdbc_georaster_query>
</theme>
```

2.3.4.1 Creating Predefined GeoRaster Themes

To create a predefined GeoRaster theme, you must store the definition of the GeoRaster theme in the database by inserting a row into the USER_SDO_THEMES view (described in Section 2.9.2). Example 2–15 stores the definition of a GeoRaster theme.

Example 2–15 Creating a Predefined GeoRaster Theme

```
INSERT INTO user_sdo_themes  VALUES (
   'GEOR_BANDS_012',
   'Band 0 for red, 1 for green, 2 for blue',
   'GEOR_TABLE',
   'GEOR_COLUMN',
   '<?xml version="1.0" standalone="yes"?>
    <styling_rules theme_type="georaster" raster_table="RDT_PCI"
                   raster_id="1" raster_bands="0,1,2">
    </styling_rules>' );
```

Example 2–15 creates a GeoRaster theme named `GEOR_BANDS_012`, in which band 0 is assigned to the red channel, band 1 to the green channel, and band 2 to the blue channel. The GeoRaster table name (GEOR_TABLE in this example) is inserted in the BASE_TABLE column of the USER_SDO_THEMES view, the GeoRaster column name (GEOR_COLUMN in this example) is inserted in the GEOMETRY_COLUMN column, and an XML document with one <styling_rules> element is inserted in the STYLING_RULES column.

In the <styling_rules> element for a GeoRaster theme, `theme_type` must be `georaster` in order for this theme to be recognized as a GeoRaster theme.

The <styling_rules> element for a GeoRaster theme can contain the attributes described in Section 2.3.4, including raster_bands, raster_pyramid, raster_id, and raster_table, as shown in Example 2–15. Alternatively, the <styling_rules> element for a GeoRaster theme can be a rule definition. For example, to create a GeoRaster theme that selects a GeoRaster object from the GeoRaster table satisfying the WHERE clause condition georid=1, replace the <styling_rules> element in Example 2–15 with the following:

```
<styling_rules theme_type="georaster">
  <rule>
    <features> georid=1
    </features>
  </rule>
</styling_rules>
```

The <styling_rules> element for a GeoRaster theme can also specify one or more bitmap masks, as explained in Section 2.3.4.2.

The DTD for the <styling_rules> element is presented in Section A.7.

Example 2–16 prepares GeoRaster data for use with a GeoRaster theme that is stored in the database. Comments in the code example briefly describe the main steps. For detailed information about requirements and steps for using GeoRaster data, see *Oracle Spatial and Graph GeoRaster Developer's Guide*.

Example 2–16 Preparing GeoRaster Data for Use with a GeoRaster Theme

```
connect scott
Enter password: password

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 100
SET PAGESIZE 10000
SET SERVEROUTPUT ON SIZE 5000
SET LONG 20000
SET TIMING ON
call dbms_java.set_output(5000);

-------------------------------------------------------------------
-- Create a GeoRaster table (a table that has a
-- column of SDO_GEORASTER object type).
-------------------------------------------------------------------

create table georaster_table
    (georid     number primary key,
     type       varchar2(32),
     georaster  sdo_georaster);

-------------------------------------------------------------------
-- Create the GeoRaster DML trigger on the GeoRaster table, if
-- the Oracle Database release is before 11.1. (In Release 11.1 and later
-- this trigger is created automatically, so you do not need to create
-- it manually.)
-------------------------------------------------------------------

call sdo_geor_utl.createDMLTrigger('georaster_table', 'georaster');

-------------------------------------------------------------------
-- Create a raster data table (RDT).
```

```
--
-- It is used to store cell data of GeoRaster objects.
-- This step is not a requirement. If the RDT table does not
-- exist, the GeoRaster procedures or functions will generate it
-- automatically whenever needed.
-- However, for huge GeoRaster objects, some tuning and setup on those
-- tables can improve the scalability and performance significantly.
-- In those cases, it is better for users to create the RDTs.
-- The primary key must be added to the RDT if you create it.
-------------------------------------------------------------------

create table rdt_geor of sdo_raster
  (primary key (rasterId, pyramidLevel, bandBlockNumber,
                rowBlockNumber, columnBlockNumber))
  lob(rasterblock) store as (nocache nologging);

commit;

----------------
-- Import the image.
----------------

connect system;
Enter password: password

call dbms_java.grant_permission('MDSYS','SYS:java.io.FilePermission',
  'lbs/demo/images/l7_ms.tif', 'read' );

call dbms_java.grant_permission('SCOTT','SYS:java.io.FilePermission',
  'lbs/demo/images/l7_ms.tif', 'read' );

connect scott;
Enter password: password

 declare
  geor SDO_GEORASTER;
begin
  delete from georaster_table where georid = 1;
 insert into georaster_table
    values( 1, 'TIFF', sdo_geor.init('rdt_geor', 1) );
 select georaster into geor
    from georaster_table where georid = 1 for update;
  sdo_geor.importFrom(geor, '', 'TIFF', 'file',
    'lbs/demo/images/l7_ms.tif');
  update georaster_table set georaster = geor where georid = 1;
  commit;
end;
/

connect system;
Enter password: password

call dbms_java.revoke_permission('MDSYS','SYS:java.io.FilePermission',
       'lbs/demo/images/l7_ms.tif', 'read' );

call dbms_java.revoke_permission('SCOTT','SYS:java.io.FilePermission',
       'lbs/demo/images/l7_ms.tif', 'read' );

connect scott;
Enter password: password
```

```
--------------------------
-- Change the GeoRaster format, if needed.
-- To do this, you can call SDO_GEOR.changeFormatCopy.
-- The following operations for pyramiding, spatial resolution setup, and
-- spatial extent generation can also be combined into one PLSQL block.
--------------------------

declare
  gr1 sdo_georaster;
begin
  --
  -- Using changeFormat with a GeoRaster object:
  --

  -- 1. Select the source GeoRaster object.
  select georaster into gr1
    from georaster_table where georid = 1;

  -- 2. Make changes. (Interleaving is application-dependent. For TIFF images,
  --    the default interleaving is BSQ.)
  sdo_geor.changeFormat(gr1, 'blocksize=(512,512,3) interleaving=BIP');

  -- 3. Update the GeoRaster object in the GeoRaster table.
  update georaster_table set georaster = gr1 where georid = 1;

  commit;
end;
/

---------------------------
-- Generate pyramid levels (strongly recommended, but optional).
---------------------------

declare
  gr sdo_georaster;
begin

  -- 1. Select the source GeoRaster object.
  select georaster into gr
    from georaster_table where georid = 1 for update;

  -- 2. Generate pyramids.
  sdo_geor.generatePyramid(gr, 'resampling=NN');

  -- 3. Update the original GeoRaster object.
  update georaster_table set georaster = gr where georid = 1;

  commit;
end;
/

-----------------------------
-- Georeference the GeoRaster object.
-----------------------------

DECLARE
  gr sdo_georaster;
BEGIN
  SELECT georaster INTO gr FROM georaster_table WHERE georid = 1 FOR UPDATE;
```

```
  sdo_geor.georeference(gr, 82216, 1,
                        sdo_number_array(30, 0, 410000.000000),
                        sdo_number_array(0, -30,3759000.000000));
  UPDATE georaster_table SET georaster = gr WHERE georid = 1;
  COMMIT;
END;
/

-----------------------------
-- Set the spatial resolutions (required for 10gR1 only)
-----------------------------
-- If you are using Oracle Database Release 10.1, set spatial resolutions. (Not
-- required if you are using Release 10.2.) The spatial resolution values of
-- (30, 30) are from the ESRI world file or from the georeferencing information;
-- however, you may have to compute these values if they are not part of
-- the original georeferencing metadata.
DECLARE
  gr sdo_georaster;
BEGIN
  SELECT georaster INTO gr FROM georaster_table WHERE georid = 1 FOR UPDATE;
  sdo_geor.setSpatialResolutions(gr, sdo_number_array(30, 30));
  UPDATE georaster_table SET georaster = gr WHERE georid = 1;
  COMMIT;
END;
/

------------------------
-- Update the spatial extent.
------------------------

DECLARE
  sptext sdo_geometry;
BEGIN
  SELECT sdo_geor.generateSpatialExtent(a.georaster) INTO sptext
          FROM georaster_table a WHERE a.georid=1 FOR UPDATE;
  UPDATE georaster_table a SET a.georaster.spatialextent = sptext WHERE
a.georid=1;
  COMMIT;
END;
/

commit;

------------------------------------------------------------------
-- Create metadata information for the GeoRaster spatial extent column.
------------------------------------------------------------------

INSERT INTO USER_SDO_GEOM_METADATA
  VALUES (
  'GEORASTER_TABLE',
  'georaster.spatialextent',
  SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT('X', 410000.0, 470000.0, 0.000005),
    SDO_DIM_ELEMENT('Y', 3699000.0,3759000., 0.000005)
     ),
  82216   -- SRID
);

------------------------
-- Create a spatial index on the spatial extent.
```

```
------------------------

CREATE INDEX georaster_idx ON georaster_table(georaster.spatialextent)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

-------------------------------------------------------
-- Create a predefined GeoRaster theme for MapViewer.
-------------------------------------------------------

INSERT INTO user_sdo_themes  VALUES (
   'GEORASTER_TABLE',
   'GeoTiff image',
   'GEORASTER_TABLE',
   'GEORASTER',
   '<?xml version="1.0" standalone="yes"?>
    <styling_rules theme_type="georaster" raster_table="RDT_GEOR"
                   raster_id="1" raster_bands="0,1,2">
    </styling_rules>' );

commit;
```

2.3.4.2 Using Bitmap Masks with GeoRaster Themes

In Oracle Spatial and Graph GeoRaster, bitmap masks can be assigned to GeoRaster layers stored in the database. A **bitmap mask** is a special one-bit deep rectangular raster grid with each pixel having either the value of 0 or 1. It is used to define an irregularly shaped region inside another image. The 1-bits define the interior of the region, and the 0-bits define the exterior of the region. For more information about bitmap masks, see *Oracle Spatial and Graph GeoRaster Developer's Guide*.

To specify a bitmap mask with a GeoRaster theme, use the <bitmap_masks> element in the <styling_rules> element for the predefined theme, as shown in Example 2–17.

Example 2–17 Bitmap Mask in Predefined GeoRaster Theme

```
<styling_rules theme_type="georaster" raster_id="1"
  raster_table="RDT_MASS_COLOR_MOSAIC">
    <bitmap_masks>
      <mask layers="1,2" zeromapping="0" onemapping="255"/>
    </bitmap_masks>
</styling_rules>
```

The <bitmap_masks> element contains one or more <mask> elements, each with a mask definition for a specific GeoRaster object. In Example 2–17, a mask is defined for layers 1 and 2 of the GeoRaster object with the raster ID of 1 in the RDT_MASS_COLOR_MOSAIC table. The <mask> element has the following attributes:

- `raster_id` specifies the raster ID value of the GeoRaster object.
- `raster_table` specifies the raster data table (RDT).
- `layers` specifies the layer numbers in the GeoRaster object to be used for the mask.
- `zeromapping` specifies the transparency value to be applied during rendering on bitmap pixels with a value of 0 (zero). The attribute value can be from 0 (completely transparent) to 255 (completely opaque).

- `onemapping` specifies the transparency value to be applied during rendering on bitmap pixels with a value of 1. The attribute value can be from 0 (completely transparent) to 255 (completely opaque).

2.3.4.3 Reprojection of GeoRaster Themes

Effective with Oracle Spatial and Graph GeoRaster for Release 11.2.0.1, GeoRaster objects can be reprojected into a different SRID. It is recommended that you apply Oracle Database patch 10259201, to avoid black boundaries for adjacent reprojected GeoRaster objects when the objects are rendered in MapViewer. For more information, see My Oracle Support document ID 1272931.1, *Black Lines After Reprojection Of Georaster Data Via Wms In Oracle Mapviewer*.

In MapViewer, a GeoRaster theme will be reprojected if its SRID is different from the map request SRID. The reprojection is just for rendering, with no changes made to the original GeoRaster object. For older databases without reprojection support, the GeoRaster object will not be reprojected.

The reprojection modes available are BILINEAR (used as default), NN, CUBIC, AVERAGE4, AVERAGE16. For more information about reprojection, see *Oracle Spatial and Graph GeoRaster Developer's Guide*.

To specify a reprojection mode with a GeoRaster theme, use the `reproj_mode` keyword in the `<styling_rules>` element for the predefined theme, as shown in Example 2–18.

Example 2–18 Reprojection Mode in Predefined GeoRaster Theme

```
<styling_rules theme_type="georaster" reproj_mode="CUBIC">
</styling_rules>
```

2.3.5 Network Themes

A **network theme** is a special kind of MapViewer theme useful for visualizing networks defined using the Oracle Spatial and Graph network data model. A network consists of a set of nodes and links. A network can be directed or undirected, although links and paths typically have direction. A network can be organized into different levels of abstraction, called a network hierarchy. MapViewer assumes that network spatial tables in a network use the same coordinate system, and that these tables are indexed and registered as described in *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*.

Network node, link, and path tables store geometries of type SDO_GEOMETRY. You can create JDBC themes that use these geometries. In addition, you can define dynamic themes that consider aspects of the network, such as the direction of links for a directed network.

The following elements and attributes apply to the definition of a network theme:

- `<jdbc_network_query>` element: Specifies that this is a dynamically defined network theme. The complete DTD for this element is included in the map request DTD in Section 3.2.
- `network_name` attribute: Specifies the name of the network.
- `network_level` attribute (optional): Specifies the network hierarchy level to which this theme applies. (For a nonhierarchical network, specify 1, which is the default value.)
- `link_style` attribute (optional): Specifies the style name to be used for links.

- direction_style attribute (optional): Specifies the style name to be used for a link direction marker (for example, a directional arrow image).
- bidirection_style attribute (optional): Specifies the style name to be used for a bidirected link.
- direction_position attribute (optional): Specifies the position of the direction marker relative to the link start, as a number between 0 and 1. For example, 0.85 indicates 85 percent of the way between the link start and end points.
- direction_markersize attribute (optional): Specifies the size (number of pixels) of the direction marker.
- direction_multimarker attribute (optional): Specifies if the direction marker should be repeated over the link: true repeats the marker at a specified start position and each subsequent interval of that distance; false (the default) does not repeat the marker.
- link_labelstyle attribute (optional): Specifies the style name to be used for link labels in the column specified in the link_labelcolumn attribute.
- link_labelcolumn attribute (optional): Specifies the name of the column containing link labels to be rendered using the style specified in the link_labelstyle attribute.
- node_style attribute (optional): Specifies the style name to be used for nodes.
- node_markersize attribute (optional): Specifies the size (number of pixels) of the node marker.
- node_labelstyle attribute (optional): Specifies the style name to be used for node labels in the column specified in the node_labelcolumn attribute.
- node_labelcolumn attribute (optional): Specifies the name of the column containing node labels to be rendered using the style specified in the node_labelstyle attribute.
- path_ids attribute (optional): Specifies one or more path ID values of stored paths to be rendered. For more than one path, use commas to delimit the path ID values. For example, path_ids="1,3,4" specifies that the paths with path ID values 1, 3, and 4 are to be rendered.
- path_styles attribute (optional): Specifies one or more style names associated with the paths specified in the path_ids attribute. For example, path_styles="C.RED,C.GREEN,C.BLUE" specifies styles to be used to render the first, second, and third paths (respectively) specified in the path_ids attribute.
- path_labelstyle attribute (optional): Specifies the style name to be used for path labels in the column specified in the path_labelcolumn attribute.
- path_labelcolumn attribute (optional): Specifies the name of the column containing path labels to be rendered using the style specified in the path_labelstyle attribute.

Additional network theme attributes related to network analysis are described in Section 2.3.5.2.

A network theme can combine attributes for links, nodes, and paths, or any combination. In such cases, MapViewer first renders the links, then the paths, and then the nodes.

Example 2–19 defines a network theme that specifies attributes for the display of links and nodes in the network named NYC_NET.

Example 2–19 Network Theme

```
<theme name="net_theme" user_clickable="false">
  <jdbc_network_query
    network_name="NYC_NET"
    network_level="1"
    jdbc_srid="8307"
    datasource="mvdemo"
    link_style="C.RED"
    direction_style="M.IMAGE105_BW"
    direction_position="0.85"
    direction_markersize="8"
    node_style="M.STAR"
    node_markersize="5"
    asis="false">
  </jdbc_network_query>
</theme>
```

2.3.5.1 Creating Predefined Network Themes

To create a predefined network theme, you must store the definition of the network theme in the database by inserting a row into the USER_SDO_THEMES view (described in Section 2.9.2). Example 2–20 stores the definition of a network theme.

Example 2–20 Creating a Predefined Network Theme

```
INSERT INTO user_sdo_themes  VALUES (
   'NYC_NET_1',
   'New York City network',
   'NYC_NET_LINK_TABLE',
   'GEOMETRY',
   '<?xml version="1.0" standalone="yes"?>
    <styling_rules
              theme_type="network"
              network_name="NYC_NET"
              network_level="1">
      <rule>
         <features>
           <link
              style="C.RED"
              direction_style="M.IMAGE105_BW"
              direction_position="0.85"
              direction_markersize="8">
           </link>
           <path
              ids="1,3"
              styles="C.BLUE,C.GREEN">
           </path>
           <node
              style="M.CIRCLE"
              markersize="5">
           </node>
         </features>
         <label>
           <link column="LINK_ID" style="T.STREET NAME"> 1 </link>
         </label>
      </rule>
    </styling_rules>' );
```

Example 2–20 creates a network theme named `NYC_NET_1` for level 1 of the network named `NYC_NET`. The network table name (NYC_NET_LINK_TABLE in this example)

is inserted in the BASE_TABLE column of the USER_SDO_THEMES view, the link geometry column name (GEOMETRY in this example) is inserted in the GEOMETRY_COLUMN column, and an XML document with one `<styling_rules>` element is inserted in the STYLING_RULES column.

In the `<styling_rules>` element for a network theme, `theme_type` must be `network` in order for this theme to be recognized as a network theme. Elements for links, paths, and nodes can be specified in the same `<features>` element, as is done in Example 2–20:

- The link feature rule specifies the style `C.RED` and direction marker attributes for all links.
- The path feature rule specifies the style `C.BLUE` for paths with the path ID value 1, and the style `C.GREEN` for paths with the path ID value 3.
- The node feature rule specifies the style `M.CIRCLE` and a marker size of 5.

Example 2–20 also contains a `<label>` element for links, specifying the link column `LINK_ID` and the label style `T.STREET NAME`.

The DTD for the `<styling_rules>` element is presented in Section A.7.

2.3.5.2 Using MapViewer for Network Analysis

The network model Java API provides several network analysis capabilities. You can define MapViewer network themes that support the shortest-path and within-cost analysis capabilities. Some attributes apply to both capabilities, and some attributes apply only to the relevant associated capability.

For all network analysis capabilities, the `<jdbc_network_query>` element and the network-related attributes described in Section 2.3.5 apply to the definition of the network theme.

For shortest-path analysis, the following attributes apply to the definition of the network theme:

- `analysis_algorithm` attribute: Specifies the shortest-path analysis algorithm to use. Must be either `DIJKSTRA` or `ASEARCH`.
- `shortestpath_style` attribute: Specifies the style name to be used for the shortest path.
- `shortestpath_startnode` attribute: Specifies the start node to be used for the analysis.
- `shortestpath_endnode` attribute: Specifies the end node to be used for the analysis.
- `shortestpath_startstyle` attribute (optional): Specifies the style name to be used for the start node.
- `shortestpath_endstyle` attribute (optional): Specifies the style name to be used for the end node.

Example 2–21 defines a network theme that can be used for shortest-path analysis.

Example 2–21 Network Theme for Shortest-Path Analysis

```
<theme name="shortest_path_theme" user_clickable="false">
  <jdbc_network_query
    network_name="BI_TEST"
    network_level="1"
    jdbc_srid="0"
```

```
    datasource="mvdemo"
    analysis_algorithm="DIJKSTRA"
    shortestpath_style="L.PH"
    shortestpath_startnode="20"
    shortestpath_endnode="101"
    shortestpath_startstyle="M.STAR"
    shortestpath_endstyle="M.CIRCLE"
    asis="false">
  </jdbc_network_query>
</theme>
```

For within-cost analysis, the following attributes apply to the definition of the network theme:

- `analysis_algorithm` attribute: Must be `WITHINCOST`.
- `withincost_startnode` attribute: Specifies the start node to be used for the analysis.
- `withincost_cost` attribute: Specifies the cost cutoff value for nodes to be included. All nodes that can be reached from the start node at a cost less than or equal to the specified value are included in the resulting display. Nodes that cannot be reached from the start node or that can be reached only at a cost greater than the specified value are not included.
- `withincost_startstyle` attribute (optional): Specifies the style name to be used for the start node.
- `withincost_style` attribute: Specifies the style name to be used for links in the displayed paths between the start node and each node that is within the specified cost cutoff value.

Example 2–22 defines a network theme that can be used for within-cost analysis.

Example 2–22 Network Theme for Within-Cost Analysis

```
<theme name="within_cost_theme" user_clickable="false">
  <jdbc_network_query
    network_name="BI_TEST"
    network_level="1"
    jdbc_srid="0"
    datasource="mvdemo"
    analysis_algorithm="WITHINCOST"
    withincost_startnode="20"
    withincost_style="L.PH"
    withincost_cost="1"
    withincost_startstyle="M.STAR"
    asis="false">
  </jdbc_network_query>
</theme>
```

2.3.6 Topology Themes

A **topology theme** is a special kind of MapViewer theme useful for visualizing topologies defined using the Oracle Spatial and Graph topology data model. The topology data model lets you work with data about nodes, edges, and faces in a topology. The spatial representations of nodes, edges, and faces are spatial geometries of type SDO_GEOMETRY. For nodes and edges, the geometries are explicitly stored; for faces, the initial lines (exterior and interior) are stored, allowing the face geometry to be generated.

In addition to the spatial representation of nodes, edges, and faces, a topology can have features. A feature (also called a topology geometry) is a spatial representation of a real-world object. Each feature is defined as an object of type SDO_TOPO_GEOMETRY, which identifies the topology geometry type, topology geometry ID, topology geometry layer ID, and topology ID. For detailed information, see *Oracle Spatial and Graph Topology Data Model and Network Data Model Graph Developer's Guide*.

MapViewer can render topology features. It can also render a theme in debug mode (explained later in this section) to show the nodes, edges, and faces of a topology. For each topology theme, MapViewer uses the topology metadata information stored in the USER_SDO_TOPO_METADATA view.

The following elements and attributes apply to the definition of a topology theme:

- <jdbc_topology_query> element: Specifies that this is a dynamically defined topology theme. The element can specify a SQL query statement (without a terminating semicolon). The complete DTD for this element is included in the map request DTD in Section 3.2.
- `topology_name` attribute: Specifies the name of the topology.
- `feature_table` attribute: Specifies the name of the feature table.
- `spatial_column` attribute: Specifies the name of the spatial feature column of type SDO_TOPO_GEOMETRY.
- `label_column` attribute: Specifies the column in the feature table that contains the text label to be used with each feature.
- `label_style` attribute: Specifies the name of the text style to be used to render the labels in the label column.
- `render_style` attribute: Specifies the name of the style to be used to render the topology.

Example 2–23 defines a topology theme that specifies attributes for the display of features and labels from the LAND_PARCELS table in the CITY_DATA topology. The SQL statement specifies the spatial feature column and the label column, and it includes all rows in the feature table.

Example 2–23 Topology Theme

```
<theme name="topo_theme" user_clickable="false">
  <jdbc_topology_query
    topology_name="CITY_DATA"
    feature_table="LAND_PARCELS"
    label_column="FEATURE_NAME"
    spatial_column="FEATURE"
    label_style="T.CITY NAME"
    render_style="C.COUNTIES"
    jdbc_srid="0"
    datasource="topology"
    asis="false">select feature, feature_name from land_parcels
  </jdbc_topology_query>
</theme>
```

MapViewer also supports a **debug mode** that renders the nodes, edges, and faces of a topology. To specify debug mode, include the `mode="debug"` attribute in the <theme> element. In addition to the <jdbc_topology_query> attributes mentioned earlier in this section, the following attributes can be used in debug mode:

- `edge_style` attribute: Specifies the name of the style to be used to render edges.

- `edge_label_style` attribute: Specifies the name of the text style to be used to render edge labels.
- `edge_marker_style` attribute: Specifies the name of the marker style to be used for edge markers.
- `edge_marker_size` attribute: Specifies the size (number of pixels) of for edge markers.
- `node_style` attribute: Specifies the name of the style to be used to render nodes.
- `node_label_style` attribute: Specifies the name of the text style to be used to render node labels.
- `face_style` attribute: Specifies the name of the style to be used to render faces.
- `face_label_style` attribute: Specifies the name of the text style to be used to render face labels.

Example 2–24 defines a debug-mode topology theme for rendering features, edges, nodes, and faces from all feature tables in the CITY_DATA topology.

Example 2–24 Topology Theme Using Debug Mode

```
<theme name="topo_theme" mode="debug" user_clickable="false">
  <jdbc_topology_query
    topology_name="CITY_DATA"
    edge_style="C.RED"
    edge_marker_style="M.IMAGE105_BW"
    edge_marker_size="8"
    edge_label_style="T.EDGE"
    node_style="M.CIRCLE"
    node_label_style="T.NODE"
    face_style="C.BLUE"
    face_label_style="T.FACE"
    jdbc_srid="0"
    datasource="topology"
    asis="false">
  </jdbc_topology_query>
</theme>
```

2.3.6.1 Creating Predefined Topology Themes

To create a predefined topology theme, you must store the definition of the topology theme in the database by inserting a row into the USER_SDO_THEMES view (described in Section 2.9.2). Example 2–25 stores the definition of a topology theme.

Example 2–25 Creating a Predefined Topology Theme

```
INSERT INTO user_sdo_themes  VALUES (
   'LANDPARCELS',
   'Topology theme for land parcels',
   'LAND_PARCELS',
   'FEATURE',
   '<?xml version="1.0" standalone="yes"?>
    <styling_rules theme_type="topology" topology_name="CITY_DATA">
      <rule>
        <features style="C.RED"></features>
        <label column="FEATURE_NAME" style="T.TEXT STYLE"> </label>
      </rule>
    </styling_rules>' );
```

Example 2–25 creates a topology theme named LANDPARCELS for the topology named CITY_DATA. The feature table name (LAND_PARCELS in this example) is inserted in the BASE_TABLE column of the USER_SDO_THEMES view, the feature column name (FEATURE in this example) is inserted in the GEOMETRY_COLUMN column, and an XML document with one <styling_rules> element is inserted in the STYLING_RULES column.

In the <styling_rules> element for a topology theme, theme_type must be topology in order for this theme to be recognized as a topology theme. The theme in Example 2–25 defines one styling rule that renders all land parcel features from the CITY_DATA topology using the C.RED style and using the T.TEXT STYLE label style for values in the FEATURE_NAME column of the feature table.

The DTD for the <styling_rules> element is presented in Section A.7.

2.3.7 WFS Themes

A **WFS theme** is a special kind of MapViewer theme that supports the rendering of data delivered using the Open GIS Consortium (OGC) Web Feature Service (WFS) protocol, specifically the WFS 1.0.0 implementation specification.

WFS theme are conceptually similar to geometry themes, and users are able to render and label features. The WFS operations GetCapabilities, DescribeFeatureType, and GetFeature are used when rendering a WFS theme. When a WFS service is accessed, MapViewer caches the information about capabilities and feature types.

- GetCapabilities retrieves the server general information, including the URL addresses to issue requests and the features available. In general, a WFS capability request has the form:

  ```
  http://localhost:1979/geoserver/wfs/GetCapabilities?SERVICE=WFS&VERSION=1.0.0&R
  EQUEST=GetCapabilities
  ```

  The result includes a <Capabilities> element with the URL addresses for the WFS requests. For example, the following includes the GetCapabilities URLs for HTTP GET and POST requests.

  ```
  <Capability>
   <Request>
    <GetCapabilities>
     <DCPType>
      <HTTP>
       <Get onlineResource="http://localhost:1979/geoserver/wfs/GetCapabilities?"
  />
      </HTTP>
     </DCPType>
     <DCPType>
      <HTTP>
       <Post
  onlineResource="http://localhost:1979/geoserver/wfs/GetCapabilities?" />
      </HTTP>
      </DCPType>
    </GetCapabilities>
  . . .
  </Capability>
  ```

- DescribeFeatureType retrieves the feature information, including attributes and types.

- GetFeature retrieves the feature geometries and attributes. The output format for GetFeature requests is GML2.

The following attributes apply to the definition of a WFS theme:

- datasource attribute: Specifies the MapViewer data source from which styles will be loaded.
- feature_attributes attribute: Specifies feature attributes (besides geometry and label columns) that can be used with advanced styles.
- feature_ids attribute: Specifies the WFS feature IDs to be retrieved. Feature IDs are represented with the fid name in the WFS responses. If feature IDs are specified, spatial filter and query conditions are not used in the WFS request.
- feature_name attribute: Specifies the WFS feature name.
- key_column attribute: Specifies the attribute to be used as a key column. Applies to predefined themes, and can be used in Oracle Maps applications. If key_column is not specified, fid is used as the key column.
- label_column attribute: Specifies the column in the feature table that contains the text label to be used with each WFS feature.
- label_style attribute: Specifies the name of the text style to be used to render the labels in the label column.
- query_condition attribute: Specifies a WHERE clause condition to be applied to the WFS theme. Cannot be a complex condition with a hierarchy of expressions defined using multiple parentheses. Each string in the query must be separated by a blank space. If the condition cannot be parsed, it is ignored on the WFS request. Any query conditions are ignored if you specify the feature_ids attribute. The following are examples of valid expressions:

  ```
  state_name = 'New Hampshire' or state_name = 'New York'
  (state_name = 'New Hampshire' or state_name = 'New York') and top_pop > 700000
  (state_name = 'New Hampshire' or state_name = 'New York') and (top_pop >
  700000)
  ```

- render_style attribute: Specifies the name of the style to be used to render the geometry.
- service_url attribute: Corresponds to the capabilities address for HTTP GET requests. The service_url parameter for MapViewer must be the online resource address for HTTP GET in the <GetCapabilities> element. In the preceding example, the value to be used is:
  http://localhost:1979/geoserver/wfs/GetCapabilities?

  Do not include the Capabilities parameters SERVICE, VERSION, and REQUEST; use just the URL from the capabilities information.
- spatial_column attribute: Specifies the name of the spatial feature column of type SDO_TOPO_GEOMETRY.
- srs attribute: Specifies the spatial reference system (coordinate system) name for the WFS feature, in EPSG or Oracle Spatial and Graph format. For example, EPSG:4325, SDO:8307, and 8307 (the Spatial and Graph SRID value) specify the same SRS. If an EPSG SRS value is specified, MapViewer tries to identify an equivalent Spatial and Graph (SDO) SRID; and if no matching SRID is found, the SRID for the theme is assumed to be zero (0). MapViewer looks for matching SRID values as follows:

1. Use any custom mapping specified in an SDO to EPSG SRID mapping file specified MapViewer configuration file, as explained inSection 1.5.2.11.
2. Use the Spatial and Graph function SDO_CS.MAP_EPSG_SRID_TO_ORACLE to get the equivalent SDO code (if this function is available in the version of Oracle Database used to store the data).
3. Use the EPSG code that is in the MDSYS.CS_SRS table, if a match can be found.

- `user` and `password` attributes can be defined to access a secured WFS server that uses basic authentication.

  If the WFS server is deployed in WebLogic Application Server, the parameter `-DUseSunHttpHandler=true` must be added to the startup script of WebLogic server.

Example 2–26 shows a request with a dynamic WFS theme. The WFS service is `geoserver`, and it is installed on the local system.

Example 2–26 WFS Request with a Dynamic WFS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request
             title="WFS MAP"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="true"
             mapfilename="wfs_map"
             format="PNG_URL">
  <center size="20.">
     <geoFeature  >
         <geometricProperty typeName="center">
             <Point>
                 <coordinates>-70., 44.</coordinates>
             </Point>
         </geometricProperty>
     </geoFeature>
  </center>

  <themes>
   <theme name="wfs" >
     <wfs_feature_request
         service_url="http://localhost:1979/geoserver/wfs/GetCapabilities?"
         srs="EPSG:4326"
         feature_name="states"
         spatial_column="the_geom"
         render_style="C.COUNTIES"
         label_column="STATE_NAME"
         label_style="T.STATE NAME"
         datasource="mvdemo" />
    </theme>
  </themes>

</map_request>
```

2.3.7.1 Creating Predefined WFS Themes

To create a predefined WFS theme, you must store the definition of the WFS theme in the database by inserting a row into the USER_SDO_THEMES view (described in Section 2.9.2). Example 2–27 stores the definition of a WFS theme.

Example 2–27 Creating a Predefined WFS Theme

```
INSERT INTO user_sdo_themes  VALUES (
   'WFS_THEME1',
   'WFS',
   'POI',
   'THE_GEOM',
'<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="wfs" service_
url="http://localhost:1979/geoserver/wfs/GetCapabilities?" srs="EPSG:4326">
    <hidden_info>
        <field column="NAME" name="name"/>
        <field column="MAINPAGE" name="mainpage"/>
  </hidden_info>
    <rule>
        <features style="M.STAR"> </features>
        <label column="NAME" style="T.STREET NAME"> 1 </label>
  </rule>
</styling_rules>' );
```

In Example 2–27, the WFS feature `POI` is used as the base table, and the attribute `THE_GEOM` is the spatial column. The styling rule information contains the `service_url` and `srs` information; and although not shown inExample 2–27, it can also specify a `key_column` value. The <features> and <label> elements of the styling rules are similar to the rules used in geometry themes. Hidden information (<hidden_info> element) can also be defined and used in Oracle Maps applications.

Example 2–28 shows a map request that uses the predefined theme created in Example 2–27.

Example 2–28 Map Request with Predefined WFS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request
             title="Predefined WFS MAP"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="true"
             format="PNG_STREAM">

  <themes>
   <theme name="wfs_theme1" />
  </themes>

</map_request>
```

See also the WFS map request examples in Section 3.1.14.

In some cases, proxy information may affect the access to WFS servers. If this occurs, specify the appropriate proxy parameters in the MapViewer configuration file.

2.3.8 WMTS Themes

A **WMTS theme** supports the rendering of data delivered using the Open GIS Consortium (OGC) Web Map Tile Service (WMTS) standard, specifically the WMTS 1.0.0 implementation standard.

A WMTS theme fetches tile images from a WMTS-enabled server over the Internet and renders the images. The tile images on a WMTS-enabled server are spatially referenced with predefined content, extent, and resolution.

The WMTS operations `GetCapabilities`, `GetTile`, and `GetFeatureInfo` are used when rendering a WMTS theme. When a WMTS service is accessed by MapViewer, it caches the capabilities information of that WMTS service. You may need to specify a proxy server in Map Builder when creating a WMTS theme and to edit the MapViewer configuration file (`MapViewerConfig.xml`) when a WMTS theme is being used.

- A `GetCapabilities` operation retrieves the server's general information, including the URL addresses to issue requests and the features available. In general, a WMTS capability request has the form:

  ```
  http://maps.opengeo.org/geowebcache/service/wmts?service=WMTS&version=1.0.0&req
  uest=GetCapabilities
  ```

  The result includes a `<Capabilities>` element with the URL addresses for the WMTS requests. For example, the following includes the `GetCapabilities` URLs for HTTP GET or POST requests:

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <Capabilities xmlns="http://www.opengis.net/wmts/1.0"
  xmlns:ows="http://www.opengis.net/ows/1.1"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:gml="http://www.opengis.net/gml"
  xsi:schemaLocation="http://www.opengis.net/wmts/1.0
  http://schemas.opengis.net/wmts/1.0/wmtsGetCapabilities_response.xsd"
  version="1.0.0">
  <ows:ServiceIdentification>
    <ows:Title>Web Map Tile Service - GeoWebCache</ows:Title>
    <ows:ServiceType>OGC WMTS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
  </ows:ServiceIdentification>
  <ows:ServiceProvider>

  <ows:ProviderName>http://maps.opengeo.org/geowebcache/service/wmts</ows:Provide
  rName>
    <ows:ProviderSite
  xlink:href="http://maps.opengeo.org/geowebcache/service/wmts" />
    <ows:ServiceContact>
      <ows:IndividualName>GeoWebCache User</ows:IndividualName>
    </ows:ServiceContact>
  </ows:ServiceProvider>
  <ows:OperationsMetadata>
    <ows:Operation name="GetCapabilities">
      <ows:DCP>
        <ows:HTTP>
          <ows:Get
  xlink:href="http://maps.opengeo.org/geowebcache/service/wmts?">
            <ows:Constraint name="GetEncoding">
              <ows:AllowedValues>
                <ows:Value>KVP</ows:Value>
  ```

```
          </ows:AllowedValues>
        </ows:Constraint>
      </ows:Get>
    </ows:HTTP>
  </ows:DCP>
</ows:Operation>
<ows:Operation name="GetTile">
  <ows:DCP>
    <ows:HTTP>
      <ows:Get
xlink:href="http://maps.opengeo.org/geowebcache/service/wmts?">
        <ows:Constraint name="GetEncoding">
          <ows:AllowedValues>
            <ows:Value>KVP</ows:Value>
          </ows:AllowedValues>
        </ows:Constraint>
      </ows:Get>
    </ows:HTTP>
  </ows:DCP>
</ows:Operation>
<ows:Operation name="GetFeatureInfo">
  <ows:DCP>
    <ows:HTTP>
      <ows:Get
xlink:href="http://maps.opengeo.org/geowebcache/service/wmts?">
        <ows:Constraint name="GetEncoding">
          <ows:AllowedValues>
            <ows:Value>KVP</ows:Value>
          </ows:AllowedValues>
        </ows:Constraint>
      </ows:Get>
    </ows:HTTP>
  </ows:DCP>
</ows:Operation>
</ows:OperationsMetadata>
<Contents>
  <Layer>
    <ows:Title>bluemarble</ows:Title>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>-180.0 -90.0</ows:LowerCorner>
      <ows:UpperCorner>180.0 90.0</ows:UpperCorner>
    </ows:WGS84BoundingBox>
    <ows:Identifier>bluemarble</ows:Identifier>
    <Style isDefault="true">
      <ows:Identifier>_null</ows:Identifier>
    </Style>
    <Format>image/png</Format>
    <Format>image/jpeg</Format>
    <TileMatrixSetLink>
      <TileMatrixSet>EPSG:4326</TileMatrixSet>
    </TileMatrixSetLink>
    <TileMatrixSetLink>
      <TileMatrixSet>EPSG:900913</TileMatrixSet>
    </TileMatrixSetLink>
  </Layer>
  ...
  <TileMatrixSet>
    <ows:Identifier>EPSG:4326</ows:Identifier>
    <ows:SupportedCRS>urn:ogc:def:crs:EPSG::4326</ows:SupportedCRS>
    <TileMatrix>
```

```
        <ows:Identifier>EPSG:4326:0</ows:Identifier>
        <ScaleDenominator>2.795411320143589E8</ScaleDenominator>
        <TopLeftCorner>90.0 -180.0</TopLeftCorner>
        <TileWidth>256</TileWidth>
        <TileHeight>256</TileHeight>
        <MatrixWidth>2</MatrixWidth>
        <MatrixHeight>1</MatrixHeight>
      </TileMatrix>
     ...
    </TileMatrixSet>
    ...
  </Contents>
  <ServiceMetadataURL
  xlink:href="http://maps.opengeo.org/geowebcache/service/wmts?REQUEST=getcapabil
  ities&amp;VERSION=1.0.0"/>
  </Capabilities>
```

- A GetTile operation retrieves a tile image from a WMTS server. A request to the server contains parameters of key-value pairs (KVP) that define the tile image.
- A GetFeatureInfo operation allows WMTS clients to request information at a specific position of a specific tile for a specific query layer.

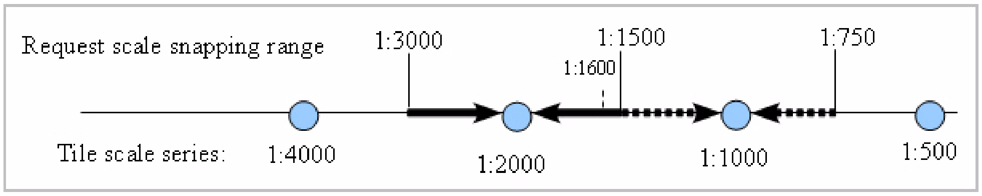
Detailed descriptions of these operations can be found at http://www.opengeospatial.org/standards/wmts.

The following attributes apply to the definition of a WMTS theme:

- current_threads attribute (optional): an integer (int) variable; the default is 8. It is the number of concurrent threads for retrieving tile images from a WMTS server. In general, a larger number of concurrent threads allows more threads to retrieve image tiles in parallel from a map server. Other constraining factors may prevent you from using a number in the hundreds, but you may try to set it 16, 32, or a slightly larger value for faster processing.
- format attribute: Specifies the tile image formats stored in the WMTS server. The tif, jpeg, png, and png8 formats are supported in MapViewer.
- layer attribute: Specifies the layer name for which the tile images are to be fetched.
- matrix_set_id attribute: Specifies the ID of the matrix set from which the tile images are to be fetched.
- service_url attribute: Corresponds to the capabilities address for HTTP GET requests. The service_url parameter for MapViewer must be the online resource address for HTTP GET in the <GetCapabilities> element. In the preceding example, the value to be used is: http://maps.opengeo.org/geowebcache/service/wmts?
- snap_to_tile_scale attribute (optional): a Boolean value (true or false); the default is false.

  When snap_to_tile_scale is set to true, a request scale (derived from a device-window size and a request-data-window size) is snapped to a closest tile scale; the map scale will be in the tile scale. For example, if there are tiles in scale series of …, 1:4000, 1:2000, 1.1000, 1:500, …., a request scale of 1:1600 will be snapped to the 1:2000 tile scale, and the map will be using the same 1:2000 scale, as shown in Figure 2–8.

Figure 2–8 snap_to_tile_scale Attribute



If a map request has more than one WMTS theme that specifies `snap_to_tile_scale` as `true`, then only the first WMTS `snap_to_tile_scale` specification is set to `true` and all others are reset to `false`. This is the logical behavior. For example, if two such themes both have the attribute set to `true`, but the two WMTS tile scale series are different from each other (the two map servers may be from different institutions using different scale series), the first theme will then use its closest tile scale to retrieve tile data and for the final request scale; at the same time, the second theme has to reset its `snap_to_tile_scale` to `false`, then has to find its own tile scale according to its `tile_resizing_option`, then retrieve tiles, and finally resize the tiles to match the first theme's tile scale.

- `style` attribute: Currently not used in MapViewer; the default is the string `default`.
- `tile_resizing_option` attribute (optional): the string `unbiased`, `expand_biased`, or `contract_biased`; the default is `unbiased`. If `snap_to_tile_scale` is set to `true`, `tile_resizing_option` is ignored.

  For more information, see Section 2.3.8.1, "How the tile_resizing_option Attribute Works".
- `timeout` attribute (optional): an integer specifying a request's timeout period in milliseconds; the default is 0, (that is, no limit for a request to wait for a response).

  Specifying this attribute ensures that a map request is terminated when the map server does not respond within the specified time period, and thus frees the resources. You might try a value of 30000 (30,000 milliseconds, or 30 seconds).
- `top_left_corner_x` attribute (optional): the x coordinate value of the top left corner of the whole tile images' extent that is served by a WMTS server. If not specified, the value retrieved from the WMTS server is used. If you specify this attribute, also specify the `top_left_corner_y` attribute.
- `top_left_corner_y` attribute (optional): the y coordinate value of the top left corner of the whole tile images' extent that is served by a WMTS server. If not specified, the value retrieved from the WMTS server is used. If you specify this attribute, also specify the `top_left_corner_x` attribute.
- `version` attribute: the version of the WMTS specification implemented by the WMTS server.

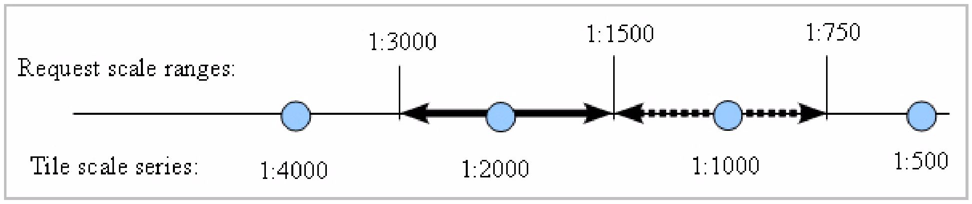
2.3.8.1 How the tile_resizing_option Attribute Works

If a WMTS theme's `snap_to_tile_scale` attribute is false (the default) or omitted, a request scale is always honored and the `tile_resizing_option` attribute value (specified or defaulted) is used when choosing a proper tile scale. However, if snap_to_tile_scale is true, the `tile_resizing_option` attribute value is ignored.

The `tile_resizing_option` attribute value can be one of the following string values: string `unbiased` (the default), `expand_biased`, or `contract_biased`.

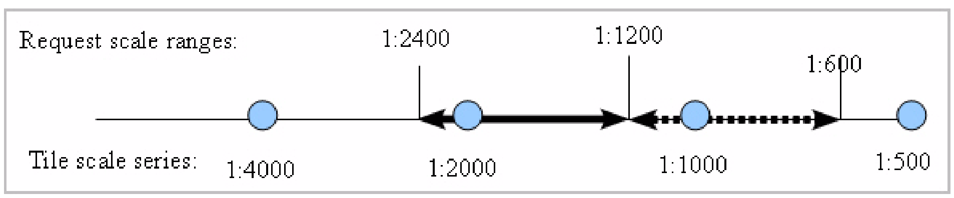
- `unbiased` (the default): The closest tile scale level is chosen, and then the tile images are expanded or contracted to generate a map in the request scale. For example, in Figure 2–9 the tile scale 1:2000 is used to generate any request scale map if a request scale falls within a scale range of 1:3000 and 1:1500.

Figure 2–9 unbiased tile_resizing_option Value



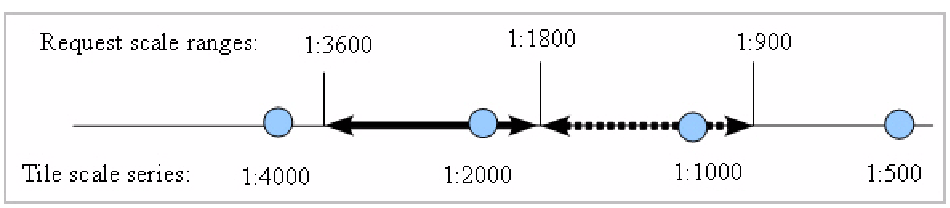
- `expand_biased`: When identifying a proper tile scale to generate a map in a request scale, a preference for expanding tile images to render a request map is used. In other words, you prefer to use a smaller tile scale to generate a map for a request scale. An 8:2 preference ratio for expanding is currently implemented. For example, in Figure 2–10 the tile scale 1:2000 is used for generating a request map if its scale is in the range of 1:2400 to 1:1200.

Figure 2–10 expand_biased tile_resizing_option Value



- `contract_biased`: When identifying a proper tile scale to generate a map in request scale, a preference for contracting tiles to render a request map is used. In other words, you prefer to use a larger tile scale to generate a map for a request scale. An 8:2 preference ratio for contracting is currently implemented. For example, in Figure 2–11 the tile scale 1:2000 is used for generating a request map if its scale is in the range of 1:3600 to 1:1800.

Figure 2–11 contract_biased tile_resizing_option Value



See also Section 2.3.8.2, "snap_to_tile_scale and tile_resizing_option Attribute Usage Guidelines".

2.3.8.2 snap_to_tile_scale and tile_resizing_option Attribute Usage Guidelines

This section presents general guidelines for using the `snap_to_tile_scale` and `tile_resizing_option` attributes to generate better quality maps. Because

mapping has a wide variety of applications, you may also use your domain knowledge to set attribute values that best meet your needs.

Whenever possible, set `snap_to_tile_scale` to `true` to use a closest tile scale instead of a request scale, because original tile maps have the best map quality.

However, if you must honor a request scale, consider the following when setting the attributes:

- If you set a `tile_resizing_option` value, omit the `snap_to_tile_scale` attribute or set it to `false` (the default). If `snap_to_tile_scale` is set to `true`, a request scale will not be honored, and instead the closest tile scale will be used.
- If a map is a topographic map with annotations and thin linear features, you may want to use the `expand_biased` option.
- If a map is a thematic map, such as a land cover map, you may want to use the `contract_biased` option.
- If a map is a satellite image, you may want to use the `unbiased` or `contract_biased` option.

Regarding the tile_resizing_option possible values:

- In general, a `contract_biased` option may generate maps with more details, but these maps may need more tiles than maps from an `expand_biased` option, and retrieving more tiles takes more time.
- If a request scale is honored (when the `snap_to_tile_scale` attribute is set to `false`), when a request scale is close enough to a tile scale, then the same operation will be employed (either expanding or contracting), regardless of the specified `tile_resizing_option`. For example, if a request scale is 1:1900 in the figures in Section 2.3.8.1, "How the tile_resizing_option Attribute Works", tile maps in 1:2000 will be retrieved and expanded to render the request map in a map scale of 1:1900, regardless of whether the `unbiased`, `expand_biased`, or `contract_biased` option is specified; similarly, if a request scale is 1:2100, then tile maps in 1:2000 will also be retrieved and then contracted to render the requested map in a scale of 1:2100.

Example 2–29 shows a request with a dynamic WMTS theme, in which `snap_to_tile_scale="true"` is specified.

Example 2–29   Request with a Dynamic WMTS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request title="OpenGeo wmts theme (bluemarble)"
  datasource="mvdemo"
  width="1024"
  height="900"
  mapfilename="Bluemarble"
  format="PNG_STREAM">
  <center size="80.0">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>-112, 42.0</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
<themes>
  <theme name="wmtstheme: Earth" timeout="10000" snap_to_tile_scale="true">
```

```
    <wmts_gettile_request>
      <service_url> http://maps.opengeo.org/geowebcache/service/wmts </service_
url>
      <version> 1.0.0 </version>
      <layer> bluemarble </layer>
      <matrix_set_id> EPSG:4326 </matrix_set_id>
      <format> image/png </format>
      <style> default </style>
      <top_left_corner_x> -180.0 </top_left_corner_x>
      <top_left_corner_y> 90.0 </top_left_corner_y>
    </wmts_gettile_request>
  </theme></themes>
</map_request>
```

2.3.8.3 Creating Predefined WMTS Themes

To create a predefined WMTS theme, you must store the definition of the WMTS theme in the database by inserting a row into the USER_SDO_THEMES view (described in Section 2.9.2). Example 2–30 stores the definition of a WMTS theme.

Example 2–30 Creating a Predefined WMTS Theme

```
INSERT INTO user_sdo_themes  VALUES (
'earth_image',
'Opengeo.org demo',
'table_spaceholder',
'geom_col_spaceholder',
'<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="wmts">
    <version> 1.0.0 </version>
    <service_url> http://maps.opengeo.org/geowebcache/service/wmts </service_url>
    <layer> bluemarble </layer>
    <matrix_set_id> EPSG:4326 </matrix_set_id>
    <format> image/png </format>
    <style> default </style>
</styling_rules>');
```

In Example 2–30, `earth_image` is the name of the WMTS theme, and `table_spaceholder` and `geom_col_spaceholder` are dummy values to fill the BASE_TABLE and GEOMETRY_COLUMN columns in the USER_SDO_THEMES view. The styling rule information contains the `service_url`, `layer`, `matrix_set_id`, `format`, and `style` information. The `top_left_corner_x` and `top_left_corner_y` attributes are not specified, so by default the values retrieved from the WMTS server are used.

Example 2–31 shows a map request that uses the predefined theme created in Example 2–30.

Example 2–31 Map Request with Predefined WMTS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="OpenGeo predefined wmts theme"
  datasource="mvdemo"
  width="1024"
  height="768"   mapfilename="Bluemarle: Earth"
  format="PNG_STREAM">
  <center size="10.0">
    <geoFeature>
      <geometricProperty typeName="center">
```

```
        <Point>
          <coordinates>-75.0,42.0</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
<themes>
  <theme name="earth_image" />
</themes>
</map_request>
```

2.3.9 Custom Geometry Themes

Custom geometry themes are associated with external spatial data (spatial data in a native format other than Oracle Spatial and Graph, such as shapefile). A custom geometry theme uses a spatial provider class to retrieve the native data, and the external provider must use the spatial data provider plug-in mechanism. MapViewer provides a spatial provider interface class that the external provider must implement. The interface implementation has the following methods (some of them mainly provide information that can be used in user interfaces of applications like Map Builder):

```
public interface SDataProvider
{
  /**
   * Returns the initialization parameters for the provider.
   * @return String[] - array with initialization parameter names
   */
  public String[] getInitParameterNames();

  /**
   * Returns runtime parameter names. Runtime parameters are additional parameters
   * that the provider may use when retrieving the data objects.
   * @return String[] - array with runtime parameter names
   */
  public String[] getRuntimeParameterNames();

  /**
   * Returns a value that gives a hint for the runtime parameter value.
   * This hin can be used as a tooltip in user intefaces.
   * @param runtimeParam
   * @return   a String representing the hint value, or null if no hint is
available
   */
  public String getRuntimeParameterHintValue(String runtimeParam);

  /**
   * This method is used to set the initialization parameters for the specific
   * data provider.
   * @param params - parameters to be used by the initialization method.
   * @return boolean - true if success; false otherwise
   */
  public boolean init(Properties params);

  /**
   * This method creates and returns an instance of SDataSet which contains
   * all the Spatial data produced by this provider, based on
   * the given parameters for a specific incoming map request.
   * <br>
   * MapViewer calls this method on the custom theme producer implementation.
```

```
   *
   * @param queryWin the search area to retrieve spatial objects. The window is
   * assumed to be already on data provider spatial reference system.
   * @param nonSpatialColumns - the list of attributes that will return with
objects.
   * @param queryCondition - query condition expression (may have binding
parameters).
   * @param bindingParameters - binding variables for query condition with binding
parameters.
   * @param params  - parameters that the provider may use to retrieve the data.
   * @return SDataObject - an instance of SDataSet class; null if failed.
   */
  public SDataSet buildDataSet(Rectangle2D queryWin,String []nonSpatialColumns,
                             String queryCondition,Object[] bindingParameters,
                              Properties params);

  /**
   * Returns the list of existing attributes for this data provider.
   * @param params  parameters that the provider may use to get the attribute
list.
   * @return Field[] - array of attributes for this provider.
   */
  public Field[] getAttributeList(Properties params);

  /**
   * Returns the data set spatial extent MBR.
   * @param params  parameters that the provider may use to get the data extents
   * @return Rectangle2D - data spatial extent for this provider.
   */
  public Rectangle2D getDataExtents(Properties params);

 /**
   * Returns if provider can build spatial indexes.
   * If true, means that buildSpatialIndex method can be called.
   * @return
   */
  public boolean canBuildSpatialIndex();

  /**
   * Builds a spatial index on the data set.
   * @param params  parameters that the provider may use to build the spatial
index.
   * @return boolean - true if saptial index creation is successful.
   */
  public boolean buildSpatialIndex(Properties params);

  /**
   * Clears provider internal caches (if provider implement caches).
   */
  public void clearCache();


  /**
   * Returns the parameter names that can be used to query for spatial tables.
   * Can be used more as information for user interfaces.
   * @return
   */
  public String[] getParametersToQuerySpatialMetadata();

  /**
```

```
   * Returns the spatial tables and spatial columns.
   * @param params must define the parameters returned from
   * getParametersToQuerySpatialMetadata.
   * @return  an array list defining the table name(index [0])
   *         and spatial column (index[1])
   */
  public String[][] getSpatialTables(Properties params);
}
```

The `init` and `buildDataSet` methods must be implemented. The other method implementations can be empty; however applications (such as the Oracle Map Builder Tool) can make use of these methods to handle the information about spatial data providers. A provider can implement its own spatial indexing mechanism; MapViewer offers an implementation for the shapefile provider, and the `buildSpatialIndex` method creates an indexing file with the `.oix` extension in the `shapefile` directory. Appendix D contains an example of how to implement and register a sample spatial provider with MapViewer.

To render native data in MapViewer with custom geometry themes, follow these steps:

1. Implement a spatial provider class based on the plug-in interface, and generate a jar file with the provider implementation. Copy the jar file to a directory that is part of the MapViewer CLASSPATH definition.
2. Register the provider in the MapViewer configuration. MapViewer and Map Builder ships with a shapefile provider to access ESRI shapefiles, a GDAL-OGR provider to access data formats supported by OGR, and a Teradata provider to access data stored in a Teradata database. (See the GDAL-OGR and Teradata documentation for detailed information about handling spatial data in these environments.) The GDAL-OGR library `gdal.jar`, and Teradata libraries `terajdbc4.jar` and `tdgssconfig.jar`, must be on server classpath. The registration section in MapViewer configuration file looks like this:

   ```
   <s_data_provider
     id="shapefileSDP"
     class="oracle.sdovis.ShapefileDataProvider"
     >
     <parameters>
       <parameter name="datadir" value="/temp/data" />
     </parameters>
   </s_data_provider>
   ```

   Each provider must have `id` and `class` names defined: `id` is a unique name that identifies the provider, and `class` corresponds to the Java class implementation. The `<parameters>` element defines the initialization parameters of the provider.

   For the shapefile provider, the initialization parameter `datadir` defines where MapViewer will look for the data files, and thus it should be a directory that is accessible to MapViewer. MapViewer first looks for data files based on the theme definition information; and if the data path defined in the theme definition is not accessible, MapViewer looks for the data path defined in the configuration file.
3. Create custom geometry themes associated with the external spatial data provider.

Although the external spatial data is outside the Oracle database, you still need to have a database connection to render this data. The database is used to store the metadata information related with the theme, as well as the styling information used to render and to label the data.

Example 2–32 shows the definition for a dynamic custom geometry theme. The XML element `<custom_geom_theme>` identifies a custom geometry theme. The `<parameters>` element defines the runtime parameters to be used by the provider. In this case `"filename"` is a runtime parameter, and `"/lbs/demo/shapefile/parcel.shp"` defines the file path. MapViewer first attempts to use this file path definition; but if it is not accessible, it uses the data directory value defined in the configuration file for the shapefile spatial provider.

The runtime parameters for the available spatial providers are as follows (note that Map Builder provides the option to encrypt parameter values):

- For a shapefile provider:
  - `filename`: full path to the shapefile (.shp) on disk
- For GDAL-OGR:
  - `datasource`: a full OGR data source string. Depending on the data source format, this string can vary.(see the GDAL-OGR documentation for detailed information about connecting to different formats supported by GDAL-OGR.)

    For file formats, this parameter's value is usually the full path to the archive.

    For database connections, enter the full connection string to access the spatial table and spatial column. For example, for a Postgis table name STATES with one spatial column, the value might be:

    ```
    datasource = PG:dbname='template_postgis' host='localhost' port='5432'
    user='postgres' password='manager' tables=states
    ```
- For a Teradata provider:
  - `jdbcurl`: the JDBC connection to Teradata database. The string format is:

    ```
    jdbc:teradata://<host_address>/DATABASE=<db_name>,DBS_PORT=<db_
    port>,TMODE=ANSI,CHARSET=UTF8
    ```
  - `user`: database user.
  - `password`: database password.
  - `containerds`: Optional. Name of Teradata container data source defined on the application server (WebLogic, for example). If defined, this will be used first in MapViewer, instead of the `jdbcurl` value.
  - `basetable`: name of the spatial table.
  - `geomcolumn`: name of the spatial column.
  - `fetchsize`: Optional. Number of rows to be prefetched.

Example 2–32 Defining a Dynamic Custom Geometry Theme

```
<theme name="custom_geom_theme_1" >
   <custom_geom_theme
     provider_id="shapefileSDP"
     srid="26986"
     render_style="C.RED"
     label_column="parcel_id"
     label_style="T.CITY NAME"
     datasource="mvdemo">
   <parameters>
    <parameter name="filename" value="/lbs/demo/shapefile/parcel.shp"/>
   </parameters>
  </custom_geom_theme>
```

```
</theme>
```

The available attributes for a dynamic custom geometry theme are:

- `provider_id` specifies the spatial provider.
- `datasource` specifies the Oracle database connection. This connection is used to retrieve the styles to render the spatial data.
- `srid` specifies the spatial reference system (Oracle Spatial and Graph coordinate system).
- `render_style` specifies the style to be used when rendering the features.
- `label_column` specifies the name of the column containing label text to be used with the theme.
- `label_style` specifies the style to be used when labeling the features.
- `feature_attributes` specifies additional attributes that can be used with advanced styles.
- `key_column` specifies a key attribute that can be used in Oracle Maps applications.
- `query_condition` specifies the WHERE clause to filter feature selection. Shapefile providers do not support a query condition; however an OGR provider can be used to render shapefiles with a query condition. The query condition expression can define binding parameters (for example, `attr = :1`).

Example 2–33 shows how to store a predefined custom geometry theme definition. Use GEOMETRY as the geometry column name, and you can specify any name for the base table name. The `"theme_type=geom_custom"` attribute identifies the theme as a custom theme. The `<rule>` element has the same function as for an Oracle Spatial and Graph geometry theme. The `<parameters>` element defines the runtime parameters that the provider accepts. For the shapefile provider, the runtime `parameter` filename defines the path to the shapefile data.

Example 2–33 Storing a Predefined Custom Geometry Theme

```
insert into user_sdo_themes values (
'SHAPE_THEME',
'Shapefile theme',
'CUSTOM_TABLE',
'GEOMETRY',
'<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="geom_custom" srid="26986" provider_id="shapefileSDP">
  <rule>
    <features style="C.RED"> </features>
    <label column="PARCEL_ID" style="T.CITY NAME"> 1 </label>
  </rule>
  <parameters>
    <parameter name="filename" value="/lbs/demo/shapefile/parcel.shp"/>
  </parameters>
</styling_rules>'
);
```

You can override the runtime parameters section of a predefined custom geometry theme by the specifying the parameters in a map_request. For example, you can include the following in a `<map_request>` element:

```
<theme name="CUSTOM_THEME" >
  <parameters>
```

```
    <parameter name="filename" value="/lbs/demo/shapefile/counties.shp"/>
  </parameters>
</theme>
```

2.3.10 Annotation Text Themes

Oracle Spatial and Graph supports annotation text as specified in the *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture*, which defines **annotation text** as "simply placed text that can carry either geographically-related or ad-hoc data and process-related information as displayable text. This text may be used for display in editors or in simpler maps. It is usually lacking in full cartographic quality, but may act as an approximation to such text as needed by any application."

Oracle Spatial and Graph provides the ST_ANNOTATION_TEXT object type for storing annotation text, and the USER_ANNOTATION_TEXT_METADATA and ALL_ANNOTATION_TEXT_METADATA views for storing metadata related to annotation text. For more information about annotation text support, see *Oracle Spatial and Graph Developer's Guide*.

Each annotation text object may have one or more elements, and each element is defined by the following:

- Value: Text associated with element. If the value is null, the text is derived from the first non-null preceding element value. If all preceding elements have null values, the text is a text expression value derived from the metadata.
- Location: Spatial location associated with the annotation text object.
- Leader line: Linear feature associated with the annotation text object.
- Attributes: Graphic attributes used to display the text. If the value is null, graphic attributes are derived from the attributes value in the metadata.

The text expression in the metadata views can be any of the following:

- A column name.
- A function applied to a column name. For example: `substr(my_col,1,3)`
- The concatenation of two or more column names. For example: `column_1 || column_2 || column_3`
- A text value that is unrelated to a column name. In this case, it is treated as a simple text string that is used for any text element that has a null value.

Annotation text themes in MapViewer are associated with database tables that have a column of type ST_ANNOTATION_TEXT. For each annotation text element, MapViewer will render:

- The value (if not null) of the annotation text element as a string, using a text style that is created at real time based on the element attributes.
- The leader line (if not null) associated with the annotation text element. In this case, users can select a MapViewer style to render the leader line.

Each annotation text element has an envelope represented by a geometry, and which is used for spatial indexing. Therefore, you must do the following to use spatial indexing with annotation text tables in MapViewer:

1. Insert a row into the USER_ANNOTATION_TEXT_METADATA view that specifies the name of the annotation text table and the PRIVATEENVELOPE

attribute of the annotation text column (that is, the column of type ST_ANNOTATION_TEXT).

The following example inserts a row for a table named ANNOTEXT_TABLE with an annotation text column named TEXTOBJ:

```
INSERT INTO USER_SDO_GEOM_METADATA
  VALUES (
  'ANNOTEXT_TABLE',
  'TEXTOBJ.PRIVATEENVELOPE',
  SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT('X', 0.0, 10.0, 0.0005),
    SDO_DIM_ELEMENT('Y', 0.0,10.0, 0.0005)
     ),
  null   -- SRID
);
```

2. Create a spatial index on the annotation text envelope of the annotation text table.

   The following example creates a spatial index named ANNO_TEXT_IDX on the annotation envelope of the table named ANNOTEXT_TABLE:

```
CREATE INDEX anno_text_idx ON annotext_table(textobj.privateenvelope)
  INDEXTYPE IS mdsys.spatial_index;
```

For themes with valid SRID information, if the metadata base map scale is defined, the element text sizes will be scaled as maps zoom in or out.

Example 2–34 defines the styling rules for a predefined annotation text theme in MapViewer. The structure is similar to other MapViewer themes. Currently, just one styling rule is processed for each annotation theme. In this example, the theme type is `annotation`, the feature style L.PH is used to render leader lines, and the query condition `(id = 1 or id = 2)` is appended on the final query.

Example 2–34 Styling Rules for a Predefined Annotation Text Theme

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="annotation">
    <rule>
        <features style="L.PH"> (id = 1 or id = 2) </features>
  </rule>
</styling_rules>
```

Example 2–35 shows the theme definition for a dynamic annotation text theme. The parameters defined are:

- `datasource`: the data source name
- `jdbc_srid`: the spatial reference identifier
- `annotation_table`: the annotation text table
- `annotation_column`: the annotation text column
- `leaderline_style`: the leader line style to be used

Example 2–35 Dynamic Annotation Text Theme Definition

```
<themes>
  <theme name="theme1" >
     <jdbc_annotation_query
       datasource="tilsmenv"
       jdbc_srid="0"
```

```
        annotation_table="ANNOTEXT_TABLE"
        annotation_column="textobj"
        leaderline_style="L.PH"
        >select textobj from annotext_table
      </jdbc_annotation_query>
   </theme>
</themes>
```

Example 2–36 is similar to Example 2–35, but it adds the behavior that if the `annotation_column` column contains a null value, then the value in the `textexpr_column` is used for the annotation instead. In Example 2–36, assume that the ANNOTATION_TABLE table contains a column named DEFAULT_ANNOTATION (which is used in Example 2–37). This additional column is specified in the `textexpr_column` attribute and in the SELECT statement.

Example 2–36 Dynamic Annotation Text Theme with Default Annotation Column

```
<themes>
  <theme name="theme1" >
     <jdbc_annotation_query
        datasource="tilsmenv"
        jdbc_srid="0"
        annotation_table="ANNOTEXT_TABLE"
        annotation_column="textobj"
        textexpr_column="default_annotation"
        leaderline_style="L.PH"
        >select textobj, default_annotation from annotext_table
      </jdbc_annotation_query>
   </theme>
</themes>
```

Example 2–37 creates an annotation text table and prepares it to be used with MapViewer.

Example 2–37 Script to Generate Annotation Text Data

```
SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 100
SET PAGESIZE 10000
SET SERVEROUTPUT ON SIZE 5000
SET LONG 20000
SET TIMING ON
call dbms_java.set_output(5000);

---------------------------------------------------------------------
-- Create an annotation text table (a table that has a
-- column of ST_ANNOTATION_TEXT object type), and insert some records.
---------------------------------------------------------------------

create table annotext_table (
  id number,
  default_annotation varchar2(32),
  textobj ST_ANNOTATION_TEXT);

insert into annotext_table values (1,'Text_1',
ST_ANNOTATION_TEXT(
   ST_ANNOTATIONTEXTELEMENT_ARRAY(
          ST_ANNOT_TEXTELEMENT_ARRAY(
```

```
                ST_ANNOTATIONTEXTELEMENT('Sample Label 1',
                   SDO_GEOMETRY(2001, null, sdo_point_type(1,1,null),null,null),
                   SDO_GEOMETRY(2002,null,null,
                       SDO_ELEM_INFO_ARRAY(1,2,1),
                       SDO_ORDINATE_ARRAY(0,0, 1,1)), NULL)))));

insert into annotext_table values (2,'Text_2',
ST_ANNOTATION_TEXT(
   ST_ANNOTATIONTEXTELEMENT_ARRAY(
          ST_ANNOT_TEXTELEMENT_ARRAY(
                ST_ANNOTATIONTEXTELEMENT('Sample Label 2',
                   SDO_GEOMETRY(2001,null,sdo_point_type(10,10,null),null,null),
                   SDO_GEOMETRY(2002,null,null,
                       SDO_ELEM_INFO_ARRAY(1,2,1),
                       SDO_ORDINATE_ARRAY(5,10, 10,10)), NULL)))));

insert into annotext_table values (3, 'Text_3',
ST_ANNOTATION_TEXT(
   ST_ANNOTATIONTEXTELEMENT_ARRAY(
          ST_ANNOT_TEXTELEMENT_ARRAY(
                ST_ANNOTATIONTEXTELEMENT(null,
                   SDO_GEOMETRY(2002, null, null,
                       SDO_ELEM_INFO_ARRAY(1,2,1),
                       SDO_ORDINATE_ARRAY(2,5,4,5,6,5)),
                   SDO_GEOMETRY(2002,null,null,
                       SDO_ELEM_INFO_ARRAY(1,2,1),
                       SDO_ORDINATE_ARRAY(4,3, 4,5)),
'<?xml version="1.0" encoding="UTF-8" ?>
<textAttributes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="../../annotation_text.xsd">
 <textStyle fontFamily="Dialog" fontSize="14" fill="blue"/>
 <textlayout/>
</textAttributes>'
)))));

---------------------------------------------------------------------
-- Register the annotation text table in the user metadata view.
---------------------------------------------------------------------

insert into USER_ANNOTATION_TEXT_METADATA values(
  'ANNOTEXT_TABLE', 'TEXTOBJ', null, null, null);

---------------------------------------------------------------------
-- Update the metadata information.
---------------------------------------------------------------------

update user_annotation_text_metadata set
text_expression='default_annotation',
text_attributes =
'<?xml version="1.0" encoding="UTF-8" ?>
<textAttributes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="../../annotation_text.xsd">
 <textStyle fontFamily="Serif" fontSize="14" fill="#ff0000"/>
 <textlayout/>
</textAttributes>';

---------------------------------------------------------------------
-- Register the annotation text geometry envelope on the user
-- metadata view of geometries.
---------------------------------------------------------------------
```

```
INSERT INTO USER_SDO_GEOM_METADATA
 VALUES (
 'ANNOTEXT_TABLE',
 'TEXTOBJ.PRIVATEENVELOPE',
 SDO_DIM_ARRAY(
   SDO_DIM_ELEMENT('X', 0.0, 10.0, 0.0005),
   SDO_DIM_ELEMENT('Y', 0.0,10.0, 0.0005)
    ),
 null   -- SRID
);

---------------------------------------------------------------------
-- Create a spatial index on the annotation text envelope.
---------------------------------------------------------------------

create index anno_text_idx on annotext_table(textobj.privateenvelope)
  indextype is mdsys.spatial_index;

---------------------------------------------------------------------
-- Insert a predefined theme into MapViewer's theme view.
---------------------------------------------------------------------

INSERT INTO user_sdo_themes  VALUES (
  'ANNOTEXT_THEME',
  'Annotation text',
  'ANNOTEXT_TABLE',
  'TEXTOBJ',
  '<?xml version="1.0" standalone="yes"?>
   <styling_rules theme_type="annotation">
     <rule >
      <features style="L.PH"> </features>
     </rule>
   </styling_rules>' );

commit;
```

2.3.11 Thematic Mapping

Thematic mapping refers to the drawing of spatial features based on their attribute values. MapViewer uses thematic mapping to create maps in which colors or symbols are applied to features to indicate their attributes. For example, a `Counties` theme can be drawn using colors with different hues that map directly to the population density of each county, or an `Earthquakes` theme can be plotted with filled circles whose sizes map to the scale or damage of each earthquake.

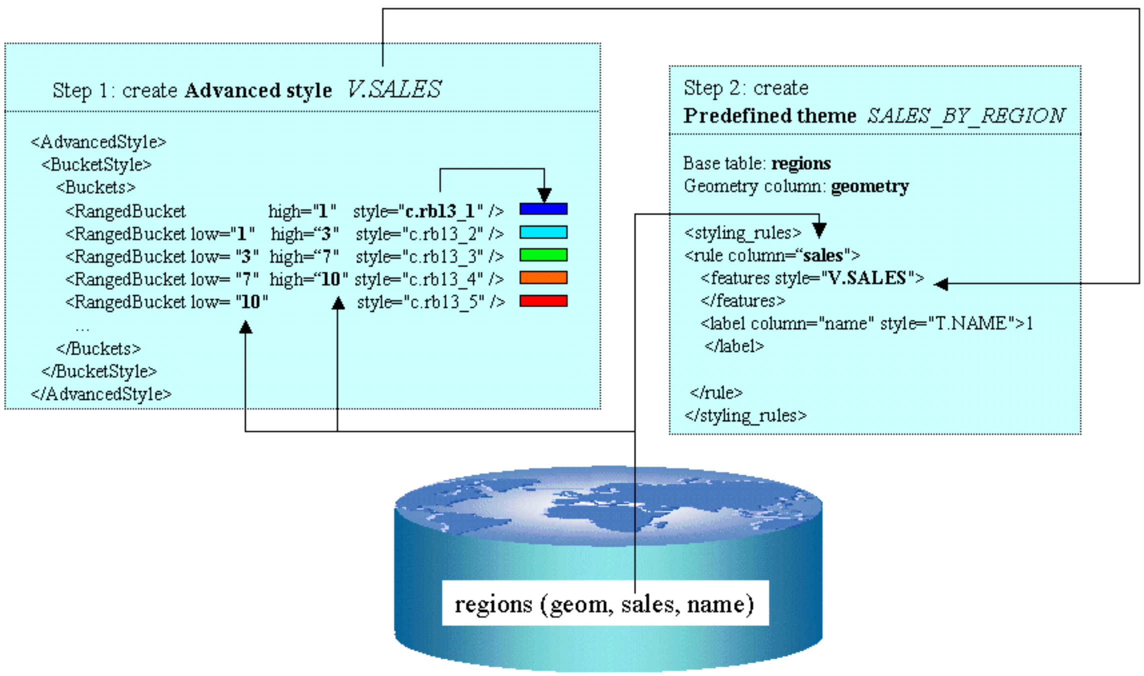
To achieve thematic mapping, you must first create an advanced style that is suitable for the type of thematic map, and then create a theme for the features specifying the advanced style as the rendering style. In the styling rules for the theme, you must also specify attribute columns in the table or view whose values will be used to determine exactly how a feature will be rendered thematically by the advanced style.

For example, assume that you wanted to display a map in which the color used for each region reflects the level of sales for a particular product. To do this, create an advanced style that defines a series of individual range-based buckets (see Section A.6.1.2), where each bucket contains a predefined range of sales values for a product, and each bucket has an associated rendering style. (Each region will be rendered using the style associated with the range in which that region's sales value falls.) Also specify the name of the column or columns that provide the attribute

values to be checked against the ranges. In other words, the advanced style defines how to map regions based on their sales values, and the theme's styling rules tie together the advanced style and the attribute column containing the actual sales values.

Figure 2–12 shows the relationship between an advanced style and a theme, and how the style and the theme relate to the base table. In this figure, the advanced style named `V.SALES` defines the series of buckets. The predefined theme named `SALES_BY_REGION` specified the `V.SALES` style in its styling rules. The theme also identifies the SALES column in the REGIONS table as the column whose value is to be compared with the bucket ranges in the style. (Each bucket could be associated with a labeling style in addition to or instead of a rendering style, as explained in Section 2.2.2.)

Figure 2–12 Thematic Mapping: Advanced Style and Theme Relationship



In addition to the individual range-based buckets shown in Figure 2–12, MapViewer supports other bucket styles, as explained in Section A.6.1. You can also use more than one attribute column for thematic mapping, such as when drawing pie charts (explained in Section 3.1.9).

The rest of this section presents additional examples of thematic mapping.

Example 2–38 is the XML definition for an `Earthquakes` theme.

Example 2–38 XML Definition of Styling Rules for an Earthquakes Theme

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="nature">
  <rule column="RICHTER_SCALE">
    <features style="v.earthquakes">
    </features>
  </rule>
</styling_rules>
```

The theme in Example 2–38 has only one rule. The `<rule>` element includes an attribute named `column` that does not appear in the `Airport` theme in Example 2–6. The `column` attribute specifies one or more columns (comma-delimited) that provide the attribute values needed for thematic mapping. The style specified for the `<features>` element is named `v.earthquakes`, and it is an advanced style.

Another part of the definition of the `Earthquakes` theme specifies the table that contains the data to be rendered. This table must contain a column named RICHTER_SCALE in addition to a column (of type SDO_GEOMETRY) for the spatial data. (The table and the column of type SDO_GEOMETRY must be identified in the BASE_TABLE and GEOMETRY_COLUMN columns, respectively, of the USER_SDO_THEMES view, which is described in Section 2.9.2.) The RICHTER_SCALE column must be of type NUMBER. To understand why, look at the advanced style definition in Example 2–39.

Example 2–39 Advanced Style Definition for an Earthquakes Theme

```
<?xml version="1.0" ?>
<AdvancedStyle>
  <VariableMarkerStyle basemarker="m.circle" startsize="7" increment="4">
     <Buckets>
          <RangedBucket seq="0"  label="less than 4"  high="4"/>
          <RangedBucket seq="1"  label="4 - 5" low="4" high="5"/>
          <RangedBucket seq="2"  label="5 - 6" low="5" high="6"/>
          <RangedBucket seq="3"  label="6 - 7" low="6" high="7"/>
          <RangedBucket seq="4"  label="7 and up" low="7"/>
      </Buckets>
  </VariableMarkerStyle>
</AdvancedStyle>
```

This style specifies that the marker named `m.circle` is used to indicate the location of an earthquake. The size of the marker to be rendered for an earthquake depends on the numeric value of the RICHTER_SCALE column for that row. In this example there are five buckets, each covering a predetermined range of values. For example, if an earthquake is of magnitude 5.7 on the Richter scale, the marker size will be 15 pixels (7 + 4 + 4), because the value 5.7 falls in the third bucket (5 - 6) and the starting marker size is 7 pixels (`startsize="7"`) with an increment of 4 for each range (`increment="4"`).

Note: The `label` attribute value (for example, `label="less than 4"`) is not displayed on the map, but is used only in a label that is compiled for an advanced style.

The `seq` attribute value (for example, `seq="0"`) is ignored by MapViewer, which determines sequence only by the order in which elements appear in a definition.

Example 2–39 used the `<VariableMarkerStyle>` tag. The following examples use the `<ColorSchemeStyle>` tag in creating thematic maps of census blocks in California. Example 2–40 illustrates the use of a graduated color scale for a thematic mapping of population density. Example 2–41 is a thematic mapping of average household income using a graduated color scale. Example 2–42 is also a thematic mapping of average household income, but it uses a specific color style for each income range rather a graduated scale.

Example 2–40 Mapping Population Density Using a Graduated Color Scheme

```
# ca pop density usbg_hhinfo
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="political">
<rule column="densitycy">
   <features style="v.CA Pop density">
   </features>
 </rule>
</styling_rules>
```

The table named USBG_HHINFO includes a column named DENSITYCY (used in Example 2–40). The definition of the style (`v.CA Pop density`) that corresponds to this population density theme is as follows:

```
<?xml version="1.0" ?>
<AdvancedStyle>
  <ColorSchemeStyle basecolor="#ffff00" strokecolor="#00aaaa">
     <Buckets low="0.0" high="20000.0" nbuckets="10"/>
  </ColorSchemeStyle>
</AdvancedStyle>
```

The base color (`basecolor`) and the stroke color (`strokecolor`) are 24-bit RGB (red-green-blue) values specified using a hexadecimal notation. The base color value is used for the first bucket. The color value for each subsequent bucket is obtained by first converting the base color from the RGB to the HSB (hue-saturation-brightness) model and then reducing the brightness by a fixed increment for each bucket. Thus, the first bucket is the brightest and the last is the darkest.

As in Example 2–40, Example 2–41 illustrates the use of a base color and a graduated color scheme, this time to show household income.

Example 2–41 Mapping Average Household Income Using a Graduated Color Scheme

```
<?xml version="1.0" standalone="yes"?>
<!-- # ca hh income theme  table = usbg_hhinfo  -->
<styling_rules>
<rule column="avghhicy">
   <features style="v.ca income">
   </features>
 </rule>
</styling_rules>
```

The table named USBG_HHINFO includes a column named AVGHHICY (used in Example 2–41 and Example 2–42). The definition of the style (`v.ca income`) that corresponds to this average household income theme is as follows:

```
<?xml version="1.0" ?>
<AdvancedStyle>
  <ColorSchemeStyle basecolor="#ffff00" strokecolor="#00aaaa">
  <!-- # income range with a color gradient -->
     <Buckets>
          <RangedBucket seq="0" label="less than 10k"  high="10000"/>
          <RangedBucket seq="1" label="10-15k" low="10000" high="15000"/>
          <RangedBucket seq="2" label="15-20k" low="15000" high="20000"/>
          <RangedBucket seq="3" label="20-25k" low="20000" high="25000"/>
          <RangedBucket seq="4" label="25-35k" low="25000" high="35000"/>
          <RangedBucket seq="5" label="35-50k" low="35000" high="50000"/>
          <RangedBucket seq="6" label="50-75k" low="50000" high="75000"/>
          <RangedBucket seq="7" label="75-100k" low="75000" high="100000"/>
          <RangedBucket seq="8" label="100-125k" low="100000" high="125000"/>
```

```
          <RangedBucket seq="9" label="125-150k" low="125000" high="150000"/>
          <RangedBucket seq="10" label="150-250k" low="150000" high="250000"/>
          <RangedBucket seq="11" label="250-500k" low="250000" high="500000"/>
          <RangedBucket seq="12" label="500k and up"   low="500000"/>
      </Buckets>
  </ColorSchemeStyle>
</AdvancedStyle>
```

For individual range-based buckets, the lower-bound value is inclusive, while the upper-bound value is exclusive (except for the range that has values greater than any value in the other ranges; its upper-bound value is inclusive). No range is allowed to have a range of values that overlaps values in other ranges.

Example 2–42 uses specific color styles for each average household income range.

Example 2–42 Mapping Average Household Income Using a Color for Each Income Range

```
<?xml version="1.0" standalone="yes"?>
<!-- # ca hh income theme  table = usbg_hhinfo  -->
<styling_rules>
<rule column="avghhicy">
   <features style="v.ca income 2">
   </features>
 </rule>
</styling_rules>
```

The definition of the `v.ca income 2` style is as follows:

```
<?xml version="1.0" ?>
<AdvancedStyle>
 <BucketStyle>
  <Buckets>
   <!-- # income ranges with specific colors -->
   <RangedBucket seq="0" label="less than 10k"  high="10000" style="c.rb13_1"/>
   <RangedBucket seq="1" label="10-15k" low="10000" high="15000" style="c.rb13_2"/>
   <RangedBucket seq="2" label="15-20k" low="15000" high="20000" style="c.rb13_3"/>
   <RangedBucket seq="3" label="20-25k" low="20000" high="25000" style="c.rb13_4"/>
   <RangedBucket seq="4" label="25-35k" low="25000" high="35000" style="c.rb13_5"/>
   <RangedBucket seq="5" label="35-50k" low="35000" high="50000" style="c.rb13_6"/>
   <RangedBucket seq="6" label="50-75k" low="50000" high="75000" style="c.rb13_7"/>
   <RangedBucket seq="7" label="75-100k" low="75000" high="100000" style="c.rb13_8"/>
   <RangedBucket seq="8" label="100-125k" low="100000" high="125000" style="c.rb13_9"/>
   <RangedBucket seq="9" label="125-150k" low="125000" high="150000" style="c.rb13_10"/>
   <RangedBucket seq="10" label="150-250k" low="150000" high="250000" style="c.rb13_11"/>
   <RangedBucket seq="11" label="250-350k" low="250000" high="350000" style="c.rb13_12"/>
   <RangedBucket seq="12" label="350k and up"   low="350000" style="c.rb13_13"/>
  </Buckets>
 </BucketStyle>
</AdvancedStyle>
```

Each `<RangedBucket>` definition has a specified style.

The following examples create an advanced style to identify gasoline stations operated by different oil companies, and a theme that uses the style. A `<CollectionBucket>` tag is used to associate a column value (`Shell`; `Esso`; `Texaco`; `BP`; any of `Avia`, `Benzinex`, `Q8`, `Total`, `Witte Pomp`; and all others for a default category) with a style appropriate for that company's stations, as shown in Example 2–43.

Example 2–43 Advanced Style Definition for Gasoline Stations Theme

```
<?xml version="1.0" ?>
<AdvancedStyle>
```

```
<BucketStyle>
 <Buckets>
  <CollectionBucket seq="0" label="Shell" style="m.shell gasstation">
   Shell
  </CollectionBucket>
  <CollectionBucket seq="1" label="Esso" style="m.esso gasstation">
   Esso
  </CollectionBucket>
  <CollectionBucket seq="2" label="Texaco"  style="m.texaco gasstation">
   Texaco
 </CollectionBucket>
  <CollectionBucket seq="3" label="BP"  style="m.bp gasstation">
   BP
  </CollectionBucket>
  <CollectionBucket seq="4" label="Other"  style="m.generic gasstation">
   Avia,Benzinex,Q8,Total,Witte Pomp
  </CollectionBucket>
  <CollectionBucket seq="5" label="DEFAULT" style="m.default gasstation">
   #DEFAULT#
  </CollectionBucket>
  </Buckets>
</BucketStyle>
</AdvancedStyle>
```

Notes on Example 2–43:

- `m.esso gasstation`, `m.texaco gasstation`, and the other style names have a space between the words in their names.
- The names are not case-sensitive. Therefore, be sure not to use case as a way of differentiating names. For example, `m.esso gasstation` and `M.ESSO GASSTATION` are considered the same name.
- A default collection bucket can be specified by using `#DEFAULT#` as its value. This bucket is used for any column values (gas station names) that are not specified in the other buckets.

A theme (`theme_gasstation`) is then defined that specifies the column (MERK) in the table that contains company names. The styling rules of the theme are shown in Example 2–44.

Example 2–44 Styling Rules of Theme Definition for Gasoline Stations

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule column="merk">
    <features style="v.gasstations">
    </features>
    <label column="merk" style="t.SansSerif red 10">
      1
    </label>
  </rule>
</styling_rules>
```

This theme depends on a table named NED_GASSTATIONS, which has the columns shown in Table 2–2 (with column names reflecting the fact that the developer's language is Dutch).

Table 2–2 Table Used with Gasoline Stations Theme

Column	Data Type
FID	NOT NULL NUMBER
ID	NUMBER
NAAM	VARCHAR2(31)
STRAAT_	VARCHAR2(30)
NR	NUMBER
TV	VARCHAR2(1)
AAND	VARCHAR2(2)
PCODE	VARCHAR2(6)
PLAATS	VARCHAR2(10)
GEOM	SDO_GEOMETRY
MERK	VARCHAR2(40)

In this table, the GEOM column contains spatial geometries, and the MERK column contains company names (`Shell`, `Esso`, and so on).

The styling rules for the `theme_gasstation` theme specify that the marker (style `v.gasstations`) at a location specified by the content of the GEOM column is determined by the value of the MERK column for that row. The style `v.gasstations` (see Example 2–43) specifies that if the column value is `Shell`, use the style `m.shell gasstation`; if the column value is `Esso`, use the style `m.esso gasstation`; and so on, including if the column value is any one of `Avia`, `Benzinex`, `Q8`, `Total`, and `Witte Pomp`, use the style `m.generic gasstation;` and if the column value is none of the preceding, use the style `m.default gasstation`.

2.3.11.1 Thematic Mapping Using External Attribute Data

Previous discussion of thematic mapping has assumed that both the attribute data (such as population of sales totals) and the geospatial data (geometry objects representing boundaries, locations, and so on) are in the same database. However, the attribute data can come from a source outside the current database; for example, the attribute data might reflect aggregated results of a business intelligence (BI) query performed on a different database, or the attribute data might come from a comma-delimited list of sales values exported from a spreadsheet. Such attribute data, from outside the database that contains the geospatial data, is called **external attribute data**.

To use external attribute data with MapViewer, you must use the **nonspatial data provider** plug-in mechanism, in which a custom data provider is associated with a MapViewer theme (predefined or dynamic) in the same map request. When MapViewer process the theme, it calls the nonspatial data provider to join nonspatial attribute data with the spatial data that has been fetched for the theme.

To use a nonspatial data provider, follow these steps:

1. Implement your Java nonspatial data provider by implementing the MapViewer defined interface `oracle.mapviewer.share.ext.NSDataProvider`.
2. Register the nonspatial data provider implementation with MapViewer (in its configuration file). There you can also specify a set of global parameters that your

implementation may depend on. Each custom data provider implementation class must have a unique ID that you assign.

3. Place a library containing the nonspatial data provider implementation classes in the library path of MapViewer, such as its `web/WEB-INF/lib` directory.

4. Include the nonspatial data provider implementation in a map request by invoking the following method on the MapViewer Java client API class MapViewer:

```
addNSDataProvider(java.lang.String providerId,
                  java.lang.String forTheme,
                  java.lang.String spatialKeyColumn,
                  java.lang.String customRenderingStyle,
                  java.util.Properties params,
                  long timeout)
```

   For information about the `addNSDataProvider` parameters, see the Javadoc reference information for MapViewer, available at a URL in the form `http://`*`host:port`*`/mapviewer/mapclient`, where *host* and *port* indicate where OC4J or Oracle Fusion Middleware listens for incoming requests. For example: `http://www.mycorp.com:8888/mapviewer/mapclient`

Example 2–45 shows a simple nonspatial data provider implementation. This implementation is also supplied with MapViewer as a default nonspatial data provider.

Example 2–45 Nonspatial (External) Data Provider Implementation

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Properties;
import java.util.Vector;

import oracle.mapviewer.share.ext.NSDataSet;
import oracle.mapviewer.share.ext.NSDataProvider;
import oracle.mapviewer.share.ext.NSRow;
import oracle.lbs.util.Logger;

import oracle.mapviewer.share.Field;

/**
 * A simple implementation of the NSDataProvider interface. When invoked, it
supplies tabular attribute data to MapViewer out
 * of a file or URL. The data in the file must be orgazined as following: <br>
 * <UL>
 *   <LI> The first line contain a single character which is the delimiter
 *        between columns in the subsequent lines.
 *   <LI> Each line after the first in the file represent one data row
 *   <LI> Each field in the row must be separated by the delimiter char only
 *   <LI> The first field in each line must be a string (key) that serves as the
 *         key; the rest of the fields must be numeric values
 * </UL>
 *
 * When incorporating this data provider in a map request, one of the following
 * two parameters must be specified:
 * <UL>
 *   <LI> file  if the custom data is stored in a local file; this parameter
 *        specifies the full path to that file
 *   <LI> url  if the custom data can be accessed from a web; this parameter
 *        specifeis the full URL to the data file.
```

```
 * </UL>
 *
 *
 */
public class NSDataProviderDefault implements NSDataProvider
{
  private static Logger log = Logger.getLogger("oracle.sdovis.nsdpDefault");

  public boolean init(Properties params)
  {
    return true;
  }

  public NSDataSet buildDataSet(Properties params)
  {
    String file = params.getProperty("file");
    if(file!=null)
      return readFromFile(file);

    String url = params.getProperty("url");
    if(url!=null)
      return readFromUrl(url);

    log.error("Must supply either file or url for default NS data provider.");
    return null;
  }

  public void destroy()
  {
  }

  protected NSDataSet readFromFile(String file)
  {
    BufferedReader in = null;
    try{
      in   = new BufferedReader(new FileReader(file));
      String line = in.readLine();
      String delimiter = line.substring(0,1);

      Vector rows = new Vector();

      while ( (line=in.readLine()) != null)
      {
        NSRow row = buildRow(line, delimiter);
        if(row!=null)
          rows.add(row);
      }

      NSDataSet res = new NSDataSet(rows);
      return res;
    }catch(Exception ex)
    {
      log.error(ex);
      return null;
    } finally
    {
      try{
        if(in!=null)
          in.close();
      }catch(Exception e){}
```

```
    }
  }

  protected NSDataSet readFromUrl(String url)
  {
    log.error("url not supported yet.");
    return null;
  }

  protected NSRow buildRow(String line, String delimiter)
  {
    if(line==null || line.length()<1)
      return null;

    String[] fields = line.split(delimiter);
    if(fields==null || fields.length==0)
      return null;

    Field[] row = new Field[fields.length];

    Field a = new Field(fields[0]);
    a.setKey(true);

    row[0] = a;

    for (int i = 1; i < fields.length; i++)
    {
      try{
        double d = Double.parseDouble(fields[i]);
        a = new Field(d);
        row[i] = a;
      }catch(Exception e)
      {
        log.warn("invalid row field (key="+fields[0]+")");
        return null;
      }
    }

    return new NSRow(row);

  }
}
```

2.3.12 Attributes Affecting Theme Appearance

Some attributes of the `<theme>` element affect only the appearance of the map display, rather than determining the data to be associated with the theme. These appearance-related attributes control whether and how the theme is processed and rendered when a map is generated. Examples include the following attributes:

- `min_scale` and `max_scale` determine whether or not a theme is displayed at various map scales (levels of resolution). For example, if you are displaying a map of streets, there are certain map scales at which the streets would become too dense for a usable display, such as when viewing an entire state or province. In this case, you should create a theme for streets, and specify minimum and maximum scale values to ensure that individual streets affected by the theme are displayed when the scale is appropriate and otherwise are not displayed.
- `labels_always_on` determines whether or not labels for the theme will be displayed if they would overlap another label. By choosing appropriate `labels_`

always_on values and choosing an appropriate order of themes to be processed within a map request, you can control how cluttered the labels might become and which labels have priority in getting displayed.

- fast_unpickle determines the unpickling (unstreaming) method to be used, which can involve a trade-off between performance and precision in the display.
- fixed_svglabel, visible_in_svg, selectable_in_svg, onclick, onmousemove, onmouseover, and onmouseout affect the appearance of SVG maps.

To specify any appearance-related attributes, use the <theme> element (described in Section 3.2.20) with the XML API or the JavaBean-based API (see especially Section 4.3).

2.4 Maps

A map can consist of a combination of elements and attributes, such as the following:

- Background image
- Title
- Legend
- Query window
- Footnote (such as for a copyright notice)
- Base map
- Predefined themes (in addition to any in the base map)
- JDBC themes (with dynamic queries)
- Dynamically defined (temporary) styles

These elements and attributes, when specified in a map request, define the content and appearance of the generated map. Chapter 3 contains detailed information about the available elements and attributes for a map request.

A map can have a base map and a stack of themes rendered on top of each other in a window. A map has an associated coordinate system that all themes in the map must share. For example, if the map coordinate system is 8307 (for *Longitude / Latitude (WGS 84)*, the most common system used for GPS devices), all themes in the map must have geometries defined using that coordinate system.

You can add themes to a map by specifying a base map name or by using the programming interface to add themes. The order in which the themes are added determines the order in which they are rendered, with the last specified theme on top, so be sure you know which themes you want in the background and foreground.

All base map names and definitions for a database user are stored in that user's USER_SDO_MAPS view, which is described in Section 2.9 and Section 2.9.1. The DEFINITION column in the USER_SDO_MAPS view contains an XML definition of a base map.

Example 2–46 shows a base map definition.

Example 2–46 XML Definition of a Base Map

```
<?xml version="1.0" ?>
<map_definition>
<theme name="theme_us_states"    min_scale="10"  max_scale="0"/>
```

```
<theme name="theme_us_parks"     min_scale="5"    max_scale="0"/>
<theme name="theme_us_highways"  min_scale="5"    max_scale="0"/>
<theme name="theme_us_streets"   min_scale="0.05" max_scale="0"/>
</map_definition>
```

Each theme in a base map can be associated with a visible scale range within which it is displayed. In Example 2–46, the theme named `theme_us_streets` is not displayed unless the map request is for a map scale of 0.05 or less and greater than 0 (in this case, a scale showing a great deal of detail). If the `min_scale` and `max_scale` attributes are not specified, the theme is displayed whenever the base map is displayed. (For more information about map scale, see Section 2.4.1.)

The display order of themes in a base map is the same as their order in the base map definition. In Example 2–46, the `theme_us_states` theme is rendered first, then `theme_us_parks`, then `theme_us_highways`, and finally (if the map scale is within all specified ranges) `theme_us_streets`.

This section contains the following major subsections:

- Section 2.4.1, "Map Size and Scale"
- Section 2.4.2, "Map Legend"

2.4.1 Map Size and Scale

Map size is the height of the map in units of the map data space. For example, if the map data is in WGS 84 geographic coordinates, the map center is (-120.5, 36.5), and the size is 2, then the height of the map is 2 decimal degrees, the lower Y (latitude) value is 35.5 degrees, and the upper Y value is 37.5 decimal degrees.

Map scale is expressed as units in the user's data space that are represented by 1 inch on the screen or device. Map scale for MapViewer is actually the denominator value in a popular method of representing map scale as 1/*n*, where:

- 1, the numerator, is 1 unit (1 inch for MapViewer) on the displayed map.
- *n*, the denominator, is the number of units of measurement (for example, decimal degrees, meters, or miles) represented by 1 unit (1 inch for MapViewer) on the displayed map.

For example:

- If 1 inch on a computer display represents 0.5 decimal degree of user data, the fraction is 1/0.5. The decimal value of the fraction is 2.0, but the scale value for MapViewer is 0.5.
- If 1 inch on a computer display represents 2 miles of user data, the fraction is 1/2. The decimal value of the fraction is 0.5, but the scale value for MapViewer is 2.
- If 1 inch on a computer display represents 10 miles of user data, the fraction is 1/10. The decimal value of the fraction is 0.1, but the scale value for MapViewer is 10.

The `min_scale` and `max_scale` attributes in a `<theme>` element describe the visible scale range of a theme. These attributes control whether or not a theme is displayed, depending on the current map scale. The default scale value for `min_scale` is positive infinity, and the default value for `max_scale` is negative infinity (or in other words, by default display the theme for all map scales, if possible given the display characteristics).

- `min_scale` is the value to which the display must be zoomed in for the theme to be displayed. For example, if parks have a `min_scale` value of 5 and if the

current map scale value is 5 or less but greater than the `max_scale` value, parks will be included in the display; however, if the display is zoomed out so that the map scale value is greater than 5, parks will not be included in the display.

- `max_scale` is the value beyond which the display must be zoomed in for the theme not to be displayed. For example, if counties have a `max_scale` value of 3 and if the current map scale value is 3 or less, counties will not be included in the display; however, if the display is zoomed out so that the map scale value is greater than 3, counties will be included in the display.

A high `min_scale` value is associated with less map detail and a smaller scale in cartographic terms, while a high `max_scale` value is associated with greater map detail and a larger scale in cartographic terms. (Note that the MapViewer meaning of map scale is different from the popular meaning of cartographic map scale.) The `min_scale` value for a theme should be larger than the `max_scale` value. Example 2–46 in Section 2.4 includes `min_scale` and `max_scale` values.

You also assign scale values for theme labels, to enable the showing or hiding of labels with values different from the base theme scales, by using the theme label scale parameters `label_min_scale` and `label_max_scale`. These parameters are similar to the `min_scale` and `max_scale` parameters, but the labels are shown if the map scale is in the visible range defined by `label_min_scale` and `label_max_scale`. (The label scale values are ignored if the theme is not in the visible scale range defined by `min_scale` and `max_scale`.) The following is a theme definition with label scale values; the labels will be shown when the map scale is between 5 and 2, but the theme features will be shown when the map scale is between 10 and 0:

```
<theme name="theme_us_states" min_scale="10" max_scale="0"
     label_min_scale="5" label_max_scale="2"/>
```

To determine the current map scale for a map returned by MapViewer, first find the map size, namely the height (vertical span) of the map in terms of the coordinate system associated with the map data. For example, assume that a map with a height of 10 (miles, meters, decimal degrees, or whatever unit of measurement is associated with the data) is requested, and that the map is drawn on a device with a size of 500 by 350 pixels, where 350 is the height. MapViewer assumes a typical screen resolution of 96 dpi. Because 96 pixels equals 1 inch, the height of the returned map is 3.646 inches (350/96 = 3.646). In this example, the size of the map is 10, and therefore the map scale is approximately 2.743 (10/3.646 = 2.743).

Alternatively, you can request a map using a map scale value without specifying a unit, such as 50000 for a scale of 1:50000, by specifying the `scale_mode` theme attribute value as `ratio`. (If the `scale_mode` theme attribute value is `screen_inch`, the scale refers to a unit.) To use a scale defined without a unit, request the map specifying the center and ratio scale.

To find the equivalent MapViewer screen inch scale for a ratio scale, follow these steps:

1. Find the numerical fraction of a meter associated with one screen pixel. For example, if the screen resolution is 96 dpi (dots per inch), the number of meters on the screen for each screen pixel is 0.000265 (that is, 0.0254/96).
2. Find the map scale for one screen pixel (the `mapdotScale` value), as follows:
   - For projected data (meters), multiply the result of step 1 by the ratio scale. For example, if the ratio scale is 50000 (50 thousand) and the screen resolution is 96 dpi, the result is 13.25 meters for each pixel (50000 * 0.000265).

  - For geodetic data (degrees), multiply the result of step 1 by the number of meters (on the surface of the Earth) for each degree. (This number will depend on the coordinate system associated with the data.) For example, if one degree = 111195 meters and if the screen resolution is 96 dpi, the result is 29.466675 meters for each pixel (111195 * 0.000265).
  - For data using any other unit, use the approach for projected data using meters.

3. Because the MapViewer scale is per screen inch instead of per screen pixel, multiply the result of step 2 by the dpi value. For example, if the result of step 2 is 13.25 meters at 96 dpi, the number of meters for each screen inch is 1272 (13.25 * 96).

2.4.2 Map Legend

A **map legend** is an inset illustration drawn on top of the map and describing what various colors, symbols, lines, patterns, and so on represent. You have flexibility in specifying the content and appearance of the legend. You can:

- Customize the background, border style, and font
- Have one or more columns in the legend
- Add space to separate legend entries
- Indent legend entries
- Use any MapViewer style, including advanced styles

Example 2–47 is an excerpt from a request that includes a legend.

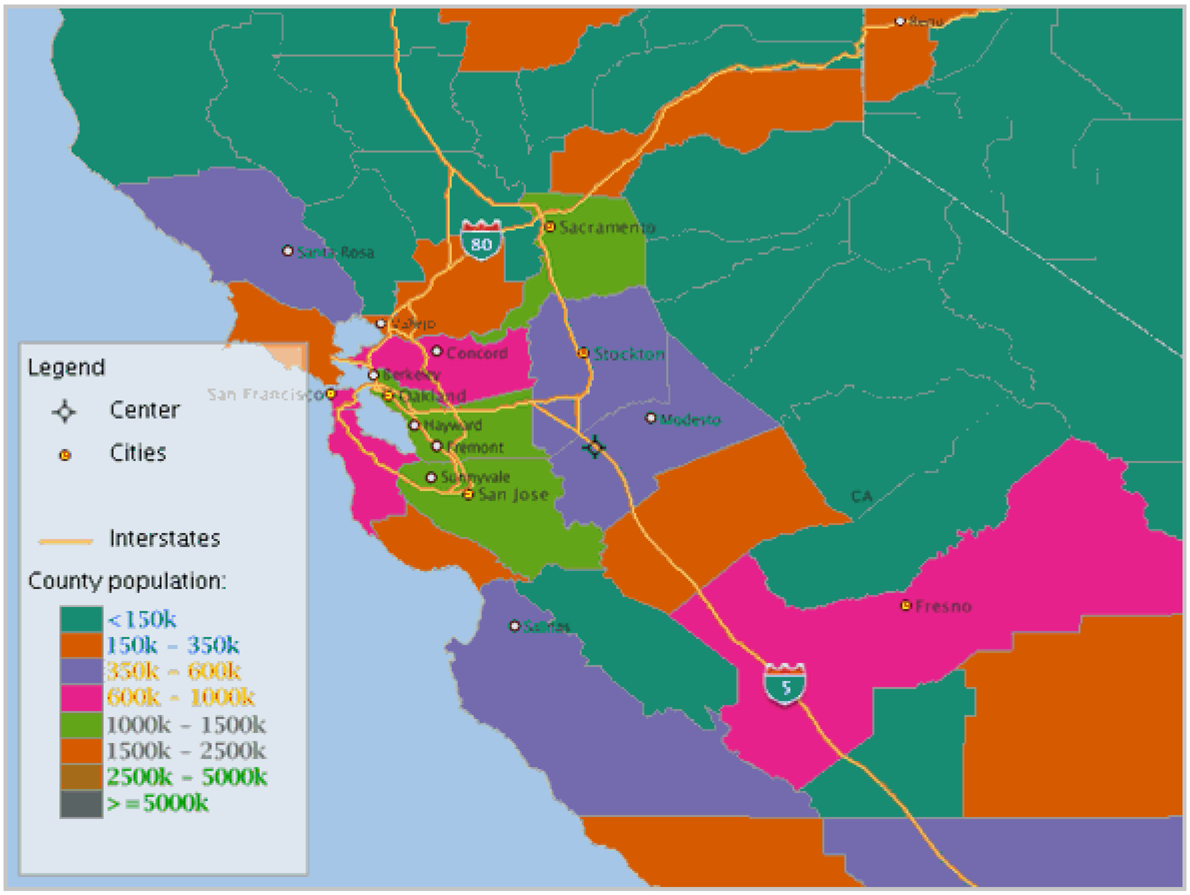
Example 2–47 Legend Included in a Map Request

```
<?xml version="1.0" standalone="yes"?>
<map_request
             basemap="density_map"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="false"
             format="PNG_STREAM">
  <center size="4.2">
     <geoFeature render_style="m.image134_bw">
         <geometricProperty typeName="center">
             <Point srsName="SDO:8307">
                 <coordinates>-121.2615, 37.5266</coordinates>
             </Point>
         </geometricProperty>
     </geoFeature>
  </center>

  <legend bgstyle="fill:#ffffff;fill-opacity:100;stroke:#a3a3a3" profile="MEDIUM"
position="SOUTH_WEST">
          <column>
            <entry text="Legend" is_title="true" />
            <entry style="M.IMAGE134_BW" text="Center" />
            <entry style="M.ALL_CITY_L2" text="Cities" />
            <entry is_separator="true" />
            <entry style="L.S04_ROAD_INTERSTATE" text="Interstates" />
            <entry text="County population:" />
```

```
            <entry style="V.COUNTY_POP_DENSITY_8" tab="1" />
          </column>
  </legend>

  <!--
  <themes>

  </themes>
  -->

</map_request>
```

Figure 2–13 shows a map with the legend specified in Example 2–47.

Figure 2–13 Map with Legend



Notes on Example 2–47 and Figure 2–13:

- This example shows a legend with a single column, although you can create multiple columns in a legend.
- Each entry in the column definition can identify label text and whether the text is the legend title (`is_title="true"`), a style name and associated text, or a separator (`is_separator="true"`) for vertical blank space to be added (after the *cities* entry in this example).

As an alternative to specifying the legend content in one or more <column> elements, you can request an **automatic legend** based on the map request. With an automatic legend, you specify the legend header, and MapViewer generates the legend based on

the themes that have any interaction with the map area. Themes from the map request and from the base map are considered. (Some legend items might not be visible, though, such as if a theme interacts with the query window but no features of the theme are visible on the map.)

Example 2–48 is a map request that requests an automatic legend (because the `<legend>` element does not include any `<column>` elements).

Example 2–48 Map Request with Automatic Legend

```
<?xml version="1.0" standalone="yes"?>
<map_request
             title="Automatic legend"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="false"
             format="PNG_STREAM">
  <center size="4.5">
     <geoFeature >
         <geometricProperty typeName="center">
             <Point>
                 <coordinates>-122.2615, 37.5266</coordinates>
             </Point>
         </geometricProperty>
     </geoFeature>
  </center>

  <themes>
    <theme name="THEME_COUNTIES_3397829" />
    <theme name="THEME_US_ROAD1" />
    <theme name="THEME_US_AIRPORT" />
  </themes>

 <legend bgstyle="fill:#ffffff;fill-opacity:128;stroke:#ff0000;stroke-opacity:128"
profile="medium" font="Courier">
  </legend>

</map_request>
```

Example 2–49 requests an automatic legend in which the `<legend>` elements specifies the themes to be used to generate the legend items. In this example, even if the map result shows more themes, the legend items are based on the `THEME_COUNTIES_3397829` and `THEME_US_AIRPORT` themes specified in the `<legend>` element.

Example 2–49 Automatic Legend with Themes Specified

```
<map_request
             title="Legend with themes defined"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="false"
             format="PNG_STREAM">
  <center size="4.5">
     <geoFeature >
         <geometricProperty typeName="center">
             <Point>
```

```
                <coordinates>-122.2615, 37.5266</coordinates>
            </Point>
        </geometricProperty>
    </geoFeature>
  </center>

  <themes>
    <theme name="THEME_COUNTIES_3397829" />
    <theme name="THEME_US_ROAD1" />
    <theme name="THEME_US_AIRPORT" />
  </themes>

 <legend bgstyle="fill:#ffffff;fill-opacity:128;stroke:#ff0000;stroke-opacity:128"
profile="medium" font="Courier">
  <themes>
    <theme name="THEME_COUNTIES_3397829" />
    <theme name="THEME_US_AIRPORT" />
  </themes>
 </legend>

</map_request>
```

You cannot combine an automatic legend with the use of `<column>` elements. If the `<legend>` element contains any `<column>` elements, a column/entry legend is created.

MapViewer used the following considerations when it builds automatic legend items:

- Each legend column has a maximum of five entries (an advanced style is considered one entry).
- The legend text for simple rendering styles comes from the theme description if defined, otherwise from the theme name.
- If a rendering style is used in more than one theme, the style is repeated in the legend but with text related to the theme to which it applies.
- Labeling styles are not repeated in the legend. The style text for labeling styles comes from the style description.
- Advanced styles are not repeated in the legend.

For detailed information about adding a legend to a map request, see Section 3.2.11.

If you also specify a map title, note, or logo (or any combination), be sure that the legend and the other features have different positions. (Map titles, notes, and logos are explained in Section 1.5.2.5.) The default position for a legend is `SOUTH_WEST`.

2.5 Data Sources

A data source corresponds to a database schema or user. Before you can draw any spatial data in a database schema, you must first define (create) a data source for the schema, either permanently or dynamically:

- You can define a data source permanently by specifying its connection information and user login credentials in the MapViewer configuration file (`mapViewerConfig.xml`).
- You can define or modify a data source dynamically using the MapViewer administration (Admin) page.

Each map request must specify a master data source. You can, however, specify a different data source for individual themes added to the map request. This makes it easy to aggregate data stored across different database schemas. If a theme has no specified data source, it is associated with the master data source. A base map (and thus the themes included in it) is always associated with the master data source. When a theme is processed, all of its underlying data, as well as the styles referenced in its definition, must be accessible from the data source or sources associated with the theme.

Each data source has associated renderers (sometimes called mappers or map makers), the number of which is determined by the `number_of_mappers` attribute in the <map_data_source> element. This attribute (described in Section 1.5.2.14) affects the number of database connections created for each data source when map requests are processed. The number of renderers specified in a data source also is the maximum number of concurrent requests that can be processed for that data source. Each additional renderer requires only a small amount of memory, so the main potential disadvantage of specifying a large number of renderers (such as 100) is that the underlying CPU resource might be strained if too many map requests are allowed to come through, thus affecting the performance of the entire MapViewer server.

Each data source has its own internal metadata cache. The metadata cache holds the definitions of all accessed styles, as well as of all predefined themes that originate from the data source. This eliminates the need to query the database repeatedly for the definition of a style or predefined theme whenever it is needed.

2.6 How a Map Is Generated

When a map request arrives at the MapViewer server, the server picks a free renderer associated with the master data source in the request. This section describes the process that the MapViewer server follows to generate a map. In brief, MapViewer performs the following steps:

1. Parse and process the incoming XML map request.
2. Prepare the data for each theme (executed in parallel).
3. Render and label each theme.
4. Generate final images or files.

Each map generated by MapViewer results from its receiving a valid XML map request. (If you use the JavaBean-based API, the request is automatically converted to an XML document and passed to the MapViewer server.) The XML map request is parsed and its content is validated. MapViewer then creates any dynamic styles specified in the XML request. It builds a theme list from all themes included in the base map (if a base map is specified), as well as any specified predefined or JDBC themes. All individual features in the request are grouped into a single temporary theme. In other words, after parsing the incoming request, all data that must be shown on the map is presented in a list of themes to the MapViewer rendering engine.

The ordering of the themes in the list is important, because it determines the order in which the themes are rendered. All themes included in the base map (when present) are added to the list first, followed by all specified themes (predefined or JDBC). The theme that contains all the individual features is added as the last theme on the list. Any other requested features of a map (such as legend, map title, or footnote), are created and saved for rendering later.

For each theme in the request, MapViewer then creates a separate execution thread to prepare its data, so that preparation of the themes takes place in parallel. For a

predefined theme, this means formulating a query based on the theme's definition and any other information, such as the current map request window. This query is sent to the database for execution, and the result set is returned. MapViewer creates individual renderable objects based on the result set.

- For predefined themes that are fully cached, no query is sent to the database, because all renderable objects are readily available.
- For JDBC themes, the query supplied by the user is either executed as is (when the `asis` attribute value is `TRUE` in the JDBC theme definition) or with a spatial filter subquery automatically applied to it. The spatial filter part is used to limit the results of the user's query to those within the current requested window.
- For themes that already have renderable features (such as the one containing all individual features in a request), there is no need to create renderable objects.

After all themes for the map request have been prepared and all necessary data has been collected, MapViewer starts to render the map. It creates an empty new in-memory image to hold the result map, and paints the empty image with the necessary backgrounds (color or image). It then renders all of the themes in the theme list.

> **Note:** All image or GeoRaster themes are always rendered first, regardless of their position in the theme list. All other themes, however, are rendered in the order in which they appear in the theme list.

For each theme, features are rendered in an order determined internally by MapViewer. The rendering of each feature involves invoking the drawing methods of its rendering style. After all themes have been rendered, the labeling process starts. For each theme whose features must be labeled with text, MapViewer invokes algorithms to label each feature, with the specific algorithm depending on the type of feature (such as polygon or line).

After all themes have been rendered and (when needed) labeled, MapViewer plots any additional map features (such as a legend) on the internal map image. MapViewer then converts that image into the desired format (such as PNG or GIF) specified in the original map request; however, for SVG maps, instead of using an internal image, MapViewer initially creates an empty SVG map object, then creates an SVG document as a result of the rendering process, and inserts it into the map object.

2.7 Cross-Schema Map Requests

A database user can issue a map request specifying a theme that uses data associated with another database user, to select data from tables that the other data source user is authorized to access. For example, assume that user SCOTT wants to issue a map request using data associated with user MVDEMO. In general, user SCOTT must be granted SELECT access on relevant tables owned by user MVDEMO, and the `<theme>` element should generally specify any tables in *schema-name.table-name* format. In this example scenario:

- For a geometry table, grant the SELECT privilege on the geometry table of MVDEMO to SCOTT (see Example 2–50).
- For a GeoRaster table, grant the SELECT privilege on the GeoRaster table and raster data table or tables of MVDEMO to SCOTT (see Example 2–51).

- For a topology data model table, grant the SELECT privilege on the topology table, topology column index table, and related topology information tables (*topology-name*_EDGE$, *topology-name*_NODE$, *topology-name*_FACE$, *topology-name*_RELATION$) of MVDEMO to SCOTT (see Example 2–52).
- For network data model tables, grant the SELECT privilege on the network link, node, path, and path-link tables of MVDEMO to SCOTT (see Example 2–53).

Example 2–50 shows a dynamic theme that accesses the MVDEMO.STATES geometry table from a data source defined on the SCOTT user.

Example 2–50 Cross-Schema Access: Geometry Table

```
SQL> grant select on STATES to SCOTT;
. . .
<themes>
  <theme name="theme1">
     <jdbc_query
       datasource="scottds"
       spatial_column="geom"
       render_style="MVDEMO:C.COUNTIES"
       jdbc_srid="8265"
       >SELECT geom from MVDEMO.STATES</jdbc_query>
   </theme>
</themes>
```

Example 2–51 shows a dynamic theme that accesses the MVDEMO.GEORASTER_TABLE GeoRaster table and its RDT from a data source defined on the SCOTT user. Specify the base (GeoRaster) table in *schema-name.table-name* format.

Example 2–51 Cross-Schema Access: GeoRaster Table

```
SQL> grant select on GEORASTER_TABLE to SCOTT;
SQL> grant select on RDT_GEOR1 to SCOTT;
. . .
<themes>
  <theme name="georaster_theme">
    <jdbc_georaster_query
       georaster_table="MVDEMO.georaster_table"
       georaster_column="georaster"
       raster_table="rdt_geor1"
       raster_id="1"
       jdbc_srid="8307"
       datasource="scottds"
       asis="false">
     </jdbc_georaster_query>
  </theme>
</themes>
```

Example 2–52 shows a dynamic theme that accesses the MVDEMO.LAND_PARCELS topology table and information tables for the CITY_DATA topology from a data source defined on the SCOTT user. Specify the feature table and the topology in *schema-name.object-name* format, if they are owned by a different schema than the one associated with the data source.

Example 2–52 Cross-Schema Access: Topology Feature Table

```
SQL> grant select on CITY_DATA_FACE$ to SCOTT;
SQL> grant select on CITY_DATA_EDGE$ to SCOTT;
SQL> grant select on CITY_DATA_NODE$ to SCOTT;
```

```
SQL> grant select on CITY_DATA_RELATION$ to SCOTT;
SQL> grant select on LAND_PARCELS to SCOTT;
SQL> grant select on <topology-column-index-table-name> to SCOTT;
. . .
<themes>
  <theme name="topo_theme" >
     <jdbc_topology_query
       topology_name="MVDEMO.CITY_DATA"
       feature_table="MVDEMO.LAND_PARCELS"
       spatial_column="FEATURE"
       render_style="MVDEMO:C.COUNTIES"
       jdbc_srid="0"
       datasource="scottds"
       asis="false">select feature from MVDEMO.land_parcels
     </jdbc_topology_query>
   </theme>
</themes>
```

In Example 2–52, you must grant SELECT on the topology column index table name (<*topology-column-index-table-name*>) because the spatial index table associated with the feature table topology column is used by MapViewer in topology queries. You can determine the topology column index table name as follows. Assume the following information:

- Topology feature table owner: MVDEMO
- Topology feature table name: LAND_PARCELS
- Topology feature table topology column name: FEATURE

The following query returns the index table name (in this example, MDTP_14E60$):

```
SQL> select sdo_index_table from all_sdo_index_info
     where table_owner = 'MVDEMO'
     and table_name = 'LAND_PARCELS'
     and column_name = 'FEATURE'

SDO_INDEX_TABLE
--------------------------------
MDTP_14E60$
```

Then, modify the last GRANT statement in Example 2–52, "Cross-Schema Access: Topology Feature Table" to specify the <*topology-column-index-table-name*>. In this case:

```
SQL> grant select on MDTP_14E60$ to SCOTT;
```

Example 2–53 shows a dynamic theme that accesses the MVDEMO.BI_TEST network and its link, node, path, and path-link tables. Specify the network name in *schema-name.network-name* format.

Example 2–53 Cross-Schema Access: Network Tables

```
SQL> grant select on BI_TEST_LINK$ to SCOTT;
SQL> grant select on BI_TEST_NODE$ to SCOTT;
SQL> grant select on BI_TEST_PATH$ to SCOTT;
SQL> grant select on BI_TEST_PLINK$ to SCOTT;
. . .
<themes>
  <theme name="net_theme" >
     <jdbc_network_query
       network_name="MVDEMO.BI_TEST"
       network_level="1"
```

```
         jdbc_srid="0"
         datasource="scottds"
         link_style="MVDEMO:C.RED"
         node_style="MVDEMO:M.CIRCLE"
         node_markersize="5"
         asis="false">
      </jdbc_network_query>
   </theme>
</themes>
```

2.8 Workspace Manager Support in MapViewer

Workspace Manager is an Oracle Database feature that lets you version-enable one or more tables in the database. After a table is version-enabled, users in a workspace automatically see the correct version of database rows in which they are interested. For detailed information about Workspace Manager, see *Oracle Database Workspace Manager Developer's Guide*.

You can request a map from a specific workspace, at a specific savepoint in a workspace, or at a point close to a specific date in a workspace. The following attributes of the `<theme>` element are related to support for Workspace Manager:

- `workspace_name` attribute: specifies the name of the workspace from which to get the map data.
- `workspace_savepoint` attribute: specifies the name of the savepoint to go to in the specified workspace.
- `workspace_date` attribute: specifies the date to go to (that is, a point at or near the specified date) in the specified workspace.
- `workspace_date_format` attribute: specifies the date format. The default is `mmddyyyyhh24miss`. This attribute applies only if you specified the `workspace_date` attribute.
- `workspace_date_nlsparam` attribute: specifies globalization support options. The options and default are the same as for the `nlsparam` argument to the `TO_CHAR` function for date conversion, which is described in *Oracle Database SQL Language Reference*.
- `workspace_date_tswtz` attribute: specifies a Boolean value. `TRUE` means that the input date is in timestamp with time zone format; `FALSE` (the default) means that the input date is a date string.

The `workspace_name` attribute is required for the use of Workspace Manager support in MapViewer.

If you specify neither the `workspace_savepoint` nor `workspace_date` attribute, MapViewer goes to the latest version of the workspace defined. If you specify both the `workspace_savepoint` and `workspace_date` attributes, MapViewer uses the specified date instead of the savepoint name.

Example 2–54 shows the definition of a dynamic theme that uses attributes (shown in bold) related to Workspace Manager support. In this example, MapViewer will render the data related to workspace `wsp_1` at the savepoint `sp1`.

Example 2–54 Workspace Manager-Related Attributes in a Map Request

```
<?xml version="1.0" standalone="yes"?>
<map_request
 . . .
```

```
  <themes>
    <theme name="wmtheme" user_clickable="false"
       workspace_name="wsp_1" workspace_savepoint="sp1" >
       <jdbc_query
         spatial_column="GEOM"
         render_style="stylename"
         jdbc_srid="8307"
         datasource="mvdemo"
         asis="false"> select GEOM,ATTR from GEOM_TABLE
       </jdbc_query>
     </theme>
  </themes>
 . . .
</map_request>
```

The following considerations apply to MapViewer caching of predefined themes (explained in Section 2.3.1.5) and the use of Workspace Manager-related MapViewer attributes:

- The Workspace Manager-related attributes are ignored for predefined themes if the `caching` attribute is set to `ALL` in the <styling_rules> element for the theme.
- No caching data is considered if you specify the `workspace_name` attribute.

For MapViewer administrative requests (discussed in Chapter 7), the following elements are related to Workspace Manager support:

- <list_workspace_name>
- <list_workspace_session>

The <list_workspace_name> element returns the name of the current workspace, as specified with the `workspace_name` attribute in the most recent map request. If no workspace has been specified (that is, if the `workspace_name` attribute has not been specified in a map request in the current MapViewer session), or if the `LIVE` workspace has been specified, the `LIVE` workspace is returned. If Workspace Manager is not currently installed in Oracle Database, the request fails.

Example 2–55 uses the <list_workspace_name> element in an administrative request.

Example 2–55 <list_workspace_name> Element in an Administrative Request

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <list_workspace_name data_source="mvdemo"/>
</non_map_request>
```

If `wsp_1` is the current workspace, the response for Example 2–55 will be:

```
<?xml version="1.0" ?>
<non_map_response>
  <workspace_name succeed="true" name="wsp_1"/>
</non_map_response>
```

If no workspace has been specified or if the `LIVE` workspace has been specified, the response for Example 2–55 will be:

```
<?xml version="1.0" ?>
<non_map_response>
  <workspace_name succeed="true" name="LIVE"/>
</non_map_response>
```

If Workspace Manager is not currently installed in Oracle Database, the response for Example 2–55 will be:

```
<?xml version="1.0" ?>
<non_map_response>
  <workspace_name succeed="false"/>
</non_map_response>
```

The <list_workspace_session> element returns the names of the current workspace and current context. If no workspace has been specified (that is, if the workspace_name attribute has not been specified in a map request in the current MapViewer session), or if the LIVE workspace has been specified, information for the LIVE workspace is returned. If Workspace Manager is not currently installed in Oracle Database, the request fails.

Example 2–56 uses the <list_workspace_session> element in an administrative request.

Example 2–56 <list_workspace_session> Element in an Administrative Request

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <list_workspace_session data_source="mvdemo"/>
</non_map_request>
```

If wsp_1 is the current workspace and if the context is LATEST, the response for Example 2–56 will be:

```
<?xml version="1.0" ?>
<non_map_response>
  <workspace_session succeed="true" name="wsp_1" context="LATEST"
     context_type="LATEST"/>
</non_map_response>
```

If no workspace has been specified or if the LIVE workspace has been specified, and if the context is LATEST, the response for Example 2–56 will be:

```
<?xml version="1.0" ?>
<non_map_response>
  <workspace_session succeed="true" name="LIVE" context="LATEST"
     context_type="LATEST"/>
</non_map_response>
```

If Workspace Manager is not currently installed in Oracle Database, the response for Example 2–56 will be:

```
<?xml version="1.0" ?>
<non_map_response>
  <workspace_session succeed="false"/>
</non_map_response>
```

2.9 MapViewer Metadata Views

The mapping metadata describing base maps, themes, and styles is stored in the global tables SDO_MAPS_TABLE, SDO_THEMES_TABLE, and SDO_STYLES_TABLE, which are owned by MDSYS. However, you should never directly update these tables. Each MapViewer user has the following views available in the schema associated with that user:

- USER_SDO_MAPS and ALL_SDO_MAPS contain information about base maps. These views are described in Section 2.9.1.
- USER_SDO_THEMES and ALL_SDO_THEMES contain information about themes. These views are described in Section 2.9.2.
- USER_SDO_STYLES and ALL_SDO_STYLES contain information about styles. These views are described in Section 2.9.3.

Note: You can use the Map Builder tool (described in Chapter 9) to manage most mapping metadata. However, for some features you must use SQL statements to update the MapViewer metadata views.

The USER_SDO_xxx views contain metadata information about mapping elements (styles, themes, base maps) owned by the user (schema), and the ALL_SDO_xxx views contain metadata information about mapping elements on which the user has SELECT permission.

The ALL_SDO_xxx views include an OWNER column that identifies the schema of the owner of the object. The USER_SDO_xxx views do not include an OWNER column.

All styles defined in the database can be referenced by any user to define that user's themes, markers with a text style, or advanced styles. However, themes and base maps are not shared among users; so, for example, you cannot reference another user's themes in a base map that you create.

The following rules apply for accessing the mapping metadata:

- If you need to add, delete, or modify any metadata, you must perform the operations using the USER_SDO_xxx views. The ALL_SDO_xxx views are automatically updated to reflect any changes that you make to USER_SDO_xxx views.
- If you need only read access to the metadata for all styles, you should use the ALL_SDO_STYLES view. Both the OWNER and NAME columns make up the primary key; therefore, when you specify a style, be sure to include both the OWNER and NAME.

The preceding MapViewer metadata views are defined in the following file:

```
$ORACLE_HOME/lbs/admin/mapdefinition.sql
```

MapViewer also uses some other metadata views, which may be defined in other files. You should never modify the contents of these views, which include the following:

- MDSYS.USER_SDO_CACHED_MAPS is used by the map tile server, which is part of Oracle Maps (described in Chapter 8).
- MDSYS.USER_SDO_TILE_ADMIN_TASKS includes information about long tasks related to map tile management. If you stop a long map tile layer task such as prefetching and then restart the task, MapViewer uses the information in the USER_SDO_TILE_ADMIN_TASKS view to resume the task rather than start over at the beginning.

2.9.1 xxx_SDO_MAPS Views

The USER_SDO_MAPS and ALL_SDO_MAPS views have the columns listed in Table 2–3.

Table 2–3 xxx_SDO_MAPS Views

Column Name	Data Type	Description
OWNER	VARCHAR2	Schema that owns the base map (ALL_SDO_MAPS only)
NAME	VARCHAR2	Unique name to be associated with the base map
DESCRIPTION	VARCHAR2	Optional descriptive text about the base map
DEFINITION	CLOB	XML definition of the list of themes and their scale value range information to be associated with the base map

2.9.2 xxx_SDO_THEMES Views

The USER_SDO_THEMES and ALL_SDO_THEMES views have the columns listed in Table 2–4.

Table 2–4 xxx_SDO_THEMES Views

Column Name	Data Type	Description
OWNER	VARCHAR2	Schema that owns the theme (ALL_SDO_THEMES only)
NAME	VARCHAR2	Unique name to be associated with the theme
DESCRIPTION	VARCHAR2	Optional descriptive text about the theme
BASE_TABLE	VARCHAR2	Table or view containing the spatial geometry column
GEOMETRY_ COLUMN	VARCHAR2	Name of the spatial geometry column (of type SDO_ GEOMETRY)
STYLING_ RULES	CLOB	XML definition of the styling rules to be associated with the theme

2.9.3 xxx_SDO_STYLES Views

The USER_SDO_STYLES and ALL_SDO_STYLES views have the columns listed in Table 2–5.

Table 2–5 xxx_SDO_STYLES Views

Column Name	Data Type	Description
OWNER	VARCHAR2	Schema that owns the style (ALL_SDO_STYLES only)
NAME	VARCHAR2	Unique name to be associated with the style
TYPE	VARCHAR2	One of the following values: COLOR, MARKER, LINE, AREA, TEXT, or ADVANCED
DESCRIPTION	VARCHAR2	Optional descriptive text about the style
DEFINITION	CLOB	XML definition of the style
IMAGE	BLOB	Image content (for example, `airport.gif`) for marker or area styles that use image-based symbols (for markers) or fillers (for areas)
GEOMETRY	SDO_ GEOMETRY	(Reserved for future use)

Depending on the Oracle Database release, the ALL_SDO_STYLES view may contain sample styles owned by the MDSYS schema. If these styles are defined on your system, you can specify them in theme definitions and map requests, and you can examine the XML definitions for ideas to use in defining your own styles.

To specify a style (or other type of MapViewer object) that is owned by a schema other than the one for the current user, you must specify the schema name, and you must use a colon (:), not a period, between the schema name and the object name. The following excerpt from a <jdbc_query> element refers to the style named `C.RED` owned by the MDSYS schema:

```
<jdbc_query . . . render_style="MDSYS:C.RED">
. . .
 </jdbc_query>
```

Example 2–57 finds the names of all currently defined styles owned by the MDSYS schema, and it displays the type, description, and XML definition of one of the styles. (The example output is reformatted for readability.)

Example 2–57 Finding Styles Owned by the MDSYS Schema

```
SELECT owner, name FROM all_sdo_styles
  WHERE owner = 'MDSYS';

OWNER                            NAME
-------------------------------- --------------------------------
MDSYS                            C.BLACK
MDSYS                            C.BLACK GRAY
MDSYS                            C.BLUE
MDSYS                            C.COUNTIES
MDSYS                            C.FACILITY
. . .
MDSYS                            L.MAJOR STREET
MDSYS                            L.MAJOR TOLL ROAD
MDSYS                            L.MQ_ROAD2
MDSYS                            L.PH
MDSYS                            L.POOR_ROADS
MDSYS                            L.PTH
MDSYS                            L.RAILROAD
MDSYS                            L.RAMP
MDSYS                            L.SH
MDSYS                            L.STATE BOUNDARY
. . .
MDSYS                            M.REDSQ
MDSYS                            M.SMALL TRIANGLE
MDSYS                            M.STAR
MDSYS                            M.TOWN HALL
MDSYS                            M.TRIANGLE
MDSYS                            T.AIRPORT NAME
MDSYS                            T.CITY NAME
MDSYS                            T.MAP TITLE
MDSYS                            T.PARK NAME
MDSYS                            T.RED STREET
MDSYS                            T.ROAD NAME
MDSYS                            T.SHIELD1
MDSYS                            T.SHIELD2
MDSYS                            T.STATE NAME
MDSYS                            T.STREET NAME
. . .

-- Display the type, description, and XML definition of one style.
SET LONG 4000;
SELECT owner, name, type, description, definition
  FROM all_sdo_styles WHERE name = 'L.PH';
```

```
OWNER   NAME     TYPE       DESCRIPTION
------  -----    ------     ------------------
MDSYS   L.PH     LINE       Primary highways

DEFINITION
---------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<desc></desc>
<g class="line" style="fill:#33a9ff;stroke-width:4">
<line class="parallel" style="fill:#aa55cc;stroke-width:1.0"/>
</g>
</svg>
```

3

MapViewer Map Request XML API

This chapter explains how to submit map requests in XML format to MapViewer, and it describes the XML document type definitions (DTDs) for the map requests (input) and responses (output). XML is widely used for transmitting structured documents using the HTTP protocol. If an HTTP request (`GET` or `POST` method) is used, it is assumed the request has a parameter named `xml_request` whose value is a string containing the XML document for the request.

(In addition to map requests, the MapViewer XML API can be used for administrative requests, such as adding new data sources. Administrative requests are described in Chapter 7.)

As shown in Figure 1–1 in Section 1.1.1, the basic flow of action with MapViewer is that a client locates a remote MapViewer instance, binds to it, sends a map request, and processes the map response returned by the MapViewer instance.

A request to the MapViewer servlet has the following format:

```
http://hostname[:port]/MapViewer-servlet-path?xml_request=xml-request
```

In this format:

- *`hostname`* is the network path of the server on which MapViewer is running.
- *`port`* is the port on which the web server listens.
- *`MapViewer-servlet-path`* is the MapViewer servlet path (for example, `mapviewer/omserver`).
- *`xml-request`* is the URL-encoded XML request submitted using the HTML `GET` or `POST` method.

The input XML is required for all requests. The output depends on the content of the request: the response can be either an XML document, or a binary object containing the (generated image) file requested by the user.

In an input request, you must specify a data source, and you can specify one or more of the following:

- Themes and styles.
- A center point or a box for the map display, and options such as highlight, label, and styles.
- A predefined base map, which can be reused and overlaid with custom data.
- A custom theme with the user data points (or any geometry) retrieved dynamically and plotted directly from an accessible database.

- Custom features (point, circles, or any geometry) specified in the XML request string to be plotted. These require that you provide the dynamic data in the format of the `<geoFeature>` element (described in Section 3.2.5), as defined in the DTD. The geometry portion of the `<geoFeature>` element adopts the Geometry DTD as specified in Open GIS Consortium Geography Markup Language Version 1.0 (OGC GML v1.0).
- Thematic mapping.

You can manage the definition of base maps, themes, and styles (individual symbologies) using the Map Builder tool, which is described in Chapter 9.

For the current release, MapViewer accepts only a coordinate pair to identify the location for a map request; it cannot take a postal address as direct input for a map.

This chapter first presents some examples of map requests (see Section 3.1), and then presents detailed explanations of the following XML DTDs for requests and other operations:

- Map Request DTD
- Information Request DTD
- Map Response DTD
- MapViewer Exception DTD
- Geometry DTD (OGC)

3.1 Map Request Examples

This section provides examples of map requests. It refers to concepts, elements, and attributes that are explained in detail in Section 3.2. It contains sections with the following examples:

- Section 3.1.1, "Simple Map Request"
- Section 3.1.2, "Map Request with Dynamically Defined Theme"
- Section 3.1.3, "Map Request with Base Map, Center, and Additional Predefined Theme"
- Section 3.1.4, "Map Request with Center, Base Map, Dynamically Defined Theme, and Other Features"
- Section 3.1.5, "Map Request for Point Features with Attribute Value and Dynamically Defined Variable Marker Style"
- Section 3.1.6, "Map Request with an Image Theme"
- Section 3.1.7, "Map Request for Image of Map Legend Only"
- Section 3.1.8, "Map Request with SRID Different from Data SRID"
- Section 3.1.9, "Map Request Using a Pie Chart Theme"
- Section 3.1.15, "Java Program Using MapViewer"
- Section 3.1.16, "PL/SQL Program Using MapViewer"

3.1.1 Simple Map Request

Example 3–1 is a very simple map request. It requests a map consisting of a blank blue image (from the `mvdemo` data source) with the string *Hello World* drawn on top. (The

`datasource` attribute is required for a map request, even though this specific map request does not retrieve any map data from the data source.)

Example 3–1 Simple Map Request ("Hello World")

```
<?xml version="1.0" standalone="yes"?>
<map_request  title="Hello World" datasource = "mvdemo"/>
```

3.1.2 Map Request with Dynamically Defined Theme

Example 3–2 is a simple map request with one dynamically defined theme. It requests a map of all Oracle Spatial and Graph geometries from the COUNTIES table.

Example 3–2 Simple Map Request with a Dynamically Defined Theme

```
<?xml version="1.0" encoding="UTF-8" ?>
<map_request datasource="lbs_data">
  <themes>
    <theme name="t1">
      <jdbc_query spatial_column = "GEOM"
                  datasource = "lbs_data">
      SELECT geom FROM counties
      </jdbc_query>
    </theme>
  </themes>
</map_request>
```

3.1.3 Map Request with Base Map, Center, and Additional Predefined Theme

Example 3–3 requests a map with a specified center for the result map, and specifies a predefined theme (`poi_theme_us_restaurants`) to be rendered in addition to the predefined themes that are part of the base map (`basemap="us_base"`).

Example 3–3 Map Request with Base Map, Center, and Additional Predefined Theme

```
<?xml version="1.0" encoding="UTF-8" ?>
<map_request datasource="lbs_data" title="LBS CUSTOMER MAP"
      basemap="us_base" width="500" height="375"
      bgcolor="#a6cae0" format="GIF_URL">
   <center size="1">
      <geoFeature typeName="mapcenter" label="Motel 1" text_style="T.MOTEL"
         render_style="M.MOTEL" radius="300">
         <geometricProperty>
            <Point>
              <coordinates>-122.2615, 37.5266</coordinates>
            </Point>
         </geometricProperty>
      </geoFeature>
   </center>
   <srs>SDO:8265</srs>
   <themes>
      <theme name="poi_theme_us_restaurants"/>
   </themes>
</map_request>
```

Notes on Example 3–3:

- Because `basemap` is specified, MapViewer first draws all predefined themes for that base map before drawing the specified theme (`poi_theme_us_restaurants`).

- The center will be drawn with a marker of the `M.MOTEL` style and the label `Motel 1` in the `T.MOTEL` style.
- A circle with a radius of 300 meters will be drawn around the center.

3.1.4 Map Request with Center, Base Map, Dynamically Defined Theme, and Other Features

Example 3–4 requests a map with a specified center, a predefined theme named `theme_lbs_customers`, a dynamically defined theme named `sales_by_region`, and all base themes in the base map `us_base_road`, plus two features: a polygon representing the top sales region, and a point. The requested map will be stored at the MapViewer host and a URL to that GIF image (`format="GIF_URL"`) will be returned to the requester.

Example 3–4 Map Request with Center, Base Map, Dynamically Defined Theme, Other Features

```
<?xml version="1.0" encoding="UTF-8" ?>
<map_request datasource="lbs_data2" title="LBS CUSTOMER MAP 2"
      width="400" height="300" format="GIF_URL" basemap="us_base_road">
   <center size="1.5">
      <geoFeature typeName="nil">
         <geometricProperty>
            <Point>
              <coordinates>-122.2615, 37.5266</coordinates>
            </Point>
         </geometricProperty>
      </geoFeature>
   </center>
   <themes>
      <theme name="theme_lbs_customers"/>
      <theme name="sales_by_region">
         <jdbc_query spatial_column ="region"
                            label_column="manager"
                            render_style="V.SALES COLOR"
                            label_style="T.SMALL TEXT"
                            jdbc_host="data.my_corp.com"
                            jdbc_sid="orcl"
                            jdbc_port="1521"
                            jdbc_user="scott"
                            jdbc_password="password"
                            jdbc_mode="thin"
            > select region, sales, manager from my_corp_sales_2001
         </jdbc_query>
      </theme>
   </themes>
   <geoFeature typeName="nil" label="TopSalesRegion"
         text_style="9988" render_style="2837">
      <geometricProperty>
         <Polygon srsName="SDO:8265">
            <outerBoundaryIs>
               <LinearRing>
                  <coordinates>42.9,71.1 43.2,72.3 39.2,73.0 39.0,
                  73.1 42.9,71.1</coordinates>
               </LinearRing>
            </outerBoundaryIs>
         </Polygon>
      </geometricProperty>
```

```
    </geoFeature>
    <geoFeature render_style="1397" text_style="9987">
      <geometricProperty>
        <Point>
          <coordinates>-122.5615, 37.3266</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
</map_request>
```

In Example 3–4, sales_by_region is a dynamically defined theme. For information about dynamically defining a theme, see Section 3.2.20 and Section 3.2.9.

3.1.5 Map Request for Point Features with Attribute Value and Dynamically Defined Variable Marker Style

Example 3–5 shows a map request to render point features with a dynamically defined variable marker style. The attribute_values attribute defines the value that will be used to find the appropriate bucket (for the range into which the value falls), as defined in the variable marker style.

Example 3–5 Map Request for Point Features with Attribute Value and Dynamically Defined Variable Marker Style

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="Point Features with Variable Marker Style"
  datasource="mvdemo"
  srid="0"
  width="500"
  height="375"
  bgcolor="#a6caf0"
  antialiase="true"
  format="PNG_URL">
  <center size="19.2">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>-116.65,38.92</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
  <geoFeature
     render_style="varmarkerpf"
     attribute_values="50000.0">
    <geometricProperty>
      <Point>
        <coordinates>-112.0,43.0</coordinates>
      </Point>
    </geometricProperty>
  </geoFeature>
  <geoFeature
     render_style="varmarkerpf"
     attribute_values="125000.0">
    <geometricProperty>
      <Point>
        <coordinates>-123.0,40.0</coordinates>
      </Point>
```

```
    </geometricProperty>
  </geoFeature>
  <geoFeature
     render_style="varmarkerpf"
     attribute_values="200000.0">
    <geometricProperty>
      <Point>
        <coordinates>-116.64,38.92</coordinates>
      </Point>
    </geometricProperty>
  </geoFeature>
  <geoFeature
     render_style="varmarkerpf"
   attribute_values="300000.0">
    <geometricProperty>
      <Point>
        <coordinates>-112.0,35.0</coordinates>
      </Point>
    </geometricProperty>
  </geoFeature>
  <styles>
    <style name="varmarkerpf">
        <AdvancedStyle>
           <VariableMarkerStyle basemarker="mkcircle" startsize="10"
                                increment="5">
             <Buckets>
             <RangedBucket label="less than 100k" high="100000.0"/>
             <RangedBucket label="100k - 150k" low="100000.0" high="150000.0"/>
             <RangedBucket label="150k - 250k" low="150000.0" high="250000.0"/>
             <RangedBucket label="250k - 350k" low="250000.0" high="350000.0"/>
              </Buckets>
           </VariableMarkerStyle>
        </AdvancedStyle>
    </style>

    <style name="mkcircle">
      <svg>
        <g class="marker" style="stroke:blue;fill:red;">
          <circle r="20"/>
          </g>
        </svg>
    </style>

  </styles>
</map_request>
```

3.1.6 Map Request with an Image Theme

Example 3–6 requests a map in which an image theme is to be plotted underneath all other regular vector data. The image theme is specified in the <jdbc_image_query> element as part of the <theme> element in a map request. (For an explanation of image themes, see Section 2.3.3.)

Example 3–6   Map Request with an Image Theme

```
<?xml version="1.0" encoding="UTF-8" ?>
<map_request datasource="lbs_data" title="LBS Image MAP"
  basemap="us_roads" format="GIF_STREAM">
    <center size="1">
      <geoFeature>
```

```
        <geometricProperty>
          <Point>
            <coordinates>-122.2615, 37.5266</coordinates>
          </Point>
        </geometricProperty>
      </geoFeature>
    </center>
  <themes>
    <theme name="anImageTheme">
      <jdbc_image_query  image_format="ECW"
                         image_column="image"
                         image_mbr_column="img_extent"
                         jdbc_srid="33709"
                         datasource="lbs_data">
        SELECT image, img_extent, image_id FROM my_images
      </jdbc_image_query>
    </theme>
  </themes>
</map_request>
```

MapViewer processes the request in Example 3–6 as follows:

1. MapViewer retrieves the image data by executing the user-supplied query (`SELECT image, img_extent, image_id FROM my_images`) in the current map window context.
2. MapViewer checks its internal list of all registered image renderers to see if one supports the ECW format (`image_format="ECW"`). Because MapViewer as supplied by Oracle does not support the ECW format, you must implement and register a custom image renderer that supports the format, as explained in Appendix C.
3. MapViewer calls the `renderImages` method, and image data retrieved from the user-supplied query is passed to the method as one of its parameters.
4. MapViewer retrieves and renders any requested vector data on top of the rendered image.

3.1.7 Map Request for Image of Map Legend Only

Example 3–7 requests a map with just the image of the map legend, but without rendering any spatial data. In this example, the legend explains the symbology used for identifying cities, state boundaries, interstate highways, and county population density. (Map legends are explained in Section 3.2.11.)

Example 3–7 Map Request for Image of Map Legend Only

```
<?xml version="1.0" standalone="yes"?>
<map_request
             datasource = "mvdemo"
             format="PNG_URL">

  <legend bgstyle="fill:#ffffff;stroke:#ff0000" profile="MEDIUM" position="SOUTH_
EAST">
          <column>
            <entry text="Map Legend" is_title="true"/>
            <entry style="M.STAR" text="center point"/>
            <entry style="M.CITY HALL 3" text="cities"/>
            <entry is_separator="true"/>
            <entry style="C.ROSY BROWN STROKE" text="state boundary"/>
```

```
            <entry style="L.PH" text="interstate highway"/>
            <entry text="County population:"/>
            <entry style="V.COUNTY_POP_DENSITY" tab="1"/>
          </column>
  </legend>

</map_request>
```

Generating just the map legend image, as in Example 3–7, can save processing time if you display the stored map legend image on a web page separately from the actual displayed maps. This avoids the need to generate a legend each time there is a map request.

3.1.8 Map Request with SRID Different from Data SRID

Example 3–8 requests a map displayed in a coordinate system (`srid="32775"` for US - Equal Area Projection) that is different from the coordinate system associated with the county theme data (`jdbc_srid="8265"` for Longitude/Latitude - NAD 83). As a result, during the rendering process, MapViewer converts all geometries from the data SRID to the map request SRID.

If no coordinate system is associated with the theme data, MapViewer assumes that the data is associated with the coordinate system of the map request, and no conversion occurs.

Example 3–8 Map Request with SRID Different from Data SRID

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="US Counties: Equal-Area Projection (SRID=32775)"
  datasource="mvdemo"
  srid="32775"
  width="500"
  height="375"
  bgcolor="#a6caf0"
  antialiase="true"
  format="PNG_URL">
  <center size="4000000.0">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>-218191.9643,1830357.1429</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
  <themes>
    <theme name="county_th" user_clickable="false">
       <jdbc_query
         spatial_column="geom"
         render_style="C.COUNTIES"
         jdbc_srid="8265"
         datasource="mvdemo"
         asis="false">select geom from counties</jdbc_query>
     </theme>
  </themes>
</map_request>
```

3.1.9 Map Request Using a Pie Chart Theme

This section shows how to use thematic mapping with a pie chart theme. The result is a map in which each county contains a pie chart in which the size of each slice reflects the proportion of the population in a specified household income level category (low, medium, or high) in the county.

The basic steps are as follows.

1. Create an advanced style that defines the characteristics of the pie charts to be used. The following example creates an advanced style named V.PIECHART1.

```
INSERT INTO user_sdo_styles VALUES (
'V.PIECHART1', 'ADVANCED', null,
'<?xml version="1.0" ?>
<AdvancedStyle>
        <PieChartStyle pieradius="10">
                <PieSlice name="low"   color="#ff0000"/>
                <PieSlice name="medium"  color="#ffff00"/>
                <PieSlice name="high" color="#00ff00"/>
        </PieChartStyle>
</AdvancedStyle>',  null, null);
```

   When the style defined in the preceding example is applied to a geographic feature, a pie chart is created with three slices. The pieradius attribute specifies the size of each pie chart in pixels. Each slice (<PieSlice> element) has a color defined for it. The name attribute for each slice is ignored by MapViewer.

2. Create a new theme that uses the style that you created, as in the following example:

```
INSERT INTO user_sdo_themes VALUES (
'THEME_PIE_CHART', null, 'COUNTIES', 'GEOM',
'<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule column="INC_LOW,INC_MED,INC_HIGH">
    <features style="C.US MAP YELLOW">  </features>
    <label  column="''dummy''" style="V.PIECHART1"> 1 </label>
  </rule>
</styling_rules>');
```

   In the theme definition in the preceding example, the <label> element of the styling rule specifies style="V.PIECHART1", to indicate that this pie chart style (the style created in Step 1) is used to label each geometry displayed on the map.

   The column attribute (column="''dummy''" in this example) is required, even though it has no effect on the resulting map. The column attribute value can be dummy or any other string, and the value must be enclosed on both sides by two single quotation marks.

   Because the V.PIECHART1 style is defined with three slices, the preceding example must specify the names of three columns from the COUNTIES table, and these columns must have a numeric data type. The column names are INC_LOW, INC_MED, and INC_HIGH. These columns will supply the value that will be used to determine the size of each pie slice.

3. Issue a map request that uses the theme that you created. Example 3–9 requests a map that uses the THEME_PIE_CHART theme that was created in Step 2.

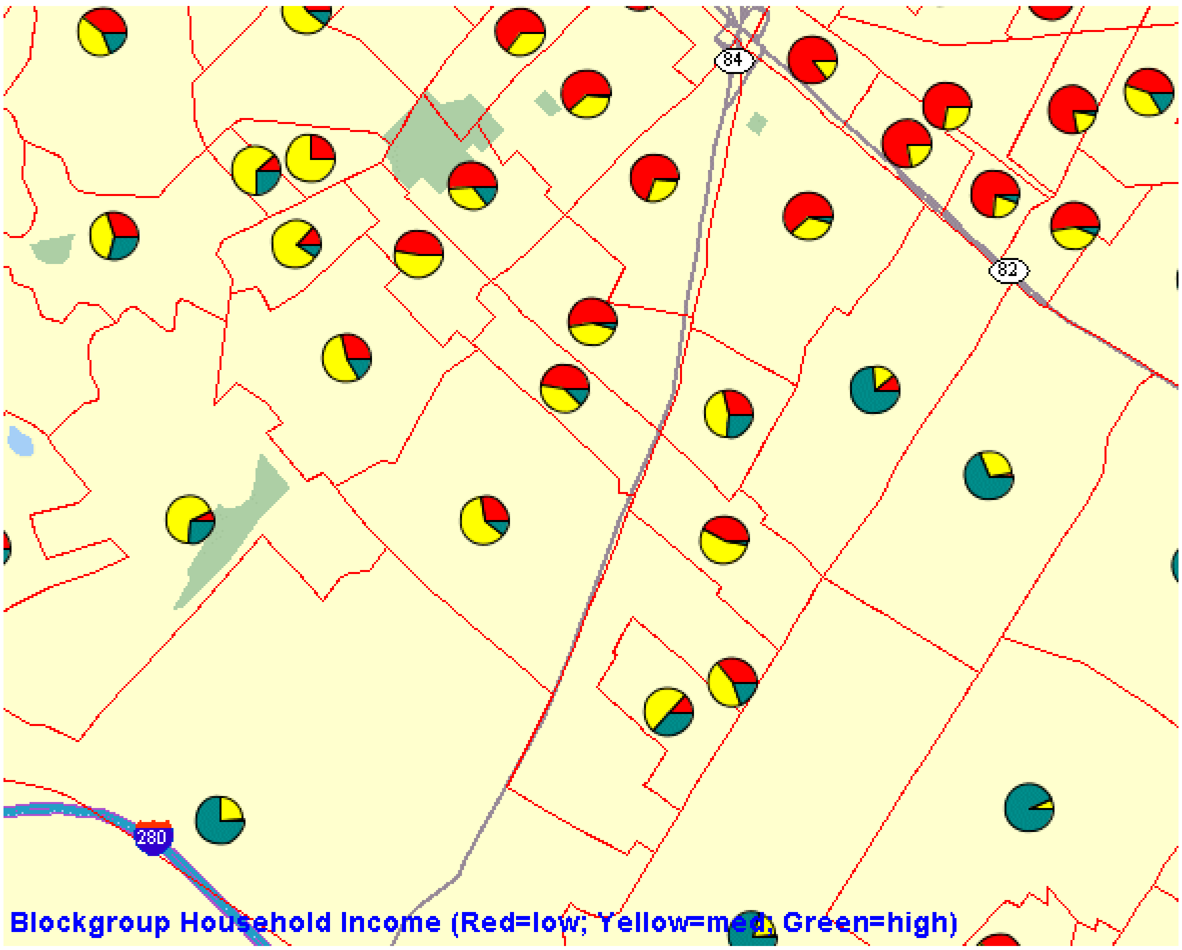
Example 3–9 Map Request Using a Pie Chart Theme

```
<?xml version="1.0" standalone="yes"?>
```

```
<map_request datasource = "mvdemo"
             format="PNG_STREAM">
  <themes>
    <theme name="THEME_PIE_CHART"/>
  </themes>
</map_request>
```

Figure 3–1 shows part of a display resulting from the map request in Example 3–9.

Figure 3–1 Map Display Using a Pie Chart Theme



You can also use the pie chart style in a dynamic (JDBC) theme when issuing a map request. You must specify the complete SQL query for a JDBC theme in the map request, because you must identify the attribute columns that are needed by the pie chart style. Any columns in the SELECT list that are not SDO_GEOMETRY columns or label columns are considered to be attribute columns that can be used by an advanced style.

Example 3–10 is a sample request with a JDBC theme using a pie chart style. The SQL query (`SELECT geom, 'dummy', sales, service, training FROM support_centers`) is included in the theme definition.

Example 3–10 JDBC Theme Using a Pie Chart Style

```
<?xml version="1.0" standalone="yes"?>
```

```
<map_request
             basemap="CA_MAP"
             datasource = "mvdemo"
             format="PNG_URL">
  <themes>
    <theme name="support_center">
     <jdbc_query spatial_column="geom" datasource="tilsmenv"
                                 label_column="dummy",
                                 label_style="V.PIECHART1">
                SELECT geom, 'dummy', sales, service, training
                FROM support_centers
        </jdbc_query>
    </theme>
  </themes>
</map_request>
```

3.1.10 Map Request Using Ratio Scale and Mixed Theme Scale Modes

Example 3–11 requests a map specifying a center and a ratio scale to define the map area. Two themes are used: a predefined theme named `THEME_US_COUNTIES1`, which uses the default screen inch scale mode, and a JDBC theme names `STATES_TH`, which uses the ratio mode.

Example 3–11 Map Request Using Ratio Scale and Mixed Theme Scale Modes

```
<?xml version="1.0" standalone="yes"?>
<map_request
title="States (ratio), counties (screen inch), center and scale"
datasource="tilsmenv"
width="500"
height="400"
bgcolor="#a6caf0"
antialiase="true"
format="PNG_URL"
>
<center scale="5000000">
  <geoFeature>
    <geometricProperty typeName="center">
      <Point>
        <coordinates>-90.0,32.0</coordinates>
      </Point>
    </geometricProperty>
  </geoFeature>
</center>
<themes>
  <theme name="STATES_TH" min_scale="5.0E7" max_scale="1.0E7" scale_mode="ratio">
     <jdbc_query
       label_column="STATE"
       spatial_column="geom"
       label_style="T.STATE NAME"
       render_style="C.COUNTIES"
       jdbc_srid="8265"
       datasource="tilsmenv"
       asis="false">select geom,state from states
     </jdbc_query>
   </theme>
  <theme name="THEME_US_COUNTIES1" min_scale="2.286" />
</themes>
</map_request>
```

3.1.11 Map Request Using Predefined Theme (Binding Parameter and Custom Type)

Example 3–12 requests a map using a predefined theme with a styling rule that selects all counties where a state abbreviation is in the selection list. When the predefined theme is created, the selection list is represented as a binding parameter, as follows:

```
INSERT INTO user_sdo_themes VALUES (
    'COUNTIES_BY_STATES', null, 'COUNTIES', 'GEOM',
    '<styling_rules>
    <rule>
        <features style="C.COUNTIES"> (state_abrv in (select column_value from
table(:1))) </features>
        <label column="COUNTY" style="T.CITY NAME"> 1 </label>
  </rule>
</styling_rules>');
```

This binding parameter can accept one or more values, for which you can create a custom SQL data type that represents this set of values, as follows:

```
CREATE OR REPLACE TYPE string_array AS TABLE OF VARCHAR2(64);
```

Then, you can use this custom data type on the binding parameter of the map request, as shown in Example 3–12.

Example 3–12 Map Request Using Predefined Theme (Binding Parameter and Custom Type)

```
<?xml version="1.0" standalone="yes"?>
<map_request
             title="Binding Parameters and STRING_ARRAY type"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="false"
             format="PNG_STREAM">

  <themes>
    <theme name="COUNTIES_BY_STATES" >
     <binding_parameters>
            <parameter value="FL,ME,CA,OH" type="STRING_ARRAY"/>
     </binding_parameters>
     </theme>
  </themes>

</map_request>
```

3.1.12 Map Request Using Advanced Styles and Rendering Rules

Example 3–13 requests a map using the <rendering> element, and it combines two advanced styles that are based on different columns. In this example, an advanced style named `POPVMK` is based on column POP90, and another advanced style named `EQRBRANK` is based on column RANK90. Point features (from the CITIES table) are rendered. The shape of the feature is defined by the advanced style associated with column POP90, and the feature color is defined by the advanced style associated with column RANK90.

Example 3–13 Map Request Using Advanced Styles and Rendering Rules

```
<?xml version="1.0" standalone="yes"?>
```

```
<map_request
  title="Cross advanced styles"
  datasource="mvdemo"
  width="640"
  height="480"
  bgcolor="#a6caf0"
  antialiase="false"
  format="PNG_STREAM"
>
  <center size="7.7">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>-72.96,41.25</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>

  <themes>
    <theme name="cities">
       <jdbc_query
         label_column="city"
         spatial_column="location"
         label_style="T.CITY NAME"
         jdbc_srid="8265"
         datasource="mvdemo"
         asis="false">select location,city,pop90,rank90 from cities
       </jdbc_query>
        <rendering>
           <style name="POPVMK" value_columns="POP90">
              <substyle name="EQRBRANK" value_columns="RANK90" changes="FILL_
COLOR"/>
           </style>
         </rendering>
     </theme>
  </themes>

  <styles>
   <style name="STAR_TRANSP">
<svg width="1in" height="1in">
    <desc/>
    <g class="marker"
style="stroke:#000000;fill:#FF0000;fill-opacity:0;width:15;height:15;font-family:D
ialog;font-size:12;font-fill:#FF0000">
        <polyline
points="138.0,123.0,161.0,198.0,100.0,152.0,38.0,198.0,61.0,123.0,0.0,76.0,76.0,76
.0,100.0,0.0,123.0,76.0,199.0,76.0"/>
  </g>
</svg>
   </style>

   <style name="POPVMK">
     <AdvancedStyle>
       <VariableMarkerStyle basemarker="STAR_TRANSP" startsize="7" increment="5">
        <Buckets>
            <RangedBucket seq="0" label="100217 - 1905803.75" low="100217"
high="1905803.75"/>
            <RangedBucket seq="1" label="1905803.75 - 3711390.5" low="1905803.75"
high="3711390.5"/>
```

```
            <RangedBucket seq="2" label="3711390.5 - 5516977.25" low="3711390.5"
high="5516977.25"/>
            <RangedBucket seq="3" label="5516977.25 - 7322564" low="5516977.25"
high="7322565"/>
        </Buckets>
       </VariableMarkerStyle>
     </AdvancedStyle>
   </style>

   <style name="EQRBRANK">
     <AdvancedStyle>
       <BucketStyle>
         <Buckets low="1" high="196" nbuckets="4" styles="C.RED,C.RB13_1,C.RB13_
6,C.SEQ6_01"/>
       </BucketStyle>
     </AdvancedStyle>
   </style>
  </styles>

  <legend bgstyle="fill:#ffffff;fill-opacity:50;stroke:#ff0000" profile="SMALL"
position="SOUTH_EAST">
          <column>
            <entry text="Map Legend" is_title="true" />
            <entry text="POP90:" />
            <entry style="POPVMK" tab="1" />
            <entry text="RANK90:" />
            <entry style="EQRBRANK" tab="1" />
          </column>
  </legend>
</map_request>
```

3.1.13 Map Request Using Stacked Styles

Example 3–14 requests a map using the <rendering> element, and it defines multiple styles (C.COUNTIES and PIECHART1) to be applied on each theme feature.

Example 3–14 Map Request Using Stacked Styles

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="Theme with Stacked Styles"
  datasource="mvdemo"
  width="600"
  height="450"
  bgcolor="#a6caf0"
  antialiase="true"
  format="PNG_STREAM"
>
  <center size="18">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>-122.729,40.423</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
  <themes>
    <theme name="STACKEDSTYLES">
       <jdbc_query
```

```
         label_column="state"
         spatial_column="geom"
         label_style="T.STATE NAME"
         jdbc_srid="8265"
         datasource="mvdemo"
         asis="false">select geom,state,HHI0_10,HHI10_15,HHI100UP,HHI15_25,HHI25_
35 from states
       </jdbc_query>
        <rendering>
           <style name="C.COUNTIES"/>
           <style name="PIECHART1" value_columns="HHI0_10,HHI10_15,HHI100UP,HHI15_
25,HHI25_35"/>
        </rendering>
    </theme>
  </themes>

  <styles>
    <style name="piechart1">
      <AdvancedStyle>
        <PieChartStyle pieradius="10">
           <PieSlice name="A" color="#FFFF00"/>
           <PieSlice name="B" color="#000000"/>
           <PieSlice name="H" color="#FF00FF"/>
           <PieSlice name="I" color="#0000FF"/>
           <PieSlice name="W" color="#FFFFFF"/>
       </PieChartStyle>
     </AdvancedStyle>
    </style>
  </styles>

</map_request>
```

3.1.14 WFS Map Requests

This section contains examples of WFS map requests, one using a predefined theme and one using a dynamic theme.

Example 3–15 requests a map using a predefined WFS theme named `BC_MUNICIPALITY`, which is defined as follows:

```
INSERT INTO user_sdo_themes  VALUES (
   'BC_MUNICIPALITY',
   'WFS theme',
   'BC_MUNICIPALITY',
   'THE_GEOM',
'<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="wfs" service_
url="http://www.refractions.net:8080/geoserver/wfs/GetCapabilities?"
srs="EPSG:3005">
    <rule>
        <features style="C.BLUE"> </features>
        <label column="name" style="T.CITY NAME"> 1 </label>
  </rule>
</styling_rules>');
```

Example 3–15 shows a map request that renders this predefined WFS theme.

Example 3–15 Map Request Using Predefined WFS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request
```

```
              title="Predefined WFS MAP"
              datasource = "mvdemo"
              width="640"
              height="480"
              bgcolor="#a6cae0"
              antialiase="true"
              format="PNG_STREAM">

  <center size="76000">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>1260500,470000</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>

  <themes>
   <theme name="bc_municipality" />
  </themes>

</map_request>
```

Example 3–16 shows a map request that uses a dynamic WFS theme.

Example 3–16 Map Request Using Dynamic WFS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request
              title="WFS MAP"
              datasource = "mvdemo"
              width="640"
              height="480"
              bgcolor="#a6cae0"
              antialiase="true"
              format="PNG_STREAM">

  <center size="76000">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>1260500,470000</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>

   <themes>
   <theme name="wfs" >
     <wfs_feature_request
         service_
url="http://www.refractions.net:8080/geoserver/wfs/GetCapabilities?"
         srs="EPSG:3005"
         feature_name="bc_hospitals"
         spatial_column="the_geom"
         render_style="M.STAR"
         label_column="name"
         label_style="T.CITY NAME"
         datasource="mvdemo" />
```

```
    </theme>
  </themes>

</map_request>
```

Example 3–17 shows a map request for a dynamic WFS theme with an advanced style to render features.

Example 3–17 Map Request Using Dynamic WFS Theme with an Advanced Style

```
<?xml version="1.0" standalone="yes"?>
<map_request
            title="WFS Theme with Advanced Style"
            datasource = "mvdemo"
            width="640"
            height="480"
            bgcolor="#a6cae0"
            antialiase="true"
            format="PNG_STREAM">
 <center size="10.">
    <geoFeature  >
        <geometricProperty typeName="center">
            <Point>
                <coordinates>-70., 44.</coordinates>
            </Point>
        </geometricProperty>
    </geoFeature>
 </center>

 <themes>
  <theme name="wfs" >
    <wfs_feature_request
        service_url="http://199.29.1.81:8181/miwfs/GetFeature.ashx?"
        srs="EPSG:4326"
        feature_name="usa"
        spatial_column="obj"
        render_style="CBSTATES"
        label_column="STATE_NAME"
        label_style="T.CITY NAME"
        feature_attributes="state"
        datasource="mvdemo" />
   </theme>
 </themes>

 <styles>
   <style name="CBSTATES">
     <AdvancedStyle>
      <BucketStyle>
       <Buckets default_style="C.COUNTIES">
           <CollectionBucket seq="0" type="string" style="C.RB13_
13">MA</CollectionBucket>
           <CollectionBucket seq="1" type="string" style="C.RB13_
1">NH</CollectionBucket>
           <CollectionBucket seq="2" type="string" style="C.RB13_
7">ME</CollectionBucket>
      </Buckets>
     </BucketStyle>
    </AdvancedStyle>
   </style>
 </styles>
```

```
</map_request>
```

3.1.15 Java Program Using MapViewer

Example 3–18 uses the `java.net` package to send an XML request to MapViewer and to receive the response from MapViewer. (Note, however, most programmers will find it more convenient to use the JavaBean-based API, described in Chapter 4, or the JSP tag library, described in Chapter 5.)

Example 3–18 Java Program That Interacts with MapViewer

```
import java.net.*;
import java.io.*;

/**
 * A sample program that shows how to interact with MapViewer
 */
public class MapViewerDemo
{
    private HttpURLConnection mapViewer = null;

    /**
     * Initializes this demo with the URL to the MapViewer server.
     * The URL is typically http://my_corp.com:8888/mapviewer/omserver.
     */
    public MapViewerDemo(String mapViewerURLString)
    {
        URL url;

        try
        {
            url = new URL(mapViewerURLString);
            mapViewer = (HttpURLConnection) url.openConnection();
            mapViewer.setDoOutput(true);
            mapViewer.setDoInput(true);
            mapViewer.setUseCaches(false);
        }
        catch (Exception e)
        {
            e.printStackTrace(System.err);
            System.exit(1);
        }
    }

    /**
     * Submits an XML request to MapViewer.
     * @param xmlreq   the XML document that is a MapViewer request
     */
    public void submitRequest(String xmlreq)
    {
        try
        {
            mapViewer.setRequestMethod("POST");  //Use HTTP POST method.
            OutputStream os = mapViewer.getOutputStream();
            //MapViewer expects to find the request as a parameter
            //named "xml_request".
            xmlreq = "xml_request="+URLEncoder.encode(xmlreq);
            os.write(xmlreq.getBytes());
            os.flush();
            os.close();
```

```
        }
        catch (Exception e)
        {
            e.printStackTrace(System.err);
            System.exit(1);
        }
    }

    /**
     * Receives an XML response from MapViewer.
     */
    public String getResponse()
    {
        ByteArrayOutputStream content = new ByteArrayOutputStream();
        InputStream is = null;
        try
        {
            is = mapViewer.getInputStream();
            int c;
            while ((c = is.read()) != -1)
              content.write(c);
            is.close();
            content.flush();
            content.close();
            return content.toString();
        }
        catch (Exception e)
        {
            e.printStackTrace(System.err);
            return null;
        }
    }

    // A simple main program that sends a list_data_sources XML
    // request to MapViewer through HTTP POST
    public static void main(String[] args)
    {
        if(args.length<1)
        {
          System.out.println("Usage: java  MapViewerDemo <mapviewer url>");
          System.out.println("Example: java MapViewerDemo http://my_
corp.com/mapviewer/omserver");
          System.exit(1);
        }

        // A sample XML request for MapViewer
        String
        listDataSources = "<?xml version=\"1.0\" standalone=\"yes\"?>" +
                          "  <non_map_request>" +
                          "    <list_data_sources/>" +
                          "  </non_map_request>";

        MapViewerDemo tester = null;
        tester = new MapViewerDemo(args[0]);
        System.out.println("submitting request:\n"+listDataSources);
        tester.submitRequest(listDataSources);
        String response = tester.getResponse();
        System.out.println("response from MapViewer: \n" + response);
    }
}
```

3.1.16 PL/SQL Program Using MapViewer

Example 3–19 is a sample PL/SQL program that sends an XML request to the MapViewer server.

Example 3–19   PL/SQL Program That Interacts with MapViewer

```
set serverout on size 1000000;

--
--  Author: Clarke Colombo
--
declare

    l_http_req    utl_http.req;
    l_http_resp   utl_http.resp;
    l_url         varchar2(4000):=  'http://my_corp.com:8888/mapviewer/omserver';


    l_value       varchar2(4000);
    img_url       varchar2(4000);
    response      sys.xmltype;

    output        varchar2(255);

    map_req       varchar2(4000);

begin

  utl_http.set_persistent_conn_support(TRUE);

  map_req := '<?xml version="1.0" standalone="yes"?>
  <map_request  title="MapViewer Demonstration"
                datasource="mvdemo"
                basemap="course_map"
                width="500"
                height="375"
                bgcolor="#a6cae0"
                antialiasing="false"
                format="GIF_URL">
    <center size="5">
      <geoFeature>
        <geometricProperty>
          <Point>
            <coordinates>-122.2615, 37.5266</coordinates>
          </Point>
        </geometricProperty>
      </geoFeature>
    </center>
  </map_request>';

  l_http_req := utl_http.begin_request(l_url, 'POST', 'HTTP/1.0');

  --
  -- Sets up proper HTTP headers.
  --
  utl_http.set_header(l_http_req, 'Content-Type',
'application/x-www-form-urlencoded');
  utl_http.set_header(l_http_req, 'Content-Length', length('xml_request=' || map_
req));
  utl_http.set_header(l_http_req, 'Host', 'my_corp.com');
```

```
  utl_http.set_header(l_http_req, 'Port', '8888');
  utl_http.write_text(l_http_req, 'xml_request=' || map_req);
  --
  l_http_resp := utl_http.get_response(l_http_req);

  utl_http.read_text(l_http_resp, l_value);

  response := sys.xmltype.createxml (l_value);

  utl_http.end_response(l_http_resp);

  img_url := response.extract('/map_response/map_image/map_
content/@url').getstringval();

  dbms_output.put_line(img_url);

end;
/
```

3.2 Map Request DTD

The following is the complete DTD for a map request, which is followed by reference sections that describe each element and its attributes.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <box> is defined in OGC GML v1.0 -->
<!ELEMENT map_request ((box | center | bounding_themes)?, srs?, legend?, themes?,
  styles?, scale_bar?, north_arrow?, geoFeature*)>
<!ATTLIST map_request
  datasource    CDATA #REQUIRED
  srid          CDATA #IMPLIED
  basemap       CDATA #IMPLIED
  width         CDATA #IMPLIED
  height        CDATA #IMPLIED
  antialiasing  (TRUE|FALSE) "FALSE"
  imagescaling  (TRUE|FALSE) "TRUE"
  format        (GIF|GIF_URL|GIF_STREAM|JAVA_IMAGE|
                 PNG_STREAM|PNG_URL|PNG8_STREAM|PNG8_URL|
                 JPEG_STREAM|JPEG_URL|PDF_STREAM|PDF_URL|
                 SVG_STREAM|SVGZ_STREAM|SVGTINY_STREAM|
                 SVG_URL|SVGZ_URL|SVGTINY_URL) "GIF_URL"
  transparent   (TRUE|FALSE) "FALSE"
  title         CDATA #IMPLIED
  bgcolor       (CDATA) "#A6CAF0"
  bgimage       CDATA #IMPLIED
  zoomlevels    CDATA #IMPLIED
  zoomfactor    CDATA #IMPLIED
  zoomratio     CDATA #IMPLIED
  initscale     CDATA #IMPLIED
  navbar        (TRUE|FALSE) "TRUE"
  infoon        (TRUE|FALSE) "TRUE"
  onclick       CDATA #IMPLIED
  onmousemove   CDATA #IMPLIED
  rasterbasemap (TRUE|FALSE) "FALSE"
  onrectselect  CDATA #IMPLIED
  onpolyselect  CDATA #IMPLIED
  use_cached_basemap (TRUE|FALSE) "FALSE"
  snap_to_cache_scale (TRUE|FALSE) "FALSE"
  title_style   CDATA #IMPLIED
  footnote      CDATA #IMPLIED
```

```
  footnote_style CDATA #IMPLIED
  rotation       CDATA #IMPLIED*
 >
<!ELEMENT center (geoFeature)>
<!ATTLIST center
    size CDATA #REQUIRED
>
<!ELEMENT box (coordinates) >
<!ATTLIST box
    ID                    CDATA #IMPLIED
    srsName               CDATA #REQUIRED
    preserve_aspect_ratio (TRUE|FALSE) "FALSE"
>
<!ELEMENT bounding_themes (#PCDATA) >
<!ATTLIST bounding_themes
    border_margin          CDATA  #IMPLIED
    preserve_aspect_ratio  CDATA  "TRUE"
    size_hint              CDATA  #IMPLIED
>
<!ELEMENT srs (#PCDATA) >

<!ELEMENT themes (theme+) >
<!ELEMENT theme (jdbc_query | jdbc_image_query | jdbc_georaster_query
                            | jdbc_network_query | jdbc_topology_query
                            | map_tile_theme
)? >
<!ATTLIST theme
    name                  CDATA #REQUIRED
    datasource            CDATA #IMPLIED
    max_scale             CDATA #IMPLIED
    min_scale             CDATA #IMPLIED
    label_always_on       (TRUE|FALSE) "FALSE"
    fast_unpickle         (TRUE|FALSE) "TRUE"
    mode                  CDATA #IMPLIED
    min_dist              CDATA #IMPLIED
    fixed_svglabel        (TRUE|FALSE) "FALSE"
    visible_in_svg        (TRUE|FALSE) "TRUE"
    selectable_in_svg     (TRUE|FALSE) "FALSE"
    part_of_basemap       (TRUE|FALSE) "FALSE"
    simplify_shapes       (TRUE|FALSE) "TRUE"
    onclick               CDATA #IMPLIED
    onmousemove           CDATA #IMPLIED
    onmouseover           CDATA #IMPLIED
    onmouseout            CDATA #IMPLIED
    workspace_name        CDATA #IMPLIED
    workspace_savepoint   CDATA #IMPLIED
    workspace_date        CDATA #IMPLIED
    workspace_date_format CDATA #IMPLIED
>
<!ELEMENT jdbc_query (#PCDATA, hidden_info?)>
<!ATTLIST jdbc_query
    asis               (TRUE|FALSE) "FALSE"
    spatial_column     CDATA #REQUIRED
    key_column         CDATA #IMPLIED
    label_column       CDATA #IMPLIED
    label_style        CDATA #IMPLIED
    render_style       CDATA #IMPLIED
    datasource         CDATA #IMPLIED
    jdbc_host          CDATA #IMPLIED
    jdbc_port          CDATA #IMPLIED
```

```
    jdbc_sid           CDATA #IMPLIED
    jdbc_user          CDATA #IMPLIED
    jdbc_password      CDATA #IMPLIED
    jdbc_srid          CDATA #IMPLIED
    jdbc_mode          (thin|oci8) "thin"
>
<!ELEMENT hidden_info (field+)>
<!ELEMENT field (#PCDATA)>
<!ATTLIST field
    column  CDATA #REQUIRED
    name    CDATA #IMPLIED
>
<!ELEMENT jdbc_image_query (#PCDATA) >
<!ATTLIST jdbc_image_query
    asis              (TRUE|FALSE) "FALSE"
    image_format      CDATA  #REQUIRED
    image_column      CDATA  #REQUIRED
    image_mbr_column  CDATA #REQUIRED
    image_resolution  CDATA #IMPLIED
    image_unit        CDATA #IMPLIED
    datasource        CDATA #IMPLIED
    jdbc_host         CDATA #IMPLIED
    jdbc_port         CDATA #IMPLIED
    jdbc_sid          CDATA #IMPLIED
    jdbc_user         CDATA #IMPLIED
    jdbc_password     CDATA #IMPLIED
    jdbc_srid         CDATA #IMPLIED
    jdbc_mode         (thin|oci8) "thin"
>
<!ELEMENT jdbc_georaster_query (#PCDATA) >
<!ATTLIST jdbc_georaster_query
    asis               (TRUE|FALSE) "FALSE"
    georaster_table    CDATA #REQUIRED
    georaster_column   CDATA #REQUIRED
    raster_id          CDATA #IMPLIED
    raster_table       CDATA #IMPLIED
    raster_pyramid     CDATA #IMPLIED
    raster_bands       CDATA #IMPLIED
    datasource         CDATA #IMPLIED
    polygon_mask       CDATA #IMPLIED
    transparent_nodata CDATA #IMPLIED
    jdbc_host          CDATA #IMPLIED
    jdbc_port          CDATA #IMPLIED
    jdbc_sid           CDATA #IMPLIED
    jdbc_user          CDATA #IMPLIED
    jdbc_password      CDATA #IMPLIED
    jdbc_srid          CDATA #IMPLIED
    jdbc_mode          (thin|oci8) "thin">
<!ELEMENT jdbc_network_query (#PCDATA) >
<!ATTLIST jdbc_network_query
    asis                    (TRUE|FALSE) "FALSE"
    network_name            CDATA #REQUIRED
    network_level           CDATA #IMPLIED
    link_style              CDATA #IMPLIED
    direction_style         CDATA #IMPLIED
    direction_position      CDATA #IMPLIED
    direction_markersize    CDATA #IMPLIED
    link_labelstyle         CDATA #IMPLIED
    link_labelcolumn        CDATA #IMPLIED
    node_style              CDATA #IMPLIED
```

```
    node_markersize         CDATA #IMPLIED
    node_labelstyle         CDATA #IMPLIED
    node_labelcolumn        CDATA #IMPLIED
    path_ids                CDATA #IMPLIED
    path_styles             CDATA #IMPLIED
    path_labelstyle         CDATA #IMPLIED
    path_labelcolumn        CDATA #IMPLIED
    analysis_algorithm      CDATA #IMPLIED
    shortestpath_style      CDATA #IMPLIED
    shortestpath_startnode  CDATA #IMPLIED
    shortestpath_endnode    CDATA #IMPLIED
    shortestpath_startstyle CDATA #IMPLIED
    shortestpath_endstyle   CDATA #IMPLIED
    withincost_startnode    CDATA #IMPLIED
    withincost_style        CDATA #IMPLIED
    withincost_cost         CDATA #IMPLIED
    withincost_startstyle   CDATA #IMPLIED
    datasource              CDATA #IMPLIED
    jdbc_host               CDATA #IMPLIED
    jdbc_port               CDATA #IMPLIED
    jdbc_sid                CDATA #IMPLIED
    jdbc_user               CDATA #IMPLIED
    jdbc_password           CDATA #IMPLIED
    jdbc_srid               CDATA #IMPLIED
    jdbc_mode               (thin|oci8) "thin"
>
<!ELEMENT jdbc_topology_query (#PCDATA)>
<!ATTLIST jdbc_topology_query
    asis               (TRUE|FALSE) "FALSE"
    topology_name      CDATA #REQUIRED
    feature_table      CDATA #REQUIRED
    spatial_column     CDATA #REQUIRED
    label_column       CDATA #IMPLIED
    label_style        CDATA #IMPLIED
    render_style       CDATA #IMPLIED
    datasource         CDATA #IMPLIED
    edge_style         CDATA #IMPLIED
    edge_marker_style  CDATA #IMPLIED
    edge_marker_size   CDATA #IMPLIED
    edge_label_style   CDATA #IMPLIED
    node_style         CDATA #IMPLIED
    node_label_style   CDATA #IMPLIED
    face_style         CDATA #IMPLIED
    face_label_style   CDATA #IMPLIED
    jdbc_host          CDATA #IMPLIED
    jdbc_port          CDATA #IMPLIED
    jdbc_sid           CDATA #IMPLIED
    jdbc_user          CDATA #IMPLIED
    jdbc_password      CDATA #IMPLIED
    jdbc_srid          CDATA #IMPLIED
    jdbc_mode          (thin|oci8) "thin"
>
<!ELEMENT map_tile_theme (#PCDATA)>
<!ATTLIST map_tile_theme
    map_tile_layer      CDATA # REQUIRED
    snap_to_tile_scale  (TRUE|FALSE) "FALSE"
>
<!ELEMENT geoFeature (description?, property*,
 geometricProperty)>
<!ATTLIST geoFeature
```

```
   typeName          CDATA #IMPLIED
   id                CDATA #IMPLIED
   render_style      CDATA #IMPLIED
   text_style        CDATA #IMPLIED
   label             CDATA #IMPLIED
   label_always_on   (TRUE|FALSE) "FALSE"
   marker_size       CDATA #IMPLIED
   radius            CDATA #IMPLIED
   attribute_values  CDATA #IMPLIED
   orient_x          CDATA #IMPLIED
   orient_y          CDATA #IMPLIED
   orient_z          CDATA #IMPLIED
   selectable_in_svg (TRUE|FALSE) "FALSE"
   onclick           CDATA #IMPLIED
   hidden_info       CDATA #IMPLIED
>
<!ELEMENT legend column+ >
<!ATTLIST legend
    bgstyle    CDATA #implied
    font       CDATA #implied
    location_x CDATA #implied
    location_y CDATA #implied
    offset_x   CDATA #implied
    offset_y   CDATA #implied
    profile   (MEDIUM|SMALL|LARGE)  "MEDIUM"
    position  (SOUTH_WEST|SOUTH_EAST|SOUTH|NORTH|
              NORTH_WEST|NORTH_EAST|EAST|WEST|CENTER)  "SOUTH_WEST"
>
<!ELEMENT column entry+ >
<!ATTLIST entry
    is_title      (true|false) "false"
    is_separator  (true|false) "false"
    tab           CDATA  "0"
    style         CDATA  #implied
    text          CDATA  #implied
>
<!ELEMENT scale_bar >
<!ATTLIST scale_bar
   mode              (METRIC_MODE|US_MODE|DUAL_MODES) "METRIC_MODE"
   position          (SOUTH_WEST|SOUTH_EAST|SOUTH|NORTH|
                      NORTH_WEST|NORTH_EAST)  "NORTH_EAST"
   offset_y          CDATA #implied
   offset_y          CDATA #implied
   color1            CDATA #implied
   color1_opacity    CDATA #implied
   color2            CDATA #implied
   color2_opacity    CDATA #implied
   length_hint       CDATA #implied
   label_color       CDATA #implied
   label_font_family CDATA #implied
   label_font_size   CDATA #implied
   label_halo_size   CDATA #implied
   label_position    (TOP|BOTTOM) "TOP"
>
<!ELEMENT styles (style+) >
<!ELEMENT style (svg | AdvancedStyle)?>
<!ATTLIST style
    name  CDATA #REQUIRED
>
<!ELEMENT north_arrow (style, location?, size?) >
```

The main elements and attributes of the map request DTD are explained in sections that follow. The <map_request> element is described in Section 3.2.1. The remaining related elements are described, in alphabetical order by element name, in the following sections:

- Section 3.2.2, "bounding_themes Element"
- Section 3.2.3, "box Element"
- Section 3.2.4, "center Element"
- Section 3.2.5, "geoFeature Element"
- Section 3.2.6, "jdbc_georaster_query Element"
- Section 3.2.7, "jdbc_image_query Element"
- Section 3.2.8, "jdbc_network_query Element"
- Section 3.2.9, "jdbc_query Element"
- Section 3.2.10, "jdbc_topology_query Element"
- Section 3.2.11, "legend Element"
- Section 3.2.12, "map_tile_theme Element"
- Section 3.2.13, "north_arrow Element"
- Section 3.2.14, "operation Element"
- Section 3.2.15, "operations Element"
- Section 3.2.16, "parameter Element"
- Section 3.2.17, "scale_bar Element"
- Section 3.2.18, "style Element"
- Section 3.2.19, "styles Element"
- Section 3.2.20, "theme Element"
- Section 3.2.21, "themes Element"

3.2.1 map_request Element

The <map_request> element has the following definition:

```
<!ELEMENT map_request ((box | center | bounding_themes)?, srs?, legend?, themes?,
  styles?, geoFeature*)>
<!ELEMENT map_request ((box | center | bounding_themes)?, srs?, legend?, themes?,
  styles?, geoFeature*, north_arrow?)>
```

The root element of a map request to MapViewer is always named map_request.

<map_request> can have a child element that is <box> (see Section 3.2.3), <center> (see Section 3.2.4), or <bounding_themes> (see Section 3.2.2), which specifies the range of the user data to be plotted on a map. If none of these child elements is specified, the result map is drawn using all data available to MapViewer.

The optional <srs> child element is ignored by the current version of MapViewer.

The optional <legend> element (see Section 3.2.11) is used to draw a legend (map inset illustration) on top of a generated map, to make the visual aspects of the map more meaningful to users.

The optional <themes> element (see Section 3.2.21) specifies predefined or dynamically defined themes.

The optional <styles> element (see Section 3.2.19) specifies dynamically defined styles.

The <geoFeature> element (see Section 3.2.5) can be used to specify any number of individual geometries and their rendering attributes.

The optional <north_arrow> element (see Section 3.2.13) is used to draw a north arrow marker based on the request rotation.

MapViewer first draws the themes defined in a base map (if a base map is specified as an attribute in the root element), then any user-provided themes, and finally any `geoFeature` elements.

3.2.1.1 map_request Attributes

The root element <map_request> has a number of attributes, some required and the others optional. The attributes are defined as follows:

```
<!ATTLIST map_request
  datasource     CDATA #REQUIRED
  srid           CDATA #IMPLIED
  basemap        CDATA #IMPLIED
  width          CDATA #IMPLIED
  height         CDATA #IMPLIED
  antialiasing   (TRUE|FALSE) "FALSE"
  imagescaling   (TRUE|FALSE) "TRUE"
  format         (GIF|GIF_URL|GIF_STREAM|JAVA_IMAGE|
                  PNG_STREAM|PNG_URL|PNG8_STREAM|PNG8_URL|
                  JPEG_STREAM|JPEG_URL|PDF_STREAM|PDF_URL|
                  SVG_STREAM|SVGZ_STREAM|SVGTINY_STREAM|
                  SVG_URL|SVGZ_URL|SVGTINY_URL) "GIF_URL"
  transparent    (TRUE|FALSE) "FALSE"
  title          CDATA #IMPLIED
  title_style    CDATA #IMPLIED
  footnote       CDATA #IMPLIED
  footnote_style CDATA #IMPLIED
  rotation       CDATA #IMPLIED*
  bgcolor        (CDATA) "#A6CAF0"
  bgimage        CDATA #IMPLIED
  zoomlevels     CDATA #IMPLIED
  zoomfactor     CDATA #IMPLIED
  zoomratio      CDATA #IMPLIED
  initscale      CDATA #IMPLIED
  navbar         (TRUE|FALSE) "TRUE"
  infoon         (TRUE|FALSE) "TRUE"
  onclick        CDATA #IMPLIED
  onmousemove    CDATA #IMPLIED
  rasterbasemap  (TRUE|FALSE) "FALSE"
  onrectselect   CDATA #IMPLIED
  onpolyselect   CDATA #IMPLIED
  keepthemesorder CDATE #IMPLIED
  use_cached_basemap  (TRUE|FALSE) "FALSE"
  snap_to_cache_scale (TRUE|FALSE) "FALSE"
  title_style    CDATA #IMPLIED
  footnote       CDATA #IMPLIED
  footnote_style CDATA #IMPLIED
  rotation       CDATA #IMPLIED*
>
```

datasource is a required attribute that specifies a data source. A data source provides information to MapViewer about where to fetch the user data (and the mapping metadata) that is required to render a map.

srid is an optional attribute. If it is specified, it provides the SRID value of the coordinate system (spatial reference system) for the map request. If necessary, theme geometries will be converted to the specified coordinate system before being rendered, although geometries with an undefined coordinate system will not be converted. If this attribute is not specified, MapViewer uses the coordinate system of the first theme to be rendered as the coordinate system for the map request.

basemap is an optional attribute. When it is specified, MapViewer renders all themes that are specified for this base map. The definition of a base map is stored in the user's USER_SDO_MAPS view, as described in Section 2.9.1. Use this attribute if you will always need a background map on which to plot your own themes and geometry features.

width and height are optional attributes that together specify the size (in device units) of the resulting map image. This size is different from the size specified in the center element or box element, which is the range of the window into a user's source data. The default width and height values are 500 and 375 pixels, respectively. The unit is in pixels except for PDF formats, in which case pt is used as the unit, and the relationship with pixels is approximately 1 pt = 1.333 px (or, 1px = 0.75 pt). Thus, for example, if you request a map with size 500x375 "pt" in PDF format, this should generate an image of approximately 667x500 pixels.

antialiasing is an optional attribute. When its value is TRUE, MapViewer renders the map image in an antialiased manner. This usually provides a map with better graphic quality, but it may take longer for the map to be generated. The default value is FALSE (for faster map generation). (For backward compatibility, antialiase is a synonym for antialiasing, but you are encouraged to use antialiasing.)

imagescaling is an optional attribute. When its value is TRUE (the default), MapViewer attempts to scale the images to fit the current querying window and the generated map image size. When its value is FALSE, and if an image theme is included directly or indirectly (such as through a base map), the images from the image theme are displayed in their original resolution. This attribute has no effect when no image theme is involved in a map request.

format is an optional attribute that specifies the file format of the returned map image. The default value is GIF_URL, which is a URL to a GIF image stored on the MapViewer host system.

- If you specify GIF, the generated GIF image data is embedded in a MapResponse object and returned to the client. If you specify GIF_STREAM, the generated image map content is returned directly to the client through the HTTP MIME type image/gif.
- If you specify JAVA_IMAGE, a Java 2D BufferedImage object with a color model of TYPE_INT_RGB is embedded in a MapResponse object and returned to the client.
- If you specify PNG_STREAM, the stream of the image in nonindexed PNG format is returned directly; if you specify PNG_URL, a URL to a nonindexed PNG image stored on the MapViewer host system is returned. (The PNG image format has some advantages over the GIF format, including faster image encoding and true color support.)
- If you specify PNG8_STREAM, the stream of the image in indexed PNG format is returned directly; if you specify PNG8_URL, a URL to an indexed PNG image

stored on the MapViewer host system is returned. (The PNG image format has some advantages over the GIF format, including faster image encoding and true color support. The indexed PNG format limits the total number of colors available for displaying the map to 256.)

- If you specify `JPEG_STREAM`, the stream of the image in JPEG format is returned directly; if you specify `JPEG_URL`, a URL to a JPEG image stored on the MapViewer host system is returned.
- If you specify `PDF_STREAM`, the stream of the image in PDF document format is returned directly; if you specify `PDF_URL`, a URL to a PDF document stored on the MapViewer host system is returned.
- If you specify `SVG_STREAM`, the stream of the image in SVG Basic (SVGB) format is returned directly; if you specify `SVG_URL`, a URL to an SVG Basic image stored on the MapViewer host system is returned.
- If you specify `SVGZ_STREAM`, the stream of the image in SVG Compressed (SVGZ) format is returned directly; if you specify `SVGZ_URL`, a URL to an SVG Compressed image stored on the MapViewer host system is returned. SVG Compressed format can effectively reduce the size of the SVG map by 40 to 70 percent compared with SVG Basic format, thus providing better performance.
- If you specify `SVGTINY_STREAM`, the stream of the image in SVG Tiny (SVGT) format is returned directly; if you specify `SVGTINY_URL`, a URL to an SVG Tiny image stored on the MapViewer host system is returned. (The SVG Tiny format is designed for devices with limited display capabilities, such as cell phones.)

`transparent` is an optional attribute that applies to indexed PNG (`PNG8_STREAM` or `PNG8_URL`) formats only. When its value is `TRUE`, MapViewer makes the map background color completely transparent. The default value is `FALSE`.

`title` is an optional attribute that specifies the map title to be displayed on the top of the resulting map image.

`title_style` is an optional attribute that specifies the name of the text style to be used when rendering the title.

`footnote` is an optional attribute that specifies the footnote text to be added on the final map.

`footnote_style` is an optional attribute that specifies the name of the text style to be used when rendering the footnote.

`bgcolor` is an optional attribute that specifies the background color in the resulting map image. The default is water-blue (RGB value #A6CAF0). It must be specified as a hexadecimal value.

`bgimage` is an optional attribute that specifies the background image (GIF or JPEG format only) in the resulting map image. The image is retrieved at runtime when a map request is being processed, and it is rendered before any other map features, except that any `bgcolor` value is rendered before the background image.

`zoomlevels` is an optional attribute that specifies the number of zoom levels for an SVG map. The default is 4.

`zoomfactor` is an optional attribute that specifies the zoom factor for an SVG map. The zoom factor is the number by which to multiply the current zoom ratio for each integer increment (a zoomin operation) in the zoom level. The inverse of the `zoomfactor` value is used for each integer decrement (a zoomout operation) in the zoom level. For example, if the `zoomfactor` value is 2 (the default), zooming in from zoom level 4 to 5 will enlarge the detail by two; for example, if 1 inch of the map at

zoom level 4 represents 10 miles, 1 inch of the map at zoom level 5 will represent 5 miles. The zoom ratio refers to the relative scale of the SVG map, which in its original size (zoom level 0) has a zoom ratio of 1.

`zoomratio` is an optional attribute that specifies the zoom ratio when an SVG map is initially displayed. The default value is 1, which is the original map size (zoom level 0). Higher zoom ratio values show the map zoomed in, and lower values show the map zoomed out.

`initscale` is an optional attribute that specifies the initial scale when an SVG map is first displayed. The default value is 1, which is the original map size (zoom level 0). Higher values will show the SVG map zoomed in when it is first displayed.

`navbar` is an optional attribute that specifies whether to display the built-in navigation bar on an SVG map. If its value is `TRUE` (the default), the navigation bar is displayed; if it is set to `FALSE`, the navigation bar is not displayed.

`infoon` is an optional attribute that specifies whether to display hidden information when the mouse moves over features for which hidden information is provided. If its value is `TRUE` (the default), hidden information is displayed when the mouse moves over such features; if it is set to `FALSE`, hidden information is not displayed when the mouse moves over such features. Regardless of the value, however, hidden information is always rendered in an SVG map; this attribute only controls whether hidden information can be displayed. (To specify the hidden information for a feature, use the `hidden_info` attribute in the `<geoFeature>` element, as explained in Section 3.2.5.)

`onclick` is an optional attribute that specifies the name of the JavaScript function to be called when a user clicks on an SVG map. The JavaScript function must be defined in the HTML document outside the SVG definition. This function must accept two parameters: x and y, which specify the coordinates inside the SVG window where the click occurred. The coordinates are defined in the local SVG window coordinate system, which starts at (0, 0) at the upper-left corner and ends at (*width*, *height*) at the lower-right corner. For information about using JavaScript functions with SVG maps, see Appendix B.

`onmousemove` is an optional attribute that specifies the name of the JavaScript function to be called when a user moves the mouse on an SVG map. The JavaScript function must be defined in the HTML document outside the SVG definition. This function must accept two parameters: x and y, which specify the coordinates inside the SVG window where the move occurred. The coordinates are defined in the local SVG window coordinate system, which starts at (0, 0) at the upper-left corner and ends at (*width*, *height*) at the lower-right corner. For information about using JavaScript functions with SVG maps, see Appendix B.

`rasterbasemap` is an optional attribute. If the map format is SVG and the value of this attribute is `TRUE`, MapViewer renders the base map as a raster image. In this case, the base map image becomes the background image for the SVG map, and all other vector features are rendered on top of it.

`onrectselect` is an optional attribute that specifies the name of the JavaScript function to be called when a user draws a rectangular selection area by clicking and dragging the mouse (to indicate two diagonally opposite corners) on an SVG map. The JavaScript function must be defined in the HTML document outside the SVG definition. This function must not accept any parameters. For information about using JavaScript functions with SVG maps, see Appendix B.

`onpolyselect` is an optional attribute that specifies the name of the JavaScript function to be called when a user draws a polygon-shaped selection area by clicking and dragging the mouse (to indicate more than two vertices) on an SVG map. The

JavaScript function must be defined in the HTML document outside the SVG definition. This function must not accept any parameters. For information about using JavaScript functions with SVG maps, see Appendix B.

`keepthemesorder` is an optional attribute. If the map format is not SVG and the value of this attribute is `TRUE`, MapViewer always renders the themes in the order specified in the map request; if the value of this attribute is `FALSE`, raster themes will be rendered before vector themes.

`use_cached_basemap` is an optional attribute. If the value of this attribute is `TRUE` and if a map tile layer caches the same base map specified by the `basemap` attribute, MapViewer tries to use the map images cached by the map tile server to render the map specified by the map request. For information about the map tile server, see Section 8.2.

`snap_to_cache_scale` is an optional attribute that is effective only when the `use_cached_basemap` attribute value is `TRUE`. It affects the behavior of MapViewer only when the map scale specified by the map request does not match that of any predefined cached zoom level. If this attribute is `FALSE`, MapViewer uses the cached map images to render the base map only when the map scale specified by the map request matches the scale of a cached predefined zoom level. If this attribute is `TRUE`, MapViewer always uses the cached map images to render the base map and adjusts the map scale to fit that of a cached predefined zoom level when the request map scale does not match any of the cached predefined zoom levels.

`title_style` is an optional attribute that defines the text style to be used for the title.

`footnote` is an optional attribute that defines the text for a footnote to be added to the map.

`footnote_style` is an optional attribute that defines the text style to be used for the footnote text.

`rotation` is an optional attribute defined in degrees to apply a rotation on the map. Positive values means counterclockwise rotation of the map. Rotation values are ignored if the request does not have a window defined (no center and size defined, or using bounding themes). Rotation is not supported for requests using base maps coming from the Oracle Maps cache.

3.2.2 bounding_themes Element

The <bounding_themes> element has the following definition:

```
<!ELEMENT bounding_themes (#PCDATA) >
<!ATTLIST bounding_themes
     border_margin          CDATA  #IMPLIED
     preserve_aspect_ratio  CDATA  "TRUE"
     size_hint              CDATA  #IMPLIED
>
```

You can specify one or more themes as the bounding themes when you cannot predetermine the data size for a map. For example, you may have one dynamic theme that selects all data points that meet certain criteria, and you then want to plot those data points on a map that is just big enough to enclose all the selected data points. In such cases, you can use the <bounding_themes> element to specify the names of such dynamic themes. MapViewer first processes any themes that are specified in the <bounding_themes> element, generates a bounding box based on the resulting features of the bounding themes, and then prepares other themes according to the new bounding box.

The <bounding_themes> element is ignored if you specify the <box> or <center> element in the map request.

border_margin is an optional attribute that specifies the percentage to be added to each margin of the generated bounding box. For example, if you specify a value of 0.025, MapViewer adds 2.5% of the width to the left and right margins of the generated bounding box (resulting in a total 5% width expansion in the x-axis); similarly, 2.5% of the height is added to the top and bottom margins. The default value is 0.05, or 5% to be added to each margin.

preserve_aspect_ratio is an optional attribute that indicates whether or not the bounding box generated after processing the bounding themes should be further modified so that it has the same aspect ratio as the map image or device. The default is TRUE, which modifies the bounding box to preserve the aspect ratio, so as not to distort the resulting map image.

size_hint is an optional attribute that specifies the vertical span of the map in terms of the original data unit. For example, if the user's data is in decimal degrees, the size_hint attribute specifies the number of decimal degrees in latitude. If the user's data is projected with meter as its unit, MapViewer interprets size_hint in meters.

The size_hint attribute can be used to extend the boundary limit. This is useful when the bounding theme has just one point feature. For example, the bounding theme can be a point resulting from a geocoding query, and you want to place this point in the middle of the map and extend the boundary from that point.

The element itself contains a comma-delimited list of names of the bounding themes. The theme names must exactly match their names in the map request or the base map used in the map request. The following example shows a map request with two bounding themes, named theme1 and theme3, and with 2.3 percent (border_margin="0.023") added to all four margins of the minimum bounding box needed to hold features associated with the two themes:

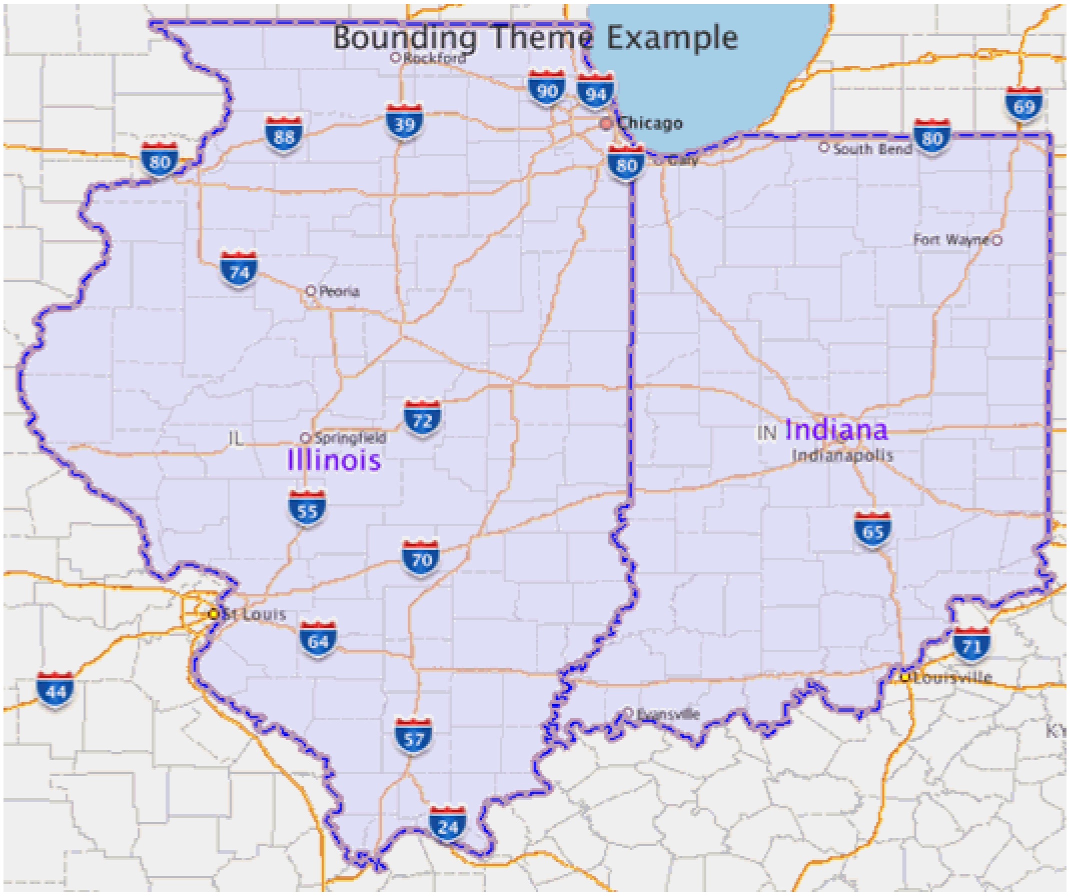
```
<?xml version="1.0" standalone="yes"?>
<map_request
            title="Bounding Theme Example"
            title_style="titleText"
            datasource = "mvdemo"
            basemap="demo_map"
            width="600"
            height="500"
            bgcolor="#a6cae0"
            antialiase="false"
            mapfilename="tilsmq202"
            format="PNG_STREAM">

  <bounding_themes border_margin="0.023">theme1, theme3</bounding_themes>
  <themes>
    <theme name="theme1" min_scale="5.0E7" max_scale="0.0">
       <jdbc_query
         datasource="mvdemo"
         jdbc_srid="8307"
         spatial_column="geom" label_column="STATE"
         render_style="myPattern" label_style="myText"
         >SELECT geom, state from states where state_abrv='IL'</jdbc_query>
     </theme>
    <theme name="theme3" min_scale="5.0E7" max_scale="0.0">
       <jdbc_query
         datasource="mvdemo"
         jdbc_srid="8307"
         spatial_column="geom" label_column="STATE"
```

```
         render_style="myPattern" label_style="myText"
         >SELECT geom,state from states where state_abrv='IN'</jdbc_query>
     </theme>
  </themes>
  <styles>
   <style name="myPattern">
     <svg width="1in" height="1in">
     <desc></desc>
       <g class="area"
          style="stroke:#6666e0;fill:#d6ccff;fill-opacity:128;line-style:L.STATE
BOUNDARY">
       </g>
     </svg>
   </style>
   <style name="myText">
     <svg width="1in" height="1in">
        <g class="text" float-width="1.0"
           style="font-style:bold;font-family:Arial;font-size:16pt;fill:#6600ff">
          Hello World!
        </g>
     </svg>
   </style>
   <style name="titleText">
     <svg width="1in" height="1in">
        <g class="text" float-width="1.0"

style="font-style:bold;font-family:Helvetica;font-size:18pt;fill:#333333">
          Hello World!
        </g>
     </svg>
   </style>
  </styles>
</map_request>
```

The preceding example displays a map in which the states of Illinois and Indiana are displayed according to the specifications in the two `<theme>` elements, both of which specify a rendering style named `myPattern`. In the `myText` style, the text "Hello World!" is displayed only when the style is being previewed in a style creation tool, such as the Map Builder tool. When the style is applied to a map, it is supplied with an actual text label that MapViewer obtains from a theme.

Figure 3–2 shows the display from the preceding example.

Figure 3–2 Bounding Themes



3.2.3 box Element

The <box> element has the following definition:

```
<!ELEMENT box (coordinates) >
<!ATTLIST box
    ID                    CDATA #IMPLIED
    srsName               CDATA #REQUIRED
    preserve_aspect_ratio (TRUE|FALSE)  "FALSE"
>
```

The <box> element is used to specify the bounding box of a resulting map. It uses a <coordinates> element to specify two coordinate value pairs that identify the lower-left and upper-right corners of the rectangle. The coordinate values are interpreted in terms of the user's data. For example, if the user's data is geodetic and is specified in decimal degrees of longitude and latitude, a <coordinates> specification of `-72.84, 41.67, -70.88, 42.70` indicates a bounding box with the lower-left corner at longitude-latitude coordinates (-72.84, 41.67) and the upper-right corner at coordinates (-70.88, 42.70), which are in the New England region of the United States. However, if the data is projected with meter as its unit of measurement, the coordinate values are interpreted in meters.

`preserve_aspect_ratio` is an optional attribute that indicates whether or not the box coordinates should be further modified so that it has the same aspect ratio as the map image or device. The default is `FALSE`, in order to keep compatibility with

previous versions that do not have this attribute. If this value is set to TRUE, the box is modified to preserve the aspect ratio, so as not to distort the resulting map image.

3.2.4 center Element

The <center> element has the following definition:

```
<!ELEMENT center (geoFeature)>
<!ATTLIST center
   size CDATA #REQUIRED
>
```

The <center> element is used to specify the center of a resulting map. It has a required attribute named size, which specifies the vertical span of the map in terms of the original data unit. For example, if the user's data is in decimal degrees, the size attribute specifies the number of decimal degrees in latitude. If the user's data is projected with meter as its unit, MapViewer interprets the size in meters.

The center itself must embed a <geoFeature> element, which is specified in Section 3.2.5.

3.2.5 geoFeature Element

The <geoFeature> element has the following definition:

```
<!ELEMENT geoFeature (description?, property*,
 geometricProperty)>
<!ATTLIST geoFeature
   typeName         CDATA #IMPLIED
   id               CDATA #IMPLIED
   render_style     CDATA #IMPLIED
   text_style       CDATA #IMPLIED
   label            CDATA #IMPLIED
   label_always_on  (TRUE|FALSE) "FALSE"
   marker_size      CDATA #IMPLIED
   radius           CDATA #IMPLIED
   attribute_values CDATA #IMPLIED
   orient_x         CDATA #IMPLIED
   orient_y         CDATA #IMPLIED
   orient_z         CDATA #IMPLIED
   selectable_in_svg (TRUE|FALSE) "FALSE"
   onclick          CDATA #IMPLIED
   hidden_info      CDATA #IMPLIED
>
```

<geoFeature> elements are used to provide individual geospatial entities to be rendered on a map. The main part of a <geoFeature> element is the geometry (<geometricProperty> element), which must be supplied in compliance with the OGC GML v1.0 Geometry DTD (described in Section 3.6).

typeName is an optional attribute that is ignored by the current release of MapViewer.

id is an optional attribute that can be used to uniquely identify the feature among all the geospatial features on the SVG map. (See the explanation of the selectable_in_svg attribute.) Otherwise, this attribute is ignored by MapViewer.

render_style is an optional attribute. When it is omitted, the geoFeature is not rendered. If it is supplied, its value must be the name of a style stored in the user's USER_SDO_STYLES view.

text_style is an optional attribute. If it is supplied (and if the render_style and label attributes are present and valid), it identifies the style to be used in labeling the feature. If it is not specified, a default text style is used.

label is an optional attribute. If it is supplied (and if the render_style and label attributes are present and valid), it identifies text that is used to label the feature.

label_always_on is an optional attribute. If it is set to TRUE, MapViewer labels the features even if two or more labels will overlap in the display of a theme. (MapViewer always tries to avoid overlapping labels.) If label_always_on is FALSE (the default), when it is impossible to avoid overlapping labels, MapViewer disables the display of one or more labels so that no overlapping occurs. The label_always_on attribute can also be specified for a theme (theme element, described in Section 3.2.20). Specifying label_always_on as TRUE for a feature in the geoFeature element definition gives you control over which features will have their labels displayed if label_always_on is FALSE for a theme and if overlapping labels cannot be avoided.

marker_size is an optional attribute. If it is supplied with a point feature, and if render_style is a marker-type style, the specified size is used by MapViewer in rendering this feature. This provides a mechanism to override the default value specified for a marker style.

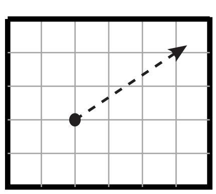
radius is an optional attribute. If it is supplied, it specifies a number or a comma-delimited list of numbers, with each number representing the radius of a circle to be drawn centered on this feature. For geodetic data, the unit is meters; for non-geodetic data, the unit is the unit of measurement associated with the data.

attribute_values is an optional attribute. If it is supplied, it specifies a value or a comma-delimited list of values to be used with bucket ranges of an advanced style (for example, values for pie chart segments or bucket values for variable markers).

orient_x and orient_y optionally specify a virtual end point to indicate an orientation vector for rotating a marker symbol (such as a shield symbol to indicate a highway) or text at a specified point. (orient_z is reserved for future use by Oracle.) The value for each must be from -1 to 1. The orientation start point is assumed to be (0,0), and it is translated to the location of the physical point to which it corresponds.

Figure 3–3 illustrates an orientation vector of approximately 34 degrees (counterclockwise from the x-axis), resulting from specifying orient_x="0.3" orient_y="0.2". (To have an orientation that more precisely matches a specific angle, refer to the cotangent or tangent values in the tables in a trigonometry textbook.)

Figure 3–3 Orientation Vector



selectable_in_svg is an optional attribute that specifies whether or not the feature is selectable on an SVG map. The default is FALSE; that is, the feature is not selectable on an SVG map. If this attribute is set to TRUE and if theme feature selection is allowed, the feature can be selected by clicking on it. If the feature is selected, its color is changed and its ID is recorded. You can get a list of the ID values of all selected features by calling the JavaScript function getSelectedIdList() defined in the

SVG map. (For feature selection to work correctly, the id attribute value of the feature must be set to a value that uniquely identifies it among all the geospatial features on the SVG map.) For information about using JavaScript functions with SVG maps, see Appendix B.

onclick is an optional attribute that specifies the name of the JavaScript function to be called when a user clicks on the feature. The JavaScript function must be defined in the HTML document outside the SVG definition. This function must accept only four parameters: the theme name, the key of the feature, and x and y, which specify the coordinates (in pixels) of the clicked point on the SVG map. For information about using JavaScript functions with SVG maps, see Appendix B.

hidden_info is an optional attribute that specifies an informational note or tip to be displayed when the mouse is moved over the feature. To specify multiple lines, use "\n" between lines. For example, hidden_info="State park with\nhistorical attractions" specifies a two-line tip. (To enable the display of hidden information in the map, you must specify infoon="true" in the <map_request> element, as explained in Section 3.2.1.1.)

The following example shows a <geoFeature> element specification for a restaurant at longitude and latitude coordinates (-78.1234, 41.0346). In this case, the feature will be invisible because the render_style and text_style attributes are not specified.

```
<geoFeature typeName="Customer" label="PizzaHut in Nashua">
   <geometricProperty>
      <Point srsName="SDO:8265">
         <coordinates>-78.1234,41.0346</coordinates>
      </Point>
   </geometricProperty>
</geoFeature>
```

The following example shows a <geoFeature> element specification for a point of interest at longitude and latitude coordinates (-122.2615, 37.5266). The feature will be rendered on the generated map because the render_style attribute is specified. The example specifies some label text (A Place) and a text style for drawing the label text. It also instructs MapViewer to draw two circles, centered on this feature, with radii of 1600 and 4800 meters. (In this case, the srsName attribute of the <Point> element must be present, and it must specify an Oracle Spatial and Graph SRID value using the format "SDO:*<srid>*". Because SRID value 8265 is associated with a geodetic coordinate system, the radius values are interpreted as 1600 and 4800 meters.)

```
<geoFeature render_style="m.star"
            radius="1600,4800"
            label="A Place"
            text_style="T.Name">
    <geometricProperty>
        <Point srsName="SDO:8265">
            <coordinates>-122.2615, 37.5266</coordinates>
        </Point>
    </geometricProperty>
</geoFeature>
```

Figure 3–4 is a map drawn using the <geoFeature> element in the preceding example. The feature is labeled with the text A Place, and it is represented by a red star marker surrounded by two concentric circles.

Figure 3–4 Map with <geoFeature> Element Showing Two Concentric Circles

3.2.6 jdbc_georaster_query Element

The `<jdbc_georaster_query>` element, which is used to define a GeoRaster theme, has the following definition:

```
<!ELEMENT jdbc_georaster_query (#PCDATA) >
<!ATTLIST jdbc_georaster_query
    asis                (TRUE|FALSE) "FALSE"
    georaster_table     CDATA #REQUIRED
    georaster_column    CDATA #REQUIRED
    raster_id           CDATA #IMPLIED
    raster_table        CDATA #IMPLIED
    raster_pyramid      CDATA #IMPLIED
    raster_bands        CDATA #IMPLIED
    datasource          CDATA #IMPLIED
    polygon_mask        CDATA #IMPLIED
    transparent_nodata  CDATA #IMPLIED
    jdbc_host           CDATA #IMPLIED
    jdbc_port           CDATA #IMPLIED
    jdbc_sid            CDATA #IMPLIED
    jdbc_user           CDATA #IMPLIED
    jdbc_password       CDATA #IMPLIED
    jdbc_srid           CDATA #IMPLIED
    jdbc_mode           (thin|oci8) "thin"
>
```

For detailed usage and reference information about GeoRaster themes, see Section 2.3.4.

3.2.7 jdbc_image_query Element

The `<jdbc_image_query>` element, which is used to define an image theme (described in Section 2.3.3), has the following definition:

```
<!ELEMENT jdbc_image_query (#PCDATA) >
<!ATTLIST jdbc_image_query
    asis               (TRUE|FALSE) "FALSE"
    image_format       CDATA #REQUIRED
```

```
    image_column      CDATA #REQUIRED
    image_mbr_column  CDATA #REQUIRED
    image_resolution  CDATA #IMPLIED
    image_unit        CDATA #IMPLIED
    datasource        CDATA #IMPLIED
    jdbc_host         CDATA #IMPLIED
    jdbc_port         CDATA #IMPLIED
    jdbc_sid          CDATA #IMPLIED
    jdbc_user         CDATA #IMPLIED
    jdbc_password     CDATA #IMPLIED
    jdbc_srid         CDATA #IMPLIED
    jdbc_mode         (thin|oci8) "thin"
>
```

To define a theme dynamically, you must supply a valid SQL query as the content of the <jdbc_image_query> element. You must specify the JDBC connection information for an image theme (either `datasource` or the combination of `jdbc_host`, `jdbc_port`, `jdbc_sid`, `jdbc_user`, and `jdbc_password`).

`jdbc_srid` is an optional attribute that specifies the coordinate system (`SDO_SRID` value) of the data to be rendered.

`jdbc_mode` identifies the Oracle JDBC driver (`thin` or `oci8`) to use to connect to the database.

`asis` is an optional attribute. If it is set to `TRUE`, MapViewer does not attempt to modify the supplied query string. If `asis` is `FALSE` (the default), MapViewer embeds the SQL query as a subquery of its spatial filter query. For example, assume that you want a map centered at (-122, 37) with size 1, and the supplied query is:

```
SELECT geometry, sales FROM crm_sales WHERE sales < 100000;
```

If `asis` is `FALSE`, the actual query that MapViewer executes is similar to:

```
SELECT * FROM
   (SELECT geometry, sales FROM crm_sales WHERE sales < 100000)
WHERE sdo_filter(geometry, sdo_geometry(. . . -122.5, 36.5, -123.5, 37.5 . . .)
='TRUE';
```

In other words, the original query is further refined by a spatial filter query for the current map window. However, if `asis` is `TRUE`, MapViewer executes the query as specified, namely:

```
SELECT geometry, sales FROM crm_sales WHERE sales < 100000;
```

`image_format` identifies the format (such as GIF or JPEG) of the image data. If the image format is not supported by MapViewer, you must create and register a custom image renderer for the format, as explained in Appendix C.

`image_column` identifies the column of type BLOB where each image is stored.

`image_mbr_column` identifies the column of type SDO_GEOMETRY where the footprint (minimum bounding rectangle, or MBR) of each image is stored.

`image_resolution` is an optional attribute that identifies the original image resolution (number of `image_unit` units for each pixel).

`image_unit` is an optional attribute, except it is required if you specify the `image_resolution` attribute. The `image_unit` attribute specifies the unit of the resolution, such as `M` for meter. The value for this attribute must be one of the values in the SDO_UNIT column of the MDSYS.SDO_DIST_UNITS table. In Example 2–12 in Section 2.3.3.1, the image resolution is 2 meters per pixel.

For an example of using the <jdbc_image_query> element to specify an image theme, see Example 3–6 in Section 3.1.6.

3.2.8 jdbc_network_query Element

The <jdbc_network_query> element, which is used to define a network theme, has the following definition:

```
<!ELEMENT jdbc_network_query (#PCDATA) >
<!ATTLIST jdbc_network_query
    asis                     (TRUE|FALSE) "FALSE"
    network_name             CDATA #REQUIRED
    network_level            CDATA #IMPLIED
    link_style               CDATA #IMPLIED
    direction_style          CDATA #IMPLIED
    bidirection_style        CDATA #IMPLIED
    direction_position       CDATA #IMPLIED
    direction_markersize     CDATA #IMPLIED
    direction_multimarker   (TRUE|FALSE) "FALSE"
    link_labelstyle          CDATA #IMPLIED
    link_labelcolumn         CDATA #IMPLIED
    node_style               CDATA #IMPLIED
    node_markersize          CDATA #IMPLIED
    node_labelstyle          CDATA #IMPLIED
    node_labelcolumn         CDATA #IMPLIED
    path_ids                 CDATA #IMPLIED
    path_styles              CDATA #IMPLIED
    path_labelstyle          CDATA #IMPLIED
    path_labelcolumn         CDATA #IMPLIED
    analysis_algorithm       CDATA #IMPLIED
    shortestpath_style       CDATA #IMPLIED
    shortestpath_startnode   CDATA #IMPLIED
    shortestpath_endnode     CDATA #IMPLIED
    shortestpath_startstyle CDATA #IMPLIED
    shortestpath_endstyle    CDATA #IMPLIED
    withincost_startnode     CDATA #IMPLIED
    withincost_style         CDATA #IMPLIED
    withincost_cost          CDATA #IMPLIED
    withincost_startstyle    CDATA #IMPLIED
    datasource               CDATA #IMPLIED
    jdbc_host                CDATA #IMPLIED
    jdbc_port                CDATA #IMPLIED
    jdbc_sid                 CDATA #IMPLIED
    jdbc_user                CDATA #IMPLIED
    jdbc_password            CDATA #IMPLIED
    jdbc_srid                CDATA #IMPLIED
    jdbc_mode                (thin|oci8) "thin"
>
```

For detailed usage and reference information about network themes, see Section 2.3.5.

3.2.9 jdbc_query Element

The <jdbc_query> element is used to define a theme dynamically. This element and its associated <hidden_info> element have the following definitions:

```
<!ELEMENT jdbc_query (#PCDATA, hidden_info?)>
<!ATTLIST jdbc_query
    asis                (TRUE|FALSE) "FALSE"
    spatial_column     CDATA #REQUIRED
```

```
    key_column        CDATA #IMPLIED
    label_column      CDATA #IMPLIED
    label_style       CDATA #IMPLIED
    render_style      CDATA #IMPLIED
    x_column          CDATA #IMPLIED
    y_column          CDATA #IMPLIED
    datasource        CDATA #IMPLIED
    jdbc_host         CDATA #IMPLIED
    jdbc_port         CDATA #IMPLIED
    jdbc_sid          CDATA #IMPLIED
    jdbc_user         CDATA #IMPLIED
    jdbc_password     CDATA #IMPLIED
    jdbc_srid         CDATA #IMPLIED
    jdbc_mode         (thin|oci8) "thin"
>
<!ELEMENT hidden_info (field+)>
<!ELEMENT field (#PCDATA)>
<!ATTLIST field
    column  CDATA #REQUIRED
    name    CDATA #IMPLIED
>
```

To define a theme dynamically, you must supply a valid SQL query as the content of the <jdbc_query> element. You must specify the spatial_column (column of type SDO_GEOMETRY) and the JDBC connection information for a dynamically defined theme (either datasource or the combination of jdbc_host, jdbc_port, jdbc_sid, jdbc_user, and jdbc_password).

If the selectable_in_svg attribute value is TRUE in the <theme> element, you must use the key_column attribute in the <jdbc_query> element to specify the name of a column that can uniquely identify each selected feature from the JDBC query. The specified column must also appear in the SELECT list in the JDBC query.

render_style and label_style are optional attributes. For render_style, for point features the default is a red cross rotated 45 degrees, for lines and curves it is a black line 1 pixel wide, and for polygons it is a black border with a semitransparent dark gray interior.

x_column and y_column are optional attributes. If specified, they are used to define a point JDBC theme based on two columns in a table, so that MapViewer can render a point theme based on values in these columns. For more information, see Section 2.3.2.1.

jdbc_srid is an optional attribute that specifies the coordinate system (SDO_SRID value) of the data to be rendered.

jdbc_mode identifies the Oracle JDBC driver (thin or oci8) to use to connect to the database.

asis is an optional attribute. If it is set to TRUE, MapViewer does not attempt to modify the supplied query string. If asis is FALSE (the default), MapViewer embeds the SQL query as a subquery of its spatial filter query. For example, assume that you want a map centered at (-122, 37) with size 1, and the supplied query is:

```
SELECT geometry, sales FROM crm_sales WHERE sales < 100000;
```

If asis is FALSE, the actual query that MapViewer executes is similar to:

```
SELECT * FROM
   (SELECT geometry, sales FROM crm_sales WHERE sales < 100000)
WHERE sdo_filter(geometry, sdo_geometry(. . . -122.5, 36.5, -123.5, 37.5. . . )
='TRUE';
```

In other words, the original query is further refined by a spatial filter query using the current map window. However, if asis is TRUE, MapViewer executes the query as specified, namely:

```
SELECT geometry, sales FROM crm_sales WHERE sales < 100000;
```

The <hidden_info> element specifies the list of attributes from the base table to be displayed when the user moves the mouse over the theme's features. The attributes are specified by a list of <field> elements.

Each <field> element must have a column attribute, which specifies the name of the column from the base table, and it can have a name attribute, which specifies the display name of the column. (The name attribute is useful if you want a text string other than the column name to be displayed.)

For examples of using the <jdbc_query> element to define a theme dynamically, see Example 3–2 in Section 3.1.2 and Example 3–4 in Section 3.1.4.

3.2.10 jdbc_topology_query Element

The <jdbc_topology_query> element, which is used to define a topology theme, has the following definition:

```
<!ELEMENT jdbc_topology_query (#PCDATA)>
<!ATTLIST jdbc_topology_query
    asis                (TRUE|FALSE) "FALSE"
    topology_name       CDATA #REQUIRED
    feature_table       CDATA #REQUIRED
    spatial_column      CDATA #REQUIRED
    label_column        CDATA #IMPLIED
    label_style         CDATA #IMPLIED
    render_style        CDATA #IMPLIED
    datasource          CDATA #IMPLIED
    edge_style          CDATA #IMPLIED
    edge_marker_style   CDATA #IMPLIED
    edge_marker_size    CDATA #IMPLIED
    edge_label_style    CDATA #IMPLIED
    node_style          CDATA #IMPLIED
    node_label_style    CDATA #IMPLIED
    face_style          CDATA #IMPLIED
    face_label_style    CDATA #IMPLIED
    jdbc_host           CDATA #IMPLIED
    jdbc_port           CDATA #IMPLIED
    jdbc_sid            CDATA #IMPLIED
    jdbc_user           CDATA #IMPLIED
    jdbc_password       CDATA #IMPLIED
    jdbc_srid           CDATA #IMPLIED
    jdbc_mode           (thin|oci8) "thin"
>
```

For detailed usage and reference information about topology themes, see Section 2.3.6.

3.2.11 legend Element

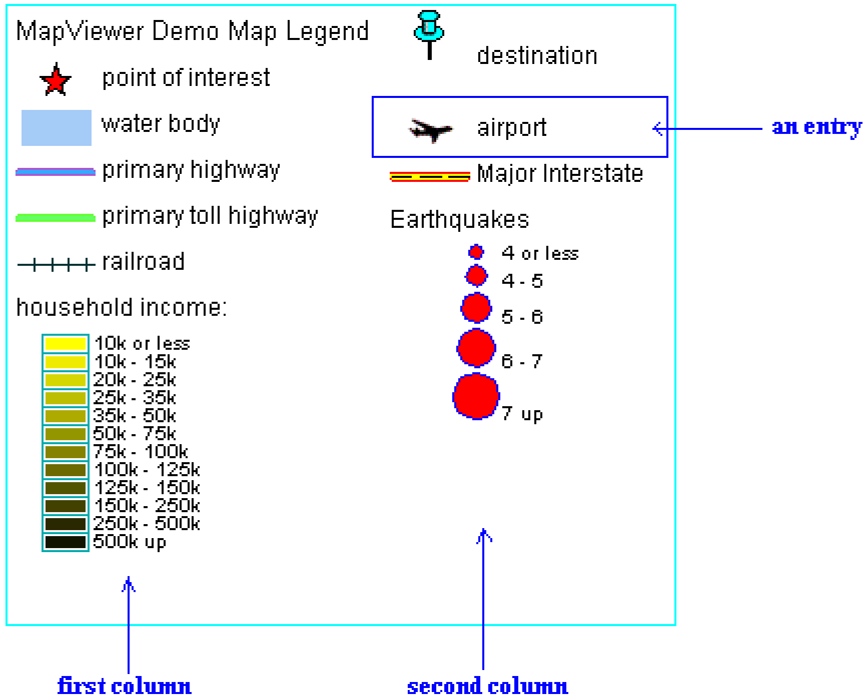
The <legend> element has the following definition:

```
<!ELEMENT legend (column,themes)? >
<!ATTLIST legend
    bgstyle    CDATA #implied
    font       CDATA #implied
```

```
    location_x CDATA #implied
    location_y CDATA #implied
    offset_x   CDATA #implied
    offset_y   CDATA #implied
    profile    (MEDIUM|SMALL|LARGE)  "MEDIUM"
    position   (SOUTH_WEST|SOUTH_EAST|SOUTH|NORTH|
               NORTH_WEST|NORTH_EAST|EAST|WEST|CENTER)  "SOUTH_WEST"
>
<!ELEMENT column entry+ >
<!ATTLIST entry
    is_title      (true|false) "false"
    is_separator  (true|false) "false"
    tab           CDATA  "0"
    style         CDATA  #implied
    text          CDATA  #implied
    text_size     CDATA  #implied
    width         CDATA  #implied
    height        CDATA  #implied
>
<!ELEMENT themes theme+ >
<!ATTLIST theme
    name          CDATA #REQUIRED
>
```

<legend> elements are used to draw a legend (map inset illustration) on top of a generated map, to make the visual aspects of the map more meaningful to users. The main part of a <legend> element is one or more <column> elements, each of which defines a column in the legend. (If no <column> elements are present, an *automatic legend* is created, as explained in Section 2.4.2.) A one-column legend will have all entries arranged from top to bottom. A two-column legend will have the two columns side by side, with the first column on the left, and each column having its own legend entries. Figure 2–13 in Section 2.4.2 shows a one-column legend. Figure 3–5 shows a two-column legend.

Figure 3–5 Two-Column Map Legend



`bgstyle` is an optional attribute that specifies the overall background style of the legend. It uses a string with syntax similar to scalable vector graphics (SVG) to specify the fill and stroke colors for the bounding box of the legend. If you specify an opacity (`fill-opacity` or `stroke-opacity`) value, the fill and stroke colors can be transparent or partially transparent. The following example specifies a background that is white and half transparent, and a stroke (for the legend box boundary) that is red:

```
bgstyle="fill:#ffffff;fill-opacity:128;stroke:#ff0000"
```

`font` is an optional attribute that specifies the name of the font to be used for text that appears in the legend image. You can specify a logical font name that is supported by Java (`serif`, `sansserif`, `monospaced`, `dialog`, or `dialoginput`). You can also specify the name of a physical font that is available on the system where the MapViewer server is running.

`location_x` and `location_y` are optional attributes that specify the X and Y coordinates (in screen units) of the start of the legend. If you specify these attributes, they override any specification for the `position` attribute.

`offset_x` and `offset_y` are optional attributes to be used with the `position` attribute. The default distance from the borders for the position hint corresponds to 10 pixels. You can use these offset parameters to override the default value.

`profile` is an optional attribute that specifies a relative size of the legend on the map, using one of the following keywords: `SMALL`, `MEDIUM` (the default), or `LARGE`.

`position` is an optional attribute that specifies where the legend should be drawn on the map. The default is `SOUTH_WEST`, which draws the legend in the lower-left corner of the resulting map.

is_title is an optional attribute of the <entry> element. When its value is TRUE, the entry is used as the title for the column, which means that the description text appears in a more prominent font than regular legend text, and any other style attribute defined for the entry is ignored. The default is FALSE.

is_separator is an optional attribute of the <entry> element. When its value is TRUE, the entry is used to insert a blank line for vertical spacing in the column. The default is FALSE.

tab is an optional attribute of the <entry> element. It specifies the number of tab positions to indent the entry from the left margin of the column. The default is 0 (zero), which means no indentation.

style is an optional attribute of the <entry> element. It specifies the name of the MapViewer style (such as a color or an image) to be depicted as part of the entry.

text is an optional attribute of the <entry> element. It specifies the description text (for example, a short explanation of the associated color or image) to be included in the entry.

text_size is an optional attribute of the <entry> element. It specifies the size (in display units) of the description text to be included in the entry. The specified value overrides the MapViewer predefined profile size.

width and height are optional attributes that together specify the size (in device units) of the legend entry Any specified values override the defaults, which depend on the MapViewer profile values for small, medium, and large text.

The following example shows the <legend> element specification for the legend in Figure 2–13 in Section 2.4.2.

```
<legend bgstyle="fill:#ffffff;fill-opacity:128;stroke:#ff0000"
            position="NORTH_WEST">
  <column>
    <entry text="Map Legend" is_title="true"/>
    <entry style="M.STAR" text="center point"/>
    <entry style="M.CITY HALL 3" text="cities"/>
    <entry is_separator="true"/>
    <entry style="C.ROSY BROWN STROKE" text="state boundary"/>
    <entry style="L.PH" text="interstate highway"/>
    <entry text="County population:"/>
    <entry style="V.COUNTY_POP_DENSITY" tab="1"/>
  </column>
</legend>
```

In the preceding example:

- The background color has an opacity value of 128 (fill-opacity:128), which means that the white background will be half transparent.
- The legend boundary box will be red (stroke:#ff0000).
- The legend boundary box will be positioned in the upper-left part of the display (position="NORTH_WEST").
- The legend will be the default size, because the profile attribute (which has a default value of MEDIUM) is not specified.
- The legend will have a single column, with entries arranged from top to bottom.
- The first entry is the legend title, with the text *Map Legend*.
- The fourth entry is a separator for adding a blank line.

- The seventh entry is description text (*County population:*) that users of the generated map will associate with the next (and last) entry, which specifies an advanced style. The *County population:* text entry is helpful because advanced styles usually have their own descriptive text, and you do not want users to become confused about which text applies to which parts of the legend.
- The last entry specifies an advanced style (`style="V.COUNTY_POP_DENSITY"`), and it is indented one tab position (`tab="1"`) so that the colors and text identifying various population density ranges will be easy for users to distinguish from the preceding *County population:* description text.

3.2.12 map_tile_theme Element

The `<map_tile_theme>` element is used to define a map tile theme, which produces a map image layer rendered by the map tile server with pregenerated map image tiles. The map image tiles can be served by any internal or external map service providers. This element has the following definition:

```
<!ELEMENT map_tile_theme (#PCDATA)>
<!ATTLIST map_tile_theme
    map_tile_layer      CDATA # REQUIRED
    snap_to_tile_scale  (TRUE|FALSE) "FALSE"
>
```

`map_tile_name` specifies the name of the map tile layer that has been predefined with MapViewer.

`snap_to_tile_scale` is an optional attribute that specifies whether to adjust the map scale to fit that of one of the predefined map tile layer zoom levels. If this attribute is `FALSE`, the scale of the result map is always the same as what the map request specifies; and if the map request scale does not fit any of the predefined map tile layer zoom levels, the map tile images are scaled to fit the map request scale. If this attribute is `TRUE`, the scale of the result map is adjusted to fit one of the predefined map tile layer zoom levels when the request map scale does not fit any of the predefined zoom levels.

3.2.13 north_arrow Element

The `<north_arrow>` element specifies a style (usually a marker) to point to the north direction on the map. It uses the map request rotation attribute to define its orientation. This element has the following definition:

```
<!ELEMENT north_arrow (style, location?, size?) >
```

The `<style>` element specifies the name of the style (typically a marker style) for the north arrow.

The `<location>` element is optional. It specifies the X and Y coordinate values (in pixels) of the position on the map for the north arrow. The default value is (25, 25).

The `<size>` element is optional. It specifies the width and height (in pixels) to be used by MapViewer in rendering the north arrow. The default value is (16, 32).

Example 3–20 shows a north arrow definition using style `m.image41_bw`, located at position (35, 35) of the map image, and with width 16 and height 32.

Example 3–20 North Arrow

```
<north_arrow>
   <style> m.image41_bw </style>
```

```
    <location> 35,35 </location>
    <size> 16,32 </size>
</north_arrow>
```

3.2.14 operation Element

The `<operation>` element enables you to perform additional transformations on the original data during rendering. The `<operation>` element has the following definition:

```
<!ELEMENT operation (parameter+) >
<!ATTLIST parameter
  name  CDATA #REQUIRED
>
```

Currently this element is used in GeoRaster themes (described in Section 2.3.4). You can perform some image processing operations on the original image, such as normalization, equalization, linear stretch, piecewise linear stretch, brightness and contrast adjustment, and threshold change.

Example 3–21 specifies the normalization operation with a GeoRaster theme.

Example 3–21 Normalization Operation with a GeoRaster Theme

```
<theme name="geor_theme" >
    <jdbc_georaster_query
        jdbc_srid="0"
        datasource="mvdemo"
        georaster_table="dem"
        georaster_column="georaster"
       asis="false"> select georaster from dem
    </jdbc_georaster_query>
    <operations>
     <operation name="normalize">
      </operation>
     </operations>
  </theme>
```

The following code segment shows a manual linear stretch operation. (For automatic linear stretch, include the `<operation>` element but no `<parameter>` elements.)

```
     <operation name="linearstretch">
         <parameter name="autostretch" value="false"/>
         <parameter name="lowstretch" value="50"/>
         <parameter name="highstretch" value="150"/>
     </operation>
```

Table 3–1 lists the image processing operations, their <operation> element name keyword values, and (where relevant) associated <parameter> element values.

Table 3–1 Image processing Options for GeoRaster Theme Operations

Operation	<operation> name value	<parameter> values
Normalization	`normalize`	(Not applicable)
Equalization	`equalize`	(Not applicable)
Linear stretch	`linearstretch`	`name=autostretch` (automatic)
		`name=lowstretch` (low stretch)
		`name=highstretch` (high stretch)

Table 3–1 (Cont.) Image processing Options for GeoRaster Theme Operations

Operation	<operation> name value	<parameter> values
Piecewise linear stretch	`piecewiselinearstretch`	(Not applicable)
Brightness	`brightness`	`value=[number]`
Contrast	`contrast`	`value=[number]`
Change threshold	`changethreshold`	`name=threshold` (threshold) `name=lowsthreshold` (low threshold) `name=highthreshold` (high threshold)

3.2.15 operations Element

The `<operations>` element specifies one or more `<operation>` elements (described in Section 3.2.14). The `<operations>` element has the following definition:

```
<!ELEMENT operations (oepration+) >
```

For a predefined GeoRaster theme, the <operations> element will be part of the styling rule definition. Example 3–21 shows the styling rules for a GeoRaster theme that uses the normalization operation.

Example 3–22 Styling Rules with Normalization Operation in a GeoRaster Theme

```
<styling_rules theme_type="georaster" raster_table="RDT_DEM"
               raster_id="1">
   <operations>
      <operation name="normalize"/>
    </operations>
</styling_rules>
```

3.2.16 parameter Element

The `<parameter>` element defines values to be used in an operation to be applied on themes. (The operation is specified in an `<operations>` element, described in Section 3.2.14.) The `<parameter>` element has the following definition:

```
<!ELEMENT parameter >
<!ATTLIST parameter
  name   CDATA #REQUIRED
  value  CDATA #REQUIRED
>
```

Each parameter must have a name and value associated with it.

3.2.17 scale_bar Element

The `<scale_bar>` element defines a scale bar (to show how many kilometers or miles are represented by a distance marked on the bar) to be added to the map request, if the map has a known spatial reference system (SRS). You can specify a single display mode (Metric or US) or dual mode (both Metric and US). The `<scale_bar>` element has the following definition:

```
<!ELEMENT scale_bar >
<!ATTLIST scale_bar
   mode              (METRIC_MODE|US_MODE|DUAL_MODES) "METRIC_MODE"
   position          (SOUTH_WEST|SOUTH_EAST|SOUTH|NORTH|
                      NORTH_WEST|NORTH_EAST)  "NORTH_EAST"
```

```
   offset_y         CDATA #implied
   offset_y         CDATA #implied
   color1           CDATA #implied
   color1_opacity   CDATA #implied
   color2           CDATA #implied
   color2_opacity   CDATA #implied
   length_hint      CDATA #implied
   label_color      CDATA #implied
   label_font_family CDATA #implied
   label_font_size  CDATA #implied
   label_halo_size  CDATA #implied
   label_position   (TOP|BOTTOM) "TOP"
>
```

All `<scale_bar>` attributes are optional.

`mode` specifies if the scale bar should be in metric or US mode, or in both modes. The default is `METRIC_MODE`.

`position` defines the relative location on the map to place the scale bar. The default is `NORTH_EAST`.

`offset_x and offset_y` define the X and Y values to offset the scale bar position from the map margin. The default value for each is 0.

`color1`, `color1_opacity`, `color2`, and `color2_opacity` define the colors to be used when rendering the scale bar. `color1` and `color2` have a default value for red, green, blue; `color1_opacity` has a default value of (0x44, 0x44, 0x44, 210); and color `2_opacity` has a default value of (0xee, 0xee, 0xee, 210).

`length_hint` defines the preferred number of pixels to be used to render the scale bar. The default is approximately 17% of the map width.

`label_color`, `label_font_family`, `label_font_size`, and `label_halo_size` affect the scale bar text. The defaults are black color, Serif font family, 12pt font size, and no halo (0 halo size).

label_position defines the position of the text relative to the scale bar (`TOP` or `BOTTOM`). The default is `TOP`.

Example 3–23 defines a scale bar.

Example 3–23 Scale Bar

```
<scale_bar
   position="SOUTH_WEST"
   mode="US_MODE"
   color1="#ff0000"
   color1_opacity="128"
   color2="#00ffff"
   label_font_family="Dialog"
   label_font_size="15"
   label_font_style="italic"
   label_font_weight="bold"
   label_halo_size="2.8"
   label_position="bottom"
   offset_y="5"
/>
```

3.2.18 style Element

The `<style>` element has the following definition:

```
<!ELEMENT style (svg | AdvancedStyle)?>
<!ATTLIST style
    name  CDATA #REQUIRED
>
```

The `<style>` element lets you specify a dynamically defined style. The style can be either of the following:

- An SVG description representing a color, line, marker, area, or text style
- An advanced style definition (see Section A.6) representing a bucket, a color scheme, or a variable marker style

The `name` attribute identifies the style name.

The following example shows an excerpt that dynamically defines two styles (a color style and an advanced style) for a map request:

```
<map_request . . .>
  . . .
  <styles>
    <style name="color_red">
      <svg width="1in" height="1in">
        <g class="color"
           style="stroke:red;stroke-opacity:100;fill:red;fill-opacity:100">
          <rect width="50" height="50"/>
        </g>
      </svg>
    </style>

    <style name="ranged_bucket_style">
        <AdvancedStyle>
           <BucketStyle>
             <Buckets>
             <RangedBucket seq="0" label="less than 100k"
               high="100000.0" style="C.RB13_13"/>
             <RangedBucket seq="1" label="100k - 150k" low="100000.0"
               high="150000.0" style="C.RB13_1"/>
             <RangedBucket seq="2" label="150k - 250k" low="150000.0"
               high="250000.0" style="C.RB13_4"/>
             <RangedBucket seq="3" label="250k - 350k" low="250000.0"
               high="350000.0" style="C.RB13_7"/>
             <RangedBucket seq="4" label="350k - 450k" low="350000.0"
               high="450000.0" style="C.RB13_10"/>
              </Buckets>
           </BucketStyle>
        </AdvancedStyle>
    </style>
  </styles>
</map_request>
```

3.2.19 styles Element

The `<styles>` element has the following definition:

```
<!ELEMENT styles (style+) >
```

The `<styles>` element specifies one or more `<style>` elements (described in Section 3.2.18).

3.2.20 theme Element

The <theme> element has the following definition:

```
<!ELEMENT theme (jdbc_query | jdbc_image_query | jdbc_georaster_query
                 | jdbc_network_query | jdbc_topology_query | map_tile_theme)?,
                 operations? >
<!ATTLIST theme
    name                  CDATA #REQUIRED
    datasource            CDATA #IMPLIED
    template_theme        CDATA #IMPLIED
    max_scale             CDATA #IMPLIED
    min_scale             CDATA #IMPLIED
    label_max_scale       CDATA #IMPLIED
    label_min_scale       CDATA #IMPLIED
    label_always_on       (TRUE|FALSE) "FALSE"
    fast_unpickle         (TRUE|FALSE) "TRUE"
    mode                  CDATA #IMPLIED
    min_dist              CDATA #IMPLIED
    fixed_svglabel        (TRUE|FALSE) "FALSE"
    visible_in_svg        (TRUE|FALSE) "TRUE"
    selectable_in_svg     (TRUE|FALSE) "FALSE"
    part_of_basemap       (TRUE|FALSE) "FALSE"
    simplify_shapes       (TRUE|FALSE) "TRUE"
    transparency          CDATA #IMPLIED
    minimum_pixels        CDATA #IMPLIED
    onclick               CDATA #IMPLIED
    onmousemove           CDATA #IMPLIED
    onmouseover           CDATA #IMPLIED
    onmouseout            CDATA #IMPLIED
    workspace_name        CDATA #IMPLIED
    workspace_savepoint   CDATA #IMPLIED
    workspace_date        CDATA #IMPLIED
    workspace_date_format CDATA #IMPLIED
    fetch_size            CDATA #IMPLIED
    timeout               CDATA #IMPLIED
>
```

The <theme> element lets you specify a predefined or dynamically defined theme.

- For a predefined theme, whose definition is already stored in your USER_SDO_THEMES view, only the theme name is required.
- For a dynamically defined theme, you must provide the information in one of the following elements: <jdbc_query> (described in Section 3.2.9), <jdbc_image_query> (described in Section 3.2.7), <jdbc_georaster_query> (described in Section 2.3.4), <jdbc_network_query> (described in Section 2.3.5), or <jdbc_topology_query> (described in Section 2.3.6).
- For a GeoRaster theme, you can define some image processing options (described in Section 3.2.14).

The name attribute identifies the theme name. For a predefined theme, the name must match a value in the NAME column of the USER_SDO_THEMES view (described in Section 2.9.2). For a dynamically defined theme, this is just a temporary name for referencing the jdbc_query-based theme.

datasource is an optional attribute that specifies a data source for the theme. If you do not specify this attribute, the data source for the map request is assumed (see the datasource attribute explanation in Section 3.2.1.1). By specifying different data sources for different themes, you can use multiple data sources in a map request.

template_theme is an optional attribute that can be used to render two or more themes when a predefined theme has same name in multiple data sources. You cannot repeat theme names in a map request, but if you have two different data sources with same predefined theme name, you can use this attribute to render both themes. The following example specifies two themes that are based on a US_STATES theme that exists in two data sources, but that has a different content in each data source.

```
<themes>
  <theme name="US_STATES" datasource="dsrc"/>
  <theme name="OTHER_US_STATES" template_theme="US_STATES" datasource="other_dsrc"
/>
</themes>
```

The max_scale and min_scale attributes affect the visibility of this theme. If max_scale and min_scale are omitted, the theme is always rendered, regardless of the map scale. (See Section 2.4.1 for an explanation of max_scale and min_scale.)

The label_max_scale and label_min_scale attributes affect the visibility of feature labels of this theme. If label_max_scale and label_min_scale are omitted, the theme feature labels are always rendered when the map scale is within the visible range of theme scales (that is, within the max_scale and min_scale range). (See Section 2.4.1 for an explanation of label_max_scale and label_min_scale.)

label_always_on is an optional attribute. If it is set to TRUE, MapViewer labels all features of the theme even if two or more labels will overlap in the display. (MapViewer always tries to avoid overlapping labels.) If label_always_on is FALSE (the default), when it is impossible to avoid overlapping labels, MapViewer disables the display of one or more labels so that no overlapping occurs. The label_always_on attribute can also be specified for a map feature (geoFeature element, described in Section 3.2.5), thus allowing you to control which features will have their labels displayed if label_always_on is FALSE for a theme and if overlapping labels cannot be avoided.

fast_unpickle is an optional attribute. If it is TRUE (the default), MapViewer uses its own fast unpickling (unstreaming) algorithm instead of the generic JDBC conversion algorithm to convert SDO_GEOMETRY objects fetched from the database into a Java object accessible to MapViewer. This process improves performance, but occasionally the coordinates may lose some precision (around 0.00000005), which can be significant in applications where all precision digits of each coordinate must be kept. If fast_unpickle is set to FALSE, MapViewer uses the generic JDBC conversion algorithm. This process is slower than MapViewer's fast unpickling process, but there is never any loss of precision.

mode is an optional attribute. For a topology theme, you can specify mode="debug" to display edges, nodes, and faces, as explained in Section 2.3.6. The mode attribute is ignored for other types of themes.

min_dist is an optional attribute. It specifies the minimum on-screen distance (number of pixels) between two adjacent shape points on a line string or polygon for rendering of separate shape points. If the on-screen distance between two adjacent shape points is less than the min_dist value, only one shape point is rendered. The default value is 0.5. You can specify higher values to reduce the number of shape points rendered on an SVG map, and thus reduce the size of the resulting SVG file. You can specify different values in different theme definitions, to allow for customized levels of detail in SVG maps.

fixed_svglabel is an optional attribute that specifies whether to display the labels on an SVG map using the original "fixed" labels, but having them appear larger or

smaller as the zoom level increases (zoomin) or decreases (zoomout), or to use different labels with the same text but different actual sizes so that the apparent size of each label remains the same at all zoom levels. If the `fixed_svglabel` value is specified as `TRUE`, the same theme labels are displayed on the map at all zoom levels, with the labels zoomed in and out as the map is zoomed in and out. If the value is `FALSE` (the default), different theme labels are displayed at different zoom levels so that the size of each displayed label appears not to change during zoomin and zoomout operations.

`visible_in_svg` is an optional attribute that specifies whether or not to display the theme on an SVG map. If its value is `TRUE` (the default), the theme is displayed; if it is set to `FALSE`, the theme is not displayed. However, even if this attribute is set to `FALSE`, the theme is still rendered to the SVG map: the theme is initially invisible, but you can make it visible later by calling the JavaScript function `showTheme()` defined in the SVG map. For information about using JavaScript functions with SVG maps, see Appendix B.

`selectable_in_svg` is an optional attribute that specifies whether or not the theme is selectable on an SVG map. The default is `FALSE`; that is, the theme is not selectable on an SVG map. If this attribute is set to `TRUE` and if theme feature selection is allowed, each feature of the theme displayed on the SVG map can be selected by clicking on it. If the feature is selected, its color is changed and its ID (its rowid by default) is recorded. You can get a list of the ID values of all selected features by calling the JavaScript function `getSelectedIdList()` defined in the SVG map. For information about using JavaScript functions with SVG maps, see Appendix B.

`part_of_basemap` is an optional attribute. If the map format is SVG and the value of this attribute is `TRUE`, MapViewer renders the theme as part of and on top of the base map, which is rendered as a raster image.

`simplify_shapes` is an optional attribute that specifies whether or not the shapes are simplified before being rendered. Simplification is useful when you want a map display with less fine resolution than the original geometries. For example, if the display resolution cannot show the hundreds or thousands of turns in the course of a river or in a political boundary, better performance might result if the shapes were simplified to show only the major turns. The default is `TRUE`; that is, shapes are simplified before being rendered. If this attribute is set to `FALSE`, MapViewer attempts to render all vertices and line segments from the original geometries, and performance may be slower.

transparency is an optional parameter to define the basic alpha composing value to be applied on themes during rendering. The value can be from 0 to 1, with 0 meaning completely transparent and 1 (the default) meaning completely opaque (no transparency).

minimum_pixels is an optional parameter that defines the level of resolution to be used on the spatial filter query. This may be useful to avoid rendering too many elements at the same position of the screen. (See the Oracle Spatial and Graph documentation about the `min_resolution` and `max_resolution` options for the SDO_FILTER operator.) The unit for `minimum_pixels` is screen pixels. For example, `minimum_pixels=1` means that the spatial filter query will not return features with a resolution less than the amount that 1 pixel represents for the current device window and current query window

`onclick` is an optional attribute that specifies the name of the JavaScript function to be called when a user clicks on an SVG map and theme feature selection is allowed (see the `selectable_in_svg` attribute explanation). The JavaScript function must be defined in the HTML document that has the SVG map embedded. This function must accept only four parameters: the theme name, the key of the feature, and x and y,

which specify the coordinates (in pixels) of the clicked point on the SVG map. For information about using JavaScript functions with SVG maps, see Appendix B.

`onmousemove` is an optional attribute that specifies the name of the JavaScript function to be called when a user moves the mouse on top of any feature of the theme on an SVG map. The JavaScript function must be defined in the HTML document that has the SVG map embedded. This function must accept only four parameters: the theme name, the key of the feature, and x and y, which specify the coordinates (in pixels) of the point for the move on the SVG map. For information about using JavaScript functions with SVG maps, see Appendix B.

`onmouseover` is an optional attribute that specifies the name of the JavaScript function to be called when a user moves the mouse into a feature of the theme on an SVG map. (Unlike the `onmousemove` function, which is called whenever the mouse moves inside the theme, the `onmouseover` function is called only once when the mouse moves from outside a feature of the theme to inside a feature of the theme.) The JavaScript function must be defined in the HTML document that has the SVG map embedded. This function must accept only four parameters: the theme name, the key of the feature, and x and y, which specify the coordinates (in pixels) of the point at which the mouse moves inside a feature on the SVG map. For information about using JavaScript functions with SVG maps, see Appendix B.

`onmouseout` is an optional attribute that specifies the name of the JavaScript function to be called when a user moves the mouse out of a feature of the theme on an SVG map. The JavaScript function must be defined in the HTML document that has the SVG map embedded. This function must accept only four parameters: the theme name, the key of the feature, and x and y, which specify the coordinates (in pixels) of the point at which the mouse moves out of a feature on the SVG map. For information about using JavaScript functions with SVG maps, see Appendix B.

`workspace_name`, `workspace_savepoint`, `workspace_date`, and `workspace_date_format` are optional attributes related to support for Workspace Manager in Mapviewer, which is explained in Section 2.8.

`fetch_size` is an optional attribute that specifies how many rows will be prefetched into memory. The default value is 100.

`timeout` is an optional attribute that specifies the number of milliseconds to wait for the connection to the WMS or WFS server.

3.2.21 themes Element

The `<themes>` element has the following definition:

```
<!ELEMENT themes (theme+) >
```

The `<themes>` element specifies one or more `<theme>` elements (described in Section 3.2.20). If you have specified a base map (`basemap` attribute of the `map_request` element), any themes that you specify in a `<themes>` element are plotted after those defined in the base map. If no base map is specified, only the specified themes are rendered.

Inside this `<themes>` element there must be one or more `<theme>` child elements, which are rendered in the order in which they appear.

3.2.22 theme_modifiers Element

The `<theme_modifiers>` element has the following definition:

```
<!ELEMENT theme_modifiers (theme_decorations)? >
```

The theme modifiers enable you to override the theme definition on a base map, without having to edit and change the base map definition. The <theme_decorations> element has the same attributes as the <theme> element (described in Section 3.2.20).

The following example overrides the `labels_always_on` attribute for the `theme_us_airport` theme on the base map `FORCED_LABELING`.

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="Override labeling on map definition"
  basemap="FORCED_LABELING"
  datasource="tilsmenv"
  width="500"
  height="375"
  bgcolor="#a6caf0"
  antialiase="true"
  format="PNG_URL">
  <center size="15.0">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>-122.4,37.8</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
  <theme_modifiers>
    <theme_decorations name="theme_us_airport" label_always_on="false"/>
  </theme_modifiers>
</map_request>
```

3.3 Information Request DTD

In addition to issuing map requests (see Section 3.2) and administrative requests (see Chapter 7), you can issue information requests to MapViewer. An information request is an XML request string that you can use to execute SQL queries and obtain the result as an array of strings or an XML document. The SQL query must be a SELECT statement and must select only primitive SQL types (for example, not LOB types or user-defined object types).

The following is the DTD for a MapViewer information request.

```
<!ELEMENT info_request (#PCDATA) >
<!ATTLIST info_request
               datasource CDATA #REQUIRED
               format     (strict | non-strict)  "strict"
>
```

`datasource` is a required attribute that specifies the data source for which to get the information.

`format` is an optional attribute. If it is `strict` (the default), all rows are formatted and returned in an XML document. If `format` is set to `non-strict`, all rows plus a column heading list are returned in a comma-delimited text string.

Example 3–24 shows an information request to select the city, 1990 population, and state abbreviation from the CITIES table, using the connection information in the

mvdemo data source and returning the information as an XML document (format="strict").

Example 3–24 MapViewer Information Request

```
<?xml version="1.0" standalone="yes"?>
<info_request datasource="mvdemo"  format="strict">
        SELECT city, pop90 population, state_abrv state FROM cities
</info_request>
```

Example 3–24 returns an XML document that includes the following:

```
<?xml version="1.0" encoding="UTF-8"?>
  <ROWSET>
    <ROW num="1">
      <CITY>New York</CITY>
      <POPULATION>7322564</POPULATION>
      <STATE>NY</STATE>
    </ROW>
    <ROW num="2">
      <CITY>Los Angeles</CITY>
      <POPULATION>3485398</POPULATION>
      <STATE>CA</STATE>
    </ROW>
    <ROW num="3">
      <CITY>Chicago</CITY>
      <POPULATION>2783726</POPULATION>
      <STATE>IL</STATE>
    </ROW>
    <ROW num="4">
      <CITY>Houston</CITY>
      <POPULATION>1630553</POPULATION>
      <STATE>TX</STATE>
    </ROW>
     . . .
  </ROWSET>
```

3.4 Map Response DTD

The following is the DTD for the map response resulting from normal processing of a map request. (Section 3.5 shows the DTD for the response if there was an exception or unrecoverable error.)

```
<!ELEMENT map_response (map_image)>
<!ELEMENT map_image (map_content, box, themes, WMTException)>
<!ELEMENT map_content EMPTY>
<!ATTLIST map_content url CDATA #REQUIRED>
<!ELEMENT WMTException (#PCDATA)>
<!ATTLIST WMTException  version CDATA "1.0.0"
                        error_code (SUCCESS|FAILURE) #REQUIRED
>
```

The response includes the URL for retrieving the image, as well as any error information. When a valid map is generated, its minimum bounding box is also returned, along with the list of themes that have features within the minimum bounding rectangle (MBR) that intersects with the bounding box.

Example 3–25 shows a map response.

Example 3–25 Map Response

```
<?xml version="1.0" encoding="UTF-8" ?>
<map_response>
   <map_image>
     <map_content url="http://map.oracle.com/output/map029763.gif"/>
     <box srsName="default">
        <coordinates>-122.260443,37.531621 -120.345,39.543</coordinates>
     </box>
     <themes>
        <theme name="US_STATES" />
        <theme name="US_HIGHWAYS" />
     </themes>
     <WMTException version="1.0.0" error_code="SUCCESS">
     </WMTException>
   </map_image>
</map_response>
```

3.5 MapViewer Exception DTD

The following DTD is used by the output XML when an exception or unrecoverable error is encountered while processing a map request:

```
<!ELEMENT oms_error (#PCDATA)>
```

The exception or error message is embedded in this element.

3.6 Geometry DTD (OGC)

MapViewer supports the Geometry DTD as defined in the Open Geospatial Consortium (OGC) GML v1.0 specification. This specification has the following copyright information:

This specification includes the following status information, although its current official status is Deprecated Recommendation Paper:

```
This document is an OpenGIS® Consortium Recommendation Paper. It is similar to a
proposed recommendation in other organizations. While it reflects a public
statement of the official view of the OGC, it does not have the status of a OGC
Technology Specification. It is anticipated that the position stated in this
document will develop in response to changes in the underlying technology.
Although changes to this document are governed by a comprehensive review
procedure, it is expected that some of these changes may be significant.

The OGC explicitly invites comments on this document. Please send them to
gml.rfc@opengis.org
```

The following additional legal notice text applies to this specification:

The OGC Geometry DTD in this specification is as follows:

```
<!-- ============================================================== -->
<!--       G e o g r a p h y                                        -->
<!--       M a r k u p                                              -->
<!--       L a n g u a g e                                          -->
<!--                                                                -->
<!--       ( G M L )                                                -->
<!--                                                                -->
<!--       G E O M E T R Y   D T D                                  -->
<!--                                                                -->
<!--  Copyright (c) 2000 OGC All Rights Reserved.                   -->
<!-- ============================================================== -->

<!-- the coordinate element holds a list of coordinates as parsed character
data. Note that it does not reference a SRS and does not constitute a proper
geometry class. -->
<!ELEMENT coordinates (#PCDATA) >
<!ATTLIST coordinates
   decimal  CDATA    #IMPLIED
   cs       CDATA    #IMPLIED
   ts       CDATA    #IMPLIED >

<!-- the Box element defines an extent using a pair of coordinates and a SRS name.
-->
<!ELEMENT Box (coordinates) >
<!ATTLIST Box
   ID       CDATA    #IMPLIED
   srsName  CDATA    #REQUIRED >

<!-- ============================================================== -->
<!--  G E O M E T R Y   C L A S S   D e f i n i t i o n s         -->
<!-- ============================================================== -->

<!-- a Point is defined by a single coordinate. -->
<!ELEMENT Point (coordinates) >
<!ATTLIST Point
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >

<!-- a LineString is defined by two or more coordinates, with linear
interoplation between them. -->
<!ELEMENT LineString (coordinates) >
<!ATTLIST LineString
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >

<!-- a Polygon is defined by an outer boundary and zero or more inner
boundaries. These boundaries are themselves defined by LinerRings. -->
<!ELEMENT Polygon (outerBoundaryIs, innerBoundaryIs*) >
<!ATTLIST Polygon
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT outerBoundaryIs (LinearRing) >
<!ELEMENT innerBoundaryIs (LinearRing) >

<!-- a LinearRing is defined by four or more coordinates, with linear
interpolation between them. The first and last coordinates must be
coincident. -->
<!ELEMENT LinearRing (coordinates) >
```

```
<!ATTLIST LinearRing
   ID       CDATA    #IMPLIED >

<!-- a MultiPoint is defined by zero or more Points, referenced through a
pointMember element. -->
<!ELEMENT MultiPoint (pointMember+) >
<!ATTLIST MultiPoint
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT pointMember (Point) >

<!-- a MultiLineString is defined by zero or more LineStrings, referenced
through a lineStringMember element. -->
<!ELEMENT MultiLineString (lineStringMember+) >
<!ATTLIST MultiLineString
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT lineStringMember (LineString) >

<!-- a MultiPolygon is defined by zero or more Polygons, referenced through a
polygonMember element. -->
<!ELEMENT MultiPolygon (polygonMember+) >
<!ATTLIST MultiPolygon
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT polygonMember (Polygon) >

<!-- a GeometryCollection is defined by zero or more geometries, referenced
through a geometryMember element. A geometryMember element may be any one of
the geometry classes. -->
<!ENTITY % GeometryClasses "(
   Point | LineString | Polygon |
   MultiPoint | MultiLineString | MultiPolygon |
   GeometryCollection )" >

<!ELEMENT GeometryCollection (geometryMember+) >
<!ATTLIST GeometryCollection
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT geometryMember %GeometryClasses; >

<!-- ============================================================== -->
<!--   G E O M E T R Y   P R O P E R T Y   D e f i n i t i o n s    -->
<!-- ============================================================== -->

<!-- GML provides an 'endorsed' name to define the extent of a feature. The
extent is defined by a Box element, the name of the property is boundedBy. -->
<!ELEMENT boundedBy (Box) >

<!-- the generic geometryProperty can accept a geometry of any class. -->
<!ELEMENT geometryProperty (%GeometryClasses;) >

<!-- the pointProperty has three descriptive names: centerOf, location and
position. -->
<!ELEMENT pointProperty (Point) >
<!ELEMENT centerOf (Point) >
<!ELEMENT location (Point) >
<!ELEMENT position (Point) >

<!-- the lineStringProperty has two descriptive names: centerLineOf and
```

```
edgeOf. -->
<!ELEMENT lineStringProperty (LineString) >
<!ELEMENT centerLineOf (LineString)>
<!ELEMENT edgeOf (LineString)>

<!-- the polygonProperty has two descriptive names: coverage and extentOf. -->
<!ELEMENT polygonProperty (Polygon) >
<!ELEMENT coverage (Polygon)>
<!ELEMENT extentOf (Polygon)>

<!-- the multiPointProperty has three descriptive names: multiCenterOf,
multiLocation and multiPosition. -->
<!ELEMENT multiPointProperty (MultiPoint) >
<!ELEMENT multiCenterOf (MultiPoint) >
<!ELEMENT multiLocation (MultiPoint) >
<!ELEMENT multiPosition (MultiPoint) >

<!-- the multiLineStringProperty has two descriptive names: multiCenterLineOf
and multiEdgeOf. -->
<!ELEMENT multiLineStringProperty (MultiLineString) >
<!ELEMENT multiCenterLineOf (MultiLineString) >
<!ELEMENT multiEdgeOf (MultiLineString) >

<!-- the multiPolygonProperty has two descriptive names: multiCoverage and
multiExtentOf. -->
<!ELEMENT multiPolygonProperty (MultiPolygon) >
<!ELEMENT multiCoverage (MultiPolygon) >
<!ELEMENT multiExtentOf (MultiPolygon) >

<!ELEMENT geometryCollectionProperty (GeometryCollection) >

<!-- ============================================================== -->
<!--    F E A T U R E   M E T A D A T A   D e f i n i t i o n s   -->
<!-- ============================================================== -->

<!-- Feature metadata, included in GML Geometry DTD for convenience; name and
description are two 'standard' string properties defined by GML. -->

<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

4

MapViewer JavaBean-Based API

This chapter describes the JavaBean-based MapViewer API. This API exposes all capabilities of MapViewer through a single JavaBean, `oracle.lbs.mapclient.MapViewer`. This bean is a lightweight client that handles all communications with the actual MapViewer service running on the middle tier on behalf of a user making map requests.

All communications between the bean and the actual MapViewer service follow a request/response model. Requests are always sent as XML documents to the service. Depending on the type and nature of a request, the response received by the bean is either an XML document or some binary data. However, using the MapViewer bean is easier than manipulating XML documents for forming and sending MapViewer requests, as well as for extracting information from the responses.

The bean delegates most of map data processing and rendering to the MapViewer service. All the bean does is formulate user requests into valid MapViewer XML requests and send them to a MapViewer service for processing.

This chapter contains the following major sections:

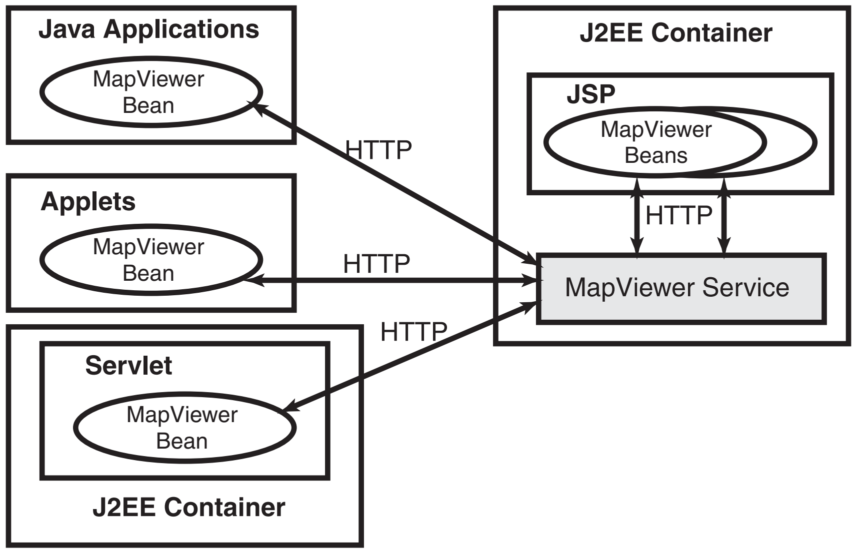
- Section 4.1, "Usage Model for the MapViewer JavaBean-Based API"
- Section 4.2, "Preparing to Use the MapViewer JavaBean-Based API"
- Section 4.3, "Using the MapViewer Bean"

4.1 Usage Model for the MapViewer JavaBean-Based API

The MapViewer bean can be created and used in either server-side objects such as JavaServer Pages (JSP) and servlets, or in client-side objects such as Java applets or standalone Java applications. The bean is a lightweight class that maintains an active HTTP connection to the MapViewer service and the current map request and map response objects. In most cases, you will create only one MapViewer bean and use it for all subsequent tasks; however, you can create more than one bean and use these beans simultaneously. For example, you may need to create a web page where a small overview map displays the whole world and a large map image displays a more detailed map of the region that is selected on the overview map. In this case, it is probably easier to create two MapViewer beans, one dedicated to the smaller overview map, and the other to the larger detail map.

Figure 4–1 shows some possible usage scenarios for the MapViewer bean.

Figure 4–1 MapViewer Bean Usage Scenarios



The MapViewer bean can communicate through the HTTP protocol with the MapViewer service in several usage scenarios, the following of which are shown in Figure 4–1:

- In a Java application
- In a Java applet
- In a servlet within a Java EE (Java Platform, Enterprise Edition) container different from the Java EE container that contains the MapViewer service
- In JavaServer Pages (JSP) code within the Java EE container that contains the MapViewer service

In all usage models, the same JavaBean class is used, and most of its methods apply. However, some methods work or are useful only in a JSP HTML-based context, and other methods work or are useful only in an interactive standalone Java application or applet context (thick clients). For example, consider the following methods of the bean:

- `java.awt.Image getGeneratedMapImage`
- `String getGeneratedMapImageURL`

Both methods extract the generated map image information from a response received from a MapViewer service; however, the first method returns the actual binary image data that is a `java.awt.BufferedImage` class, and the second method returns an HTTP URL string to the generated map image that is stored in the host running the MapViewer service. Clearly, if your application is a JavaServer Page, you should use the second method, because otherwise the JSP page will not know how to handle the `BufferedImage`. However, if you are programming a standalone Java application where you have a Java panel or window for displaying the map, you can use the first method to get the actual image and render it inside your panel or window, plus any other features that you may have created locally and want to render on top of the map.

The set of methods that are only applicable in the thick client context, which are designed to achieve optimal performance for such clients, are described in more detail in Section 4.3.10.

4.2 Preparing to Use the MapViewer JavaBean-Based API

Before you can use the MapViewer JavaBean, the MapViewer `mvclient.jar` library must be in a directory that is included in the CLASSPATH definition. After you deploy the `mapviewer.ear` file in OC4J or Oracle Fusion Middleware, the `mvclient.jar` file is located in the *$MAPVIEWER*`/web/WEB-INF/lib` directory. (*$MAPVIEWER* is the base directory that the `mapviewer.ear` file is unpacked into by OC4J. In a typical OC4J installation, if you placed the `mapviewer.ear` file in *$OC4J_HOME*`/j2ee/home/applications`, the base directory for unpacked MapViewer is *$OC4J_HOME*`/j2ee/home/applications/mapviewer`.)

Before you use the MapViewer JavaBean, you should examine the Javadoc-generated API documentation and try the JSP demo:

- Javadoc documentation for the MapViewer bean API is available at a URL with the following format:

  ```
  http://host:port/mapviewer/mapclient
  ```

  In this format, *host* and *port* indicate where OC4J or Oracle Fusion Middleware listens for incoming requests.

- A demo supplied with MapViewer shows how to use the bean. After you have set up the MapViewer demo data set (which can be downloaded from the Oracle Technology Network) by importing it into a database and running all necessary scripts, you can try the JSP demo. The URL for the JSP demo has the following format:

  ```
  http://host:port/mapviewer/demo/mapinit.jsp
  ```

  In this format, *host* and *port* indicate where OC4J or Oracle Fusion Middleware listens for incoming requests. This JSP page confirms the MapViewer service URL and then proceeds to the real demo page, `map.jsp`.

4.3 Using the MapViewer Bean

To use the MapViewer bean, you must create the bean (see Section 4.3.1), after which you can invoke methods to do the following kinds of operations:

- Set up parameters of the current map request (see Section 4.3.2)
- Add themes or features to the current map request (see Section 4.3.3)
- Add dynamically defined styles to a map request (see Section 4.3.4)
- Manipulate the themes in the current map request (see Section 4.3.5)
- Send a request to the MapViewer service (see Section 4.3.6)
- Extract information from the current map response (see Section 4.3.7)
- Obtain information about data sources (see Section 4.3.8)
- Query nonspatial attributes in the current map window (see Section 4.3.9)
- Use optimal methods for thick clients (see Section 4.3.10)

The sections about methods for kinds of operations provide introductory information about what the bean can do. For detailed descriptions of each method, including its parameters and return type, see the Javadoc-generated API documentation (described in Section 4.2).

4.3.1 Creating the MapViewer Bean

The first step in any planned use of the MapViewer bean is to create the bean, as shown in the following example:

```
import oracle.lbs.mapclient.MapViewer;
MapViewer mv = new MapViewer("http://my_corp.com:8888/mapviewer/omserver");
```

The only parameter to the constructor is a URL to an actual MapViewer service. Unless you change it to something else using `setServiceURL`, the MapViewer service at this URL will receive all subsequent requests from this bean. When a MapViewer bean is created, it contains an empty current map request. There are a few parameters in the current request that are initialized with default values, such as the width and height of the map image and the background color for maps. These default values are explained in the XML API element and attribute descriptions in Chapter 3.

4.3.2 Setting Up Parameters of the Current Map Request

As explained in Chapter 3, a map request can have many parameters that affect the final look of the generated map image. When you use the MapViewer JavaBean, such parameters can be set through a group of methods whose names start with *set*. Many of these parameters have a corresponding method that starts with *get*. For example, `setAntiAliasing` sets antialiasing on or off, and `getAntiAliasing` returns the current antialiasing setting.

The methods for setting parameters of the current map request include the following:

- `setAntiAliasing(boolean aa)` specifies whether or not the map should be rendered using the antialiasing technique.
- `setBackgroundColor(java.awt.Color bg)` sets the background color for the map to be generated.
- `setBackgroundImageURL(java.lang.String bgImgUrl)` sets the URL for the background image to be rendered in the map.
- `setBaseMapName(java.lang.String name)` sets the name of the base map to be rendered before any explicitly added themes.
- `setBoundingThemes(String[] themeNames, double borderMargin, boolean preserveAspectRatio)` sets the bounding themes for the current map request. Any previous center point and box settings will be cleared as a result of calling this method.
- `setBox(double xmin, double ymin, double xmax, double ymax)` sets the map query window box in the data coordinate space. Any previous center point and size settings will be lost as a result of calling this method.
- `setCenter(double cx, double cy)` sets the center point for this map request. The coordinates must be in the user data space.
- `setCenterAndSize(double cx, double cy, double size)` sets the map center and size for the map to be generated. All data must be in the user data space.
- `setDataSourceName(java.lang.String name)` sets the name of the data source to be used when loading data for the map.
- `setDefaultStyleForCenter(java.lang.String defRenderStyleName, java.lang.String defLabelStyleName, java.lang.String defLabel, double[] defRadii)` sets the default styling and labeling information for the center (point) of the map. Each subsequent map generated will

have its center point rendered and optionally labeled with circles of the specified radii.

- `setDeviceSize(java.awt.Dimension dsz)` sets the image dimension of the map to be generated.
- `setFullExtent()` tells the MapViewer server not to impose any center and size restriction for the next map request. This effectively removes the current map center and size settings. The resulting map will be automatically centered at the full extent of all features being displayed.
- `setImageFormat(int f)` sets the image format that MapViewer should use when generating the map. For JSP pages, you should always set it to `FORMAT_PNG_URL` or `FORMAT_GIF_URL`.
- `setImageScaling(boolean is)` specifies whether images in an image theme should automatically be rescaled to fit the current query window. The default is `TRUE`. If you specify `FALSE`, the images will be rendered without any scaling by MapViewer; however, the original query window may be slightly modified to allow other (vector) themes to overlay properly with the images. In all cases, the map center is not changed.
- `setMapLegend(java.lang.String legendSpec)` sets the map legend (in XML format) to be plotted with current map. The legend must be specified in the `legendSpec` parameter, in the format for the `<legend>` element that is documented in Section 3.2.11.
- `setMapLegend(java.lang.String fill, java.lang.String fillopacity, java.lang.String stroke, java.lang.String profile, java.lang.String position, java.lang.String fontFamily, java.lang.String[][][] legenddata)` sets the map request legend to be plotted with current map. The `legenddata` attribute contains the legend items, and its structure is `String [x][y][z] legenddata`, where `x` is the number of legend columns, `y` is the number of column items, and `z` is the legend attributes (index 0 = legend text, index 1 = style name, index 2 = is title or not, index 3 = tab, index 4 = is separator or not).
- `setMapLegend(java.lang.String fill, java.lang.String fillopacity, java.lang.String stroke, java.lang.String profile, java.lang.String position, java.lang.String[][][] legenddata)` is the same as the preceding method, but without the `fontFamily` attribute.
- `setMapRequestSRID(int d)` sets the map request output SRID, which must match an SRID value in the MDSYS.CS_SRS table. Themes whose SRID value is different from the map request SRID will be automatically converted to the output SRID if the theme SRID is not null or not equal to 0. If no map request SRID is defined (equal to zero), MapViewer will use the theme's SRID as reference, but no transformation will be performed if the themes have different SRID values.
- `setMapResultFileName(String mapFile)` sets the name of the resulting map image file on the server side. If the name is set to null (the default), MapViewer will generate map image files based on the prefix `omsmap` and a counter value. You do not need to specify the extension (`.gif` or `.png`) when specifying a custom map file name.
- `setMapTitle(java.lang.String title)` sets the map title for the map to be generated.
- `setServiceURL(java.lang.String url)` sets the MapViewer service URL.

- `setSize(double size)` sets the height (size) in the user data space for the map to be generated.
- `setShowSVGNavBar(boolean s)` specifies whether or not to show the built-in SVG navigation bar. The default value is `TRUE` (that is, show the navigation bar).
- `setSVGOnClick(java.lang.String onClick)` sets the `onClick` function for an SVG map. The `onClick` function is a JavaScript function defined in the web page in which the SVG map is embedded. The `onClick` function is called whenever the SVG map is clicked if both theme feature selection and window selection are disabled. For information about using JavaScript functions with SVG maps, see Appendix B.
- `setSVGShowInfo(boolean info)` specifies whether or not to display hidden information when the mouse moves over features for which hidden information is provided. If its value is `TRUE` (the default), hidden information is displayed when the mouse moves over such features; if it is set to `FALSE`, hidden information is not displayed when the mouse moves over such features. Regardless of the value, however, hidden information is always rendered in an SVG map; this method only controls whether hidden information can be displayed.
- `setSVGZoomFactor(double zfactor)` sets the zoom factor for an SVG map. The zoom factor is the number by which to multiply the current zoom ratio for each integer increment (a zoomin operation) in the zoom level. The inverse of the zoom factor value is used for each integer decrement (a zoomout operation) in the zoom level. For example, if the `zfactor` value is 2 (the default), zooming in from zoom level 4 to 5 will enlarge the detail by two; for example, if 1 inch of the map at zoom level 4 represents 10 miles, 1 inch of the map at zoom level 5 will represent 5 miles. The zoom ratio refers to the relative scale of the SVG map, which in its original size (zoom level 0) has a zoom ratio of 1.
- `setSVGZoomLevels(int zlevels)` sets the number of zoom levels for an SVG map.
- `setSVGZoomRatio(double s)` sets the zoom factor to be used when an SVG map is initially loaded. The default value is 1, which is the original map size (zoom level 0). Higher zoom ratio values show the map zoomed in, and lower values show the map zoomed out.
- `setWebProxy(java.lang.String proxyHost, java.lang.String proxyPort)` sets the web proxy to be used when connecting to the MapViewer service. This is needed only if there is a firewall between the web service and this bean.

You can remove the map legend from the current map request by calling the `deleteMapLegend` method.

4.3.3 Adding Themes or Features to the Current Map Request

Besides specifying a base map to be included in a map request, you can add themes or individual point and linear features, such as a point of interest or a dynamically generated route, to the current map request. The themes can be predefined themes whose definitions are stored in the database, or dynamic themes where you supply the actual query string when you add the theme to the current request.

There are several kinds of dynamic themes: to retrieve geometric data (JDBC theme), to retrieve image data (image theme), to retrieve GeoRaster data (GeoRaster theme), to retrieve network data (network theme), and to retrieve topology data (topology theme). For dynamic themes and features, you must explicitly specify the styles you

want to be used when rendering them. Being able to add dynamic themes and features gives you flexibility in adapting to application development needs.

The methods for adding themes or features to the current map request have names that start with *add*, and they include the following:

- `addGeoRasterTheme` and its variants add GeoRaster data to the current map request. In some cases you supply the query string to retrieve the raster data; in other cases you supply the necessary GeoRaster information to retrieve a specific image. (Section 2.3.4 explains GeoRaster themes.)
- `addImageTheme` and its variants add an image theme, for which you must supply the query string for retrieving the image data to be rendered as part of the map. (Section 2.3.3 explains image themes.)
- `addJDBCTheme` and its variants add a JDBC theme, for which you must supply the query string for retrieving the geometric data. (Section 2.3.2 explains JDBC themes.)
- `addLinearFeature` and its variants add a single linear feature (line string) to the current map request. You must specify a rendering style. You can specify the labeling text and text style for drawing the label, and you can also specify if the label will always be present regardless of any overlapping. The coordinates must be in the user data space. There is no limit to the number of linear features that you can add.
- `addLinksWithinCost` adds a network theme to the current map request; the theme will be a result of the within-cost analysis on network data. The within-cost analysis finds all nodes that are within a specified cost, and generates the shortest path to each node.
- `addNetworkLinks` adds network links to the current map request as a network theme, for which you must supply the rendering styles.
- `addNetworkNodes` adds the network nodes to the current map request as a network theme, for which you must supply the rendering styles.
- `addNetworkPaths` adds the network paths to the current map request as a network theme, for which you must supply the rendering styles.
- `addNetworkTheme` and its variants add the network links, nodes, and paths to the current map request as a network theme, for which you must supply the rendering styles. (Section 2.3.5 explains network themes.)
- `addPointFeature` and its variants add a single feature that is a point to the current map request. This point will be rendered using the supplied rendering style on the map after all themes have been rendered. You can optionally supply a labeling text to be drawn alongside the point feature, and you can specify if the label will always be present regardless of any overlapping. You can also supply an array of radii (the units are always in meters), in which case a series of circles will be drawn centering on the point. The coordinates `x` and `y` must be in the user data space. You can assign attribute values to the point feature for use with an advanced style. For oriented point features, you can specify orientation parameters. There is no limit to the number of point features you can add.
- `addPredefinedTheme` and its variants add a predefined theme to the current map request.
- `addShortestPath` and its variants add a network theme to the current map request; the theme will be a result of the shortest-path analysis on a network data. You must supply the necessary parameters for the shortest-path algorithm.

- addThemesFromBaseMap(java.lang.String basemap) adds all predefined themes in the specified base map to the current map request. This has an advantage over setBaseMapName, in that you can manipulate the themes for the current map request, as explained in Section 4.3.5.
- addTopologyDebugTheme and its variants add the topology data structure as a topology debug-mode theme to the current map request. You must supply the rendering styles for the edges, nodes, and faces. (Section 2.3.6 explains topology themes, including the debug mode.)
- addTopologyTheme adds the topology features as a topology theme to the current map request. You must supply the query string. (Section 2.3.6 explains topology themes.)

You can remove all added point and linear features by calling the removeAllPointFeatures and removeAllLinearFeatures methods, respectively.

4.3.4 Adding Dynamically Defined Styles to a Map Request

Besides the styles stored on the USER_SDO_STYLES view, you can also add dynamically defined (temporary) styles to a map request. These dynamically defined styles provide temporary information for the map request, and they should always be added to the map request before it is sent to the server.

The methods for adding dynamically defined styles to the map request have names that start with *add*. Effective with release 11g, you can add any kind of dynamically defined style to a map request with the single method addStyle, which has the following definition:

```
public void addStyle(java.lang.String name,
                     StyleModel tempStyle)
```

In the preceding definition, StyleModel is an interface defined in the Java client package oracle.mapviewer.share.style. This package and the oracle.mapviewer.share.stylex package also contain concrete style model classes that represent the definitions of all types of styles supported by MapViewer. See the Javadoc reference documentation for information about these packages.

The following code excerpt shows how to use the addStyle method and the ColorStyleModel class to add a dynamic color style to a map request:

```
import oracle.lbs.mapclient.*;
import oracle.mapviewer.share.*
…
ColorStyleModel csm = new ColorStyleModel();
csm.setFillColor(new Color(255, 0, 0, 100));
csm.setStrokeColor(new Color(0, 0, 255, 100));
mapViewer.addStyle("my_color", csm);
```

As an alternative to using the addStyle method, you can use the following methods for adding specific types of styles:

- addBucketStyle(java.lang.String name, java.lang.String low, java.lang.String high, int nbuckets, java.lang.String []styleName) adds a bucket style with equal intervals, for which you specify the range values, the number of buckets, and the style name for each bucket.
- addCollectionBucketStyle(java.lang.String name, java.lang.String []label, java.lang.String []styleName,

java.lang.String [][]value) adds a collection bucket style, for which you specify the label, the style name, and the values for each bucket.

- addColorSchemeStyle(java.lang.String name, java.lang.String baseColor, java.lang.String strokeColor, java.lang.String low, java.lang.String high, int nbuckets) adds a color scheme style with equal intervals, for which you specify the color parameters, the range values, and the number of buckets.
- addColorSchemeStyle(java.lang.String name, java.lang.String baseColor, java.lang.String strokeColor, java.lang.String []label, java.lang.String []low, java.lang.String []high) adds a color scheme style, for which you specify the color parameters and the range values.
- addColorStyle(java.lang.String name, java.lang.String stroke, java.lang.String fill, int strokeOpacity, int fillOpacity) adds a color style with the specified color parameters.
- addImageAreaStyleFromURL(java.lang.String styleName, java.lang.String imgURL) adds a GIF or JPEG image as an area symbol to the MapViewer client.
- addImageAreaStyleFromURL(java.lang.String styleName, java.lang.String imgURL, java.lang.String lineStyle) adds a GIF or JPEG image as an area symbol to the MapViewer client. You can also specify parameters for stroking the boundary of the area being filled.
- addImageMarkerStyleFromURL(java.lang.String styleName, java.lang.String imgURL, java.lang.String strokeColor, float strokeWidth, int strokeOpacity) adds a GIF image as a marker symbol to the MapViewer client.
- addImageMarkerStyleFromURL(java.lang.String styleName, java.lang.String imgURL) adds a GIF image as a marker symbol to the MapViewer client. You can also specify parameters for the desired width and height of the image when applied to features on a map, as well as the font properties of any text label that will go inside or on top of the marker.
- addImageMarkerStyleFromURL(java.lang.String styleName, java.lang.String imgURL) adds a GIF image as a marker symbol to the MapViewer client.
- addImageMarkerStyleFromURL(java.lang.String styleName, java.lang.String imgURL, int desiredWidth, int desiredHeight, java.lang.String fontName, int fontSize, java.lang.String fontStyle, java.lang.String fontWeight, java.lang.String fontColor) adds a GIF image as a marker symbol to the MapViewer client. You can also specify parameters for the desired width and height of the image when applied to features on a map, as well as the font properties of any text label that will go inside or on top of the marker.
- addLineStyle(java.lang.String name, java.lang.String fill, java.lang.String strokeWidth, boolean hasBase, java.lang.String baseFill, java.lang.String baseStroke, java.lang.String baseDash, boolean hasParallel, java.lang.String fillParallel, java.lang.String strokeParallel, boolean hasHashMark, java.lang.String fillHash, java.lang.String dashHash) adds a line style to the MapViewer client.

- `addLineStyle(java.lang.String name, java.lang.String fill, java.lang.String strokeWidth, boolean hasBase, java.lang.String baseFill, java.lang.String baseStroke, java.lang.String baseDash, boolean hasParallel, java.lang.String fillParallel, java.lang.String strokeParallel, boolean hasHashMark, java.lang.String fillHash, java.lang.String dashHash, java.lang.String measureMarker, double measurePosition, int measureSize)` adds a line style to the MapViewer client.
- `addMarkerStyle(java.lang.String name, int mktype, java.lang.String strokeColor, java.lang.String fillColor, java.lang.String markerWidth, java.lang.String markerHeight, java.lang.String coords, java.lang.String radius)` adds a vector marker style with the given parameters. The available vector marker style types are `MARKER_POLYGON`, `MARKER_POLYLINE`, `MARKER_CIRCLE`, and `MARKER_RECT`.
- `addTextStyle(java.lang.String name, java.lang.String style, java.lang.String family, java.lang.String size, java.lang.String weight, java.lang.String fill)` adds a text style with the specified parameters.
- `addVariableMarkerStyle(java.lang.String name, java.lang.String []label, java.lang.String baseMarker, int startSize,int increment, java.lang.String []low, java.lang.String []high)` adds a variable marker style, for which you specify the parameters for the base marker, and also the label and the values for each bucket.

You can remove a dynamically defined style from the current map request by calling the `deleteStyle(java.lang.String name)` method, or you can remove all dynamically defined styles from the current map request by calling the `removeAllDynamicStyles` method.

4.3.5 Manipulating Themes in the Current Map Request

After you add themes using any of the methods that start with *add*, you can manipulate them, performing such operations as listing their names, moving them up or down in rendering order for the current request, and even disabling themes and enabling themes that had been disabled. However, you cannot manipulate themes that are implicitly included when you set a base map (using the `setBaseMapName` method), because the list of themes in the base map is not actually included until the MapViewer service processes the request.

The methods for manipulating themes in the current map request include the following:

- `deleteAllThemes` deletes all added themes from the current map request.
- `deleteTheme(java.lang.String name)` deletes an explicitly added theme from the current map request.
- `enableThemes(java.lang.String[] themes)` enables all themes whose names appear in the supplied list.
- `getActiveTheme(double currentScale)` gets the name of the active theme, that is, the top theme on the current display map.
- `getEnabledThemes` gets a list of all themes that are currently enabled.

- `getThemeEnabled(java.land.String themeName)` determines whether or not a specified theme is currently enabled.
- `getThemeNames` returns an ordered list of names of themes that have been explicitly added to the current map request.
- `getThemePosition(java.lang.String name)` returns the position in the rendering sequence of an explicitly added theme.
- `getThemeVisibleInSVG(java.lang.String name)` determines whether or not a specified theme is currently visible in an SVG map. (If the theme is not visible, it is hidden.)
- `hasThemes` checks to see if the current map request has any explicitly added themes. For example, if you have only set the name of the base map in the current request, but have not added any other theme through one of the *add*Theme* methods, this method returns `FALSE`.
- `moveThemeDown(int index)` moves a theme down one position in the list of themes to be rendered, so that it is rendered later.
- `moveThemeUp(int index)` moves a theme up one position in the list of themes to be rendered, so that it is rendered sooner.
- `setAllThemesEnabled(boolean v)` sets all themes to be enabled or disabled.
- `setGeoRasterThemePolygonMask(java.lang.String name,double []coords)` sets the polygon mask to be applied on the GeoRaster theme. The GeoRaster area outside the polygon mask will be transparent. The coordinates are defined as x1,y1,x2,y2, ... The mask coordinates must be in the data coordinate space.
- `setLabelAlwaysOn(boolean labelAlwaysOn, java.lang.String name)` controls whether or not MapViewer labels all features in a theme even if two or more labels will overlap in the display of a theme. (MapViewer always tries to avoid overlapping labels.) If `labelAlwaysOn` is `TRUE`, MapViewer displays the labels for all features even if two or more labels overlap. If `labelAlwaysOn` is `FALSE`, when it is impossible to avoid overlapping labels, MapViewer disables the display of one or more labels so that no overlapping occurs.
- `setNetworkThemeLabels(java.lang.String name, java.lang.String linkLabelStyle, java.lang.String linkLabelColumn, java.lang.String nodeLabelStyle, java.lang.String nodeLabelColumn, java.lang.String pathLabelStyle, java.lang.String pathLabelColumn)` sets network theme label parameters for links, nodes, and paths. The attribute column name must be an existing attribute of the link, node, and path tables.
- `setThemeAlpha(java.lang.String themeName, float alpha)` sets the transparency value for an image theme.
- `setThemeEnabled(boolean v, java.lang.String themeName)` sets a specified theme to be enabled or disabled in the current map request.
- `setThemeFastUnpickle(java.lang.String name, boolean noUnpickler)` specifies whether to use the MapViewer fast unpickling algorithm (`TRUE`, the default) or the generic JDBC conversion algorithm (`FALSE`) to convert SDO_GEOMETRY objects fetched from the database into a Java object accessible to MapViewer. The MapViewer fast unpickling algorithm improves performance, but occasionally the coordinates may lose some precision (around 0.00000005), which can be significant in applications where all precision digits of each coordinate must be kept. The generic JDBC conversion algorithm is slower

than the MapViewer fast unpickling process, but there is never any loss of precision.

- `setThemeOnClickInSVG (java.lang.String theme, java.lang.String onClickFunction)` sets the theme's `onClick` function for an SVG map. The `onClick` function is a JavaScript function defined in the web page in which the SVG map is embedded. The `onClick` function is called whenever the SVG map is clicked if both theme feature selection and window selection are disabled. For information about using JavaScript functions with SVG maps, see Appendix B.
- `setThemeScale(java.lang.String name, double minScale, double maxScale)` sets the minimum and maximum scale values for displaying a theme.
- `setThemeSelectableInSVG (java.lang.String theme, boolean sel)` sets the theme to be selectable (`TRUE`) or not selectable (`FALSE`) in an SVG map. If the theme is set to selectable, any feature of the theme can be selected in the SVG map by clicking on it. If the feature is selected, its color is changed and its ID (its rowid by default) is recorded. You can get a list of the ID values of all selected features by calling the JavaScript function `getSelectedIdList()` defined in the SVG map. For information about using JavaScript functions with SVG maps, see Appendix B.
- `setThemeUnitAndResolution(java.lang.String themeName, java.lang.String unit, double resolution)` sets the unit and resolution values for an image theme.
- `setThemeVisible(java.lang.String name, boolean vis)` sets the theme to be visible (TRUE) or hidden (FALSE) in an SVG map. If the theme is set to be hidden, the theme will be still rendered, but will be invisible.

4.3.6 Sending a Request to the MapViewer Service

As an application developer, you typically issue a new map request as a result of certain user input (such as a mouse click on the currently displayed map) or after you have modified some aspect of the map request (such as setting a new background color). In fact, you can issue a map request any time you want, as long as you do not overwhelm the middle-tier MapViewer service with too many rapid requests from the MapViewer bean or beans. The MapViewer service tries to process requests in the order in which they arrive; if you send a second request before receiving the response from the first one, MapViewer continues to process the first request completely before starting to process the second request.

Any modifications to the current map request, such as changing to a new background color or moving a theme down in the rendering sequence, do not take effect in the map display until you send the map request, at which time the MapViewer service actually receives the request and processes it.

The methods for sending a map request to the MapViewer service include the following:

- `run` sends the current map request to the MapViewer service, and obtains a map response as sent back by the MapViewer service.
- `pan(int x, int y)` pans to the specified device point. Each coordinate is in the screen or display unit, in this case, pixel.
- `zoomIn(double factor)` zooms in on the map without changing the other map request parameters.

- `zoomIn(int x, int y, double factor)` zooms in on the specified device point.
- `zoomIn(int x1, int y1, int x2, int y2)` zooms in on the specified device rectangle.
- `zoomOut(double factor)` zooms out on the current map without changing the other map request parameters.
- `zoomOut(int x, int y, double factor)` zooms out and recenters the current map.

Each of these methods assembles a single XML map request document based on all properties of the current map request, and then sends it to the MapViewer service. After the MapViewer bean receives the response from the MapViewer service, the bean does any necessary postprocessing and makes the response ready for your use.

As an alternative to using these methods, you can formulate an XML request string outside the bean, and then use the `sendXMLRequest(java.lang.String req)` method to send the request to the MapViewer service. However, if you use this method, you are responsible for receiving and unpacking the response using the `getXMLResponse` method, and for parsing and interpreting the response string yourself. The state of the bean remains unchanged, because the methods are only making use of the bean's capability to open an HTTP connection to send and receive documents over the connection.

All methods described in this section throw an exception if any unrecoverable error occurs during the transmission of the request or response, or in the MapViewer service during processing. You are responsible for taking care of such exceptions in any way you consider appropriate, such as by trying the request again or by reporting the problem directly to the user.

4.3.7 Extracting Information from the Current Map Response

You can extract information, such as the generated map image or the URL for the image, from the current map response. The methods for extracting information from the map response include the following:

- `getGeneratedMapImage` returns the actual map image data contained in the response from the MapViewer service. You must have set the image format to `FORMAT_RAW_COMPRESSED` using the `setImageFormat` method. The `getGeneratedMapImage` method is primarily used in thick clients, although you may also use it in a JavaServer Page or a servlet (for example, to save the image in a format that is not supported by MapViewer).
- `getGeneratedMapImageURL` returns the URL to the currently generated map image in the application server. You must have set the image format to `FORMAT_PNG_URL` or `FORMAT_GIF_URL` using the `setImageFormat` method.
- `getMapMBR` returns the MBR (minimum bounding rectangle) for the currently generated map, in the user's data space.
- `getMapResponseString` returns the last map response in XML format.

4.3.8 Obtaining Information About Data Sources

The MapViewer bean has methods that you can use to obtain information about data sources. These methods include the following:

- `dataSourceExists(java.lang.String dsrc)` checks if a given data source exists in (that is, is known to) the MapViewer service.

- `getDataSources()` lists the currently available data sources in the server. This method lists only the names and no other details about each data source (such as database host or user login information).

4.3.9 Querying Nonspatial Attributes in the Current Map Window

It is often necessary to query nonspatial attributes that are associated with the spatial features being displayed in the current map image. For example, assume that you just issued a map request to draw a map of all customer locations within a certain county or postal code. The next logical step is to find more information about each customer being displayed in the resulting map image. In the JSP or HTML environment, because you get only an image back from the MapViewer service, you will need another round-trip to the service to fetch the nonspatial information requested by the user. This section describes a set of methods that can help you do just that. (You can, however, obtain both the nonspatial attribute values of a certain theme and the resulting map image in a single request when the bean is used in a standalone Java application or applet environment, as described in Section 4.3.10.)

A typical situation is that the user clicks on a feature on the displayed map and then wants to find out more (nonspatial attributes) about the feature. This action can be essentially implemented using a query with the desired nonspatial attributes in its SELECT list, and a spatial filter as its WHERE clause. The spatial filter is an Oracle Spatial and Graph SQL operator that checks if the geometries in a table column (the column of type SDO_GEOMETRY in the customer table) spatially interact with a given target geometry (in this case, the user's mouse-click point). The spatial filter in the WHERE clause of the query selects and returns only the nonspatial attributes associated with the geometries that are being clicked on by the user.

You will need to call an Oracle Spatial and Graph operator to perform the filtering; however, you can use the MapViewer bean-based API to obtain information, and to preassemble the spatial filter string to be appended to the WHERE clause of your query. The `identify` method simplifies the task even further.

The methods for querying nonspatial attributes in the current map window include the following:

- `doQuery` and variants execute a supplied SQL query and return an array of strings representing the result set. These are convenient methods to issue your own query without manually opening a JDBC connection to the database from the bean.
- `doQueryInMapWindow` and variants are extensions of `doQuery` and its variants. They automatically subject the user-supplied query to a spatial filtering process using the current map window.
- `getSpatialFilter(java.lang.String spatialColumn, int srid, boolean pre9i)` returns a spatial filter string that can be used as a WHERE clause condition in formulating your own queries in the current map window context. The spatial filter evaluates to `TRUE` for any geometries that are being displayed in the entire map window. You can use this method to obtain information about every spatial feature of a theme that is being displayed.
- `getSpatialFilter(java.lang.String spatialColumn, int srid, double xl, double yl, double xh, double yh, boolean pre9i)` returns a spatial filter string that can be used as a query condition in formulating your queries in the given window. This filter evaluates to `TRUE` for all geometries that interact with the supplied `(xl,yl, xh,yh)` data window. The window is not in device or screen coordinate space, but in the user's data space; therefore, you must first call the `getUserPoint` method to convert the user's mouse-click

point to a point in the user data space before using the `getSpatialFilter` method.

- `getUserPoint(int x, int y)` returns the user data space point corresponding to the mouse click.
- `getUserPoint(int x, int y, java.lang.String dataSource, int outSRID)` returns the user data space point corresponding to the mouse click, using the specified coordinate system (SRID value).
- `getUserPoint(int x, int y, java.lang.String dataSource, java.lang.String themeName)` returns the user data space point corresponding to the mouse click, using the coordinate system (SRID value) associated with the specified theme.
- `getWhereClauseForAnyInteract(java.lang.String spatialColumn, int srid, double x, double y)` returns geometries that have any interaction with a specified point in the user's data space. This provides a WHERE clause string that will use a more precise spatial filtering method than the one provided by the `getSpatialFilter` method.
- `getWhereClauseForAnyInteract(java.lang.String spatialColumn, int srid, double xl, double yl, double xh, double yh)` returns the WHERE clause that can be used to find geometries that have any interaction with the specified user space window. It is similar to the `getSpatialFilter` method, but uses a more precise version of the Oracle Spatial and Graph filtering method.
- `identify` and variants provide a convenient method for identifying nonspatial attributes. This is desirable if you do not need more flexibility and control over how a nonspatial attribute query should be formulated. As with the `doQuery` methods, all `identify` methods return a `double String` array that contains the result set of the query.

4.3.10 Using Optimal Methods for Thick Clients

When you use the MapViewer bean in a JavaServer Page in an HTML file, a second round-trip to the MapViewer service is needed to obtain nonspatial attributes of features being displayed. It is also true that with a JavaServer Page in an HTML file, even if most themes remain unchanged from one map request to the next (such as when zooming in to the center of a map), all themes must still be reprocessed each time the MapViewer service processes the page, which causes the data for each theme to be retrieved again from the database. (This is mainly due to the stateless nature of the MapViewer service and the insufficient mechanism provided in the JSP context for handling user interaction, which must be based on the request/response model.)

However, when you are working in a thick client environment, such as with J2SE (Java 2 Platform Standard Edition) applications and applets, you can reduce the processing and bandwidth overhead when using the bean. This is primarily because in such environments you have greater control of how content (including the map) should be displayed, you can better respond to the user's interaction, and you can devote more resources to maintaining the states on the client side.

A key optimization available only to the thick client context is **live features**. Basically, a live feature is a spatial feature that originates from the MapViewer service but exists in the thick client. Each live feature contains the actual shape representing the geometry data, and a set of nonspatial attributes that the user might be interested in. To obtain live features, a thick client must set its parent theme to be clickable. When a map request is sent to the MapViewer service with a clickable theme, MapViewer does

not attempt to render features for that theme in the resulting map. Rather, the set of features that would have been drawn as part of the map is returned to the requesting client as an array of live feature objects. The rest of the map is still rendered and transmitted as a single image to the client. After the client has received both the live features and the base image, it must render the live features on top of the accompanying map image, using one of the methods described later in this section.

One benefit of using live features is that the thick client will not need to issue a request for the clickable theme every time a map request is sent. For example, if the request is to zoom in to the current map, the client can determine for each live feature if it should be displayed in the zoomed-in map image. Another, and probably more significant, advantage is that the nonspatial attributes for all features displayed in the map are now readily available to the user. For example, as the user moves the mouse over a range of features on the map, the thick client can immediately get the corresponding nonspatial attributes and display them in a pop-up window that follows the mouse trail. No round-trip to the MapViewer service is needed for this type of action, and the feedback to the user is more responsive.

The methods that are optimal for thick clients include the following:

- `drawLiveFeatures(java.awt.Graphics2D g2, java.awt.Color stroke, java.awt.Color fill, double pointRadius, double strokeWidth)` draws all live features that are returned to this client from MapViewer.
- `getLiveFeatureAttrs(int x, int y, int tol)` gets the nonspatial attributes of the feature being clicked on by the user.
- `getNumLiveFeatures` returns the number of live features currently available.
- `hasLiveFeatures` checks if there are any live (clickable) features.
- `highlightFeatures` and variants highlight all live features that are intersecting the user-specified rectangle. These methods also let you specify the style for highlighting features.
- `isClickable(java.lang.String themeName)` checks if the specified theme is clickable (that is, if users can click on the theme to get its attributes).
- `setClickable(boolean v, java.lang.String themeName)` sets the theme clickable (so that its features will be available to the client as live features that users can click on and get attributes of).

To obtain a set of features and keep them live at the thick client, you must first call `setClickable` to set the theme whose features you want to be live. Then, after you issue the current map request, the bean processes the response from the MapViewer service, which (if it succeeded) contains both a base map image and an array of `LiveFeature` instances. You can then call `getGeneratedMapImage` to get and draw the base image, and use `drawLiveFeatures` to render the set of live features on top of the base map. If the user clicks or moves the mouse over a certain position on the map, you can use the `highlightFeatures` method to highlight the touched features on the map. You can also use the `getLiveFeatureAttrs` method to obtain the associated nonspatial attributes of the features being highlighted. You do not have direct access to the `LiveFeature` instances themselves.

The behavior of calling the methods described in this section in the context of JSP pages is not defined.

5 MapViewer JSP Tag Library

Deprecated Feature: MapViewer JSP Library: The MapViewer JSP library is deprecated, and will not be included in future releases of the documentation.

Instead, you are encouraged to use the MapViewer Java API, which is more comprehensive and up to date. Moreover, if you prefer to use tags, consider using the GeoMap tags in the JDeveloper Application Development Framework (ADF).

This chapter explains how to submit requests to MapViewer using JavaServer Pages (JSP) tags in an HTML file. Through an XML-like syntax, the JSP tags provide a set of important (but not complete) MapViewer capabilities, such as setting up a map request, zooming, and panning, as well as identifying nonspatial attributes of user-clicked features.

Note: The MapViewer JSP tag library will not work with Oracle9*i*AS Release 9.0.2 or the standalone OC4J Release 9.0.2. The minimum version required is Oracle9*i*AS Release 9.0.3 or the standalone OC4J Release 9.0.3.

You can develop a location-based application by using any of the following approaches:

- Using the XML API (see Chapter 3)
- Using the JavaBean-based API (see Chapter 4)
- Using JSP files that contain XML or HTML tags, or both, and that include custom Oracle-supplied JSP tags (described in this chapter)

Creating JSP files is often easier and more convenient than using the XML or JavaBean-based API, although the latter two approaches give you greater flexibility and control over the program logic. However, you can include calls to the Java API methods within a JavaServer Page, as is done with the call to the `getMapTitle` method in Example 5–1 in Section 5.3.

All MapViewer JSP tags in the same session scope share access to a single MapViewer bean.

This chapter contains the following major sections:

- Section 5.1, "Using MapViewer JSP Tags"

- Section 5.2, "MapViewer JSP Tag Reference Information"
- Section 5.3, "JSP Example (Several Tags) for MapViewer"

5.1 Using MapViewer JSP Tags

Before you can use MapViewer JSP tags, you must perform one or two steps, depending on whether or not the web application that uses the tags will be deployed in the same OC4J instance that is running MapViewer.

1. If the web application will be deployed in the same OC4J instance that is running MapViewer, skip this step and go to Step 2.

   If the web application will be deployed in a separate OC4J instance, you must copy the `mvclient.jar` file (located in the $MAPVIEWER`/web/WEB-INF/lib` directory) and the `mvtaglib.tld` file (located in the $MAPVIEWER`/web/WEB-INF` directory) to that OC4J instance's application deployment directory. Then you must define a `<taglib>` element in your application's `web.xml` file, as shown in the following example:

   ```
   <taglib>
       <taglib-uri>
         http://xmlns.oracle.com/spatial/mvtaglib
       </taglib-uri>
       <taglib-location>
         /WEB-INF/mvtaglib.tld
       </taglib-location>
    </taglib>
   ```

2. Import the tag library (as you must do with any JSP page that uses custom tags), by using the `taglib` directive at the top of the JSP page and before any other MapViewer tags. For example:

   ```
   <%@ taglib uri="http://xmlns.oracle.com/spatial/mvtaglib"
                prefix="mv" %>
   ```

   The `taglib` directive has two parameters:

   - `uri` is the unique name that identifies the MapViewer tag library, and its value must be `http://xmlns.oracle.com/spatial/mvtaglib`, because it is so defined in the MapViewer `web.xml` initialization file.
   - `prefix` identifies the prefix for tags on the page that belong to the MapViewer tag library. Although you can use any prefix you want as long as it is unique in the JSP page, `mv` is the recommended prefix for MapViewer, and it is used in examples in this guide.

     The following example shows the `mv` prefix used with the setParam tag:

     ```
     <mv:setParam title="Hello World!" bgcolor="#ffffff"
        width="500" height="375" antialiasing="true"/>
     ```

The tags enable you to perform several kinds of MapViewer operations:

- To create the MapViewer bean and place it in the current session, use the init tag, which must come before any other MapViewer JSP tags.
- To set parameters for the map display and optionally a base map, use the setParam tag.
- To add themes and a legend, use the addPredefinedTheme, addJDBCTheme, importBaseMap, and makeLegend tags.

- To get information, use the getParam, getMapURL, and identify tags.
- To submit the map request for processing, use the run tag.

5.2 MapViewer JSP Tag Reference Information

This section provides detailed information about the Oracle-supplied JSP tags that you can use to communicate with MapViewer. Table 5–1 lists each tag and briefly describes the information specified by the tag.

Table 5–1 JSP Tags for MapViewer

Tag Name	Explanation
init	Creates the MapViewer bean and places it in the current session. Must come before any other MapViewer JSP tags.
setParam	Specifies one or more parameters for the current map request.
addPredefinedTheme	Adds a predefined theme to the current map request.
addJDBCTheme	Adds a dynamically defined theme to the map request.
importBaseMap	Adds the predefined themes that are in the specified base map to the current map request.
makeLegend	Creates a legend (map inset illustration) drawn on top of the generated map.
getParam	Gets the value associated with a specified parameter for the current map request.
getMapURL	Gets the HTTP URL for the currently available map image, as generated by the MapViewer service.
identify	Gets nonspatial attribute (column) values associated with spatial features that interact with a specified point or rectangle on the map display, and optionally uses a marker style to identify the point or rectangle.
run	Submits the current map request to the MapViewer service for processing. The processing can be to zoom in or out, to recenter the map, or to perform a combination of these operations.

Except where noted, you can use JSP expressions to set tag attribute values at runtime, using the following format:

```
<mv:tag attribute="<%= jspExpression %>" >
```

The following sections (in alphabetical order by tag name) provide reference information for all parameters available for each tag: the parameter name, a description, and whether or not the parameter is required. If a parameter is required, it must be included with the tag. If a parameter is not required and you omit it, a default value is used.

Short examples are provided in the reference sections for JSP tags, and a more comprehensive example is provided in Section 5.3.

5.2.1 addJDBCTheme

The addJDBCTheme tag adds a dynamically defined theme to the map request. (It performs the same operation as the <jdbc_query> element, which is described in Section 3.2.9.)

Table 5–2 lists the addJDBCTheme tag parameters.

Table 5–2 addJDBCTheme Tag Parameters

Parameter Name	Description	Required
name	Name for the dynamically defined theme. Must be unique among all themes already added to the associated MapViewer bean.	Yes
min_scale	The value to which the display must be zoomed in for the theme to be displayed, as explained in Section 2.4.1. If min_scale and max_scale are not specified, the theme is displayed for all map scales, if possible given the display characteristics.	No
max_scale	The value beyond which the display must be zoomed in for the theme not to be displayed, as explained in Section 2.4.1. If min_scale and max_scale are not specified, the theme is displayed for all map scales, if possible given the display characteristics.	No
spatial_column	Column of type SDO_GEOMETRY containing geometry objects for the map display	Yes
srid	Coordinate system (SDO_SRID value) of the data to be rendered. If you do not specify this parameter, a null coordinate system is assumed.	No
datasource	Name of the data source instance that contains information for connecting to the database	Yes[1]
jdbc_host	Host name for connecting to the database	Yes[1]
jdbc_port	Port name for connecting to the database	Yes[1]
jdbc_sid	SID for connecting to the database	Yes[1]
jdbc_user	User name for connecting to the database	Yes[1]
jdbc_password	Password for connecting to the database	Yes[1]
jdbc_mode	The Oracle JDBC driver (thin or oci8) to use to connect to the database. The default is thin.	No
asis	If set to TRUE, MapViewer does not attempt to modify the supplied query string. If FALSE (the default), MapViewer embeds the SQL query as a subquery of its spatial filter query. (For more information and an example, see Section 3.2.9.)	No
render_style	Name of the style to be used to render the spatial data retrieved for this theme. For point features the default is a red cross rotated 45 degrees, for lines and curves it is a black line 1 pixel wide, and for polygons it is a black border with a semitransparent dark gray interior.	No
label_style	Name of the text style to be used to draw labeling text on the spatial feature for this theme. If you specify label_style, you must also specify label_column. If you do not specify label_style, no label is drawn for the spatial feature of this theme.	No
label_column	The column in the SELECT list of the supplied query that contains the labeling text for each feature (row). If label_style is not specified, any label_column value is ignored.	No

[1] You must specify either datasource or the combination of jdbc_host, jdbc_port, jdbc_sid, jdbc_user, and jdbc_password.

The following example creates a new dynamic theme named `bigCities`, to be executed using the `mvdemo` data source and specifying the LOCATION column as containing spatial data. Note that the greater-than (>) character in the WHERE clause is valid here.

```
<mv:addJDBCTheme name="bigCities" datasource="mvdemo"
                     spatial_column="location">
       SELECT location, name FROM cities WHERE pop90 > 450000
</mv:addJDBCTheme>
```

5.2.2 addPredefinedTheme

The `addPredefinedTheme` tag adds a predefined theme to the current map request. (It performs the same operation as the `<theme>` element, which is described in Section 3.2.20.) The predefined theme is added at the end of the theme list maintained in the associated MapViewer bean.

Table 5–3 lists the `addPredefinedTheme` tag parameters.

Table 5–3 addPredefinedTheme Tag Parameters

Parameter Name	Description	Required
`name`	Name of the predefined theme to be added to the current map request. This theme must exist in the USER_SDO_THEMES view of the data source used by the associated MapViewer bean.	Yes
`datasource`	Name of the data source from which the theme will be loaded. If you do not specify this parameter, the default data source for the map request is used.	No
`min_scale`	The value to which the display must be zoomed in for the theme to be displayed, as explained in Section 2.4.1. If `min_scale` and `max_scale` are not specified, the theme is displayed for all map scales, if possible given the display characteristics.	No
`max_scale`	The value beyond which the display must be zoomed in for the theme not to be displayed, as explained in Section 2.4.1. If `min_scale` and `max_scale` are not specified, the theme is displayed for all map scales, if possible given the display characteristics.	No

The following example adds the theme named `THEME_DEMO_CITIES` to the current Map request:

```
<mv:addPredefinedTheme name="THEME_DEMO_CITIES"/>
```

5.2.3 getMapURL

The `getMapURL` tag gets the HTTP URL (uniform resource locator) for the currently available map image, as generated by the MapViewer service. This map image URL is kept in the associated MapViewer bean, and it does not change until after the run tag is used.

The `getMapURL` tag has no parameters.

The following example displays the currently available map image, using the `getMapURL` tag in specifying the source (`SRC` keyword value) for the image:

```
<IMG SRC="<mv:getMapURL/>" ALIGN="top">
```

5.2.4 getParam

The `getParam` tag gets the value associated with a specified parameter for the current map request.

Table 5–4 lists the `getParam` tag parameter.

Table 5–4 getParam Tag Parameter

Parameter Name	Description	Required
`name`	Name of the parameter whose value is to be retrieved. It must be one of the valid parameter names for the setParam tag. The parameter names are case-sensitive. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes

The following example displays the value of the `title` parameter for the current map request:

```
<P> The current map title is: <mv:getParam name="title"/> </P>
```

5.2.5 identify

The `identify` tag gets nonspatial attribute (column) values associated with spatial features that interact with a specified point or rectangle on the map display, and it optionally uses a marker style to identify the point or rectangle. For example, if the user clicks on the map and you capture the X and Y coordinate values for the mouse pointer when the click occurs, you can retrieve values of nonspatial columns associated with spatial geometries that interact with the point. For example, if the user clicks on a point in Chicago, your application might display the city name, state abbreviation, and population of Chicago, and it might also display a "city" marker on the map near where the click occurred.

The attributes are returned in a `String[][]` array of string arrays, which is exposed by this tag as a scripting variable.

The list of nonspatial columns to fetch must be provided in the tag body, in a comma-delimited list, which the MapViewer bean uses to construct a SELECT list for its queries.

You can optionally associate a highlighting marker with each feature that is identified by using the style attribute and specifying a marker style. To display a new map that includes the highlighting markers, use the getMapURL tag.

Table 5–5 lists the `identify` tag parameters.

Table 5–5 identify Tag Parameters

Parameter Name	Description	Required
`id`	Name for the scripting variable through which the returned nonspatial attribute values will be exposed. The first array contains the column names. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes
`datasource`	Name of the MapViewer data source from which to retrieve the nonspatial information.	No
`table`	Name of the table containing the column identified in `spatial_column`. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes

Table 5–5 (Cont.) identify Tag Parameters

Parameter Name	Description	Required
`spatial_column`	Column of type SDO_GEOMETRY containing geometry objects to be checked for spatial interaction with the specified point or rectangle. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes
`srid`	Coordinate system (SDO_SRID value) of the data in `spatial_column`. If you do not specify this parameter, a null coordinate system is assumed.	No
`x`	The X ordinate value of the point; or the X ordinate value of the lower-left corner of the rectangle if `x2` and `y2` are specified.	Yes
`y`	The Y ordinate value of the point; or the Y ordinate value of the lower-left corner of the rectangle if `x2` and `y2` are specified.	Yes
`x2`	The X ordinate value of the upper-right corner of the rectangle.	No
`y2`	The Y ordinate value of the upper-right corner of the rectangle.	No
`style`	Name of the marker style to be used to draw a marker on features that interact with the specified point or rectangle. To display a new map that includes the highlighting markers, use the getMapURL tag.	No

The following example creates an HTML table that contains a heading row and one row for each city that has any spatial interaction with a specified point (presumably, the city where the user clicked). Each row contains the following nonspatial data: city name, population, and state abbreviation. The `String[][]` array of string arrays that holds the nonspatial information about the associated city or cities is exposed through the scripting variable named `attrs`. The scriptlet after the tag loops through the array and outputs the HTML table (which in this case will contain information about one city).

```
<mv:identify id="attrs" style="M.CYAN PIN"
           table="cities" spatial_column="location"
           x="100" y="200">
    City, Pop90 Population, State_abrv State
</mv:identify>

<%
    if(attrs!=null && attrs.length>0)
    {
      out.print("<CENTER> <TABLE border=\"1\">\n");
      for(int i=0; i<attrs.length; i++)
      {
        if(i==0) out.print("<TR BGCOLOR=\"#FFFF00\">");
        else out.print("<TR>\n");
        String[] row = attrs[i];
        for(int k=0; k<row.length; k++)
          out.print("<TD>"+row[k]+"</TD>");
          out.print("</TR>\n");
       }
      out.print("</TABLE></CENTER>");
    }
%>
```

5.2.6 importBaseMap

The `importBaseMap` tag adds the predefined themes that are in the specified base map to the current map request. (This has the same effect as using the setParam tag with the `basemap` attribute.)

Table 5–6 lists the `importBaseMap` tag parameter.

Table 5–6 importBaseMap Tag Parameter

Parameter Name	Description	Required
`name`	Name of the base map whose predefined themes are to be added at the end of the theme list for the current map request. This base map must exist in the USER_SDO_MAPS view of the data source used by the associated MapViewer bean.	Yes

The following example adds the predefined themes in the base map named `demo_map` at the end of the theme list for the current map request:

```
<mv:importBaseMap name="demo_map"/>
```

5.2.7 init

The `init` tag creates the MapViewer bean and places it in the current session. This bean is then shared by all other MapViewer JSP tags in the same session. The `init` tag must come before any other MapViewer JSP tags.

Table 5–7 lists the `init` tag parameters.

Table 5–7 init Tag Parameters

Parameter Name	Description	Required
`url`	The uniform resource locator (URL) of the MapViewer service. It must be in the form `http://`*`host:port`*`/mapviewer/omserver`, where *`host`* and *`port`* identify the system name and port, respectively, on which Oracle Fusion Middleware or OC4J listens.	Yes
`datasource`	Name of the MapViewer data source to be used when requesting maps and retrieving mapping data. If you have not already created the data source, you must do so before using the `init` tag. (For information about defining a data source, see Section 1.5.2.14.)	Yes
`id`	Name that can be used to refer to the MapViewer bean created by this tag. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes

The following example creates a data source named `mvdemo` with an `id` value of `mvHandle`:

```
<mv:init url="http://mycompany.com:8888/mapviewer/omserver"
          datasource="mvdemo" id="mvHandle"/>
```

5.2.8 makeLegend

The `makeLegend` tag accepts a user-supplied XML legend specification and creates a standalone map legend image. The legend image is generated by the MapViewer

service, and a URL for that image is returned to the associated MapViewer bean. This tag exposes the URL as a scripting variable.

The body of the tag must contain a <legend> element. See Section 3.2.11 for detailed information about the <legend> element and its attributes.

Table 5–8 lists the makeLegend tag parameters.

Table 5–8 makeLegend Tag Parameters

Parameter Name	Description	Required
id	Name for the scripting variable that can be used to refer to the URL of the generated legend image. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes
datasource	Name of the MapViewer data source from which to retrieve information about styles specified in the legend request	No
format	Format of the legend image to be created on the server. If specified, must be GIF_URL (the default) or PNG_URL.	No

The following example creates a single-column legend with the id of myLegend, and it displays the legend image.

```
<mv:makeLegend id="myLegend">
  <legend bgstyle="fill:#ffffff;stroke:#ff0000" profile="MEDIUM">
    <column>
      <entry text="Legend:" is_title="true"/>
      <entry style="M.STAR" text="center point"/>
      <entry style="M.CITY HALL 3" text="cities"/>
      <entry is_separator="true"/>
      <entry style="C.ROSY BROWN STROKE" text="state boundary"/>
      <entry style="L.PH" text="interstate highway"/>
      <entry text="County population density:"/>
      <entry style="V.COUNTY_POP_DENSITY" tab="1"/>
    </column>
  </legend>
</mv:makeLegend>

<P> Here is the map legend: <IMG SRC="<%=myLegend%>"> </P>
```

5.2.9 run

The run tag submits the current map request to the MapViewer service for processing. The processing can be to zoom in or out, to recenter the map, or to perform a combination of these operations.

The run tag does not output anything to the JSP page. To display the map image that MapViewer generates as a result of the run tag, you must use the getMapURL tag.

Table 5–9 lists the run tag parameters.

Table 5–9 run Tag Parameters

Parameter Name	Description	Required
`action`	One of the following values to indicate the map navigation action to be taken: `zoomin` (zoom in), `zoomout` (zoom out), or `recenter` (recenter the map). For `zoomin` or `zoomout`, `factor` specifies the zoom factor; for all actions (including no specified action), `x` and `y` specify the new center point; for all actions (including no specified action), `x2` and `y2` specify (with `x` and `y`) the rectangular area to which to crop the resulting image. If you do not specify an action, the map request is submitted for processing with no zooming or recentering, and with cropping only if `x`, `y`, `x2`, and `y2` are specified.	No
`x`	The X ordinate value of the point for recentering the map, or the X ordinate value of the lower-left corner of the rectangular area to which to crop the resulting image if `x2` and `y2` are specified	No
`y`	The Y ordinate value of the point for recentering the map, or the Y ordinate value of the lower-left corner of the rectangular area to which to crop the resulting image if `x2` and `y2` are specified	No
`x2`	The X ordinate value of the upper-right corner of the rectangular area to which to crop the resulting image	No
`y2`	The Y ordinate value of the upper-right corner of the rectangular area to which to crop the resulting image	No
`factor`	Zoom factor: a number by which the current map size is multiplied (for `zoomin`) or divided (for `zoomout`). The default is 2. This parameter is ignored if `action` is not `zoomin` or `zoomout`.	No

The following example requests a zooming in on the map display (with the default zoom factor of 2), and recentering of the map display at coordinates (100, 250) in the device space.

```
<mv:run action="zoomin" x="100" y="250"/>
```

5.2.10 setParam

The `setParam` tag specifies one or more parameters for the current map request. You can set all desired parameters at one time with a single `setParam` tag, or you can set different parameters at different times with multiple `setParam` tags. Most of the parameters have the same names and functions as the attributes of the `<map_request>` root element, which is described in Section 3.2.1.1. The parameter names are case-sensitive.

Table 5–10 lists the `setParam` tag parameters.

Table 5–10 setParam Tag Parameters

Parameter Name	Description	Required
`antialiasing`	When its value is `TRUE`, MapViewer renders the map image in an antialiased manner. This usually provides a map with better graphic quality, but it may take longer for the map to be generated. The default value is `FALSE` (for faster map generation).	No

Table 5–10 (Cont.) setParam Tag Parameters

Parameter Name	Description	Required
`basemap`	Base map whose predefined themes are to be rendered by MapViewer. The definition of a base map is stored in the user's USER_SDO_MAPS view, as described in Section 2.9.1. Use this parameter if you will always need a background map on which to plot your own themes and geometry features.	No
`bgcolor`	The background color in the resulting map image. The default is water-blue (RGB value #A6CAF0). It must be specified as a hexadecimal value.	No
`bgimage`	The background image (GIF or JPEG format only) in the resulting map image. The image is retrieved at runtime when a map request is being processed, and it is rendered before any other map features, except that any `bgcolor` value is rendered before the background image.	No
`centerX`	X ordinate of the map center in the data coordinate space	No
`centerY`	Y ordinate of the map center in the data coordinate space	No
`height`	The height (in device units) of the resulting map image	No
`imagescaling`	When its value is `TRUE` (the default), MapViewer attempts to scale the images to fit the current querying window and the generated map image size. When its value is `FALSE`, and if an image theme is included directly or indirectly (such as through a base map), the images from the image theme are displayed in their original resolution. This parameter has no effect when no image theme is involved in a map request.	No
`size`	Vertical span of the map in the data coordinate space	No
`title`	The map title to be displayed on the top of the resulting map image	No
`width`	The width (in device units) of the resulting map image	No

The following example uses two `setParam` tags. The first `setParam` tag sets the background color, width, height, and title for the map. The second `setParam` tag sets the center point and vertical span for the map.

```
<mv:setParam bgcolor="#ff0000" width="800" height="600"
                title="My Map!"/>

<mv:setParam centerX="-122.35" centerY="37.85" size="1.5"/>
```

5.3 JSP Example (Several Tags) for MapViewer

This section presents an example of using JSP code to perform several MapViewer operations.

Example 5–1 initializes a MapViewer bean, sets up map request parameters, issues a request, and displays the resulting map image. It also obtains the associated MapViewer bean and places it in a scripting variable (`myHandle`), which is then accessed directly in the statement:

```
Displaying map:  <B> <%=myHandle.getMapTitle()%> </B>
```

Example 5–1 MapViewer Operations Using JSP Tags

```
<%@ page contentType="text/html" %>
<%@ page session="true" %>
<%@ page import="oracle.lbs.mapclient.MapViewer" %>

<%@ taglib uri="http://xmlns.oracle.com/spatial/mvtaglib"
             prefix="mv" %>
<HTML>
<BODY>
Initializing client MapViewer bean. Save the bean in the session
using key "mvHandle"....<P>
 <mv:init url="http://my_corp.com:8888/mapviewer/omserver"
            datasource="mvdemo" id="mvHandle"/>

Setting MapViewer parameters...<P>
<mv:setParam title="Hello World!" bgcolor="#ffffff" width="500" height="375"
antialiasing="true"/>

Adding themes from a base map...<P>
<mv:importBaseMap name="density_map"/>

Setting initial map center and size...<P>
<mv:setParam centerX="-122.0" centerY="37.8" size="1.5"/>

Issuing a map request... <P>
<mv:run/>

<%
  // Place the MapViewer bean in a Java variable.
  MapViewer myHandle = (MapViewer) session.getAttribute("mvHandle");
%>

Displaying map:  <B> <%=myHandle.getMapTitle()%> </B>
<IMG SRC="<mv:getMapURL/>"  ALIGN="top"/>
</BODY>
</HTML>
```

6 MapViewer PL/SQL API

Deprecated Feature: MapViewer PL/SQL API: The MapViewer PL/SQL API is deprecated, and will not be included in future releases of the documentation.

Instead, you are encouraged to either (A) use one of the other supported MapViewer APIs, or (B) use Oracle Application Express (APEX) with the MapViewer JavaScript API.

This chapter describes the PL/SQL application programming interface (API) to MapViewer. This API consists of the PL/SQL package SDO_MVCLIENT, which is intended for Oracle Database users who want to access MapViewer services from inside an Oracle database. This package exposes most capabilities of MapViewer, and it handles all communication with the actual MapViewer server running on a middle tier on behalf of a user making map requests.

6.1 Installing the SDO_MVCLIENT Package

The SDO_MVCLIENT package is not currently installed by default. Instead, you must install the package by running two SQL scripts that are supplied with MapViewer: `sdomvclh.sql` and `sdomvclb.sql`.

To perform the installation, go to the `sql` directory under the *`$MAPVIEWER_HOME`* directory, start SQL*Plus, connect as a user that has the DBA role (for example, SYSTEM), and enter the following commands:

```
@sdomvclh
@sdomvclb
```

After you run these two scripts, exit SQL*Plus, go to the *`$MAPVIEWER_HOME`*`/web/WEB-INF/lib` directory, and use the shall command `loadjava` to load the `mvclient.jar` file into the MDSYS schema. For example (and assuming the use of the SYSTEM account with the password manager):

```
loadjava -force -schema mdsys -grant PUBLIC -user system mvclient.jar
Password: password
```

Database users can now use the SDO_MVCLIENT PL/SQL package, as described in Section 6.2.

6.2 Using the SDO_MVCLIENT Package

The SDO_MVCLIENT PL/SQL package communicates with a remote MapViewer service through the HTTP protocol. For each database session, it maintains a global MapViewer client handle, as well as the current map request and map response objects.

The usage model for the SDO_MVCLIENT package is almost identical to that of MapViewer JavaBean-based API (described in Chapter 4). Most methods implemented in the MapViewer JavaBean-Based API (`oracle.lbs.mapclient.MapViewer`) are available in this PL/SQL package, and the package uses the same method names and parameters used by the JavaBean-Based API. For usage and reference information about specific functions or procedures, see the description of the associated JavaBean-Based API. methods and interfaces in Chapter 4.

The basic workflow for accessing the MapViewer service through this PL/SQL package is almost identical to that for using the Java client API, except for some initial setup. Follow these major steps, each of which is described in a section to follow:

1. Grant network access (see Section 6.2.1).
2. Create a MapViewer client handle (see Section 6.2.2).
3. Prepare a map request (see Section 6.2.3).
4. Send the request to the MapViewer service (see Section 6.2.4).
5. Optionally, extract information from the map request (see Section 6.2.5).

6.2.1 Granting Network Access

Grant network access permission to each database user that will use the SDO_ MVCLIENT package. For example, if database user SCOTT will need to use the package, you must enter a statement in the following general form while connected as a user with DBA privileges:

```
call dbms_java.grant_permission('SCOTT', 'SYS:java.net.SocketPermission',
                                'www.mycorp.com',
                                'connect, resolve' );
```

In the preceding example, change `www.mycorp.com` to the host on which the MapViewer service is running.

Depending on the Oracle Database version, you may also need to grant network access to the database user MDSYS, which owns the SDO_MVCILENT package. To do this, enter a statement in the following general form while connected as a user with DBA privileges:

```
call dbms_java.grant_permission('MDSYS', 'SYS:java.net.SocketPermission',
                                'www.mycorp.com:8888',
                                'connect, resolve' );
```

In the preceding example, change `www.mycorp.com` to the host on which the MapViewer service is running.

The call to dbms_java.grant_permission needs to be done only once for each affected database user; the permission remains valid for all subsequent database sessions for these users.

6.2.2 Creating a MapViewer Client Handle

Before each database session, you must create a MapViewer client handle before using any functions or procedures of the SDO_MVCLIENT package. The following example creates a MapViewer client handle:

```
connect scott
Enter password: password
call sdo_mvclient.createmapviewerclient(
     'http://www.mycorp.com:8888/mapviewer/omserver') ;
```

The preceding example creates, in the current session, a unique MapViewer client handle to the MapViewer service URL `http://www.mycorp.com:8888/mapviewer/omserver`. To use this example, change `www.mycorp.com` to the host on which the MapViewer service is running.

After you have created a MapViewer client handle, you can perform the following query to check that MapViewer is running correctly:

```
select sdo_mvclient.getdatasources() datasources from dual;
```

The SQL function `sdo_mvclient.getdatasources()` is part of the MapViewer PL/SQL package API; and when it is executed, it connects to the remote MapViewer server and gets a list of all known data sources. If the installation is successful and the MapViewer server is running, the result of the preceding example is output similar to the following, with the string array containing the names of the data sources that are defined in the MapViewer server:

```
DATASOURCES
--------------------------------------------------------------------------------
SDO_1D_STRING_ARRAY('mvdemo', 'wms')
```

6.2.3 Preparing a Map Request

Call various methods in the PL/SQL package to prepare a map request, which will eventually be sent to the MapViewer server for processing. You can specify the basic characteristics of the map to be created, and you can add temporary styles and multiple themes to the current map request.

Example 6–1 sets the data source and other map characteristics, adds a dynamically defined color style to the map request, and manipulates a theme.

Example 6–1 Preparing a Map Request

```
call sdo_mvclient.setDataSourceName('mvdemo');
call sdo_mvclient.setImageFormat('PNG_URL');
call sdo_mvclient.setAntiAliasing('true');
call sdo_mvclient.setBaseMapName('qa_map') ;
call sdo_mvclient.setBox(-122.3615, 37.4266, -121.1615, 37.6266);
call sdo_mvclient.setDevicesize(500,400);

call sdo_mvclient.addColorStyle('colorst', 'blue', 'yellow', 100,100);

select sdo_mvclient.addJDBCTheme('mvdemo', 'theme1',
       'select geom from states where state_abrv = ''CA''',
       'geom', '8307', 'C.RED', null, null, 'FALSE') from dual ;
```

6.2.4 Sending the Request to the MapViewer Service

The following example effectively sends the current map request to the remote MapViewer server for processing. It will return after the request has been processed at the server.

```
select sdo_mvclient.run() from dual;
```

You can also use such methods as `sdo_mvclient.zoomIn()` and `sdo_mvclient.zoomOut()` to get zoomed maps.

6.2.5 Extracting Information from the Map Request

The following example extracts the URL string of the generated map image:

```
select sdo_mvclient.getgeneratedMapImageURL() from dual;
```

After you have the URL of the map image, you can do various things, such as fetch and store the image in a database table, or present the map with other information on a HTML page.

7

MapViewer XML Requests: Administrative and Other

The main use of MapViewer is for processing various map requests. However, MapViewer also accepts through its XML API various administrative (non-map) requests, such as to add a data source, as well as other (general-purpose) requests useful in developing applications, such as to list available themes, base maps, and tile layers. All MapViewer administrative requests require that you log in to the MapViewer administration (Admin) page, for which there is a link on the main MapViewer page; the general-purpose requests can be made from an application without the requirement to log in. This section describes the format for each request and its response.

All XML requests are embedded in either a <non_map_request> or <map_cache_admin_request> element, and all responses are embedded in a <non_map_response> or <map_cache_admin_response> element, respectively. However, for all requests an exception may occur, in which case the response is an <oms_error> or an <mcs_error> element (described in Section 3.5).

The administrative requests are described in sections according to the kinds of tasks they perform:

- Managing Data Sources
- Managing Tile Layers
- Listing All Maps (General-Purpose)
- Listing Themes (General-Purpose)
- Listing Styles (General-Purpose)
- Managing In-Memory Caches
- Editing the MapViewer Configuration File (Administrative)
- Restarting the MapViewer Server (Administrative)

The section titles often indicate whether a request is administrative or general-purpose.

7.1 Managing Data Sources

You can add, remove, redefine, and list data sources. (For information about data sources and how to define them, see Section 1.5.2.14.)

7.1.1 Adding a Data Source (Administrative)

Note: This request is typically used during development or testing, when you want to add a data source quickly and dynamically without creating a permanent one (which would involve editing the mapViewerConfig.xml file). For production use, or to take full advantage of what MapViewer provides with a data source, you should always use a permanent data source.

The <add_data_source> element has the following definition:

```
<!ELEMENT non_map_request  add_data_source>
<!ELEMENT add_data_source  EMPTY>
  <!ATTLIST add_data_source
   name              CDATA #REQUIRED
   container_ds      CDATA #IMPLIED
   jdbc_tns_name     CDATA #IMPLIED
   jdbc_host         CDATA #IMPLIED
   jdbc_port         CDATA #IMPLIED
   jdbc_sid          CDATA #IMPLIED
   jdbc_user         CDATA #IMPLIED
   jdbc_password     CDATA #IMPLIED
   jdbc_mode         (oci8 | thin) #IMPLIED
   number_of_mappers INTEGER #REQUIRED
               >
```

The name attribute identifies the data source name. The name must be unique among MapViewer data sources. (Data source names are not case-sensitive.)

You must specify a container data source name, a net service name (TNS name), or all necessary connection information. That is, you must specify only one of the following:

- container_ds
- jdbc_tns_name
- jdbc_host, jdbc_port, jdbc_sid, jdbc_mode, jdbc_user, and jdbc_password

The container_ds attribute identifies a data source name that is defined in the Java EE container's Java Naming and Directory Interface (JNDI) namespace. For OC4J, it should be the ejb-location attribute of the data source defined in the data-source.xml file.

The jdbc_tns_name attribute identifies a net service name that is defined in the tnsnames.ora file.

The jdbc_host attribute identifies the database host system name.

The jdbc_port attribute identifies the TNS listener port number.

The jdbc_sid attribute identifies the SID for the database.

The jdbc_user attribute identifies the user to connect to (map).

The jdbc_password attribute identifies the password for the user specified with the jdbc_user attribute. MapViewer does not change this password string in any way; no conversion to upper or lower case is performed. If the database uses case-sensitive passwords, the specified password must exactly match the password in the database.

The jdbc_mode attribute identifies the JDBC connection mode: thin or oci8. If you specify oci8, you must have Oracle Client installed in the middle tier in which MapViewer is running. You do not need Oracle Client if thin is used for all of your data sources.

The number_of_mappers attribute identifies the number of map renderers to be created (that is, the number of requests that MapViewer can process at the same time) for this data source. Any unprocessed map requests are queued and eventually processed. For example, if the value is 3, MapViewer will be able to process at most three mapping requests concurrently. If a fourth map request comes while three requests are being processed, it will wait until MapViewer has finished processing one of the current requests. The maximum number of mappers for a single data source is 64.

Example 7–1 adds a data source named mvdemo by specifying all necessary connection information.

Example 7–1   Adding a Data Source by Specifying Detailed Connection Information

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <add_data_source
        name="mvdemo"
        jdbc_host="elocation.us.oracle.com"
        jdbc_port="1521"
        jdbc_sid="orcl"
        jdbc_user="scott"
        jdbc_password="password"
        jdbc_mode="thin"
        number_of_mappers="5"/>
</non_map_request>
```

Example 7–2 adds a data source named mvdemo by specifying the container data source name.

Example 7–2   Adding a Data Source by Specifying the Container Data Source

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <add_data_source
        name="mvdemo"
        container_ds="jdbc/OracleDS"
        number_of_mappers="5"/>
</non_map_request>
```

The DTD for the response to an add_data_source request has the following format:

```
<!ELEMENT non_map_response add_data_source>
<!ELEMENT add_data_source  EMPTY>
<!ATTLIST add_data_source
   succeed   (true | false) #REQUIRED
   comment   CDATA #IMPLIED
>
```

The comment attribute appears only if the request did not succeed, in which case the reason is in the comment attribute. In the following example, succeed="true" indicates that the user request has reached the server and been processed without any exception being raised regarding its validity. It does not indicate whether the user's intended action in the request was actually fulfilled by the MapViewer server. In this example, the appearance of the comment attribute indicates that the request failed,

and the string associated with the comment attribute gives the reason for the failure ("data source already exists").

```
<?xml version="1.0" ?>
 <non_map_response>
     <add_data_source succeed="true" comment="data source already exists"/>
</non_map_response>
```

7.1.2 Removing a Data Source (Administrative)

The <remove_data_source> element can be used to remove a permanent data source or a dynamically added data source. This element has the following definition:

```
<!ELEMENT non_map_request remove_data_source>
<!ELEMENT remove_data_source  EMPTY>
<!ATTLIST remove_data_source
   data_source    CDATA #REQUIRED
   jdbc_password  CDATA #REQUIRED
>
```

The data_source attribute identifies the name of the data source to be removed.

The jdbc_password attribute identifies the login password for the database user in the data source. jdbc_password is required for security reasons (to prevent people from accidentally removing data sources from MapViewer).

Removing a data source only affects the ability of MapViewer to use the corresponding database schema; nothing in that schema is actually removed.

Example 7–3 removes a data source named mvdemo.

Example 7–3 Removing a Data Source

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <remove_data_source data_source="mvdemo" jdbc_password="password"/>
</non_map_request>
```

The DTD for the response to a remove_data_source request has the following format:

```
<!ELEMENT non_map_response remove_data_source>
<!ELEMENT remove_data_source  EMPTY>
<!ATTLIST remove_data_source
   succeed  (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
 <non_map_response>
    <remove_data_source succeed="true"/>
</non_map_response>
```

7.1.3 Redefining a Data Source

Note: You should use request only during development or testing, and not for production work.

For convenience, MapViewer lets you redefine a data source. Specifically, if a data source with the same name already exists, it is removed and then added using the new definition. If no data source with the name exists, a new data source is added. If an existing data source has the same name, host, port, SID, user name, password, mode, and number of mappers as specified in the request, the request is ignored.

The <redefine_data_source> element has the following definition:

```
<!ELEMENT non_map_request redefine_data_source>
<!ELEMENT redefine_data_source  EMPTY>
<!ATTLIST redefine_data_source
   name              CDATA #REQUIRED
   container_ds      CDATA #IMPLIED
   jdbc_tns_name     CDATA #IMPLIED
   jdbc_host         CDATA #IMPLIED
   jdbc_port         CDATA #IMPLIED
   jdbc_sid          CDATA #IMPLIED
   jdbc_user         CDATA #IMPLIED
   jdbc_password     CDATA #IMPLIED
   jdbc_mode         (oci8 | thin) #IMPLIED
   number_of_mappers INTEGER #REQUIRED
>
```

The attributes and their explanations are the same as for the <add_data_source> element, which is described in Section 7.1.1.

The DTD for the response to a redefine_data_source request has the following format:

```
<!ELEMENT non_map_response redefine_data_source>
<!ELEMENT redefine_data_source  EMPTY>
<!ATTLIST redefine_data_source
   succeed  (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
 <non_map_response>
   <redefine_data_source succeed="true"/>
</non_map_response>
```

7.1.4 Listing All Data Sources (Administrative or General-Purpose)

The <list_data_sources> element lists all data sources known to the currently running MapViewer. It has the following definition:

```
<!ELEMENT non_map_request list_data_sources>
<!ELEMENT list_data_sources  EMPTY>
```

For example:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <list_data_sources/>
</non_map_request>
```

The DTD for the response to a list_data_sources request has the following format:

```
<!ELEMENT non_map_response map_data_source_list>
<!ELEMENT map_data_source_list  (map_data_source*) >
```

```
<!ATTLIST map_data_source_list
   succeed      (true|false) #REQUIRED
>
<!ELEMENT map_data_source  EMPTY>
<!ATTLIST map_data_source
   name         CDATA #REQUIRED
   container_ds CDATA #IMPLIED
   host         CDATA #IMPLIED
   sid          CDATA #IMPLIED
   port         CDATA #IMPLIED
   user         CDATA #IMPLIED
   mode         CDATA #IMPLIED
   numMappers   CDATA #REQUIRED
   >
```

For each data source:

- If the user issuing the request is logged in as a MapViewer administrator, all data source information except the password for the database user is returned.
- If the user issuing the request is *not* logged in as a MapViewer administrator, only the data source name is returned.

The following example is a response that includes information about two data sources when the request is issued by a MapViewer administrator.

```
<?xml version="1.0" ?>
<non_map_response>
<map_data_source_list succeed="true">
   <map_data_source name="mvdemo" host="elocation.us.oracle.com"
       sid="orcl" port="1521" user="scott" mode="thin" numMappers="3"/>
   <map_data_source name="geomedia" host="geomedia.us.oracle.com"
       sid="orcl" port="8160" user="scott" mode="oci8" numMappers="7"/>
</map_data_source_list>
</non_map_response>
```

The following example is a response when the same request is issued by a user that is *not* a MapViewer administrator.

```
<?xml version="1.0" ?>
<non_map_response>
<map_data_source_list succeed="true">
   <map_data_source name="mvdemo"/>
   <map_data_source name="geomedia"/>
</map_data_source_list>
</non_map_response>
```

7.1.5 Checking the Existence of a Data Source (General-Purpose)

The `<data_source_exists>` element lets you find out if a specified data source exists. It has the following definition:

```
<!ELEMENT non_map_request data_source_exists>
<!ELEMENT data_source_exists  EMPTY>
<!ATTLIST data_source_exists
   data_source   CDATA #REQUIRED
>
```

For example:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
```

```
  <data_source_exists data_source="mvdemo"/>
</non_map_request>
```

The DTD for the response to a data_source_exists request has the following format:

```
<!ELEMENT non_map_response data_source_exists>
<!ELEMENT data_source_exists  EMPTY>
<!ATTLIST data_source_exists
   succeed    (true | false) #REQUIRED
   exists     (true | false) #REQUIRED
>
```

The succeed attribute indicates whether or not the request was processed successfully.

The exists attribute indicates whether or not the data source exists.

For example:

```
<?xml version="1.0" ?>
<non_map_response>
   <data_source_exists succeed="true" exists="true"/>
</non_map_response>
```

7.2 Managing Tile Layers

All tile layer administration requests are embedded in a <map_cache_admin_request> element. Their responses are embedded in a <map_tile_server_response> element except for <get_client_config>, where the format for the response document can vary. When an error occurs, the response returns an <mcs_error> element containing corresponding error messages.

Two tile layer administration requests, <get_client_config> and <get_cache_status>, can also be sent in a non-administrative mode, in which cases you do not need to log in to the administration console.

Tile layer management tasks include the following:

- Getting Client Side Configuration
- Getting Cache Status
- Clearing, Prefetching, or Refreshing Cache
- Stopping, Resuming, or Removing an Existing Cache Administrative Task
- Getting the Status of an Administrative Request
- Creating or Redefining a Cache Instance
- Removing a Cache Instance
- Restarting the Tile Layer Cache Server
- Taking a Tile Layer Offline or Bringing It Online

7.2.1 Getting Client Side Configuration

The <get_client_config> element returns the configuration of a specified tile layer from the client side. It has the following definition:

```
<!ELEMENT map_cache_admin_request (get_client_config)>
<!ELEMENT get_client_config  EMPTY>
```

```
<!ATTLIST get_client_config
   map_cache_names   CDATA #REQUIRED
   format            (JSON|XML|OLD_JS) #IMPLIED
>
```

The map_cache_names attribute lists the tile layer names for which configurations are requested.

The format attribute specifies the document format in the response.

The following example gets the configuration for one tile layer, MVDEMO.DEMO_MAP, in XML format:

```
<?xml version="1.0" standalone="yes"?>
<map_cache_admin_request>
  <get_client_config map_cache_names="MVDEMO.DEMO_MAP" format="XML" />
/>
```

The response may be the following:

```
<map_tile_server_response>
  <map_tile_layer_config>
    <map_tile_layer name="DEMO_MAP" data_source="MVDEMO" format="PNG"
transparent="false">
      <coordinate_system srid="8307" type="GEODETIC" distConvFactor="0.0"
minX="-180.0" minY="-90.0" maxX="180.0" maxY="90.0"/>
      <zoom_levels>
        <zoom_level level="0" name="" scale="1.0E7" tile_width="4.285714285714286"
tile_height="4.285714285714286" tile_image_width="189" tile_image_height="189"/>
<zoom_level level="1" name="" scale="3000000.0" tile_width="1.0843373493975903"
tile_height="0.9608052463016623" tile_image_width="158" tile_image_height="140"/>
      </zoom_levels>
    </map_tile_layer>
  </map_tile_layer_config>
</map_tile_server_response>
```

The <get_client_config> request also works in non-administrative mode, and the request can be sent over an xml_request string. For example:

```
http://localhost:7001/mapviewer/mcserver?xml_request=<map_cache_admin_
request><get_client_config map_cache_names="MVDEMO.DEMO_MAP" format="XML" /></map_
cache_admin_request>
```

7.2.2 Getting Cache Status

The <get_cache_status> element lists the status of all tile layers on the server. It has the following definition:

```
<!ELEMENT map_cache_admin_request (get_cache_status)>
<!ELEMENT get_cache_status  EMPTY>
```

The following example lists the status of all tile layers on the server:

```
<?xml version="1.0" standalone="yes"?>
<map_cache_admin_request>
  <get_cache_status/>
</map_cache_admin_request>
```

The DTD for the response to a get_cache_status request has the following format:

```
<!ELEMENT map_tile_server_response (tile_server_status)>
<!ELEMENT tile_server_status  (cache_instance*)>
<!ELEMENT cache_instance EMPTY>
```

```
<!ATTLIST cache_instance
   data_source  CDATA #REQUIRED
   name         CDATA #REQUIRED
   type         (internal|external) #REQUIRED
   base_map     CDATA #REQUIRED
   zoom_levels  PCDATA #REQUIRED
   status       (ready|not ready) #REQUIRED
   online       (true|false) #REQUIRED
>
```

The `name` attribute identifies the name of the tile layer in that data source.

The `type` attribute indicates if the map source is an internal or an external map source.

The `zoom_levels` attribute indicates the total zoom levels in this tile layer.

The `base_map` attribute specifies the base map name used by this tile layer for generating map tiles, if the type is an internal map source.

The `status` attribute indicates if the tile layer is ready or not ready.

The `online` attribute indicates if the tile layer is online (`true`) or offline (`false`).

The following is an example of a `get_cache_status` response:

```
<map_tile_server_response>
  <tile_server_status>
    <cache_instance data_source="MVDEMO_PC" name="DEMO_MAP" type="internal" base_
map="DEMO_MAP" zoom_levels="10" status="ready" online="true"/>
    <cache_instance data_source="MVDEMO" name="WMTS" type="internal" base_
map="WMTS" zoom_levels="15" status="ready" online="true"/>
  </tile_server_status>
<map_tile_server_response>
```

The <get_cache_status> request also works in non-administrative mode, and the request can be sent over an `xml_request` string. For example:

```
http://localhost:7001/mapviewer/mcserver?xml_request=<map_cache_admin_
request><get_cache_status /></map_cache_admin_request>
```

7.2.3 Clearing, Prefetching, or Refreshing Cache

The <tile_admin_task> element can be used to request the server to clear the cached tiles, fetch tiles for caching, or refresh tiles (that is, delete any existing tiles and then fetch tiles). For these purposes, it has the following definition:

```
<!ELEMENT map_cache_admin_request (tile_admin_task)>
<!ELEMENT tile_admin_task  (schedule?)>
<!ELEMENT schedule EMPTY>
<!ATTLIST tile_admin_task
   operations   (clear_tiles|fetch_tiles|refresh_tiles) #REQUIRED
   map_tile_layer CDATA #REQUIRED
   zoom_levels    PCDATA #REQUIRED
   bounding_box   PCDATA #REQUIRED
>
<!ATTLIST schedule
   start_time       PCDATA #REQUIRED
   repeat_interval  PCDATA #IMPLIED
   duration         PCDATA #IMPLIED
>
```

The `operation` attribute identifies the operation to be performed: `clear_tiles`, `fetch_tiles`, or `refresh_tiles`.

The `map_tile_layer` attribute indicates the tile layer name. It takes the data source name as its prefix, followed by a period (.) and then the tile layer name. For example: `mvdemo.demo_map`

The `zoom_levels` attribute specifies a list of zoom levels for which to perform the operation.

The `bounding_box` attribute defines a rectangular region within which to perform the operation.

The `start_time` attribute indicates when the scheduled task will start.

The `repeat_interval` attribute indicates how often, in minutes, this operation will be performed. A value of 0 (zero) indicates that it is a one-time operation.

The `duration` attribute indicates how long, in minutes, an operation will last. If this attribute is empty of not specified, the task will take as long as it needs to finish.

The following example sends a `fetch_tiles` request:

```
<?xml version="1.0" standalone="yes"?>
<map_cache_admin_request>
  <tile_admin_task
    operation="fetch_tiles"
    map_tile_layer="mvdemo.demo_map"
    zoom_levels="7,8"
    bounding_box="-71.6,42.35,-71.58,42.37">
    <schedule start_time="2013-03-20 9:00:00" repeat_interval="0" duration="" />
  </tile_admin_task>
</map_cache_admin_request>
```

The DTD for the `fetch_tiles` response is as follows:

```
<!ELEMENT map_tile_server_response (request_status, request_id, estimates)>
<!ELEMENT request_status (#CDATA)>
<!ELEMENT request_id (#CDATA)>
<!ELEMENT estimates EMPTY>
<!ATTLIST estimates
   tile_remaining       CDATA #REQUIRED
   time_remaining       CDATA #REQUIRED
   disk_space_required  CDATA #REQUIRED
>
```

The `tile_remaining` attribute indicates the number of tiles remaining to be fetched.

The `time_remaining` attribute indicates the estimated time, in seconds, needed to fetch the remaining tiles.

The `disk_space_required` attribute indicates the estimated storage space, in bytes, for the remaining tiles.

For example:

```
<map_tile_server_response>
  <request_status>Submitted</request_status>
  <request_id>62</request_id>
  <estimates tile_remaining="144" time_remaining="288" disk_space_
required="990576"/>
</map_tile_server_response>
```

7.2.4 Stopping, Resuming, or Removing an Existing Cache Administrative Task

The <tile_admin_task> element can be used to request the server to stop, resume, or remove tasks. For these purposes, it has the following definition:

```
<!ELEMENT map_cache_admin_request (tile_admin_task)>
<!ELEMENT tile_admin_task  EMPTY>
<!ATTLIST tile_admin_task
   operations   (stop_task|resume_task|remove_task) #REQUIRED
   task_id      PCDATA #REQUIRED
>
```

The `operation` attribute identifies the operation to be performed: `stop_task`, `resume_task`, or `remove_task`.

The `task_id` attribute indicates the task identifier for which to perform the operation.

The following example sends a request to stop a task:

```
<?xml version="1.0" standalone="yes"?>
<map_cache_admin_request>
  <tile_admin_task
    operation="stop_task"
    task_id="12">
  </tile_admin_task>
</map_cache_admin_request>
```

The DTD for the stop_task response to a request is as follows:

```
<!ELEMENT map_tile_server_response (#CDATA)>
```

For example:

```
<map_tile_server_response>Succeeded.</map_tile_server_response>
```

7.2.5 Getting the Status of an Administrative Request

The <get_admin_request_status> element shows the status of a previously sent administrative request. It has the following definition:

```
<!ELEMENT map_cache_admin_request (get_admin_request_status)>
<!ELEMENT get_admin_request_status  EMPTY>
<!ATTLIST  get_admin_request_status
   data_source    CDATA #REQUIRED
   map_tile_layer CDATA #REQUIRED
>
```

The following example gets the administrative request status of a specified tile layer:

```
<?xml version="1.0" standalone="yes"?>
<map_cache_admin_request>
  <get_admin_request_status data_source="mvdemo" map_tile_layer="demo_map"/>
</map_cache_admin_request>
```

The DTD for the response to a `get_admin_request_status` request has the following format:

```
<!ELEMENT map_cache_admin_request  (tile_admin_task)>
<!ELEMENT tile_admin_task  (bound, schedule?, task_progress)>
<!ELEMENT bound (gml:Box)>
<!ELEMENT schedule  EMPTY>
<!ELEMENT task_progress  (zoom_level+)>
<!ELEMENT zoom_level EMPTY>
```

```
<!ATTLIST tile_admin_task
   id              PCDATA #REQUIRED
   type            (CLEAR, PREFETCH, REFRESH) #REQUIRED
   data_source     CDATA #REQUIRED
   map_tile_layer  CDATA #REQUIRED
   zoom_levels     PCDATA #REQUIRED
>
<!ATTLIST  schedule
   start_time       PCDATA #REQUIRED
   repeat_interval  PCDATA #IMPLIED
   duration         PCDATA #IMPLIED
   cron_string      CDATA #IMPLIED
>
<!ATTLIST  zoom_level
   level                 PCDATA #REQUIRED
   total_tile_number     PCDATA #REQUIRED
   processed_tile_number PCDATA #REQUIRED
   failed_tile_number    PCDATA #REQUIRED
   tile_x                PCDATA #REQUIRED
   tile_y                PCDATA #REQUIRED
>
```

The `level` attribute indicates the current zoom level of this task.

The `total_tile_number` attribute indicates the total number of tiles of this task.

The `processed_tile_number` attribute indicates the already processed number of tiles.

The `failed_tile_number` attribute indicates the number of tiles that have failed in the operation.

The `tile_x` and `tile_y` attributes indicate the last processed tile's coordinates (x, y) in the tile mesh code coordinate system. A value of -1 indicates that the task has not been started.

For example:

```
<tile_admin_task id="67" type="PREFETCH" data_source="MVDEMO" map_tile_
layer="DEMO_MAP" zoom_levels="8,9">
  <bound>
    <gml:Box xmlns:gml="http://www.opengis.net/gml" srsName="SDO:8307">
      <gml:coordinates decimal="." cs="," ts=" ">
        -71.60,42.35 -71.58,42.37
      </gml:coordinates>
    </gml:Box>
  </bound>
  <schedule repeat_interval="0" duration="0" start_time="2015-03-20 09:00:00"
cron_string=""/>
  <task_progress>
    <zoom_level level="8" total_tile_number="144" processed_tile_number="0"
failed_tile_number="0" tile_x="-1"tile_y="-1"/>
    <zoom_level level="9" total_tile_number="1122" processed_tile_number="0"
failed_tile_number="0" tile_x="-1"tile_y="-1"/>
  </task_progress>
</tile_admin_task>
```

7.2.6 Creating or Redefining a Cache Instance

The `<tile_admin_task>` element can be used to request the tile layer server to create or redefine a cache instance. For these purposes it has the following definition:

```
<!ELEMENT map_cache_admin_request (create_cache_instance, redefine_cache_
instance)>
<!ELEMENT create_cache_instance  (cache_instance)>
<!ELEMENT redefine_cache_instance  (cache_instance)>
<!ELEMENT cache_instance  (internal_map, cache_storage?|tile_storage?, coordinate_
system, tile_image, zoom_levels)>
<!ELEMENT internal_map_source  EMPTY>
<!ELEMENT cache_storage  EMPTY>
<!ELEMENT coordinate_system  EMPTY>
<!ELEMENT tile_image  EMPTY>
<!ELEMENT zoom_levels  EMPTY>
<!ATTLIST create_cache_instance
   data_source   CDATA #REQUIRED
>
<!ATTLIST  cache_instance
   name   CDATA #REQUIRED
   image_format (PNG|GIF|JPG)"PNG"
>
<!ATTLIST  internal_map_source
   base_map   CDATA #REQUIRED
>
<!ATTLIST  cache_storage
   root_path   CDATA #IMPLIED
>
<!ATTLIST  coordinate_system
   srid  PCDATA #REQUIRED
   minX  PCDATA #REQUIRED
   maxX  PCDATA #REQUIRED
   minY  PCDATA #REQUIRED
   maxY  PCDATA #REQUIRED
>
<!ATTLIST  tile_image
   width   PCDATA #REQUIRED
   height  PCDATA #REQUIRED
>
<!ATTLIST  zoom_levels
   levels     PCDATA #REQUIRED
   min_scale  PCDATA #REQUIRED
   max_scale  PCDATA #REQUIRED
>
```

The `srid` attribute specifies the projection (coordinate system) of this tile layer.

The `minX`, `maxX`, `minY`, and `maxY` attributes specify the data bounds of this tile layer in the specified projection.

The `width` and `height` attributes specify the tile image width and height.

The `levels` attribute specifies the number of zoom levels of this tile layer.

The `min_scale` and `max_scale` attributes specify the smallest and largest values of the scale denominators. Zoom 0 is always assigned with the largest scale denominator, thus, a tile covers a larger geographic region with coarse map details.

7.2.7 Removing a Cache Instance

The <remove_cache_instance> element removes a tile layer instance from the server. It has the following definition:

```
<!ELEMENT map_cache_admin_request (remove_cache_instance)>
<!ELEMENT remove_cache_instance EMPTY>
```

```
<!ATTLIST  remove_cache_instance
   map_cache_name   CDATA #REQUIRED
   clean_disk       (true|false) "false"
   remove_permanently (true|false) "false"
>
```

The `clean_disk` attribute indicates if the already cached tiles are to be deleted.

The `remove_permanently` attribute indicates if tile layer is to be deleted from the USER_SDO_CACHED_MAPS view.

The following example removes a tile layer instance from the server.

```
<?xml version="1.0" standalone="yes"?>
<map_cache_admin_request>
  <remove_cache_instance
    map_cache_name="mvdemo.demo_map"
    clean_disk="true" remove_permanently="false"/>
```

The DTD for the `remove_cache_instance` response is as follows:

```
<!ELEMENT map_tile_server_response (#CDATA)>
```

For example:

```
<map_tile_server_response>Succeeded.</map_tile_server_response>
```

7.2.8 Restarting the Tile Layer Cache Server

The `<restart_cache_server>` element restarts the tile layer server. It has the following definition:

```
<!ELEMENT map_cache_admin_request (restart_cache_server)>
<!ELEMENT restart_cache_server EMPTY>
```

The following example restarts the server:

```
<?xml version="1.0" standalone="yes"?>
<map_cache_admin_request>
  <restart_cache_server />
</map_cache_admin_request>
```

7.2.9 Taking a Tile Layer Offline or Bringing It Online

The `<take_cache_offline>` element takes a tile layer instance offline from the server, and the `<bring_cache_online>` element brings an offline tile layer instance back online. The following is the DTD definition:

```
<!ELEMENT map_cache_admin_request (take_cache_offline, bring_cache_online)>
<!ELEMENT take_cache_offline  EMPTY>
<!ATTLIST take_cache_offline
   map_cache_name  CDATA #REQUIRED
>
<!ATTLIST  bring_cache_online
   map_cache_name  CDATA #REQUIRED
>
```

The following example takes a tile layer instance offline:

```
<?xml version="1.0" standalone="yes"?>
<map_cache_admin_request>
  <take_cache_offline map_cache_name="mvdemo.demo_map" />
</map_cache_admin_request>
```

7.3 Listing All Maps (General-Purpose)

The <list_maps> element lists all base maps in a specified data source. It has the following definition:

```
<!ELEMENT non_map_request list_maps>
<!ELEMENT list_maps  EMPTY>
<!ATTLIST list_maps
   data_source   CDATA #REQUIRED
>
```

The following example lists all base maps in the data source named mvdemo.

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <list_maps data_source="mvdemo"/>
</non_map_request>
```

The DTD for the response to a list_maps request has the following format:

```
<!ELEMENT non_map_response map_list>
<!ELEMENT map_list  (map*) >
<!ATTLIST map_list
   succeed   (true | false) #REQUIRED
>
<!ATTLIST map
   name      CDATA #REQUIRED
>
```

The succeed attribute indicates whether or not the request was processed successfully.

The name attribute identifies each map.

For example:

```
<?xml version="1.0" ?>
<non_map_response>
<map_list succeed="true">
  <map name="DEMO_MAP"/>
  <map name="DENSITY_MAP"/>
</map_list>
</non_map_response>
```

7.4 Listing Themes (General-Purpose)

The <list_predefined_themes> element lists either all themes defined in a specified data source or all themes defined in a specified data source for a specified map.

The DTD for requesting all themes defined in a data source regardless of the map associated with a theme has the following definition:

```
<!ELEMENT non_map_request list_predefined_themes>
<!ELEMENT list_predefined_themes  EMPTY>
<!ATTLIST list_predefined_themes
   data_source   CDATA #REQUIRED
>
```

The following example lists all themes defined in the data source named mvdemo.

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
```

```
  <list_predefined_themes data_source="mvdemo"/>
</non_map_request>
```

The DTD for requesting all themes defined in a data source and associated with a specific map has the following definition:

```
<!ELEMENT non_map_request list_predefined_themes>
<!ELEMENT list_predefined_themes  EMPTY>
<!ATTLIST list_predefined_themes
   data_source CDATA #REQUIRED
   map         CDATA #REQUIRED
>
```

The following example lists all themes defined in the data source named `tilsmenv` and associated with the map named `QA_MAP`.

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <list_predefined_themes data_source="tilsmenv" map="QA_MAP"/>
</non_map_request>
```

The DTD for the response to a `list_predefined_themes` request has the following format:

```
<!ELEMENT non_map_response predefined_theme_list>
<!ELEMENT predefined_theme_list  (predefined_theme*) >
<!ATTLIST predefined_theme_list
   succeed   (true | false) #REQUIRED
>
<!ELEMENT predefined_theme  EMPTY>
<!ATTLIST predefined_theme
   name   CDATA #REQUIRED
>
```

The `succeed` attribute indicates whether or not the request was processed successfully.

The `name` attribute identifies each theme.

For example:

```
<?xml version="1.0" ?>
<non_map_response>
<predefined_theme_list succeed="true">
  <predefined_theme name="THEME_DEMO_CITIES"/>
  <predefined_theme name="THEME_DEMO_BIGCITIES"/>
  <predefined_theme name="THEME_DEMO_COUNTIES"/>
  <predefined_theme name="THEME_DEMO_COUNTY_POPDENSITY"/>
  <predefined_theme name="THEME_DEMO_HIGHWAYS"/>
  <predefined_theme name="THEME_DEMO_STATES"/>
  <predefined_theme name="THEME_DEMO_STATES_LINE"/>
</predefined_theme_list>
</non_map_response>
```

Note that the order of names in the returned list is unpredictable.

7.5 Listing Styles (General-Purpose)

The <list_styles> element lists styles defined for a specified data source. It has the following definition:

```
<!ELEMENT non_map_request list_styles>
```

```
<!ELEMENT list_styles  EMPTY>
<!ATTLIST list_styles
   data_source   CDATA #REQUIRED
   style_type   (COLOR|LINE|MARKER|AREA|TEXT|ADVANCED)  #IMPLIED
>
```

If you specify a value for `style_type`, only styles of that type are listed. The possible types of styles are `COLOR`, `LINE`, `MARKER`, `AREA`, `TEXT`, and `ADVANCED`. If you do not specify `style_type`, all styles of all types are listed.

The following example lists only styles of type `COLOR`:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <list_styles data_source="mvdemo"  style_type="COLOR"/>
</non_map_request>
```

The DTD for the response to a `list_styles` request has the following format:

```
<!ELEMENT non_map_response style_list>
<!ELEMENT style_list  (style*) >
<!ATTLIST style_list
   succeed   (true | false) #REQUIRED
>
<!ELEMENT style  EMPTY>
<!ATTLIST style
   name   CDATA #REQUIRED
>
```

The following example shows the response to a request for styles of type `COLOR`:

```
<?xml version="1.0" ?>
 <non_map_response>
 <style_list succeed="true">
   <style name="SCOTT:C.BLACK"/>
   <style name="SCOTT:C.BLACK GRAY"/>
   <style name="SCOTT:C.BLUE"/>
   <style name="SCOTT:C.CRM_ADMIN_AREAS"/>
   <style name="SCOTT:C.CRM_AIRPORTS"/>
</style_list>
</non_map_response>
```

Each style name in the response has the form *`OWNER:NAME`* (for example, `SCOTT:C.BLACK`), where *`OWNER`* is the schema user that owns the style.

7.6 Listing Styles Used by a Predefined Theme (General-Purpose)

The `<list_theme_styles>` element lists all the rendering styles that are referenced in a predefined theme. This is particularly useful if you want to build a legend for a theme yourself, where you need to know which styles are actually being used in that theme. This element has the following definition:

```
<!ELEMENT non_map_request list_theme_styles>
<!ELEMENT list_theme_styles  EMPTY>
<!ATTLIST list_styles
   data_source   CDATA #REQUIRED
   theme CDATA #REQUIRED
>
```

The following example requests the styles used by the `THEME_DEMO_STATES` predefined theme:

```
<non_map_request>
   <list_theme_styles data_source="mvdemo" theme="THEME_DEMO_STATES"  />
</non_map_request>
```

The following example shows the response to the preceding request:

```
<non_map_response>
  <theme_style name="C.US MAP YELLOW" type="COLOR" render="true" label="false"
    highlight="false" description="Primary color for US maps."/>
  <theme_style name="T.STATE NAME" type="TEXT" render="false" label="true"
    highlight="false" description="name for states"/>
</non_map_response>
```

The DTD for the response to a `list_theme_styles` request has the following format:

```
<!ELEMENT non_map_response (theme_style*)>
<!ELEMENT theme_style EMPTY>
<!ATTLIST theme_style
    name CDATA #REQUIRED
    type   CDATA (COLOR|LINE|MARKER|AREA|TEXT|ADVANCED)  #REQUIRED
    render CDATA (true|false)  #REQUIRED
    label CDATA (true|false)  #REQUIRED
    highlight CDATA (true|false) #REQUIRED
    description CDATA #IMPLIED
>
```

In the preceding DTD:

- The `name` attribute identifies the name of the style.
- The type attribute identifies the MapViewer style type.
- The render attribute indicates whether or not the style is used as a rendering style by the theme.
- The label attribute indicates whether or not the style is used as a labeling style.
- The highlight attribute indicates whether or not the style is used as only a highlight style.
- The description attribute identifies the description as specified in the style definition.

7.7 Managing In-Memory Caches

MapViewer uses two types of in-memory cache:

- Metadata cache for mapping metadata, such as style, theme, and base map definitions, and the SRID value for SDO_GEOMETRY columns in tables in the cache
- Spatial data cache for predefined themes (the geometric and image data used in generating maps)

The use of these caches improves performance by preventing MapViewer from accessing the database for the cached information; however, the MapViewer displays might reflect outdated information if that information has changed in the database since it was placed in the cache.

If you want to use the current information without restarting MapViewer, you can clear (invalidate) the content of either or both of these caches. If a cache is cleared, the

next MapViewer request will retrieve the necessary information from the database, and will also store it in the appropriate cache.

7.7.1 Clearing Metadata Cache for a Data Source (Administrative)

As users request maps from a data source, MapViewer caches such mapping metadata as style, theme, and base map definitions for that data source, as well as the SRID value for SDO_GEOMETRY columns in tables (such as when rendering a theme for the first time). This prevents MapViewer from unnecessarily accessing the database to fetch the mapping metadata. However, modifications to the mapping metadata, such as those you make using the Map Builder tool, do not take effect until MapViewer is restarted.

If you want to use the changed definitions without restarting MapViewer, you can request that MapViewer clear (that is, remove from the cache) all cached mapping metadata and cached table SRID values for a specified data source. Clearing the metadata cache forces MapViewer to access the database for the current mapping metadata.

The <clear_cache> element clears the MapViewer metadata cache. It has the following definition:

```
<!ELEMENT non_map_request clear_cache>
<!ELEMENT clear_cache  EMPTY>
<!ATTLIST clear_cache
   data_source  CDATA #REQUIRED
>
```

The `data_source` attribute specifies the name of the data source whose metadata is to be removed from the MapViewer metadata cache.

The following example clears the metadata for the `mvdemo` data source from the MapViewer metadata cache:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <clear_cache data_source="mvdemo"/>
</non_map_request>
```

The DTD for the response to a `clear_cache` request has the following format:

```
<!ELEMENT non_map_response clear_cache>
<!ELEMENT clear_cache  EMPTY>
<!ATTLIST clear_cache
   succeed   (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
<non_map_response>
 <clear_cache succeed="true"/>
</non_map_response>
```

7.7.2 Clearing Spatial Data Cache for a Theme (Administrative)

MapViewer caches spatial data (geometries or georeferenced images) for a predefined theme as it loads the data from the database into memory for rendering, unless it is told not to do so. (MapViewer does not cache the data for dynamic or JDBC themes.) Thus, if a predefined theme has been frequently accessed, most of its data is probably in the cache. However, if the spatial data for the theme is modified in the database, the

changes will not be visible on maps, because MapViewer is still using copies of the data from the cache. To view the modified theme data without having to restart MapViewer, you must first clear the cached data for that theme.

The <clear_theme_cache> element clears the cached data of a predefined theme. It has the following definition:

```
<!ELEMENT non_map_request clear_theme_cache>
<!ELEMENT clear_theme_cache  EMPTY>
<!ATTLIST clear_theme_cache
   data_source CDATA #REQUIRED
   theme       CDATA #REQUIRED
>
```

The data_source attribute specifies the name of the data source. The theme attribute specifies the name of the predefined theme in that data source.

The following example clears the cached spatial data for the predefined theme named STATES in the mvdemo data source:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <clear_theme_cache data_source="mvdemo" theme="STATES"/>
</non_map_request>
```

The DTD for the response to a clear_theme_cache request has the following format:

```
<!ELEMENT non_map_response clear_theme_cache>
<!ELEMENT clear_theme_cache  EMPTY>
<!ATTLIST clear_theme_cache
   succeed  (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
<non_map_response>
 <clear_theme_cache succeed="true"/>
</non_map_response>
```

7.8 Editing the MapViewer Configuration File (Administrative)

The <edit_config_file> element lets you edit the MapViewer configuration file (mapViewerConfig.xml). It has the following definition:

```
<!ELEMENT non_map_request edit_config_file>
<!ELEMENT edit_config_file  EMPTY>
```

Note: Use the <edit_config_file> element only if you are running MapViewer in the standalone OC4J environment or in a nonclustered OC4J instance with only one process started. Otherwise, the modifications that you make will be applied only to one MapViewer instance, and inconsistencies may occur.

Specify the request as follows:

```
<?xml version="1.0" standalone="yes">
<non_map_request>
  <edit_config_file/>
```

```
</non_map_request>
```

After you submit the request, you are presented with an HTML form that contains the current contents of the MapViewer configuration file. Edit the form to make changes to the content, and click the **Save** button to commit the changes. However, the changes will not take effect until you restart the MapViewer server (see Section 7.9).

7.9 Restarting the MapViewer Server (Administrative)

In general, the safest method for restarting the MapViewer server is to restart its containing OC4J instance. However, if you are running MapViewer in a standalone OC4J environment, or if the OC4J instance is not clustered and it has only one Java process started, you can use the `<restart>` element to restart MapViewer quickly without restarting the entire OC4J instance. The `<restart>` element has the following definition:

```
<!ELEMENT non_map_request edit_config_file>
<!ELEMENT restart  EMPTY>
```

Specify the request as follows:

```
<?xml version="1.0" standalone="yes">
<non_map_request>
  <restart/>
</non_map_request>
```

8 Oracle Maps

Oracle Maps is the name for a suite of technologies for developing high-performance interactive web-based mapping applications. Oracle Maps is included with MapViewer.

This chapter contains the following major sections:

- Section 8.1, "Overview of Oracle Maps"
- Section 8.2, "Map Tile Server"
- Section 8.3, "Feature of Interest (FOI) Server"
- Section 8.4, "Oracle Maps JavaScript API"
- Section 8.5, "Developing Oracle Maps Applications"
- Section 8.6, "Using Google Maps and Bing Maps"
- Section 8.7, "Transforming Data to a Spherical Mercator Coordinate System"
- Section 8.8, "Dynamically Displaying an External Tile Layer"

8.1 Overview of Oracle Maps

Oracle Maps consists of the following main components:

- A map tile server that caches and serves pregenerated map image tiles
- A feature of interest (FOI) server that renders geospatial features that are managed by Oracle Spatial and Graph
- An Ajax-based JavaScript mapping client. (Ajax is an acronym for asynchronous JavaScript and XML.) This client provides functions for browsing and interacting with maps, as well as a flexible application programming interface (API).

The map tile server (map image caching engine) automatically fetches and caches map image tiles rendered by Oracle MapViewer or other web-enabled map providers. It also serves cached map image tiles to the clients, which are web applications developed using the Oracle Maps client API. The clients can then automatically stitch multiple map image tiles into a seamless large map. Because the map image tiles are pregenerated and cached, the application users will experience fast map viewing performance.

The feature of interest (FOI) server (rendering engine) renders spatial feature layers managed by Oracle Spatial and Graph, as well as individual geospatial features of point, line, or polygon type that are created by an application. Such FOIs, which typically include both an image to be rendered and a set of associated attribute data, are then sent to the client where a user can interact with them. Unlike the cached

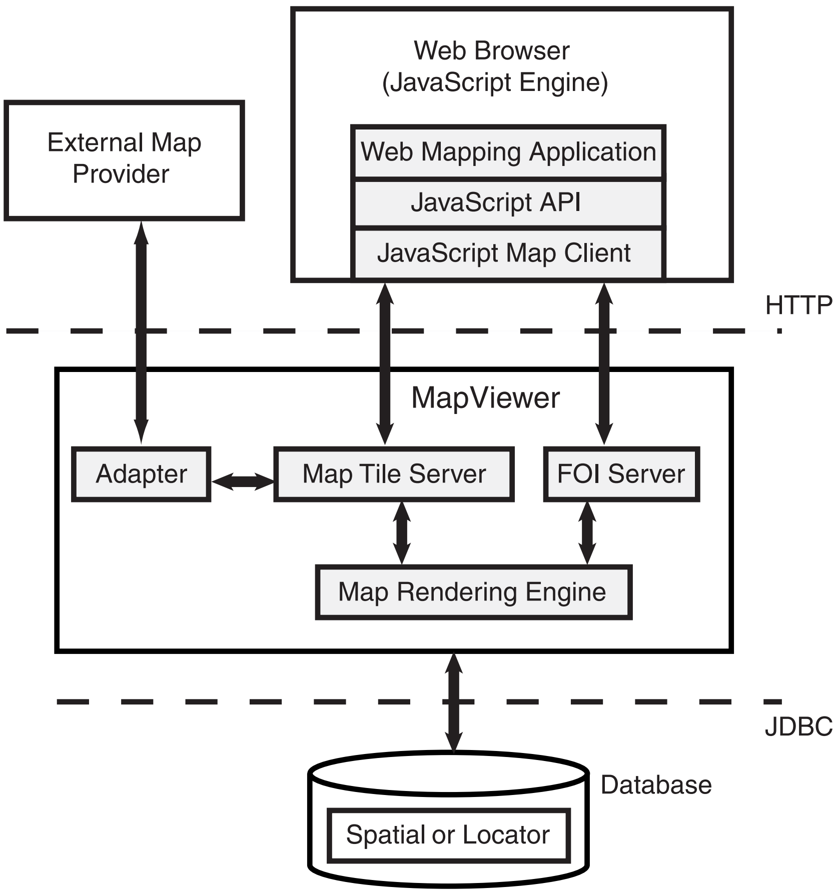
image tiles, which typically represent static content, FOIs are dynamic and represent real-time database or application contents. The dynamic FOIs and the static cached map tiles enable you to build web mapping applications.

The JavaScript mapping client is a browser side map display engine that fetches map content from the servers and presents it to client applications. It also provides customizable map-related user interaction control, such as map dragging and clicking, for the application. The JavaScript mapping client can be easily integrated with any web application or portal.

8.1.1 Architecture for Oracle Maps Applications

Figure 8–1 shows the architecture of web mapping applications that are developed using Oracle Maps.

Figure 8–1 Architecture for Oracle Maps Applications



Referring to Figure 8–1, applications interact with the Oracle Maps architecture as follows:

- The application is developed using JavaScript, and it runs inside the JavaScript engine of the web browser.
- The application invokes the JavaScript map client to fetch the map image tiles from the map tile server, and then it displays the map in the web browser.
- The application invokes the JavaScript map client to fetch dynamic spatial features from the FOI server and display them on top of the map tiles.

- The JavaScript map client controls map-related user interaction for the application.
- When the map tile server receives a map image tile request, it first checks to see if the requested tile is already cached. If the tile is cached, the cached tile is returned to the client. If the tile is not cached, the map tile server fetches the tile into the cache and returns it to the client. Tiles can be fetched either directly from the MapViewer map rendering engine or from an external web map services provider.
- When the FOI server receives a request, it uses the MapViewer map rendering engine to generate the feature images and to send these images, along with feature attributes, to the client.

8.1.2 Simple Example Using Oracle Maps

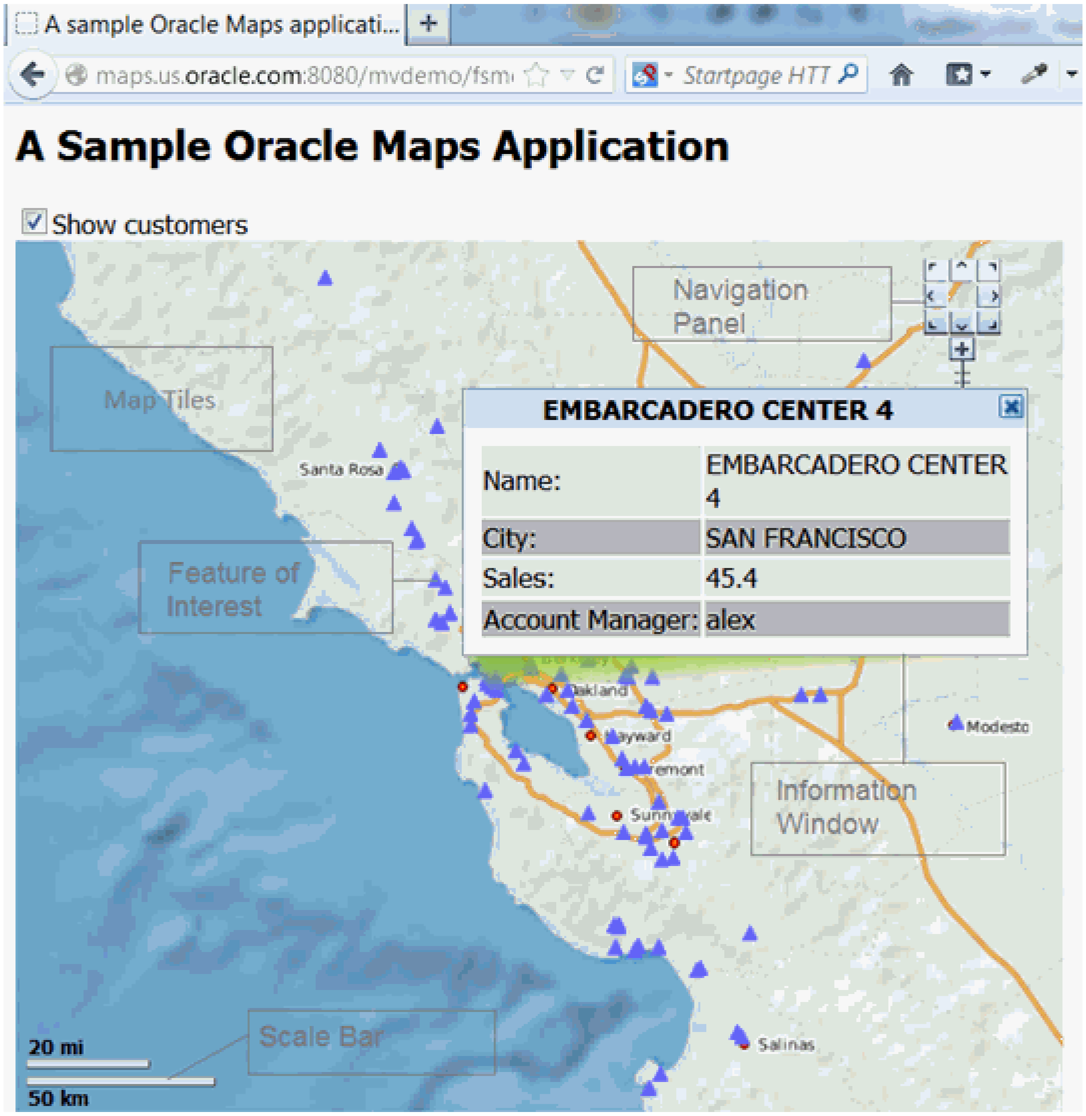
Figure 8–2 shows the interface of a simple application created using the Oracle Maps JavaScript V1 API. This example, along with sample applications, tutorials, and API documentation, is included in a separate `mvdemo.ear` file, which can be downloaded from `http://www.oracle.com/technetwork/middleware/mapviewer/`. The `mvdemo.ear` file should be deployed into the same container as the `mapviewer.ear` file.

Note: The Oracle Maps JavaScript V1 and V2 APIs are described in Section 8.4.

Section 8.1.2.1, "Simple Application Using the V2 API" describes essentially the same simple example but implemented using the V2 API.

The simple application shown in Figure 8–2 can be accessed at `http://`*`host:port`*`/mvdemo/fsmc/sampleApp.html`. To run this application, follow the instructions in `http://`*`host:port`*`/mvdemo/fsmc/tutorial/setup.html` to set up the database schema and the necessary map tile layers.

Figure 8–2 Application Created Using Oracle Maps (V1 API)



The application shown in Figure 8–2 displays customers on the map. The map consists of two layers:

- The map tile layer displays the ocean, county boundaries, cities, and highways. The whole map tile layer displayed in the web browser consists of multiple map image tiles that are rendered by the map tile server.
- The FOI layer displays customers as red dot markers on top of the map tile layer. If the user clicks on the marker for a customer, an information window is displayed showing some attributes for that customer. The customer markers and attributes are rendered by the FOI server.

In addition to these two layers, a scale bar is displayed in the lower-left corner of the map, and a navigation panel is displayed in the upper-right corner.

The application user can use the mouse to drag the map. When this happens, new image tiles and FOIs are automatically fetched for the spatial region that the map currently covers. The user can also use the built-in map navigation tool to pan and zoom the image, and can show or hide the customers (red dot markers) by checking or unchecking the **Show customers** box.

Example 8–1 shows the complete source code for the simple application shown in Figure 8–2.

Example 8–1 Source Code for the Simple Application (V1 API)

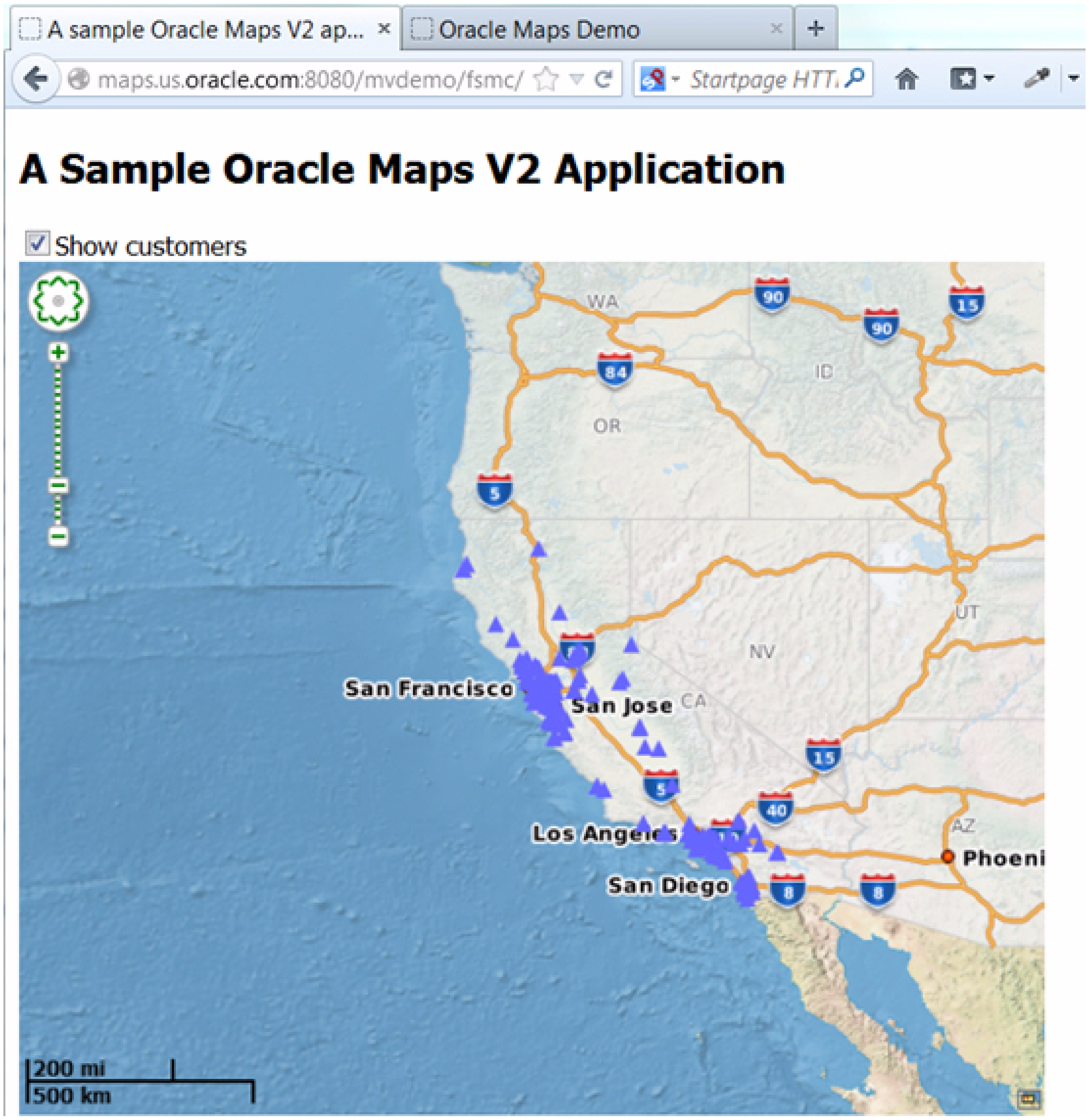
```
<html>
<head>
<META http-equiv="Content-Type" content="text/html" charset=UTF-8">
<TITLE>A sample Oracle Maps Application</TITLE>
<script language="Javascript" src="jslib/oraclemaps.js"></script>
<script language=javascript>
var themebasedfoi=null
function on_load_mapview()
{
  var baseURL  = "http://"+document.location.host+"/mapviewer";
  // Create an MVMapView instance to display the map
  var mapview = new MVMapView(document.getElementById("map"), baseURL);
  // Add a map tile layer as background.
  mapview.addMapTileLayer(new MVMapTileLayer("mvdemo.demo_map"));
  // Add a theme-based FOI layer to display customers on the map
  themebasedfoi = new MVThemeBasedFOI('themebasedfoi1','mvdemo.customers');
  themebasedfoi.setBringToTopOnMouseOver(true);
  mapview.addThemeBasedFOI(themebasedfoi);
  // Set the initial map center and zoom level
  mapview.setCenter(MVSdoGeometry.createPoint(-122.45,37.7706,8307));
  mapview.setZoomLevel(4);
  // Add a navigation panel on the right side of the map
  mapview.addNavigationPanel('east');
  // Add a scale bar
  mapview.addScaleBar();
  // Display the map.
  mapview.display();
}
function setLayerVisible(checkBox)
{
  // Show the theme-based FOI layer if the check box is checked and
  // hide the theme-based FOI layer otherwise.
  if(checkBox.checked)
    themebasedfoi.setVisible(true) ;
  else
    themebasedfoi.setVisible(false);
}
</script>
</head>
<body onload= javascript:on_load_mapview() >
<h2> A sample Oracle Maps Application</h2>
<INPUT TYPE="checkbox" onclick="setLayerVisible(this)" checked/>Show customers
<div id="map" style="width: 600px; height: 500px"></div>
</body>
</html>
```

The components of this sample application and the process for creating a client application are described in Section 8.5.1.3.

8.1.2.1 Simple Application Using the V2 API

Figure 8–3 shows a simple example with the essentially the same logic as that shown in Figure 8–2, but using the Oracle Maps JavaScript V2 API.

Figure 8–3 Application Created Using Oracle Maps (V2 API)



Example 8–2 shows the complete source code for the simple application shown in Figure 8–3.

Example 8–2 Source Code for the Simple Application (V2 API)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<TITLE>A sample Oracle Maps V2 application</TITLE>
<script language="Javascript" src="/mapviewer/jslib/v2/oraclemapsv2.js"></script>
<script language=javascript>
var customersLayer=null;
function on_load_mapview()
{
  var baseURL  = "http://"+document.location.host+"/mapviewer";
  // Create an OM.Map instance to display the map
  var mapview = new OM.Map(document.getElementById("map"),
```

```
                             {
                               mapviewerURL:baseURL
                             });
  // Add a map tile layer as background.
  var tileLayer = new OM.layer.TileLayer(
        "baseMap",
        {
            dataSource:"mvdemo",
            tileLayer:"demo_map",
            tileServerURL:baseURL+"/mcserver"
        });
  mapview.addLayer(tileLayer);
  // Set the initial map center and zoom level
  var mapCenterLon = -122.45;
  var mapCenterLat = 37.7706;
  var mapZoom = 4;
  var mpoint = new OM.geometry.Point(mapCenterLon,mapCenterLat,8307);
  mapview.setMapCenter(mpoint);
  mapview.setMapZoomLevel(mapZoom);
  // Add a theme-based FOI layer to display customers on the map
  customersLayer = new OM.layer.VectorLayer("customers",
        {
            def:
                {
                type:OM.layer.VectorLayer.TYPE_PREDEFINED,
                dataSource:"mvdemo", theme:"customers",
                url: baseURL,
                loadOnDemand: false
                }
        });
  mapview.addLayer(customersLayer);

  // Add a navigation panel on the right side of the map
  var navigationPanelBar = new OM.control.NavigationPanelBar();

navigationPanelBar.setStyle({backgroundColor:"#FFFFFF",buttonColor:"#008000",size:
12});
  mapview.addMapDecoration(navigationPanelBar);
  // Add a scale bar
  var mapScaleBar = new OM.control.ScaleBar();
  mapview.addMapDecoration(mapScaleBar);
  // Display the map.
  // Note: Change from V1. In V2 initialization and display is done just once
  mapview.init();
}
function setLayerVisible(checkBox)
{
  // Show the customers vector layer if the check box is checked and
  // hide it otherwise.
  if(checkBox.checked)
    customersLayer.setVisible(true) ;
  else
    customersLayer.setVisible(false);
}
</script>
</head>
<body onload= javascript:on_load_mapview() >
<h2>A Sample Oracle Maps V2 Application</h2>
<INPUT TYPE="checkbox" onclick="setLayerVisible(this)" checked/>Show customers
<div id="map" style="width: 600px; height: 500px"></div>
```

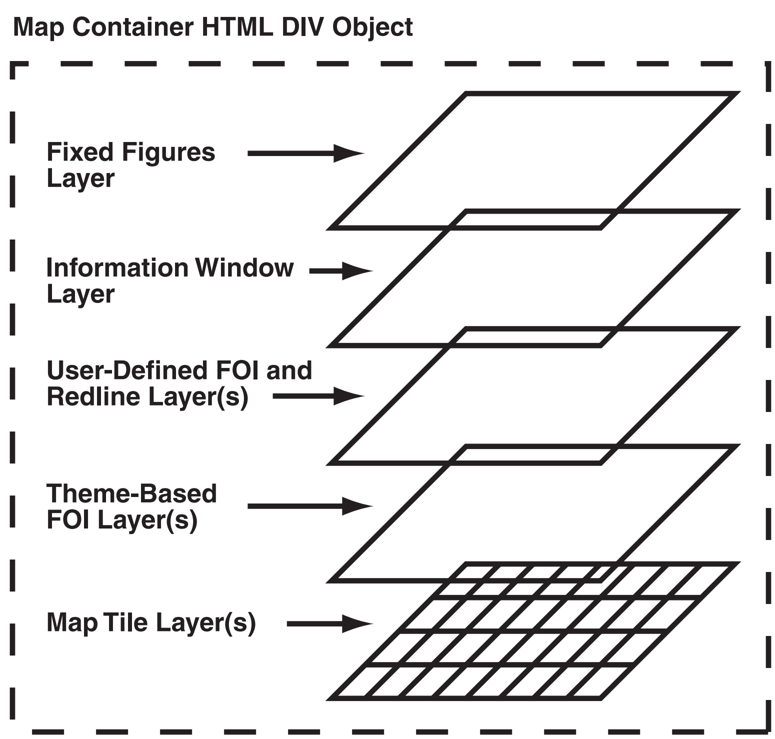
```
</body>
</html>
```

8.1.3 How Map Content Is Organized

This section describes how the JavaScript client internally organizes various map contents when displayed a map inside a web browser. An application typically places one master HTML DIV object on a web page, and the JavaScript client adds various content layers inside this DIV object.

The map content displayed by the map client is organized by layers. When the application script invokes appropriate map client API, map layers are created inside a map container. The map container is a user-defined HTML DIV object. You can customize the size and the positioning of the map container inside the web page. Figure 8–4 shows the layout of the map layers.

Figure 8–4 Layers in a Map



As shown in Figure 8–4, there are five different types of map content layers: map tiles, theme-based FOI, user-defined FOI or redline, information window, and fixed figures. All layers except the fixed figures layer are moved as a whole when the user drags the map. These movable layers are automatically updated by the map client when the map is dragged or zoomed. (The fixed figures layer is never moved.).

8.1.3.1 Map Tile Layers

A typical Oracle Maps application has at least one map tile layer, which assembles and displays pregenerated map image tiles from the map tile server. The map tile layer displays static map content that does not change very often, and it is typically used as the background map by the client application. For example, in the sample application described in Section 8.1.2 and illustrated in Figure 8–2, the ocean, county boundaries,

cities, and highways are all displayed as a map tile layer. Only limited user interaction, such as map dragging, can be performed with a map tile layer.

A map tile layer is usually associated with a MapViewer base map, and is managed by the MapViewer server. However, you can configure a map tile layer to cache map image tiles served by an external (non-MapViewer) map provider.

The Oracle Maps client can also display a custom or built-in external tile layer served directly by an external tile server. The built-in Google Maps and Microsoft Bing Maps tile layers are examples. For more information, see Section 8.6, "Using Google Maps and Bing Maps" and the JavaScript API documentation for class `MVGoogleTileLayer` and `MVBingTileLayer`. (If you need to overlay your own spatial data on top of the Google Maps or Bing Maps tile layer, see also Section 8.7, "Transforming Data to a Spherical Mercator Coordinate System".)

Map tile layers are always placed at the bottom of the layer hierarchy. These layers display static and background map contents. When multiple such layers are included, they must all have the same coordinate system and zoom level definitions.

Internally, the map tile layers are usually larger than the size of the map DIV container window. This allows additional tiles to be fetched and cached by the browser. As a result, these tiles will be immediately visible when the map layers are dragged around by the user.

8.1.3.2 Theme-Based FOI Layers

There can be one or more theme-based FOI layers. Each theme-based FOI layer consists of a collection of interactive FOIs that meet certain query criteria defined in a MapViewer predefined theme. FOIs can be points, lines, or polygons. For example, all stores with a sales volume greater than $100,000 can be displayed as a point theme-based FOI layer.

Users can interact with the FOIs by moving the mouse over them or clicking on them. The application can customize how the map client reacts to such user interaction.

All features (geographic and non-geographic) of a theme-based FOI layer are stored in the database. Features are queried and rendered by the FOI server when client applications request them. The query window for the theme-based FOI layers can be customized to be larger than the map DIV window, so that it gives some extra room for dragging the map without refreshing the theme-based FOI layers from server. For more information about theme-based FOI layers, see Section 8.3.1.

8.1.3.3 User-Defined FOI Layers

A user-defined FOI is an interactive feature defined on the client side. The FOI can be a point, line, or polygon feature. Users can interact with a user-defined FOIs in the same way they can with a theme-based FOIs. However, in contrast with a theme-based FOI layer which is rendered as a collection of features, each user-defined FOI is requested and rendered individually. All attributes of the user-defined FOI, including the geometry representation and rendering style, must be provided by the application. For example, a route geometry based on user specified start and end addresses should be displayed as a user-defined line FOI on the map.

The handling of user-defined FOI layers depends on web browser in which the application is running:

- With Microsoft Internet Explorer, all user-defined individual FOIs added by the application are placed inside a layer directly above the theme-based FOI layers. There can be at most one such layer.

- With Opera and Mozilla-based browsers such as Netscape and Firefox, all user-defined individual FOIs are placed inside two layers, one for point features and the other for non-point features such as polylines and polygons. The non-point feature layer is placed under the point feature layer.

8.1.3.4 Information Window Layer

An information window is a small pop-up window that displays customizable content in the map. All information windows, when displayed, are placed inside a layer directly above the user-defined individual FOI layer or layers. There can be at most one information window layer.

8.1.3.5 Fixed Figures Layer

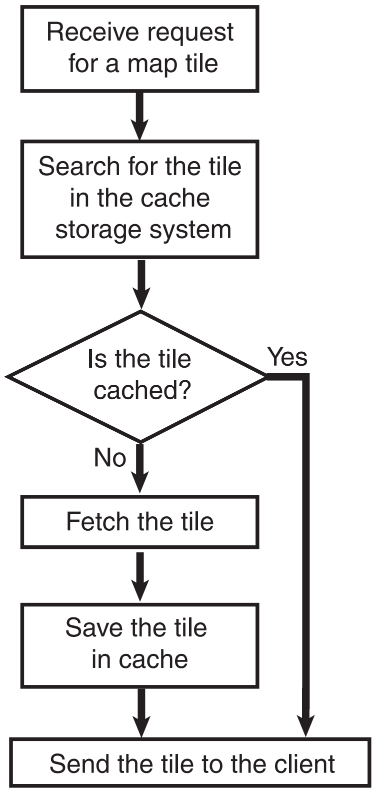
The topmost layer contains any fixed figures, which are immovable elements such as copyright notes, a scale bar, a navigation panel, and user-defined map decoration features. (A user-defined map decoration feature is an application defined element that can contain any custom HTML content, such as a map title or a custom control button.) The fixed figures layer is displayed on top of everything else, and it is not moved when the user drags the map.

8.2 Map Tile Server

The map tile server is a map image caching engine that caches and serves pregenerated, fixed-size map image tiles. It is implemented as a Java servlet that is part of the MapViewer server. The map tile server accepts requests that ask for map image tiles specified by tile zoom level and tile location (mesh code), and it sends the requested tiles back to clients.

Figure 8–5 shows the basic workflow of the map tile server.

Figure 8–5 Workflow of the Map Tile Server



As shown in Figure 8–5, when the map tile server receives a request for a map tile, it searches for the tile in the cache storage system. If the tile is cached, the map tile server sends the tile to the client. If the tile is not cached, the map tile server fetches the tile, saves in the cache, and sends it to the client.

You can use the MapViewer administration tool to manage the map tile server.

8.2.1 Map Tile Server Concepts

This section explains map tile server concepts that you need to know to be able to use Oracle Maps effectively.

8.2.1.1 Map Tile Layers and Map Tile Sources

All map tile layers are managed by the map tile server. The **map tile server** fetches and stores the map image tiles that belong to the map tile layer and returns map image tiles to the client. The map tile server can manage multiple map tile layers.

Each map tile layer can have multiple predefined zoom levels. Each zoom level is assigned a zoom level number ranging from 0 to n-1, where n is the total number of zoom levels. Zoom level 0 is the most zoomed out level and zoom level n-1 is the most zoomed in level.

The map is evenly divided into same-sized small map image tiles on each zoom level. Clients specify a map tile by its zoom level and tile mesh code.

A map tile layer can come from two different types of sources:

- Internal MapViewer base maps rendered by the MapViewer map rendering engine. A MapViewer base map consists of a set of predefined themes and must be predefined in the database view USER_SDO_MAPS.
- Maps rendered by an external web map services providers. An external web map services provider is a server that renders and serves maps upon client requests over the web. If you properly configure an adapter that can fetch maps from the external map services provider, the map tile server can fetch and cache map tiles generated by the external map services provider. (A MapViewer instance other than the MapViewer inside which the map tile server is running is also considered an external map services provider.)

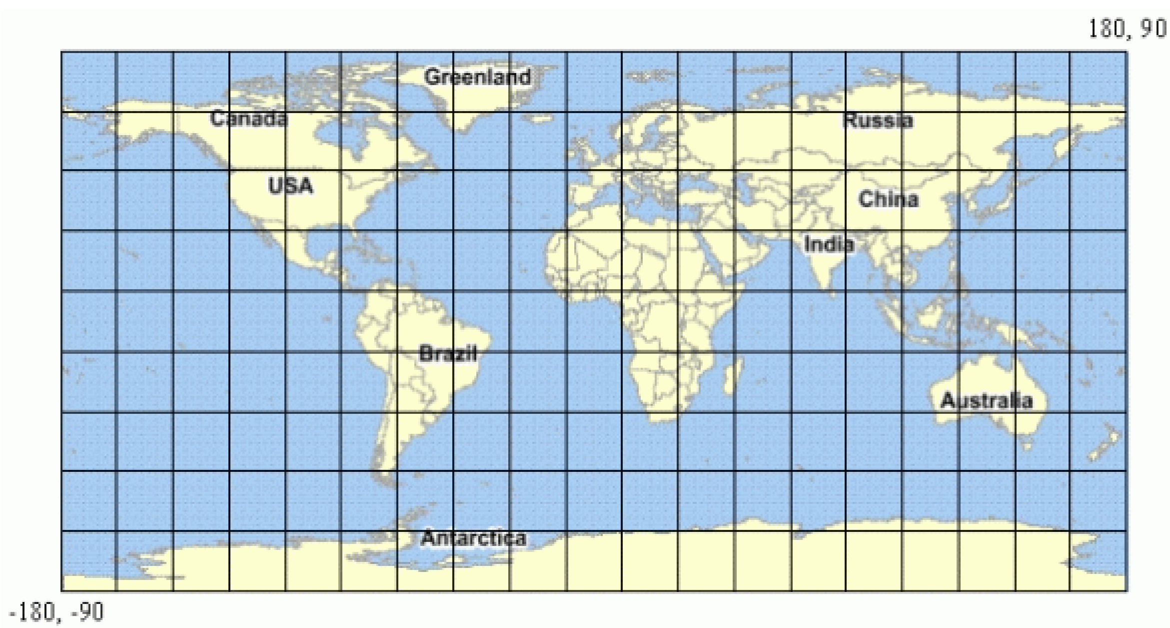
8.2.1.2 Storage of Map Image Tiles

Oracle Maps uses the local file system to store cached image tiles. You can customize the path that is used for this storage as part of the map tile server configuration settings.

8.2.1.3 Coordinate System for Map Tiles

Map images are cached and managed by the map tile server as small same-size rectangular image tiles. Currently we support tiling on any two-dimensional Cartesian coordinate system. A geodetic coordinate system can also be supported when it is mapped as if it is a Cartesian coordinate system, where longitude and latitude are treated simply as two perpendicular axes, as shown in Figure 8–6.

Figure 8–6 Tiling with a Longitude/Latitude Coordinate System



On each zoom level, the map tiles are created by equally dividing the whole map coordinate system along the two dimensions (X and Y, which inFigure 8–6 represent latitude and longitude). The map tile server needs this dimensional information of the map coordinate system in order to create map image tiles, and therefore you must include this information in the map tile layer configuration settings.

The whole map coordinate system can be represented by a rectangle, and its boundary is specified by (Xmin, Ymin) and (Xmax, Ymax), where Xmin is the minimum X value

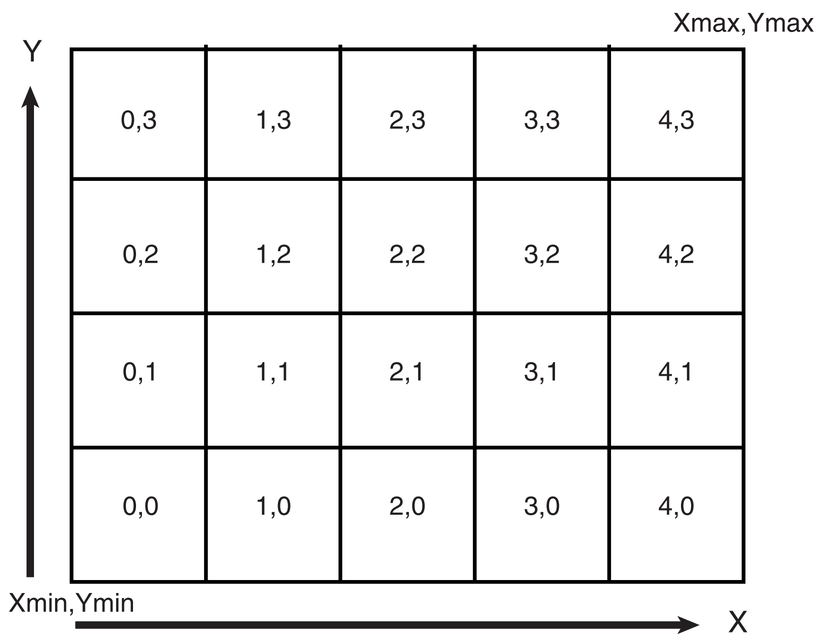
allowed in the coordinate system, Ymin is the minimum Y value allowed, Xmax is the maximum X value allowed and Ymax is the maximum Y value allowed. In Figure 8–6, Xmin is –180, Ymin is –90, Xmax is 180, and Ymax is 90.

You must also specify the spatial referencing ID (SRID) of the coordinate system to enable the map tile server to calculate map scales.

8.2.1.4 Tile Mesh Codes

Each map tile is specified by a mesh code, which is defined as a pair of integers (Mx, My), where Mx specifies the X dimension index of the tile and My specifies the Y dimension index of the tile. If the tile is the *i*th tile on X dimension starting from Xmin, then Mx should be i-1. If the tile is the *j*th tile on Y dimension starting from Ymin, then My should be j-1. Figure 8–7 shows the mesh codes of the tiles on a map.

Figure 8–7 Tile Mesh Codes



The JavaScript map client automatically calculates which tiles it needs for displaying the map in the web browser, and it sends requests with the mesh codes to the server. Mesh codes are transparent to the application, and application developers do not need to deal with mesh codes directly.

8.2.1.5 Tiling Rules

You must create tiling rules that determine how the map is divided and how tiles are created. The map tile server uses these tiling rules to divide the map into small map image tiles that are stored in the tile storage system. These rules are also used by the JavaScript map client.

Because all tiles on a given zoom level are the same size, the map tile server needs to know the following information to perform the tile division:

- The map tile image size (width and height), specified in screen pixels. This is the physical size of the tile images.

- The tile size specified according to the map coordinate system. For example, if the map uses a geodetic coordinate system, the tile width and height should be defined in degrees. The size can be specified either explicitly by tile width and height or implicitly by map scale. (Map scale, combined with tile image size, can be used to derive the tile width and height according to the map coordinate system.)

The preceding information constitutes the tiling rule for a given zoom level. Each zoom level must have its own tiling rule. You must define the tiling rules when you specify the configuration settings for the map tile server, as described in Section 8.2.2.

8.2.2 Map Tile Server Configuration

Map tile server configuration settings are stored in local configuration files and in database views. You can customize these settings.

8.2.2.1 Global Map Tile Server Configuration

Global map tile server settings, such as logging options and the default cache storage directory, are stored in the MapViewer configuration file `mapViewerConfig.xml`, which is under the directory *`$MAPVIEWER_HOME`*`/web/WEB-INF/conf`.

The map tile server configuration settings are defined in element <map_tile_server> inside the top-level <mapperConfig> element, as shown in the following example:

```
<map_tile_server>
   <tile_storage default_root_path="/scratch/tilecache/"/>
</map_tile_server>
```

The <tile_storage> element specifies the map tiles storage settings. The `default_root_path` attribute specifies the default file system directory under which the cached tile images are to be stored. If the default root directory is not set or not valid, the default root directory is *`$MAPVIEWER_HOME`*`/web/tilecache`. A subdirectory under this directory will be created and used for a map tile layer if the map tile layer configuration does not specify the map tiles storage directory for itself. The name of the subdirectory will be the same as the name of the map tile layer.

8.2.2.2 Map Tile Layer Configuration

The configuration settings for a map tile layer are stored in the USER_SDO_CACHED_MAPS metadata view. You should normally not manipulate this view directly, but should instead use the MapViewer administration tool, which uses this view to configure map tile layers.

Each database user (schema) has its own USER_SDO_CACHED_MAPS view. Each entry in this view stores the configuration settings for one map tile layer. If the map tile layer is based on an internal MapViewer base map, the base map associated with the map tile layer must be defined in the same database schema where the map tile layer configuration settings are stored.

The map tile server obtains the map source configuration by querying the USER_SDO_CACHED_MAPS view using the database connections specified by MapViewer data sources. This happens when the map tile server is started or a new data source is added to MapViewer as the result of a MapViewer administration request.

The USER_SDO_CACHED_MAPS view has the columns listed in Table 8–1.

Table 8–1 USER_SDO_CACHED_MAPS View

Column Name	Data Type	Description
NAME	VARCHAR2	Unique name of the cached map source
DESCRIPTION	VARCHAR2	Optional descriptive text about the cached map source
TILES_TABLE	VARCHAR2	(Not currently used)
IS_ONLINE	VARCHAR2	`YES` if the map tile layer is online, or `NO` if the map tile layer is offline. When a tile is missing from the cache and the map tile layer is online, the map tile server will fetch the tile and return the fetched tile to the client. When a tile is missing and the map tile layer is offline, the map tile server will not fetch the tile but will return a blank image to the client.
IS_INTERNAL	VARCHAR2	`YES` if the map source is an internal map source, or `NO` if the map source is an external map source
DEFINITION	CLOB	XML definition of the map tile layer, as described later in this section.
BASE_MAP	VARCHAR2	Name of the cached MapViewer base map, if the map source is an internal map source
MAP_ ADAPTER	BLOB	The jar file that contains the adapter Java classes of the external map services provider, as described later in this section.

For the DEFINITION column, the map source definition has the following general format:

```
<map_tile_layer
   name = "map tile layer name"
   image_format ="tile-image-format">
  <internal_map_source
    data_source="name-of-data-source"
    base_map="name-of-MapViewer-base-map"
    bgcolor="base-map-background-color"
    antialias="whether-to-turn-on-antialiasing"
    />
  </internal_map_source>
  <external_map_source
     url="external-map-service-url"
     adapter_class="name-of-adapter-class"
     proxy_host=" proxy-server-host "
     proxy_port="proxy-server-port"
     timeout="request-timeout"
     request_method="http-request-method: 'GET'|'POST'">
    <properties>
      <property name="property-name" value="property-value"/>
      …
    </properties>
  </external_map_source>
  <tile_storage
     root_path="disk-path-of-cache-root-directory"
  </tile_storage>
  <coordinate_system
     srid="coordinate-system-srid"
     minX="minimum-allowed-X-value"
     maxX="maximum-allowed-X-value"
     minY="minimum-allowed-Y-value"
     maxY="maximum-allowed-Y-value">
```

```
    </coordinate_system>
    <tile_image
       width="tile-image-width-in-screen-pixels"
       height="tile-image-height-in-screen-pixels" >
    </tile_image>
    <tile_bound>
       <coordinates> … </coordinates>
    </tile_bound>
    <zoom_levels
       levels="number-of-zoom-levels"
       min_scale="map-scale-at-highest-zoom-level"
       max_scale="map-scale-at-lowest-zoom-level"
       min_tile_width="tile-width-specified-in-map-data-units-at-
                       highest-zoom-level"
       max_tile_width="tile-width-specified-in-map-data-units-at-
                       lowest-zoom-level">
      <zoom_level
         description="zoom-level-description"
         level_name="zoom-level-name"
         scale="map-scale-of-zoom-level"
         tile_width ="tile-width-specified-in-map-data-units"
         tile_height ="tile-height-specified-in-map-data-units">
        <tile_bound>
          <coordinates> … </coordinates>
        </tile_bound>
      </zoom_level>
      …
    </zoom_levels>
  </map_tile_layer>
```

The DTD of the map tile layer definition XML is listed in Section A.9.

Example 8–3 shows the XML definition of an internal map tile layer, and Example 8–4 shows the XML definition of an external map tile layer. Explanations of the <map_tile_layer> element and its subelements follow these examples.

Example 8–3 XML Definition of an Internal Map Tile Layer

```
<?xml version = '1.0'?>
<!-- XML definition of an internal map tile layer.
-->
   <map_tile_layer image_format="PNG">
      <internal_map_source base_map="demo_map"/>
      <tile_storage root_path="/scratch/mapcache/"/>
      <coordinate_system
         srid="8307"
         minX="-180" maxX="180"
         minY="-90" maxY="90"/>
      <tile_image width="250" height="250"/>
      <zoom_levels>
         <zoom_level description="continent level" scale="10000000"/>
         <zoom_level description="country level" scale="3000000"/>
         <zoom_level description="state level" scale="1000000"/>
         <zoom_level description="county level" scale="300000"/>
         <zoom_level description="city level" scale="100000"/>
         <zoom_level description="street level" scale="30000"/>
         <zoom_level description="local street level" scale="10000"/>
      </zoom_levels>
   </map_tile_layer>
```

Example 8–4 XML Definition of an External Map Tile Layer

```
<?xml version = '1.0'?>
<!-- XML definition of an external map tile layer.
-->
   <map_tile_layer image_format="PNG">
      <external_map_source
         url="http://elocation.oracle.com/elocation/lbs"
         adapter_class="mcsadapter.MVAdapter">
         <properties>
            <property name="data_source" value="elocation"/>
            <property name="base_map" value="us_base_map"/>
         </properties>
      </external_map_source>
      <tile_storage root_path="/scratch/mapcache"/>
      <coordinate_system
         srid="8307"
         minX="-180" maxX="180"
         minY="-90" maxY="90"/>
      <tile_image width="250" height="250"/>
     <!—
         The following <zoom_levels> element does not have any
         <zoom_level> element inside it. But since it has its levels,
         min_scale and max_scale attributes set, map tile server will
         automatically generate the <zoom_level> elements for the 10
         zoom levels.
      -->
      <zoom_levels levels="10" min_scale="5000" max_scale="10000000" />
   </map_tile_layer>
```

The top-level element is <map_tile_layer>. The `image_format` attribute specifies the tile image format; the currently supported values for this attribute are `PNG`, `GIF`, and `JPG`. PNG and GIF images are generally better for vector base maps, while JPG images are generally better for raster maps, such as satellite imagery, because of a better compression ratio. Currently, only tile images in PNG format can have transparent background.

The <internal_map_source> element is required only if the map tiles are rendered by the local MapViewer instance. The `base_map` attribute is required and specifies the predefined MapViewer base map that is cached by the map tile server; its value should match an entry in the BASE_MAP column in the USER_SDO_CACHED_MAPS view. The bgcolor attribute is optional and specifies the background color of the map. If the value of this attribute is set to `NONE`, the background will be transparent. (Currently MapViewer can only render transparent PNG map tiles.)

The <external_map_source> element is required only if the map tiles are rendered by an external map services provider. This element has the following attributes:

- The `url` attribute is required and specifies the map service URL from which the map tiles can be fetched (for example, `http://myhost/mapviewer/omserver`).
- The `adapter_class` attribute is required and specifies the full name of the map adapter class, including the package names (for example, `mcsadapter.MVAdapter`).
- The `proxy_host` and `proxy_port` attributes are needed only if the external map provider server must be accessed through a proxy server; these attributes specify the host name and port number, respectively, of the proxy server. If `proxy_host` is specified as `NONE`, all map tile requests will be sent directly to the remote server without going through any proxy server. If `proxy_host` is omitted or specifies an

empty string, the global MapViewer proxy setting defined in the mapViewerConfig.xml file will be used when map tile requests are sent.

- The timeout attribute is optional and specifies the number of milliseconds for which the map tile server must wait for an external map tile image before giving up the attempt. The default timeout value is 15000.
- The request_method attribute is optional and the HTTP request method for sending map tile requests; its value can be POST (the default) or GET.

The <properties> element in the <external_map_source> element can include multiple <property> elements, each of which specifies a user-defined parameter for use by the map adapter when it fetches map tiles. The same map source adapter can use different set of parameters to fetch different map tile layers. For example, the sample MapViewer adapter mcsadapter.MVAdapter shipped with MapViewer accepts parameters defined as follows:

```
<properties>
   <property name="data_source" value="elocation"/>
   <property name="base_map" value="us_base_map"/>
</properties>
```

However, by changing the value attribute values, you can use this adapter to fetch a different base map from the same data source or a different data source.

The <tile_storage> element specifies storage settings for the map tile layer. The optional root_path attribute specifies the file system directory to be used as the root directory of the tile storage. If this attribute is omitted or invalid, the default root directory defined in the mapViewerConfig.xml file is used.

The <coordinate_system> element specifies the map coordinate system, and it has several required attributes. The srid attribute specifies the spatial reference ID of the coordinate system. The minX attribute specifies the lower bound of the X dimension; the minY attribute specifies the lower bound of the Y dimension; the maxX attribute specifies the upper bound of the X dimension; and the maxY attribute specifies the upper bound of the Y dimension. For the standard longitude/latitude (WGS 84) coordinate system, the srid value is 8307; and the minX, minY, maxX, and maxY values are -180, -90, 180, and 90, respectively.

For an internal map tile layer, the map coordinate system can be different from the data coordinate system. If the two are different, the map tile server transforms the map data into the coordinate system defined in the <coordinate_system> element and renders map tile images using this coordinate system.

The <tile_image> element specifies the tile image size settings, and it has the following required attributes: width specifies the width of the tile images in screen pixels, and height specifies the height of the tile images in screen pixels.

The optional <tile_bound> element specifies the bounding box of the cached map tiles. The map tile server only fetches tiles inside this box, and returns a blank tile if the requested tile is outside this box. The bounding box is specified by a rectangle in the map data coordinate system. The rectangle is specified by a <coordinates> element in the following format:

```
<coordinates>minX, minY, maxX, maxY</coordinates>
```

The default cache bounding box is the same bounding box specified in the <coordinate_system> element.

The <zoom_levels> element specifies the predefined zoom levels. Only image tiles at predefined zoom levels will be cached and served by the map tile server. The

<zoom_levels> element can have multiple <zoom_level> elements, each of which specifies one predefined zoom level. If there are no <zoom_level> elements, the map tile server automatically generates the <zoom_level> elements by using the following attributes inside the <zoom_levels> element. (These attributes can be omitted and will be ignored if any <zoom_level> elements exist.)

- levels specifies the total number of zoom levels.
- min_scale specifies the scale of map images at the highest (zoomed in the most) zoom level.
- max_scale specifies the scale of map images at the lowest (zoomed out the most) zoom level.
- min_tile_width specifies the width of map tiles at the highest zoom level. The width is specified in map data units.
- max_tile_width specifies the width of the map tiles at the lowest zoom level. The width is specified in map data units.

For the map tile server to be able to generate the definitions of individual zoom levels automatically, you must specify either of the following combinations of the preceding attributes:

- `levels`, `min_scale`, and `max_scale`
- `levels`, `min_tile_width`, and `max_tile_width`

When the zoom levels are defined this way, the map tile server automatically derives the definition of all the individual zoom levels and updates the XML definition with the <zoom_level> elements generated for the zoom levels. You can then make adjustments to each zoom level if you want.

Each zoom level is assigned a zoom level number by the map tile server based on the order in which the zoom levels are defined. The first zoom level defined in the <zoom_levels> element is zoom level 0, the second zoom level is zoom level 1, and so on. These zoom level numbers are used in the tile requests to refer to the predefined zoom levels.

The <zoom_level> element specifies a predefined zoom level, and it has several attributes. The `description` attribute is optional and specifies the text description of the zoom level. The `level_name` attribute is optional and specifies the name of the zoom level. The `scale` attribute specifies the map scale of the zoom level; it is required if the attributes `tile_width` and `tile_height` are not defined. The `tile_width` and `tile_height` attributes specify the tile width and height, respectively, in map data units. The `fetch_larger_tiles` attribute is optional and specifies whether to fetch larger map images instead of the small map image tiles; a value of `TRUE` (the default) means that larger map images that may consist multiple map tiles will be fetched and broken into small map image tiles, which might save network round trips between the map tile server and the map services provider.

In the <zoom_level> element, you must specify either the `scale` attribute or both the `tile_width` and `tile_height` elements.

The <tile_bound> element within the <zoom_level> element optionally specifies the bounding box of the cached map tiles for the zoom level. The map tile server only fetches tiles inside this box, and returns a blank tile if the requested tile is outside this box. The bounding box is specified by a rectangle specified in map data coordinate system. The rectangle is specified by a <coordinates> element (explained earlier in this section) If you specify the <tile_bound> element within the <zoom_level> element, it overrides the overall cache bounding box settings specified by the <tile_bound> element that is above it in the XML hierarchy.

8.2.3 External Map Source Adapter

An external map source adapter is the interface between a map tile server and an external map services provider. When a map image tile needs to be fetched from the external map services provider, the map tile server calls the adapter with information about the zoom level, size, and location of the tile. The adapter then constructs a provider-specific request, sends the request to the external map services provider, and return the resulting image tile to the map tile server.

The external map source adapter is a Java class that must extends the abstract Java class `oracle.mapviewer.share.mapcache.MapSourceAdapter`, which is defined as follows:

```
public abstract class MapSourceAdapter
{
   public abstract String getMapTileRequest(TileDefinition tile);
   public byte[] getTileImageBytes(TileDefinition tile) ;
   public Properties getProperties() ;
}
```

An adapter that extends this class must implement the following method:

- `public String getMapTileRequest(TileDefinition tile)`

  This method should implement the logic to construct the HTTP request string that can be sent to the map services provider to fetch the map image tile. For example, if the URL of a map tile is `http://myhost/mymapserver?par1=v1&par2=v2&par3=v3`, the HTTP request string returned by this method should be `par1_v1&par2=v2&par3=v3`.

  When the map tile server cannot find a specific map tile, it calls the `getTileImageBytes` method to fetch the binary data of the tile image, and that method calls the `getMapTileRequest` method to construct the map tile request before fetching the tile. The `getMapTileRequest` method takes one parameter: a `TileDefinition` object that specifies the zoom level, bounding box, image size and image format of the requested tile. This method returns the HTTP request string.

The map source adapter also inherits all methods implemented in class `MapSourceAdapter`. Among them, the following methods are more important than the others:

- `public byte[] getTileImageBytes(TileDefinition tile)`

  This method fetches the actual binary map tile image data from the external map service provider. This method is already implemented. It calls the abstract method `getMapTileRequest` to construct the map tile request and sends the request to the external map services provider. If the map tiles cannot be fetched by sending HTTP requests, you can override this method to implement the appropriate logic to fetch an image tile from the map source. This method takes one parameter: a `TileDefinition` object that specifies the zoom level, bounding box, image size, and image format of the requested tile. This method returns the binary tile image data encoded in the image format specified in the map tile layer configuration settings.

- `public Properties getProperties()`

  This method returns the provider-specific parameters defined in the map tile layer configuration settings explained in Section 8.2.2.2.

The `MapSourceAdapter` and `TileDefinition` classes are packaged inside `mvclient.jar`, which can be found under the directory *$MAPVIEWER_HOME*/web/WEB/lib.

Example 8–5 shows an external map source adapter.

Example 8–5 External Map Source Adapter

```
/**
 * This is a sample map source adapter that can be used to fetch map
 * tiles from a MapViewer instance.
 */
package mcsadapter ;

import java.awt.Dimension;
import java.net.URL;
import java.util.Properties;
import oracle.lbs.mapclient.MapViewer;
import oracle.lbs.mapcommon.MapResponse;
import oracle.mapviewer.share.mapcache.*;

/**
 * The map source adapter must extend class
 * oracle.lbs.mapcache.cache.MapSourceAdapter.
 */

public class MVAdapter extends MapSourceAdapter
{
  /**
   * Gets the map tile request string that is to be sent to the map
   * service provider URL.
   * @param tile tile definition
   * @return request string
   */
  public String getMapTileRequest(TileDefinition tile)
  {
    // Get map source specified parameters
    Properties props = this.getProperties() ;
    String dataSource = props.getProperty("data_source") ;
    String baseMap = props.getProperty("base_map") ;
    // Use oracle.lbs.mapclient.MapViewer to construct the request string
    MapViewer mv = new MapViewer(this.getMapServiceURL()) ;
    mv.setDataSourceName(dataSource);
    mv.setBaseMapName(baseMap);
    mv.setDeviceSize(new Dimension(tile.getImageWidth(),
                                   tile.getImageHeight()));
    mv.setCenterAndSize(tile.getBoundingBox().getCenterX(),
                        tile.getBoundingBox().getCenterY(),
                        tile.getBoundingBox().getHeight());
    int format = MapResponse.FORMAT_PNG_STREAM ;
    String req = null ;
    switch(tile.getImageFormat())
    {
      case TileDefinition.FORMAT_GIF:
        mv.setImageFormat(MapResponse.FORMAT_GIF_URL);
        req = mv.getMapRequest().toXMLString().replaceFirst(
                        "format=\"GIF_URL\"", "format=\"GIF_STREAM\"") ;
        break ;
      case TileDefinition.FORMAT_PNG:
        mv.setImageFormat(MapResponse.FORMAT_PNG_URL);
        req = mv.getMapRequest().toXMLString().replaceFirst(
```

```
                    "format=\"PNG_URL\"", "format=\"PNG_STREAM\"") ;
        break ;
      case TileDefinition.FORMAT_JPEG:
        mv.setImageFormat(MapResponse.FORMAT_JPEG_URL);
        req = mv.getMapRequest().toXMLString().replaceFirst(
                    "format=\"JPEG_URL\"", "format=\"JPEG_STREAM\"");
        break ;
    }

    byte[] reqStr = null ;
    try
    {
      reqStr = req.getBytes("UTF8") ;
    }
    catch(Exception e)
    {}
    // Return the request string.
    return "xml_request="+ new String(reqStr);
  }
}
```

Example 8–6 shows the implementation of the `MapSourceAdapter.getTileImageBytes` method.

Example 8–6 MapSourceAdapter.getTileImageBytes Implementation

```
/**
 * Fetches the map image tile from the external map service provider by
 * sending the HTTP map tile request to the map service provider, and
 * return the binary tile image data. You can rewrite this method so that
 * the adapter can fetch the tile from an external map service provider
 * that does not accept HTTP requests at all.
 * @param tile the tile definition
 * @return the binary tile image data.
 * @throws Exception
 */
public byte[] getTileImageBytes(TileDefinition tile)
  throws Exception
{
  // construct request string
  String request = getMapTileRequest(tile) ;

  if(request == null)
  {
    throw new Exception("Null map tile request string in map source adapter!") ;
  }

  // set proxy settings
    Proxy proxy = null ;

  /* If the proxyHost is "NONE", the request is sent directly to the
   * external server. If the proxyHost is a valid host, that host will
   * be used as the proxy server. If the proxyHost is empty of omitted,
   * the global proxy setting in mapViewerConfig.xml will be in effect.
   */
  boolean noProxy = "NONE".equalsIgnoreCase(getProxyHost()) ;
  if(getProxyHost()!=null && !noProxy)
  {
    SocketAddress addr = new InetSocketAddress(proxyHost, proxyPort);
    proxy = new Proxy(Proxy.Type.HTTP, addr);
```

```
}

// send the request and get the tile image binary
PrintWriter wr = null ;
BufferedInputStream bis = null;
try
{
  String urlStr = mapServiceURL ;
  if("GET".equalsIgnoreCase(httpMethod))
    urlStr = mapServiceURL + "?" + request ;
  log.finest("http "+httpMethod+": "+urlStr);

  URL url = new URL(urlStr);
  // Open a URL connection based on current proxy setting
  URLConnection conn =
    proxy!=null? url.openConnection(proxy):
                 (noProxy? url.openConnection(Proxy.NO_PROXY):
                           url.openConnection()) ;
  conn.setConnectTimeout(timeOut);
  if("GET".equalsIgnoreCase(getHTTPMethod()))
    conn.connect();
  else
  {
    conn.setDoOutput(true);
    wr = new PrintWriter(conn.getOutputStream());
    wr.print(request);
    wr.flush();
    wr.close();
    wr = null ;
  }
  bis = new BufferedInputStream(conn.getInputStream());
  byte[] result = toBytes(bis) ;
  bis.close();
  bis = null ;
  return result;
}
catch(Exception ioe)
{
  throw new Exception("Failed to fetch external map tile.", ioe);
}
finally
{
  try
  {
    if(bis != null)
    {
      bis.close();
      bis = null;
    }
    if(wr != null)
    {
      wr.close();
      wr = null;
    }
  }
  catch(IOException ioee)
  {
    throw ioee;
  }
}
```

```
}
```

8.3 Feature of Interest (FOI) Server

A feature of interest (FOI) is a business entity or geographical feature that can be manipulated or interacted with by a JavaScript map client running in the web browser. FOI data is dynamically displayed and is not part of the map tile layer. FOIs can be any spatial geometry type, such as points, line strings, and polygons. The ability to search, browse, inspect, and interact with FOIs is essential for location-based services.

The FOI server is a Java servlet running inside MapViewer. It responds to FOI requests from a JavaScript map client by querying the database, rendering FOI images, and sending the FOI images along with FOI attribute data to the client. The JavaScript map client displays the FOI images to the end user and provides interaction with the images.

The FOI server accepts the following types of FOI requests: theme-based and user-defined. Each type of FOI request returns a data layer appropriate for the request type.

8.3.1 Theme-Based FOI Layers

A theme-based FOI layer is a collection of spatial features that have similar characteristics and that are stored in the database. The client fetches a theme-based FOI layer by sending a theme-based FOI layer request to the FOI server. The result of this request is a collection of FOI data entries that meets certain query criteria. Each FOI data entry contains the FOI image, as well as FOI attributes that can be used by the JavaScript map client to implement client-side interactivity.

A theme-based FOI layer is based on a predefined MapViewer theme (see Section 8.3.1.1) or a dynamic JDBC query theme (see Section 8.3.1.3, which defines all information necessary for FOI data rendering. The information includes the table in which the geometry features are stored, the criteria to use during the database query, the attributes that are part of the FOI data, and the style to use when rendering the FOI images. Predefined themes can be defined and configured using the Map Builder tool, which is described in Chapter 9.

8.3.1.1 Predefined Theme-Based FOI Layers

When the client requests FOI data using a predefined theme-based FOI request, it must specify the name of a predefined theme, the scale of the feature images, and the query window used to query the geometry features. The theme name must be defined by the application, while the scale of the feature images and the query window are automatically calculated by the JavaScript map client.

For example, a predefined theme named `CUSTOMERS` could be defined on a table named CUSTOMERS, which has the following definition:

```
SQL> DESCRIBE CUSTOMERS
 Name                              Null?  Type
 --------------------------------- ------ ----------------------------
 NAME                                      VARCHAR2(64 CHAR)
 CITY                                      VARCHAR2(64 CHAR)
 COUNTY                                    VARCHAR2(64 CHAR)
 STATE                                     VARCHAR2(64 CHAR)
 LOCATION                                  SDO_GEOMETRY
 SALES                                     NUMBER
```

The LOCATION column is the spatial column that is used for rendering the customer markers.

The XML styling rules for the CUSTOMERS theme are shown in Example 8–7.

Example 8–7 XML Styling Rules for Predefined Theme Used for FOI Layer

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="CITY" name="City"/>
    <field column="SALES" name="Sales"/>
  </hidden_info>
  <rule>
    <features style="M.CIRCLE"> </features>
    <label column="NAME" style="T.TEXT"> 1 </label>
  </rule>
</styling_rules>
```

The styling rules in Example 8–7 specify the following. To see how these specifications affect the map display, see Figure 8–2, "Application Created Using Oracle Maps (V1 API)" in Section 8.1.2.

- The marker style M.CIRCLE is used to render the customers.
- The NAME column is used as the labeling attribute (label column="NAME"). The value in the NAME column (the name of the customer) is included in the information window that the JavaScript map client displays when the user moves the mouse over the customer marker.
- The information window also includes the values in columns specified in the <hidden_info> element (CITY and SALES in this example) for that customer. Each <field> element specifies two attributes: column to identify the database column and name to identify a text string to be used in the information window.

8.3.1.2 Templated Predefined Themes

The predefined MapViewer theme can be a standard predefined theme or a templated predefined theme. Both types of predefined themes are defined in the USER_SDO_THEMES view. However, the query conditions of a standard predefined theme are fixed, whereas the query conditions of a templated predefined theme can contain dynamic binding variables whose values can be changed when the theme request is issued.

Example 8–8 shows the XML styling rules for a templated predefined theme that uses two binding variables (with the relevant text shown in bold in the <features> element).

Example 8–8 XML Styling Rules for a Templated Predefined Theme

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="NAME" name="Name"/>
    <field column="CITY" name="City"/>
    <field column="SALES" name="Sales"/>
  </hidden_info>
  <rule>
    <features style="M.CIRCLE">(city=:1 and sales>:2)</features>
    <label column="NAME" style="T.TEXT"> 1 </label>
  </rule>
```

```
</styling_rules>
```

In Example 8–8, the binding variable `:1` specifies the name of the city in which the qualifying features must be located, and the binding variable `:2` specifies the minimum sales volume of the qualifying features. (That is, only customers in a specified city and with sales above a certain minimum will have store markers displayed.) The values of these two binding variables are not fixed when the theme is defined; instead, they are provided in the requests that the client sends to the server.

8.3.1.3 Dynamic JDBC Query Theme-Based FOI Layers

When the client requests FOI data using a dynamic JDBC theme-based FOI request, it must specify the complete definition of the JDBC theme. The theme definition must specify the rendering style and the SQL query that is to be used to query FOI data, including all geometry and non-geometry attributes.

Example 8–9 shows some JavaScript client code to create an FOI layer that displays a buffer around each customer location.

Example 8–9 Theme for Dynamic JDBC Query

```
var theme = '<themes><theme name="JDBC_THEME" >' +
   '<jdbc_query asis="true" spatial_column="location"
        jdbc_srid="8307" render_style="C.RED"
        datasource="mvdemo">' +
   'select sdo_geom.sdo_buffer(A.location,1,0.005,'+
   '\'unit=mile arc_tolerance=0.005\') location '+
   ' from customers A' +
   '</jdbc_query></theme></themes>' ;
buffertheme = new MVThemeBasedFOI('buffertheme',theme);
```

8.3.2 User-Defined FOI Requests

A user-defined FOI is a feature defined on the client side. Unlike the theme-based FOI layer, which is rendered as a collection of features, the user-defined FOI is requested and rendered on an individual basis.

All attributes of the user-defined FOI, including the geometry representation and rendering style, must be provided by the application. The JavaScript map client sends the request, with the geometry representation and rendering style information, to the FOI server. The FOI server renders the FOI image and returns it to the client. The rendering style must be predefined in the USER_SDO_STYLES view.

8.4 Oracle Maps JavaScript API

The Oracle Maps JavaScript client is a browser-based map visualization engine that works on top of the map tile server and the FOI server. It implements the following functions:

- Fetching map tiles from the map tile server and displaying them as a map tile layer in the web browser.
- Sending FOI requests to the FOI server, and overlaying user-defined features and Oracle Spatial and Graph query-based features on top of the map tile layer.
- Controlling user interaction, such as dragging for map navigation, clicking FOIs, drawing rectangles, and redlining.

Drawing a rectangle refers to the application user creating a rectangle by clicking and holding the mouse button at one corner of the rectangle, dragging the mouse to the diagonally opposite corner, and releasing the mouse button.

Redlining refers to the application user creating a polygon or polyline by clicking the mouse button and then moving the mouse and clicking multiple times, with each click extending the redline by a straight line. (Redline drawings are often rendered in red, although you can specify a line style that uses any color.)

To access these functions, use the JavaScript API, which consists of several JavaScript classes. The JavaScript API has two versions:

- Version 1 (V1), the traditional API that is still supported, and described in Section 8.4.1, "JavaScript API V1"
- Version 2 (V2), a new API introduced in Release 11.1.1.7, and described in Section 8.4.2, "JavaScript API V2"

For detailed information about all classes in the Oracle Maps JavaScript API (V1 and V2), see the Javadoc-style reference documentation, which is included in the `mvdemo.ear` file and is available at the following locations:

`http://`*`host:port`*`/mvdemo/api/oracle_maps_api.jsp` (for V1)

`http://`*`host:port`*`/mvdemo/api/oracle_maps_html5_api.jsp` (for V2)

Tutorials and demos for both the V1 and V2 APIs are available as a standalone packaged application with the root context path `/mvdemo`. The tutorials start with the basics (display a map tile layer, add a navigation panel, display interactive features and information windows) and move on to more complex topics such as registering event listeners, programmatically creating and using styles, and spatial filtering.

The tutorials are all based on the MVDEMO sample data set (available from the MapViewer page on the Oracle Technology Network) and assume a data source named mvdemo. The tutorial page has three panels. The left one lists the sample code, or demo, titles. Click on one and a map, or the result of executing that sample code, is displayed in the top right panel. The bottom panel has tabs titled JavaScript and HTML, which respectively show the JavaScript and HTML code fragments for the selected demo.

8.4.1 JavaScript API V1

To access the functions of the Oracle Maps JavaScript client, use the JavaScript API Version 1 (V1), which consists of several JavaScript classes, including the following:

- The `MVMapView` class is the main entry point of the API. It implements most of the map control interfaces.
- The `MVMapTileLayer` class (formerly called the `MVBaseMap` class) defines a map tile layer that displays map tiles rendered by the map tile server.
- The `MVThemeBasedFOI` class defines and controls the theme based FOI layers.
- The `FOI` class defines and controls user-defined FOIs.
- The `MVSdoGeometry` class defines a geometry object. The geometry can be in any geometry type that is supported by Oracle Spatial and Graph.
- The `MVRedLineTool` class defines and controls the redline utility.
- The `MVRectangleTool` class defines and controls the rectangle tool.

- The `MVOverviewMap` class defines and controls the overview map that displays the miniature overview of the main map as a small rectangle (which is itself inside a rectangle tool).
- The `MVMapDecoration` class defines and controls map decorations.

`MVMapView` is the main entry class for all map operations inside the web browser. `MVMapView` and the other classes provide all essential interfaces for adding logic to your web mapping applications. These logical operations can include the following:

- Create a map client instance and associate it with the map container DIV object created in the web page.
- Configure map parameters such as map center and map zoom level.
- Create and manipulate map tile layers.
- Create and manipulate theme-based FOI layers.
- Create and manipulate user-defined individual FOIs.
- Display an information window on the map.
- Create fixed map decorations, such as a map title, custom copyright notes, and control buttons.
- Access built-in utilities such as the navigation bar, scale bar, rectangle tool, redline tool, and overview map.
- Use event listeners to customize the event handling. You can add event listeners to the `MVMapView`, `MVThemeBasedFOI`, and `MVFOI` classes using the appropriate API methods.

8.4.2 JavaScript API V2

The Oracle Maps JavaScript API Version 2 (V2) takes advantage of the capabilities of modern browsers. Some of its features include:

- Built-in support of various third party map tile services, such as maps.oracle.com, Nokia Maps, Bing Maps, OpenStreet Maps, and other mapping service providers
- Rich client side rendering of geospatial data with on-the-fly application of rendering styles and effects such as gradients, animation, and drop-shadows
- Autoclustering of large number of points and client side heat map generation
- Client side feature filtering based on attribute values as well as spatial predicates (query windows)
- A rich set of built-in map controls and tools, including a customizable navigation bar and information windows, configurable layer control, and red-lining and distance measurement tools

The V2 API is *not* backward compatible with the existing Oracle Maps JavaScript V1 API applications. If you want to use V2-specific features with existing V1 applications (that is, applications written with the V1 API using classes such as `MVThemeBasedFOI`), those applications will need to be migrated first.

Note, however, that existing server-side predefined styles and themes will work with the V2 API. For example, the following code snippet creates an interactive vector layer based on a predefined theme `mvdemo.customers`, which has an associated predefined style:

```
var baseURL  = "http://"+document.location.host+"/mapviewer";
var layer = new OM.layer.VectorLayer("layer1",
```

```
{
    def:{
               type:OM.layer.VectorLayer.TYPE_PREDEFINED,
               dataSource:"mvdemo",
               theme:"customers",
               url: baseURL
           }
 });
```

The V2 API has the following top-level classes and subpackages, all of which are in the namespace `OM`:

- The `Map` class is the main class of the API.
- The `Feature` class represents individual geo features (or FOIs as they were known in V1).
- The `MapContext` class a top-level class encapsulating some essential contextual information, such as the current map center point and zoom level. It is typically passed into event listeners.
- The `control` package contains all the map controls, such as navigation bar and overview map.
- The `event` package contains all the map and layer event classes.
- The `filter` package contains all the client-side filters (spatial or relational) for selecting, or subsetting, the displayed vector layer features.
- The `geometry` package contains various geometry classes.
- The `layer` package contains various tile and vector layer classes. The tile layer classes include access to a few online map services such as Oracle, Nokia, Bing, and OpenStreetMap. The vector layers are interactive feature layers and correspond to the `MVThemeBasedFOI` and MVFOI `classes` of V1.
- The `infowindow` package contains the customizable information windows and their styles.
- The `style` package contains styles applicable to vector data on the client side. It also includes visual effects such as animation, gradients, and drop shadows.
- The `tool` package contains various map tools such as for distance measuring, red-lining, and geometry drawing.
- The `universe` package contains built-in, or predefined, *map universes*. A map universe defines the bounding box and set of zoom level definitions for the map content. It is similar to a tile layer configuration in the V1 API.
- The `util` package contains various utility classes.
- The `visualfilter` package provides an interface for the various visual effects, such as gradients and drop shadows.

`OM.Map` is the main entry class for all map operations inside the web browser. This and other classes provide interfaces for adding application-specific logic, operations, and interactivity in web mapping applications. The application logic and operations can include the following:

- Create a map client instance and associate it with the map container DIV object created in the web page.
- Configure map parameters such as map center and map zoom level.

- Optionally, create and manipulate map tile layers. Unlike in V1, a map tile layer is not required in V2. An application can have only interactive vector layers using a custom Universe that programmatically defines the zoom levels and scales.
- Create and manipulate vector layers (known as FOIs in V1).
- Display an information window on the map.
- Create fixed map decorations, such as a map title, a copyright note, and map controls.
- Access built-in utilities such as a navigation panel, rectangle or circle tool, scale bar, and overview map panel.
- Use event listeners to customize event handling and thus map interactions.

For information about developing applications using the V2 API, see Section 8.5.2, "Using the V2 API" and the Oracle-supplied tutorials and demos.

8.4.3 V1 and V2 APIs: Similarities and Differences

Both V1 and V2 APIs have major similarities:

- They have the same architecture and content organization. (Figure 8–1, "Architecture for Oracle Maps Applications" and Figure 8–4, "Layers in a Map" apply to both versions.)
- They depend on Oracle Spatial and Graph or Locator for spatial analysis (proximity, containment, nearest neighbor, and distance queries) and coordinate system support (SRIDs and transformations).

However, there are some significant differences:

- The V2 client-side rendering of interactive features (that is, using HTML5 Canvas or SVG) provides for a richer client interactivity and user experience.
- The V1 "FOI server" is in V2 a data server that streams the vector geometries and attributes for features to the client for local rendering. Therefore, the V1 "FOI layers" and called **vector layers** in V2.
- In V2, a background map tile layer is not required in order to display interactive vector layers. So in V2, for example, an application can display a thematic map of states (such as color-filled by population quintile) with no background tile layer.
- The V2 API depends on and includes `JQuery` and `JQueryUI`. So, `oraclemapsv2.js` includes `jquery-1.7.2.min.js` and `jquery-ui-1.8.16.min.js`. If your application also uses JQuery and JQueryUI and includes them already, then use the file `oraclemapsv2_core.js` in the `<script>` tag instead to load the Oracle Maps V2 library. That is, use the following:

  ```
  <script src="/mapviewer/jslib/v2/oraclemapsv2_
  core.js"></script>
  ```

  instead of:

  ```
  <script src="/mapviewer/jslib/v2/oraclemapsv2.js"></script>
  ```

Table 8–2 shows the general correspondence between the classes in V1 and V1, although the relationships are not always one-to-one.

Table 8–2 Correspondence Between V1 and V2 API Classes

V1 API Class	V2 API Class
`MVMapView`	`OM.Map`
`MVMapTileLayer`, `MVBingTileLayer`, built-in tile layers	`OM.layer.TileLayer`, `OM.layer.BingTileLayer`, `ElocationTileLayer`, `NokiaTileLayer`, `OSMTileLayer`
`MVCustomMapTileLayer`	Custom tile layers are not directly supported in the current release of V2. However, you can use custom tile layers by extending `OM.layer.TileLayer` and supplying a getTileURL callback function.
`MVThemeBasedFOI`	`OM.layer.VectorLayer`
`MVFOI`	`OM.Feature`
`MVSdoGeometry`	`OM.geometry` and its subclasses
`MVEvent`	`OM.event` and its subclasses
`MVInfoWindowTab`	`OM.infowindow.MVInfoWindowTab`
Styles (`MVStyleColor`, `MVXMLStyle`, `MVBucketStyle`, `MVBarChartStyle`, and so on)	`OM.style` and its subclasses
Tools (`MVToolbar`, `MVDistanceTool`, `MVCircleTool`, and so on)	`OM.tool` and its subclasses
Decorations and controls (`MVNavigationPanel`, `MVMapDecoration`, `MVScaleBar`, and so on)	`OM.control` and its subclasses

8.5 Developing Oracle Maps Applications

If you have all your map data stored in an Oracle database and have MapViewer deployed in Oracle Fusion Middleware, you can develop a web-based mapping application using Oracle Maps by following the instructions in the section relevant to the API version that you are using:.

- Using the V1 API
- Using the V2 API

8.5.1 Using the V1 API

To develop Oracle Maps applications using the Version 1 (V1) API, follow the instructions in these sections:

- Creating One or More Map Tile Layers
- Defining FOI Metadata
- Creating the Client Application with the V1 API

8.5.1.1 Creating One or More Map Tile Layers

For each map tile layer displayed on the client side that is served by MapViewer, you must create the corresponding map tile layer on the MapViewer server side. For example, for the sample application described in Section 8.1.2, you must create a map tile layer on the server side to display oceans, county boundaries, cities and highways

as a map tile layer on the client. However, if the tile layer is a custom or built-in eternal tile layer, you do not need to define the tile layer on the server side.

Before you can create a map tile layer, you must ensure that the map source from which the map tiles images are to be rendered is ready. If the map tile images are rendered based on map data stored in the database, you must create a MapViewer base map that consists of a set of predefined themes. (You can create the base map using the Map Builder tool, which is described in Chapter 9.) If the map tiles images are rendered by an external map provider, you must write a map source adapter that can fetch map images from the external server using the tile image definition specified by the map tile server.

When the map source is ready, you can create the map tile layer using the MapViewer administration page, as described in Section 1.5.3. When you create the map tile layer, you must provide proper coordinate system definition, map source definition (internal or external), and zoom level definition (number of zoom levels and map scales).

After you create the map tile layer, you can test it by using a JavaServer Page (JSP) demo application shipped with MapViewer. The JSP demo application can be accessed at `http://`*`host:port`*`/mapviewer/fsmc/omaps.jsp`. Based on your input, this application can display maps served by any map tile layer defined with the MapViewer instance.

8.5.1.2 Defining FOI Metadata

If your application needs to display dynamic features based on database query results as theme-based FOI layers, you must create a predefined MapViewer theme for each theme-based FOI layer. If your application needs to display individual dynamic features as user-defined FOIs, you must define the rendering style or styles used by the FOI server to render the FOI images. You can use the Map Builder tool (described in Chapter 9) to create predefined themes and rendering styles.

8.5.1.3 Creating the Client Application with the V1 API

Oracle Maps client applications running inside web browsers are pure HTML and JavaScript pages that do not require any plug-ins. Therefore, you can build the application using any web technology that delivers content as pure HTM. Such technologies include JavaServer Pages, Java Servlets, ASP, and .NET C#. This section discusses client application development only in pure HTML format, but you can easily apply this information to other web technologies.

As shown in Example 8–1 in Section 8.1.2, the source code for an Oracle Maps application is typically packaged in an HTML page, which consists of the following parts:

- A `<script>` element that loads the Oracle Maps client library into the browser JavaScript engine. In Example 8–1, this element is:

  ```
  <script language="Javascript" src="jslib/oraclemaps.js"></script>
  ```

- An HTML DIV element that is used as the map container in the web page. The size and positioning of the DIV element can be customized to suit your needs. In Example 8–1, this element is:

  ```
  <div id="map" style="left:10; top:60;width: 600px; height: 500px"></div>
  ```

- JavaScript code that creates and initializes the map client instance. It creates the map client instance, sets up the initial map content (map tile layer, FOI layers, and so on), sets the initial map center and zoom level, implements application-specific logic, displays the map, and implements other application-specific logic.

This code should be packaged inside a JavaScript function, which is executed when the HTML page is loaded from the server to the client web browser. In Example 8–1, this function is named `on_load_mapview`:

```
function on_load_mapview()
{
  var baseURL  = "http://"+document.location.host+"/mapviewer";
  // Create an MVMapView instance to display the map
  var mapview = new MVMapView(document.getElementById("map"), baseURL);
  // Add a map tile layer as background.
  mapview.addMapTileLayer(new MVMapTileLayer("mvdemo.demo_map"));
  // Add a theme-based FOI layer to display customers on the map
  var themebasedfoi = new MVThemeBasedFOI('themebasedfoi1','mvdemo.customers');
  themebasedfoi.setBringToTopOnMouseOver(true);
  mapview.addThemeBasedFOI(themebasedfoi);
  // Set the initial map center and zoom level
  mapview.setCenter(MVSdoGeometry.createPoint(-122.45,37.7706,8307));
  mapview.setZoomLevel(4);
  // Add a navigation panel on the right side of the map
  mapview.addNavigationPanel('east');
  // Add a scale bar
  mapview.addScaleBar();
  // Display the map.
  mapview.display();
}
```

This function is specified in the `onload` attribute of the `<body>` element, so that it is executed after the web page is loaded. In Example 8–1, this code is as follows:

```
<body onload= JavaScript:on_load_mapview() >
```

- Additional HTML elements and JavaScript code implement other application-specific user interfaces and control logic. In Example 8–1 in Section 8.1.2, a JavaScript function `setLayerVisible` is implemented to show or hide the theme-based FOI layer when the user checks or unchecks the **Show customers** check box. The setLayerVisible function is coded as follows:

```
function setLayerVisible(checkBox)
{
        // Show the theme-based FOI layer if the check box is checked
        // and hide the theme-based FOI layer otherwise.
  if(checkBox.checked)
    themebasedfoi.setVisible(true) ;
  else
    themebasedfoi.setVisible(false);
}
```

This function is specified in the `onclick` attribute of the `<INPUT>` element that defines the check box, so that it is executed whenever the user clicks on the check box. In Example 8–1, this code is as follows:

```
<INPUT TYPE="checkbox" onclick="setLayerVisible(this)" checked/>Show customers
```

8.5.2 Using the V2 API

Developing applications with the V2 API is similar to the process for the V1 API. If all the spatial data used for base maps, map tile layers, and interactive layers or themes is stored in an Oracle database, then the map authoring process using the Map Builder tool is the same for both APIs.

If the underlying base map and layers are managed in an Oracle database, each map tile layer displayed in the client application must have a corresponding database metadata entry in the USER_SDO_CACHED_MAPS metadata view (described in Section 8.2.2.2) . Similarly, if an interactive layer is based on database content, it must have a metadata entry in the USER_SDO_THEMES view (described in Section 2.9, especially Section 2.9.2). These tile and interactive layers, and the styles and styling rules for them, can be defined using the Map Builder tool (described in Chapter 9).

To develop Oracle Maps applications using the Version 2 (V2) API, follow these basic steps:

1. Import the `oraclemapsv2.js` library.

   The API is provided in a single JavaScript library packaged as part of the MapViewer EAR archive.

2. After MapViewer is deployed and started, load the library through a `<script>` tag, for example:

   ```
   <script  type="text/javascript"
   url="http://localhost:8080/mapviewer/jslib/v2/oraclemapsv2.js"/>
   ```

3. Create a `<DIV>` tag in the HTML page, which will contain the interactive map. (This is the same as in the V1 API.)

4. Create a client-side map instance that will handle all map display functions.

   The class is named `OM.Map` and is the main entry point of the V2 API. So, `OM.Map` in V2 is equivalent to `MVMApView` in V1.

5. Set up a map universe (unless you also do the optional next step).

   A map universe basically defines the overall map extent, the number of zoom levels, and optionally the resolution (in map units per pixel) at each zoom level. In the V1 API, this information is contained in a tile layer definition. Those will continue to work in V2; however, in V2 a predefined tile layer is not necessary in order to display interactive vector layers or themes. For example, an interactive thematic map of sales by region does not need to have a background map, or tile layer.

6. (Optional) Add a tile layer that serves as the background map.

   The tile layer can be from the database, such as `mvdemo.demo_map`, or from a supported service, such as Nokia Maps. Adding a tile layer also implicitly defines a map universe, and therefore the preceding step (setting up a map universe) is not necessary in this case.

7. Add one or more interactive vector layers.

   An `OM.layer.VectorLayer` is equivalent to `MVThemeBasedFOI` in the V1 API. The main difference in that `OM.VectorLayer` uses HTML5 (Canvas or SVG) technology to render all the data in the browser. So, unless specified otherwise, all vector layer content is loaded once and there are no subsequent database queries, or data fetching, on map zoom or pan operations.

8. Add one or more map controls, tools, and other application-specific UI controls so that users can set the displayed layers, styling, and visual effects.

For detailed instructions and related information, see the Oracle-supplied tutorials and demos.

8.5.2.1 Creating the Client Application with the V2 API

Oracle Maps V2 applications run inside web browsers and require only HTML5 (Canvas) support and JavaScript enabled. No additional plugins are required.

As shown in Example 8–1, "Source Code for the Simple Application (V1 API)"in Section 8.1.2, the source for an Oracle Maps application is typically packaged in an HTML page, which consists of the following parts:

- A `<script>` element that loads the Oracle Maps V2 client library into the browser’s JavaScript engine. For example:

  ```
  <script src="/mapviewer/jslib/v2/oraclemapsv2.js"></script>
  ```

- An HTML `<div>` element that will contain the map. For example:

  ```
  <div id="map" style="width: 600px; height: 500px"></div>
  ```

- JavaScript code that creates the map client instance and sets the initial map content (tile and vector layer), the initial center and zoom, and map controls. This code should be packaged inside a function which is executed when the HTML page is loaded or ready. The function is specified in the `onload` attribute of the `<body>` element of the HTML page. For example:

  ```
  function on_load_mapview()
  {
    var baseURL  = "http://"+document.location.host+"/mapviewer";
    // Create an OM.Map instance to display the map
    var mapview = new OM.Map(document.getElementById("map"),
                             {
                               mapviewerURL:baseURL
                             });
    // Add a map tile layer as background.
    var tileLayer = new OM.layer.TileLayer(
          "baseMap",
          {
              dataSource:"mvdemo",
              tileLayer:"demo_map",
              tileServerURL:baseURL+"/mcserver"
          });
    mapview.addLayer(tileLayer);
    // Set the initial map center and zoom level
    var mapCenterLon = -122.45;
    var mapCenterLat = 37.7706;
    var mapZoom = 4;
    var mpoint = new OM.geometry.Point(mapCenterLon,mapCenterLat,8307);
    mapview.setMapCenter(mpoint);
    mapview.setMapZoomLevel(mapZoom);
    // Add a theme-based FOI layer to display customers on the map
    customersLayer = new OM.layer.VectorLayer("customers",
          {
              def:
                  {
                  type:OM.layer.VectorLayer.TYPE_PREDEFINED,
                  dataSource:"mvdemo", theme:"customers",
                  url: baseURL,
                  loadOnDemand: false
                  }
          });
    mapview.addLayer(customersLayer);
    // Add a navigation panel on the right side of the map
    var navigationPanelBar = new OM.control.NavigationPanelBar();
  ```

```
  navigationPanelBar.setStyle(
{backgroundColor:"#FFFFFF",buttonColor:"#008000",size:12});
  mapview.addMapDecoration(navigationPanelBar);
  // Add a scale bar
  var mapScaleBar = new OM.control.ScaleBar();
  mapview.addMapDecoration(mapScaleBar);
  // Display the map.
  // Note: Change from V1. In V2 initialization and display is done just once
  mapview.init();
}
```

- Additional HTML elements and JavaScript code that implement other application-specific user interface and control logic. For example, the HTML `<input>` element and JavaScript function `setLayerVisible` together implement a layer visibility control. The `setLayerVisible` function is coded as follows:

```
function setLayerVisible(checkBox)
{
  // Show the customers vector layer if the check box is checked and
  // hide it otherwise.
  if(checkBox.checked)
    customersLayer.setVisible(true) ;
  else
    customersLayer.setVisible(false);
}
```

  The function is specified in the `onclick` attribute of the `<input>` element defining the checkbox. In the following example, the function is executed whenever the user clicks on the Show Customers check box:

```
<INPUT TYPE="checkbox" onclick="setLayerVisible(this)" checked/>Show Customers
```

8.6 Using Google Maps and Bing Maps

Applications can display Google Maps tiles or Microsoft Bing Maps tiles as a built-in map tile layer, by creating and adding to the map window an instance of `MVGoogleTileLayer` or `MVBingTileLayer`, respectively. Internally, the Oracle Maps client uses the official Google Maps or Bing Maps API to display the map that is directly served by the Google Maps or Microsoft Bing Maps server.

- To use the Google Maps tiles, your usage of the tiles must meet the terms of service specified by Google (see `https://developers.google.com/readme/terms`).
- To use the Bing Maps tiles, you must get a Bing Maps account. Your usage must meet the licensing requirement specified by Microsoft (see `http://www.microsoft.com/maps/`).

If you need to overlay your own spatial data on top of the Google Maps or Microsoft Bing Maps tile layer, see also Section 8.7, "Transforming Data to a Spherical Mercator Coordinate System".)

The following sections describe the two options for using built-in map tile layers:

- Section 8.6.1, "Defining Google Maps and Bing Maps Tile Layers on the Client Side"
- Section 8.6.2, "Defining the Built-In Map Tile Layers on the Server Side"

8.6.1 Defining Google Maps and Bing Maps Tile Layers on the Client Side

To define a built-in map tile layer on the client side, you need to create a `MVGoogleTileLayer` or `MVBingTileLayer` object, and add it to the MVMapView object. (As of Oracle Fusion Middleware Release 11.1.1.6, `MVGoogleTileLayer` uses the Google Maps Version 3 API by default, and `MVBingTileLayer` uses the Bing Maps Version 7 API by default.)

For example, to use Google tiles, add the Google tile layer to your map:

```
mapview = new MVMapView(document.getElementById("map"), baseURL);
tileLayer = new MVGoogleTileLayer() ;
mapview.addMapTileLayer(tileLayer);
```

In your application, you can invoke the method `MVGoogleTileLayer.setMapType` or `MVBingTileLayer.setMapType` to set the map type to be one of the types supported by the map providers, such as road, satellite, or hybrid.

For usage examples and more information, see the JavaScript API documentation for `MVGoogleTileLayer` and `MVBingTileLayer`, and the tutorial demos *Built-in Google Maps Tile Layer* and *Built-in Bing Maps Tile Layer*.

8.6.2 Defining the Built-In Map Tile Layers on the Server Side

You can define a built=-in map tile layer on the server side and use it as a regular MapViewer tile layer on the client side. To define a built-in map tile layer on the server side, follow these steps:

1. Log into the MapViewer Administration Page (explained in Section 1.5.1).
2. Select the Manage Map Tile Layers tab and click **Create**.
3. When you are asked to select the type of map source, choose **Google Maps** or **Bing Maps** and click **Continue**.
4. Select the data source where the tile layer is to be defined.
5. Set the license key that you have obtained from the map provider.
6. Click **Submit** to create the tile layer.

After you have created the built-in map tile layer on the server side, you can use it like any other tile layer served by MapViewer. You do not need to add any `<script>` tag to load the external JavaScript library.

The following example shows a Bing Maps tile layer defined on the server side:

```
mapview = new MVMapView(document.getElementById("map"), baseURL);
// The Bing tile layer is defined in data source "mvdemo".
tileLayer = new MVMapTileLayer("mvdemo.BING_MAP") ;
mapview.addMapTileLayer(tileLayer);
```

In your application, you can invoke the method `MVMapTileLayer.setMapType` to set the map type to be one of the types supported by the map providers, such as road, satellite, or hybrid.

8.7 Transforming Data to a Spherical Mercator Coordinate System

Popular online map services such as Google Maps and Microsoft Bing Maps use a spherical Mercator projection for their maps. If you are using an Oracle Database release earlier than 11.1.0.7, and if you need to overlay your own spatial data on top of such a tile layer, such as a Google Maps or Microsoft Bing Maps tile layer, you must

set up the database to properly handle coordinate system transformation between the coordinate system of that tile layer and your own data coordinate system, if the two coordinate systems are not the same.

Note: To perform the actions in this section, your database must be Release 10.2.0.1 or later.

Google Maps uses a Spherical Mercator coordinate system (EPSG: 3785), which is also widely used among commercial API providers such as Yahoo! Maps and Microsoft Bing Maps. This coordinate system (SRID 3785) was not provided with Oracle Spatial and Graph before Release 11.1.0.7. In order to enable MapViewer and Oracle Spatial and Graph to transform your own data to this coordinate system, you must first add this coordinate system definition into your Oracle database if it is not already defined.

To check if this coordinate system is defined, you can enter the following statement:

```
SELECT srid FROM mdsys.cs_srs WHERE srid=3785;
```

If the preceding statement returns a row, you do not need to perform the actions in this section. If the preceding statement does not return a row, you must perform the actions in this section in order to be able to overlay your own spatial data on top of the tile layer.

Follow these steps:

1. Connect to the database as a privileged user, such as one with the DBA role.
2. Run the `csdefinition.sql` script, as follows. (Replace *`$OC4J_HOME`* with the root directory of the OC4J instance where your MapViewer is deployed, and enter the command on a single line.)
   - Linux: *`$OC4J_HOME`*`/j2ee/home/applications/mapviewer/web/WEB-INF/admin/csdefinition.sql`
   - Windows: *`$OC4J_HOME`*`\j2ee\home\applications\mapviewer\web\WEB-INF\admin\csdefinition.sql`
3. If necessary, create a transformation rule to cause Oracle Spatial and Graph to skip datum conversion when transforming data from a specified coordinate system to the Spherical Mercator system. To find out if you need to create such a transformation rule, see Section 8.7.1.
4. Either pre-transform your spatial data for better performance, or let MapViewer transform the data at runtime ("on the fly"). Note that if your database release is earlier than 10.2.0.4, pre-transforming is the only option.
   - To pre-transform all your data into the Spherical Mercator coordinate system, use the SDO_CS.TRANSFORM_LAYER procedure on all the data, and use the transformed data for mapping. (See the SDO_CS.TRANSFORM_LAYER reference section in *Oracle Spatial and Graph Developer's Guide*.)
   - To let MapViewer transform the data at runtime, do not transform the data before using it for mapping.

8.7.1 Creating a Transformation Rule to Skip Datum Conversion

Spatial data is often in a coordinate system based on an ellipsoid datum, such as WGS84 or BNG. In such cases, Oracle Spatial and Graph by default applies datum

conversion when transforming the data into the Spherical Mercator system. This will introduce a small amount of mismatch or error between your data and the Google Maps other map service tiles. If you want to address this issue, you can create transformation rules that tell Oracle Spatial and Graph to skip datum conversion when transforming data from a specified coordinate system to the Spherical Mercator system.

Example 8–10 shows SQL statements that are included in the `csdefinition.sql` script and that create such transformations rules. However, if the coordinate system of your spatial data is not covered by the rules shown in Example 8–10, you can create your own rule if the coordinate system of your data is not covered by these rules. (For more information about creating coordinate system transformation rules, see *Oracle Spatial and Graph Developer's Guide*.)

Example 8–10 Transformation Rules Defined in the csdefinition.sql Script

```
-- Create the tfm_plans, that is, the transformation rules.
-- Note: This will result in an incorrect conversion since it ignores a datum
-- datum between the ellipsoid and the sphere. However, the data will match
-- up better on Google Maps.

-- For wgs84 (8307)
call sdo_cs.create_pref_concatenated_op( 83073785, 'CONCATENATED OPERATION 8307
3785', TFM_PLAN(SDO_TFM_CHAIN(8307, 1000000000, 4055, 19847, 3785)), NULL);

-- For 4326, EPSG equivalent of 8307
call sdo_cs.create_pref_concatenated_op( 43263785, 'CONCATENATED_OPERATION_4326_
3785', TFM_PLAN(SDO_TFM_CHAIN(4326, 1000000000, 4055, 19847, 3785)), NULL);

-- For OS BNG, Oracle SRID 81989
call sdo_cs.create_pref_concatenated_op( 819893785, 'CONCATENATED OPERATION 81989
3785', TFM_PLAN(SDO_TFM_CHAIN(81989, -19916, 2000021, 1000000000, 4055, 19847,
3785)), NULL);

-- For 27700, EPSG equivalent of 81989
call sdo_cs.create_pref_concatenated_op( 277003785, 'CONCATENATED_OPERATION_27700_
3785', TFM_PLAN(SDO_TFM_CHAIN(27700, -19916, 4277, 1000000000, 4055, 19847,
3785)), NULL);
commit;
```

8.8 Dynamically Displaying an External Tile Layer

The Oracle Maps JavaScript API supports dynamically defining an external tile layer without needing any server-side storage of either the definition or the tile images. Basically, you can use the class `MVCustomTileLayer` to reference and display tile layers served directly from any external map tile server on the web, such as the ESRI ArcGIS tile server, the OpenStreet map tile server, or other vendor-specific map tile servers.

To do so, you need to do the following when creating a new `MVCustomTileLayer` instance:.

- Know the configuration of the map tile layer, specifically its coordinate system, boundary, and zoom level.
- Supply a function that can translate a tile request from Oracle Maps into a tile URL from the external tile server.

The configuration of a tile layer takes the form of a JSON object, and is generally in the format illustrated by the following example:

```
var mapConfig = {mapTileLayer:"custom_map", format:"PNG",
coordSys:{srid:8307,type:"GEODETIC",distConvFactor:0.0,
minX:-180.0,minY:-90.0,maxX:180.0,maxY:90.0},
zoomLevels:
[{zoomLevel:0,name:"level0",tileWidth:15.286028158107968,tileHeight:15.28602815810
7968,tileImageWidth:256,tileImageHeight:256},
{zoomLevel:1,name:"level1",tileWidth:4.961746909541633,tileHeight:4.96174690954163
3,tileImageWidth:256,tileImageHeight:256},
{zoomLevel:2,name:"level2",tileWidth:1.6105512127664132,tileHeight:1.6105512127664
132,tileImageWidth:256,tileImageHeight:256},
{zoomLevel:3,name:"level3",tileWidth:0.5227742142726501,tileHeight:0.5227742142726
501,tileImageWidth:256,tileImageHeight:256},
{zoomLevel:4,name:"level4",tileWidth:0.16968897570090388,tileHeight:0.169688975700
90388,tileImageWidth:256,tileImageHeight:256},
{zoomLevel:5,name:"level5",tileWidth:0.05507983954154727,tileHeight:0.055079839541
54727,tileImageWidth:256,tileImageHeight:256},
{zoomLevel:6,name:"level6",tileWidth:0.017878538533723076,tileHeight:0.01787853853
3723076,tileImageWidth:256,tileImageHeight:256},
{zoomLevel:7,name:"level7",tileWidth:0.005803187729944108,tileHeight:0.00580318772
9944108,tileImageWidth:256,tileImageHeight:256},
{zoomLevel:8,name:"level8",tileWidth:0.0018832386690789012,tileHeight:0.0018832386
690789012,tileImageWidth:256,tileImageHeight:26},
{zoomLevel:9,name:"level9",tileWidth:6.114411263243185E-4,tileHeight:6.11441126324
3185E-4,tileImageWidth:256,tileImageHeight:256} ]
};
```

For the a function that can translate a tile request from Oracle Maps into a tile URL from the external tile server, specify a function such as the following example:

```
function getMapTileURL(minx, miny, width, height, level)
{
 var x = (minx-mapConfig.coordSys.minX)/mapConfig.zoomLevels[level].tileWidth ;
var y = (miny-mapConfig.coordSys.minY)/mapConfig.zoomLevels[level].tileHeight ;
return "http://localhost:8888/mapviewer/mcserver?request=gettile&format=" +
mapConfig.format + "&zoomlevel="+level+"&mapcache=mvdemo.demo_map&mx=" +
Math.round(x) + "&my=" + Math.round(y) ;
}
```

In the preceding example, the function `getMapTileURL()` is implemented by the application to supply a valid URL from the external tile server that fetches a map tile image whose top-left corner will be positioned at the map location (`minx,miny`) by the Oracle Maps client. Each map tile image is expected to have the specified size (`width,height`), and it should be for the specified zoom level (`level`). This specific example is actually returning a `gettile` URL from the local MapViewer tile server; however the approach also applies to any non-MapViewer tile servers.

The new custom tile layer is added to the client `mapViewer` just like a built-in map tile layer.

9 Oracle Map Builder Tool

This chapter briefly describes the MapViewer Map Builder tool, also referred to as Oracle Map Builder. It does not provide detailed information about the tool's interface; for that you should use see online help available when you use Oracle Map Builder.

Oracle Map Builder is a standalone application that lets you create and manage the mapping metadata (about styles, themes, and base maps) that is stored in the database. For example, use this tool to create a style or to modify the definition of a style. Besides handling the metadata, the tool provides interfaces to preview the metadata (for example, to see how a line style will appear on a map) and also spatial information.

Whenever possible, you should use Oracle Map Builder instead of directly modifying MapViewer metadata views to create, modify, and delete information about styles, themes, and maps. For any modifications made outside Oracle Map Builder, such as with SQL statements, you should refresh the database connection in Oracle Map Builder to get the current items.

To use Oracle Map Builder effectively, you must understand the MapViewer concepts explained in Chapter 2 and the information about map requests in Chapter 3.

This chapter contains the following major sections:

- Section 9.1, "Running Oracle Map Builder"
- Section 9.2, "Oracle Map Builder User Interface"

9.1 Running Oracle Map Builder

Oracle Map Builder is shipped as a JAR file (`mapbuilder.jar`). You can run it as a standalone Java application in a Java Development Kit (J2SE SDK) 1.5 or later environment, as follows:

```
% java -jar mapbuilder.jar [Options]
```

Options:

`-cache <cache_size>` specifies the size of the in-memory geometry cache. Example: `-cache 64M`

`-config <config-file>` specifies the location of the file containing Map Builder configuration and preference information. If you do not specify this option, Map Builder looks for a file named oasmapbuilder.xml in your home Java directory. For more information about the configuration and preference file, see Section 1.5.2.

`-connect` causes Map Builder at startup to register connections for all data sources specified in the `oasmapbuilder.xml` preferences file or the file specified with the

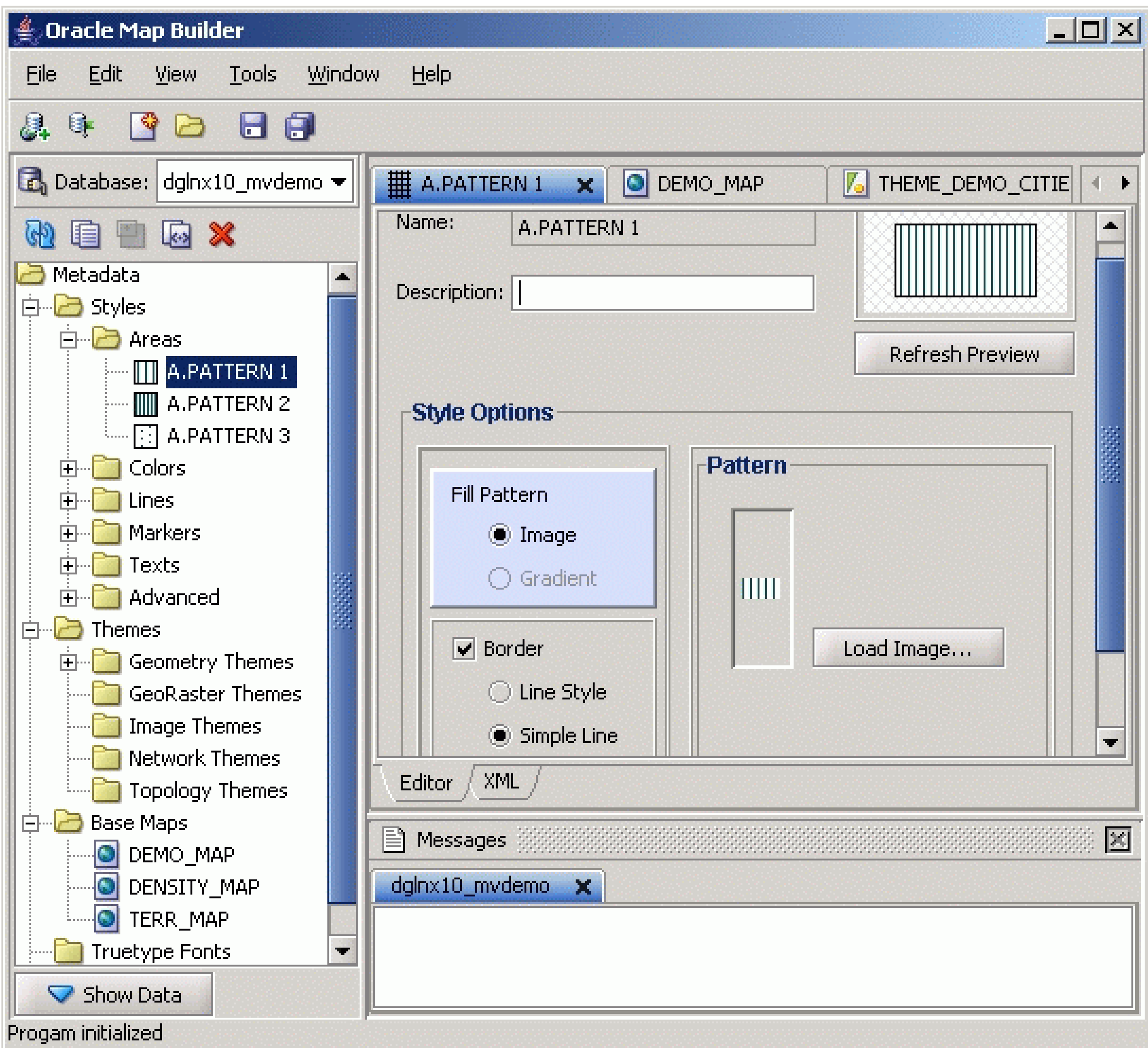
`-config` option, and it automatically connects to the first available data source. This option increases the application startup time. If this option is not defined, startup is faster, but you must then use the File menu or an icon to connect to any data sources that you want to use (see Section 9.2, "Oracle Map Builder User Interface").

`-help` displays information about the available options.

9.2 Oracle Map Builder User Interface

Oracle Map Builder generally uses the left side for navigation to find and select objects, and the right side to display information about selected objects. Figure 9–1 shows the main window of Oracle Map Builder, with the metadata navigation tree on the left and a detail pane for a selected area style on the right.

Figure 9–1 Oracle Map Builder Main Window



The menus at the top contain standard entries, plus entries for features specific to Oracle Map Builder.

You can use shortcut keys to access menus and menu items: for example Alt+F for the File menu and Alt+E for the Edit menu; or Alt+H, then Alt+A for Help, then About.

Icons under the menus perform the following actions:

- **Add new connection** creates a new database connection for Oracle Map Builder to use.
- **Load/Add/Remove connection** loads or adds database connection for Oracle Map Builder to use, or removes a database connection from the available connections that Oracle Map Builder can use.
- **Create new metadata** creates a new base map, theme, or style.
- **Open** opens a base map, theme, or style.
- **Save** saves any changes to the currently selected object.
- **Save All** saves any changes to all open objects.

The left side of the Oracle Map Builder window has the Metadata navigator, including a database connection selector, icons for performing actions, and a hierarchical tree display for the MapViewer metadata objects (categorized by object type) accessible to the currently selected database connection. To select an object, expand the appropriate tree node or nodes, then double-click the object.

The right side of the Oracle Map Builder window has tabs and panes for detail views of objects that you select or open

To switch among objects, click the desired tabs; to close a tab, click the X in the tab. If you make changes to an object and click the X, you are asked if you want to save the changes.

The Messages area is used for feedback information as appropriate (for example, results of an action, or error or warning messages).

Detailed help is available within the Oracle Map Builder interface. See the online help for more information about Oracle Map Builder, including information about specific panes and dialog boxes.

10
Oracle MapViewer Editor

This chapter describes the Oracle MapViewer Editor, also referred to as the Map Editor. It contains essentially the same information as is available in the online help available when you use Oracle Map Editor.

To see the online help, click the **Help** (question mark) icon in the Map Canvas area toolbar:

Major Topics

MapViewer Editor Concepts and Usage

MapViewer Editor Reference

10.1 MapViewer Editor Concepts and Usage

The Related Topics cover important MapViewer Editor concepts and include a suggested typical workflow to help you get started editing spatial data.

Related Topics

About the MapViewer Editor

MapViewer Editor Main Window

Editing Sessions

Getting Started: A Typical Workflow

Known Issues

See Also

MapViewer Editor Reference

10.1.1 About the MapViewer Editor

The MapViewer Editor is a web-based spatial data editing tool. It is distributed as part of the MapViewer EAR file, and can be launched as a Java applet from any web browser once the MapViewer server is up and running. The MapViewer Editor supports multiuser, multisession online data editing capabilities across an enterprise.

The MapViewer Editor assume that the spatial data to be edited consists of two-dimensional (2D) geometries of type SDO_GEOMETRY. Oriented points are supported. However, editing of 3D and LRS (Linear Referencing System) geometries is

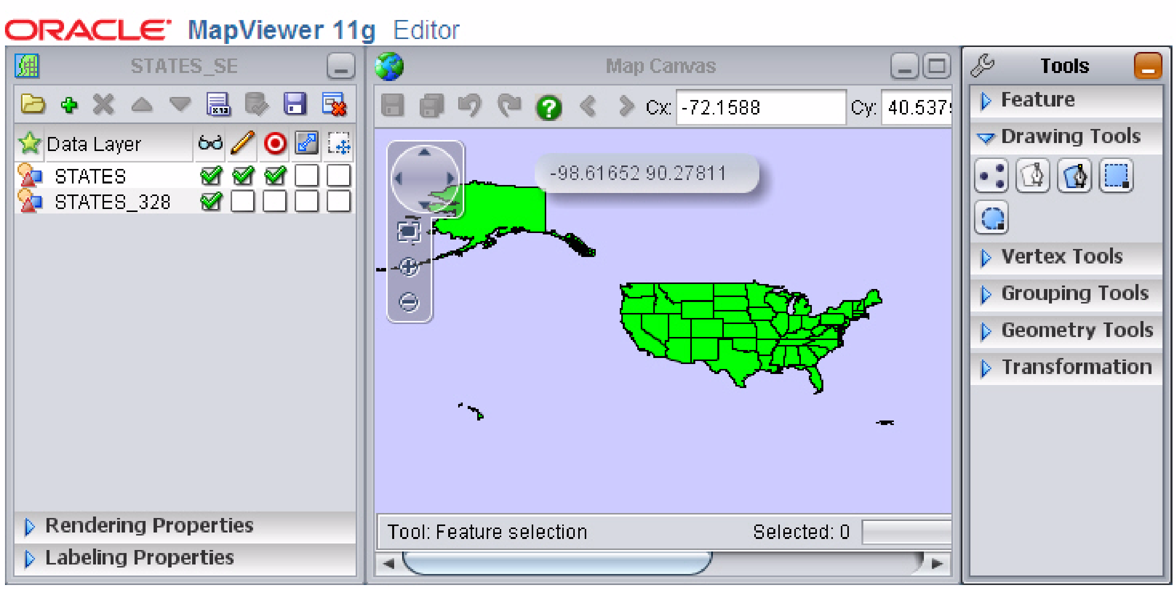
not fully supported, and editing operations on these objects may produce unknown results (that is, such data might be able to be visualized, but editing operations might not work).

To use the MapViewer Editor effectively, you must understand the concepts explained in the "Spatial Data Types and Metadata" chapter in *Oracle Spatial and Graph Developer's Guide*.

10.1.2 MapViewer Editor Main Window

The MapViewer Editor has the main window shown in Figure 10–1.

Figure 10–1 MapViewer Editor Main Window



The MapViewer Editor main window has three major areas:

- The Edit Session Area on the left side (labeled in the figure as `STATES_SE`, the current editing session name) enables you to specify operations and settings for the session and data layers, and to override the default rendering and labeling properties. The Rendering Properties and Labeling Properties panels can be expanded and collapsed.
- The Map Canvas Area in the middle is where the map is displayed. The content of this area changes to reflect properties or preferences that you set, tools that you select, and data editing operations that you perform.
- The Tools Area on the right side contains a set of collapsible panels, each containing a set of tools grouped according to their functions. The panels are for Feature Tools, Drawing Tools, Vertex Tools, Grouping Tools, Geometry Tools, and Transformation Tools.

10.1.3 Editing Sessions

With the MapViewer Editor, an **editing session** (also referred to as just a *session*) defines a personal workspace where you, the MapViewer Editor user, can edit spatial data. Each session can contain at least the following information:

- Name of the session (specified when you create a new session)

- List of data layers, including background layers and editable tables with spatial geometry data
- Data table display styles
- Name of the user that created the session (that is, the session owner)

The session definition is stored persistently in the database where edits for that session will occur. A user can have multiple editing sessions at any given time.

Within each session, you typically add one or more background layers (each of which can be any predefined theme, base map, or tile layer), plus one or more editing layers. The editing layers usually come from the same MapViewer data source (the database schema) or from published features of Web Feature Servers (WFS).

Editing sessions are stored in the USER_SDO_EDIT_SESSIONS view. Before you use the MapViewer Editor, a DBA must run the `sdedefinition.sql` script, as explained in Section 10.1.4.1, "Installing the USER_SDO_EDIT_SESSIONS View".

10.1.3.1 Editing Mode

To edit spatial data, you just perform certain basic steps as explained in Section 10.1.4, "Getting Started: A Typical Workflow", one of which is to enable **editing mode** for the session editable layer.

The editable data tables in a session may be rendered differently depending on whether the session editable layer is in editing mode. When you create or reopen a session, the layer is generally not in editing mode. When not in editing mode, the data layers are rendered by the MapViewer server, and the editor application simply displays an image generated by the server.

When you switch the layer session into editing mode, the editable data tables are rendered differently from the other layers (which are still rendered at the server side). These tables are rendered as live vector features ready to be edited. A good practice is to not set to editable mode when too many features are visible, because the application will try to load them on the client side. For editing it is recommended that you work on small areas to avoid bringing too much information to the client.

Any edits made within a session can be saved, and the saved changes, while persisted in the database, are only visible within the same session unless the session is being edited in `LIVE` workspace. If another user creates a new session that operates on the same data table but on a workspace that is not `LIVE`, that session will not see the changes made in the first session, and vice versa. Changes are only visible to others when a session with a workspace other than `LIVE` is merged.

When a session is **merged**, all the edits made within that session are published to the live data tables, and are visible by any user that queries the table. The session is effectively completed, but it is still stored in the database (for future editing), and it can be purged later.

If a session has saved edits but has not been merged, the editor can reopen it anytime from anywhere, and continue to make edits. It is typical for a session to be kept open for multiple days or even longer.

10.1.3.2 Security and Multiuser Editing Considerations

The MapViewer Editor is designed to be used by multiple users across an organization. The editor itself relies on the MapViewer server for user management and security. When a user launches the editor from the MapViewer home page, that user must be logged in to the server as a Java EE or middleware user with the **`map_edit_role`** role. New users can be added to the Java EE container by your

administrator, and the required map_edit_role role must be granted. (The map_admin_role role also enables a user to log in and use the MapViewer Editor.)

Each user must create or reopen an editing session, where that user can make changes to the data tables. In a multiuser environment, conflicts between edits across sessions may be unavoidable unless all users coordinate their work and follow clear rules.

10.1.4 Getting Started: A Typical Workflow

This topic describes a typical workflow example that involves making some edits to a spatial data table, using the MVDEMO sample data schema. The basic steps are:

1. Installing the USER_SDO_EDIT_SESSIONS View
2. Making a MapViewer Data Source Editable
3. Launching the MapViewer Editor and Logging In
4. Selecting a Data Source and Creating a New Session
5. Adding Data Layers to a Session
6. Changing Data Layer Properties
7. Navigating the Map and Enabling Editing Mode
8. Selecting a Feature for Editing
9. Saving and Merging Session Edits

These basic steps illustrate a typical work flow where you create a new session, load an existing geometry table, make a few edits then end the session and merge the changes to the live table if session is versioned.

(Alternatively, after creating a new session, you can also create a new table by clicking on the tree node to create a new geometry layer. You can then add a few background layers and start digitizing new features for your new table.)

10.1.4.1 Installing the USER_SDO_EDIT_SESSIONS View

This one-time task, which must be performed by a DBA, installs the USER_SDO_EDIT_SESSIONS system view that is required by the MapViewer Editor. To perform this installation, run the SQL script file sdedefinition.sql, which is in the MapViewer WEB-INF/admin directory.

The definitions of all editing sessions are stored in the USER_SDO_EDIT_SESSION view. Each session has the following attributes:

- name: name of the session.
- description: description of the session.
- editor: name of the owner of the session. This name is not the user schema name, but can be any name that identifies the person doing the editing.
- area: name of the area for the session (not currently used).
- workspace: name of then underlying Oracle workspace. The workspace name is automatically assigned when the session is created. If the session is not versioned, the name LIVE is assigned; if the session is versioned, the name is a combination of *<editor-name>_<session-name>_<data-source-name>*.
- definition: XML description of the session, including the general session attributes and the layer descriptions.

10.1.4.2 Making a MapViewer Data Source Editable

The MapViewer Editor will only load data sources that have been made editable. To make a data source editable, you must modify the <map_data_source> element of the MapViewer configuration file to specify `editable="true"`. For example:

```
<map_data_source name="mvdemo"
                 jdbc_host="yourhost.com"
                 jdbc_sid="lbsmain"
                 jdbc_port="37407"
                 jdbc_user="mvdemo"
                 jdbc_password="!mvdemo"
                 jdbc_mode="thin"
                 number_of_mappers="3"
                 allow_jdbc_theme_based_foi="true"
                 editable="true"
 />
```

The default value for the `editable` attribute is `false`, which means that the data source is not editable.

10.1.4.3 Launching the MapViewer Editor and Logging In

Before using the MapViewer Editor, ensure that MapViewer is deployed and the `mvdemo` data source is defined.

1. Go to the MapViewer home page, typically `http://`*`<host>:<port>`*`/mapviewer`.
2. Click the **Editor** link in the upper-right corner (between Admin and Help).
3. Log in as the administrative user for the application server (for example, `weblogic` for WebLogic Server).

A loading screen is displayed with a large Java icon. After the loading is complete, the MapViewer Editor Main Window is displayed.

10.1.4.4 Selecting a Data Source and Creating a New Session

Select a data source and create a new editing session.

1. In the Session and Data Layer Operations Toolbar on the left in the Edit Session Area, click the **Open an edit session** icon to display a dialog box with the editable data sources for this MapViewer server, shown in Figure 10–2.

Figure 10–2 Open an Edit Session

2. To select a data source on a displayed available server, click it. For example, in the data sources display tree, under `MVDEMO`, click the **Create Edit Session** node.
3. In the Create Edit Session dialog box, enter the appropriate information:

   Name: Name for the editing session. Example: `STATES_LIVE`

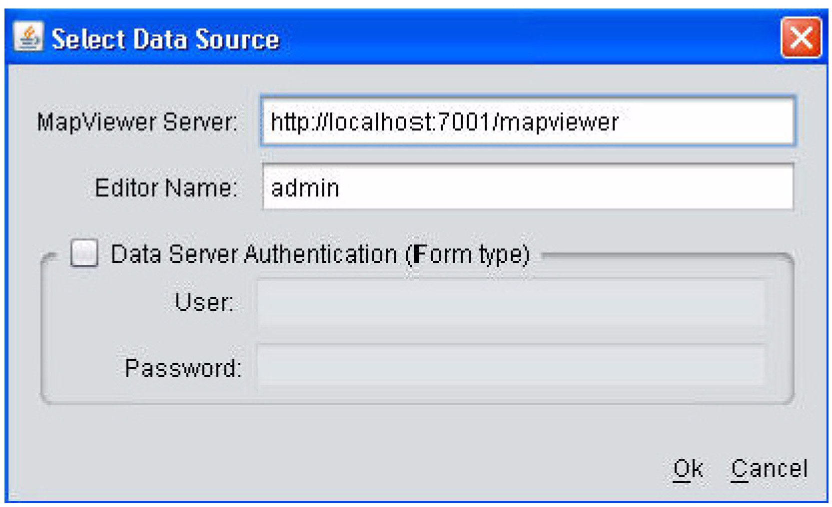
Description: Optional descriptive text about the session. Example: `Edit US states data`

Version Enabled: If this option is checked, a new workspace will be created for this session, and any changes can be merged later. If this option is not selected, the session edits will be applied directly to `LIVE` data without any version control. For versioned sessions (not `LIVE`), the workspace name will be on the form *<EDITOR_NAME>_<SESSION_NAME>_<DATASOURCE>*, so each versioned session will have its own workspace.

> **Note:** Versioned sessions involve tables that have been version-enabled using Oracle Workspace Manager. For more information, see *Oracle Database Workspace Manager Developer's Guide*.

If the data source is not on a displayed available server, you can click **Connect to Server** to display the dialog box shown in Figure 10–3.

Figure 10–3 Select Data Source



In the Select Data Source dialog box, enter the appropriate information:

- **MapViewer Server**: URL of the MapViewer server for the data source.
- **Editor Name**: Name of the session owner (to be used to filter the sessions to be displayed).
- **Data Server Authentication (Form Type)**: If the MapViewer server is a secured server.(with form authentication), check this option and enter the User and Password information.

After you click OK, all edit session names on the specified this server associated with the specified editor name will be loaded into the tree.

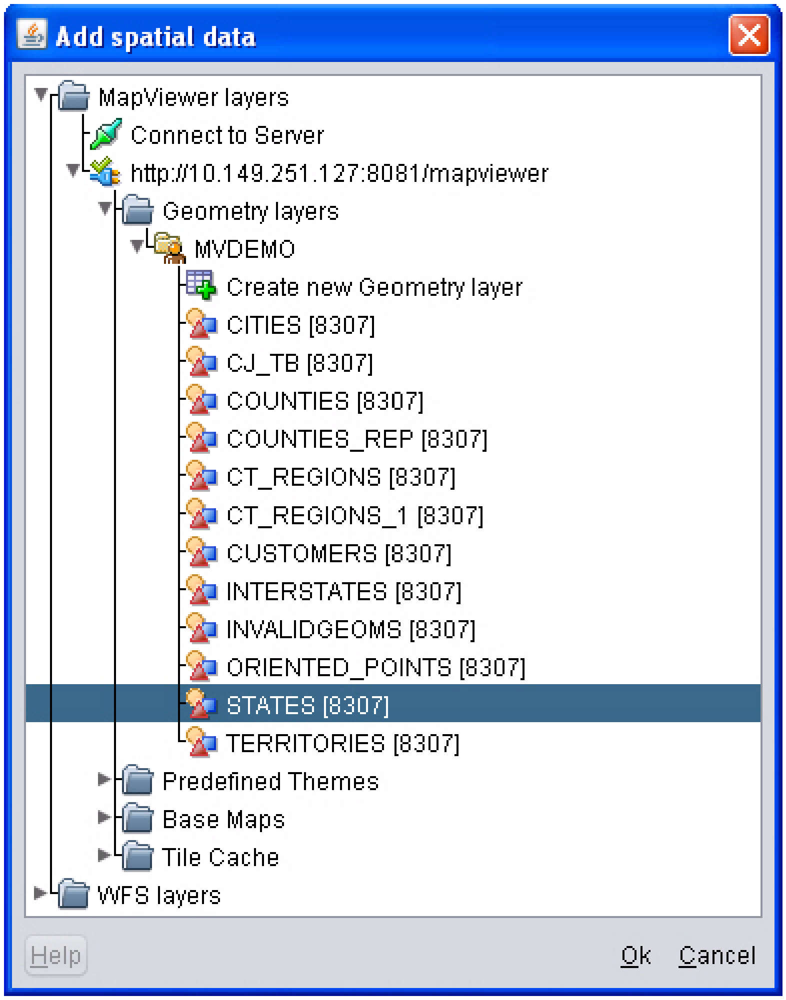
10.1.4.5 Adding Data Layers to a Session

Add one or more data layers to the editing session.

1. In the Session and Data Layer Operations Toolbar on the left in the Edit Session Area, click the **Add spatial data** icon to display the Add Spatial Data dialog box.

2. In this dialog box, shown in Figure 10–4, click the desired server, then **Geometry layers**, then the desired data source (**MVDEMO** in this example), and then the desired layer (**STATES** in this example).

Figure 10–4 Add Spatial Data



[8307] next to each layer indicates that the layer is based on SRID 8307, that is, the WGS 84 longitude-latitude coordinate system. All data used in an editing session must be based on the same SRID.

After you click **OK**, the MapViewer Editor sets an area for the map based on the metadata information for the layer table, retrieves the area covered by the data, and sets a subset area as the current map area. (If the map area does not contain data, enter other values on the map canvas, such as center X, center Y, and height or scale, and press the **Refresh** button to redraw the map.)

10.1.4.6 Changing Data Layer Properties

For editable layers based on Oracle tables, you must ensure that Key Column is correctly set for this data, and that it is not `ROWID`.

When the editable layer is added in the Edit Session Area, the MapViewer Editor automatically checks if the layer table has a primary key, and it sets the Key Column property with this value. If Key Column is set to `ROWID`, the MapViewer Editor cannot edit the data. In this case, you must set the Key Column property to another column in

the table that contains unique values. (It does not have to be the primary key of the table, but it must contain unique values.)

To change the Key Column value:, in the Session and Data Layer Operations Toolbar on the left in the Edit Session Area, click **Edit session properties** to display a dialog box for editing Session and Layer Preferences.

1. In the Session and Data Layer Operations Toolbar on the left in the Edit Session Area, click **Edit session properties** to display a dialog box for editing Session and Layer Preferences
2. In the dialog box, for **Key Column** select a text column that contains unique values. For example, for `STATES` the key column could be `STATE_ABRV` (the two-character state abbreviation).

10.1.4.7 Navigating the Map and Enabling Editing Mode

You can use the Navigation Panel in the upper-left corner of the map canvas area to pan the map and to perform zoom (marquee, in, out) operations.

You can use the Data Layers panel in the Edit Session Area on the left to control layer visibility, make layers editable, specify or change the target layer on which editing operations are applied, and make other selections that affect the editing session behavior.

Some suggested steps:

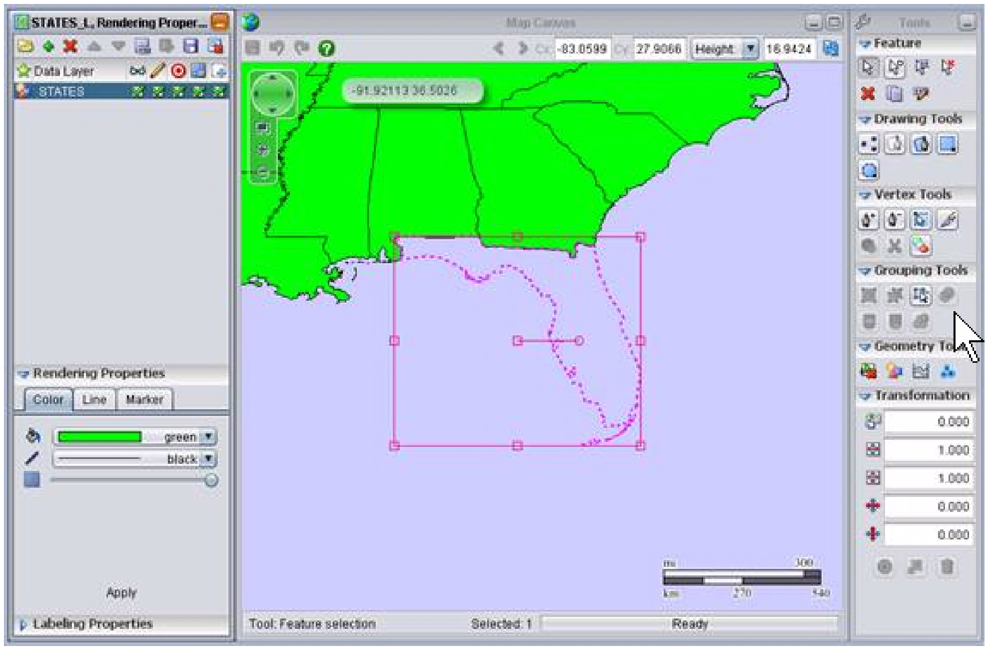
1. Pan to the specific map area that you want to edit and optionally zoom, using the Navigation Panel in the upper-left corner of the map canvas area.
2. For the desired data layer, check the **Layer is editable** (pencil) icon in the Data Layers panel to enter in editing mode for the layer.

   If the session is versioned and a data layer is made editable, the MapViewer Editor checks that the layer base table is versioned using Oracle Workspace Manager; and if it is not versioned, you will need to specify the table's primary key for versioning (if no primary key is defined).
3. For the desired data layer, check the **Current editing target** (target) icon in the Data Layers panel to specify that any editing will be applied to this layer.
4. Perform some editing operations using the tools in the Tools Area. (See also Selecting a Feature for Editing.)

10.1.4.8 Selecting a Feature for Editing

When the session is in editing or selection mode, as you mouse over the map area, the features are highlighted. When you click on a feature, it becomes selected (with an animated border and transparent interior area), for example, as the U.S. state of Florida is in Figure 10–5.

Figure 10–5 Feature Selected



The rendering of the data layer is determined by the choices made in the Rendering Properties panel. The labeling is determined by the choices made in the Labeling Properties panel. Dynamic styles are created based on the choices made in those properties panels.

The MBR (minimum bounding rectangle) of the selected feature (Florida in the preceding figure) has a manipulator around it, with nine squares that you can use to drag and resize this feature. You can also use the small red circle to rotate the feature.

The Tools Area panels in the right side of the window have tools for selection, drawing, vertex editing, grouping, and other operations.

10.1.4.9 Saving and Merging Session Edits

After you have made desired edits, save them by clicking one of these buttons on the Map Canvas Area toolbar:

- **Save edited data in the target layer** applies unsaved changes that have been made in the current target layer
- **Save all edited data** applies all unsaved changes that have been made in the current session.

It is recommended that you save edits frequently while in editing mode. When a save is requested, the edited data is sent back to server to be committed.

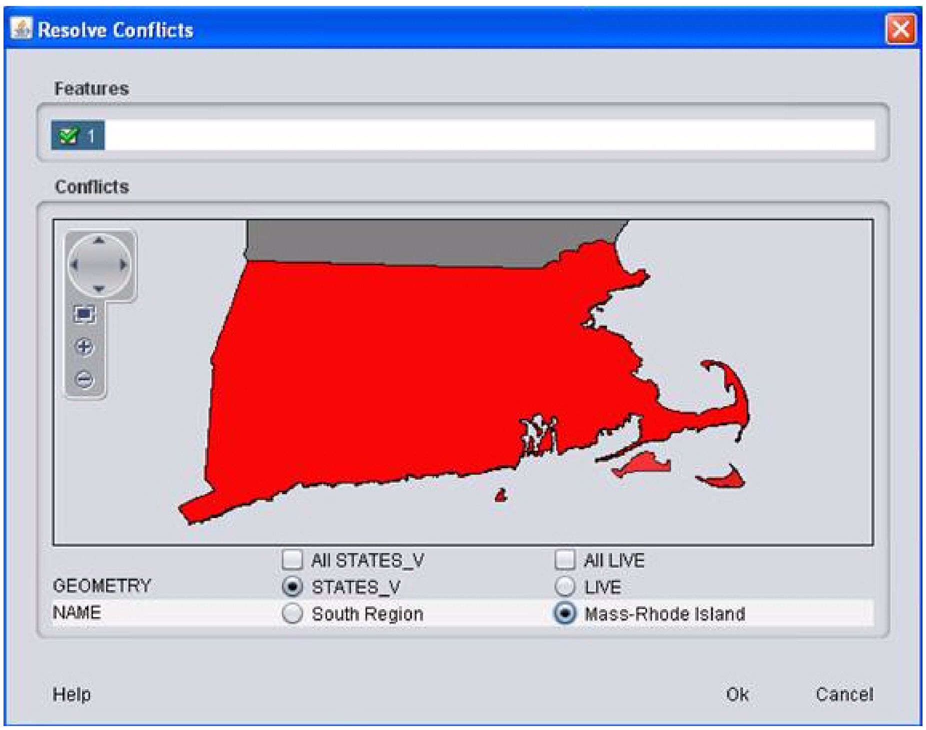
For a versioned session, remember that your edits so far are visible only in your own versioned session; they are not visible to any other users (or sessions).

Note: Versioned sessions involve tables that have been version-enabled using Oracle Workspace Manager. For more information, see *Oracle Database Workspace Manager Developer's Guide*.

If you are completely done with a versioned session, and ready to make the changes visible to the world, you will need to merge the session by clicking the **Merge session visible layers** icon in the Session and Data Layer Operations Toolbar near the top in the Edit Session Area on the left. (Only versioned layers are processed; therefore, if you have many layers in the session to be merged, you can work individually on layers by making them visible, while leaving edits on other (non-visible) layers for later.)

For a versioned editing session, once changes are saved, the editable layers base tables can be merged with LIVE data. For a merge, the editable layer the edit session workspace data is compared with current LIVE data; and if a conflict occurs, the Resolve Conflicts dialog box is displayed. For example, Figure 10–6 shows that the large feature in red has a conflict for the GEOMETRY and NAME columns.

Figure 10–6 Resolve Conflicts



Define how to resolve the conflicts by selecting the version attribute to be used as LIVE data. Note that a feature can have conflicts on both spatial and nonspatial attributes, and you can select values from the different data to resolve the conflicts. In the preceding figure, conflicts are resolved using the GEOMETRY value from the STATES_V editing session and the NAME value from the LIVE data.

When the merge process is executed and finished, the versioned edit session features will have the values selected to resolve the conflicts. The versioned editing session will still exist, but it can be purged later after final merge.

If the layer has many conflicts, you do not need to resolve all of them at once. Every time you work in the Resolve Conflicts dialog box and click OK, the affected conflicts to be resolved are processed in the server; but if the data still has one or more other conflicts, then the merge operation is not executed until all conflicts have been resolved for the layer.

Click **Yes** to confirm that you want the changes to be merged to the live data table that is visible to all users on the map data. If there are conflicts between versioned data and

`LIVE` data, a merge tool dialog is displayed for resolving the conflicts and for merging the data.

10.1.5 Known Issues

See the "readme" file for any known problems and considerations with the current release.

10.2 MapViewer Editor Reference

The reference topics provide information about MapViewer Editor preferences (properties) and the areas of the main window.

Related Topics

Session and Layer Preferences

Edit Session Area

Map Canvas Area

Tools Area

See Also

MapViewer Editor Concepts and Usage

10.2.1 Session and Layer Preferences

You can edit certain preferences for session and layer properties, to specify the default values for the associated properties. The properties for which you can set preferences are in the following categories:

```
<session-name> Properties
Control Layer Properties
  Feature Selection
  Drawing Tools
  Vertex Tools
  Background Layer
  Scale Bar Layer
  Manipulator Layer
Data Layer Properties
  <data-layer-name> (one for each data layer)
```

If you make any changes in any pane and if you want to save these changes, you ***must*** click **Apply** before you switch to another pane or before you click **Close** to close the Edit Preferences dialog box.

10.2.1.1 <session-name> Properties

Includes options for the editing session.

Logging Level: Level of information for logging: `info` for basic (limited) information or `finest` for detailed information. (`finest` provides more information, but takes longer and requires more disk storage.)

HTTP Timeout: Maximum time in seconds to wait for a response from an HTTP request (for example, when accessing the capabilities of a WFS server).

Use HTTP Proxy: If checked, specifies the host and port of the HTTP proxy server to be used.

10.2.1.2 Control Layer Properties

Includes options for the visual aspects of the editing session, such as colors and sizes for various objects.

For color-related options, click the small box that shows the current default color to display the Choose Color dialog box, in which you can specify a new color using one of several ways: Swatches, HSV, HSL, RGB, or CMYK. For each specification that you make, the Preview area of the dialog box shows how it will appear. To save a specification, click **Apply**.

10.2.1.2.1 Feature Selection **Mouse over feature color**: Color to be displayed for the feature under the mouse location.

Target feature color: Color of selected features of a target layer.

Non-target feature color: Color of selected features of a layer that is not a target layer

Feature element color: Color to highlight geometry elements of a feature. Used for validating and viewing feature geometries.

10.2.1.2.2 Drawing Tools **Cursor center color**: Cross marker color inside cursor symbol when drawing.

Cursor border color: Boundary color of cursor symbol when drawing.

Cursor point size: Cursor symbol size in pixels.

Snap point center color: Cross marker color inside cursor symbol when mouse location snaps with a snap layer.

Snap point border color: Boundary color of cursor symbol when mouse location snaps with a snap layer.

Line segment color: Segment color between points when digitizing lines or polygons.

Line segment width: Segment width between points when digitizing lines or polygons.

Rubberband segment color: Line segment color when moving the mouse to digitize points of lines or polygons.

Rubberband segment width: Line segment width when moving the mouse to digitize points of lines or polygons.

10.2.1.2.3 Vertex Tools **Show vertices**: Highlights selected feature vertices when moving the mouse around a feature boundary.

Vertex box color: Color of the rectangles representing the vertices.

Vertex box size: Size in pixels of the rectangles representing the vertices.

10.2.1.2.4 Background Layer Background color for the displayed map area.

10.2.1.2.5 Scale Bar Layer **Show scale bar**: Displays the Map Scale Bar.

10.2.1.2.6 Manipulator Layer **Color**: Boundary color of the manipulator rectangle.

Snap Highlight Color: Color to highlight a snap point. When a feature is selected and the manipulator rectangle is visible, right-clicking provides an option to set a snap point on the selected feature. This snap point can be used when moving the feature to snap at another location.

Changed Feature Color: Color of the boundary of the feature when it is being moved.

Handle Size: Size in pixels of the manipulator rectangle corner markers.

Hide When Obstructed: Hides the manipulator rectangle when it is covered by another panel.

10.2.1.3 Data Layer Properties

For each data layer, the properties for which you can set preferences depend on the type of data layer:

- Geometry Layer Properties
- WFS Layer Properties

10.2.1.3.1 Geometry Layer Properties Contains the Data Set and Rendering tabs.

Data Set tab:

MapViewer Server: URL of the MapViewer server where the layer is defined.

Key Column: Layer key column for editable features. Does not need to be the primary key, but the data values must be unique.

Geometry Column: Name of the geometry column associated with this layer.

Query Condition: Optional condition for filtering the layer features (SQL WHERE clause without the WHERE keyword).

Live Scale: (Not currently used.)

Label Column: If checked, specify the column containing text data to be used to label features.

Shared Boundary: Applies editing operations on other geometry objects that share a boundary with the object being edited.

For example, assume that two property lots share a boundary. If this option is checked and if you edit one property lot to make it larger by modifying the shared boundary, then the other property lot's definition is modified and its size becomes correspondingly smaller. If this option is not checked, then the second lot's definition is not changed (and the two lots' definitions will overlap spatially).

Create Sequence: Creates a new sequence on the MapViewer server.

Rendering tab:

Render in MapViewer when not editable: Displays the layer in MapViewer even when it is not editable. If this option is not checked, the layer is displayed only when it is editable.

10.2.1.3.2 WFS Layer Properties **Key Column**: Layer key column for editable features. Does not need to be the primary key, but the data values must be unique. Always FID for WFS 1.0.0 layers,

Geometry Column: Name of the spatial attribute of the WFS feature type.

Label Column: If checked, specify the column containing text data to be used to label features.

Polygon outer ring orientation: Specifies the polygon outer ring orientation. For Oracle Spatial and Graph data, the outer ring orientation is counterclockwise, but for external data served by WFS servers this orientation may be different. This value will be considered when building new polygons. If the polygon outer boundary digitizing has a different orientation than this value, then the coordinates will be automatically reoriented.

Shared Boundary: Applies editing operations on other features that share a boundary with the feature being edited.

For example, assume that two property lots share a boundary. If this option is checked and if you edit one property lot to make it larger by modifying the shared boundary, then the other property lot's definition is modified and its size becomes correspondingly smaller. If this option is not checked, then the second lot's definition is not changed (and the two lots' definitions will overlap spatially).

Capabilities GET URL: URL on the WFS server for GetCapabilities requests.

Version: WFS server version. (Currently only 1.0.0 is supported.)

Authentication: Specifies whether to apply Basic-type Authentication (for a secured WFS server).

10.2.2 Edit Session Area

The Edit Session area on the left side of the MapViewer Editor window enables you to specify operations and settings for the session and data layers, and to override the default rendering and labeling properties. The Rendering Properties and Labeling Properties panels can be expanded and collapsed.

- Session and Data Layer Operations Toolbar
- Data Layers
- Rendering Properties
- Labeling Properties

10.2.2.1 Session and Data Layer Operations Toolbar

At the top of the Edit Session area is a toolbar with icons for the following operations:

- **Open an edit session**: Displays a dialog box in which you select the editing session to be opened.
- **Add spatial data layer**: Displays a dialog box where you can add a spatial data layer to your session. Geometry layers from spatial tables can be edited; predefined themes, base maps, and tile layers are generally used as background layers and cannot be edited.

  Expand the MapViewer layers or WFS layers hierarchy to find the desired data layer, then click OK.
- **Remove data layer**: Removes the currently selected data layer from the Data Layers area.
- **Move data layer up**: Moves the currently selected data layer up one level in the Data Layers area.
- **Move data layer down**: Moves the currently selected data layer down one level in the Data Layers area.
- **Edit session properties**: Displays a dialog box for editing Session and Layer Preferences.
- **Merge session visible layers** (active only for versioned editing sessions) Merges merge layer workspace with LIVE workspace. (See Saving and Merging Session Edits.)

- **Save edit session definition**: Saves the current session definition to the USER_SDO_EDIT_SESSIONS view. (See Editing Sessions and Installing the USER_SDO_EDIT_SESSIONS View.)
- **Delete current edit session**: Removes the current session definition in the USER_SDO_EDIT_SESSIONS view.

  For versioned sessions, this also removes the workspace associated with the session. However, as for versioned layer base tables, if the session layer base table has not been modified in any other workspace, a dialog box is displayed letting you choose whether to unversion the table; however, if the session layer base table has been modified in any other workspace, the table is kept versioned.

10.2.2.2 Data Layers

The Data Layers panel has a header row and a row for each spatial data layer that is available for editing (that has been added and not removed). The header row has columns with icons for each of the following operations, so that you can select and deselect options for individual layers.

- **Layer is visible** (glasses icon): Controls whether the layer is visible on the map canvas or not.
- **Layer is editable** (pencil icon): Controls whether the layer is editable (that is, is in editing mode for the current editing session).

  A layer must be editable before you can modify its spatial data. One or more layers can be editable in a session. When you make a layer editable, the current features for the layer are loaded into the map area (if they are not already loaded) for editing.
- **Current editing target** (target icon): Identifies the layer on which editing operations are to be applied.

  Only zero or one layers can be the editing target, and a layer must be the editing target before you can modify its spatial data.
- **Snapping to this layer** (snap icon): Controls whether the layer is a snap layer, namely, one that can be used to snap vertices when digitizing feature geometry points. In a snap layer, if the mouse pointer moves close enough to an existing vertex, the pointer automatically snaps over to that existing vertex.

  Multiple layers can be used as snap layers.
- **Selection occurs on this layer** (selection icon): Controls whether the layer is a selection layer.

  Selection tools are used in selection layers. Features from one selection layer can be used to generate features in a different target layer, using operations such as union, intersection, and difference. One or more layers can be selection layers. When you make a layer a selection layer, the current features for the layer are loaded into the map area (if they are not already loaded) for selection.

You can use the Session and Data Layer Operations Toolbar to add and remove data layers and to move layers up and down in the display.

10.2.2.3 Rendering Properties

Rendering properties affect the rendering of editable data. Select an editable layer row in the data layer panel, and the contents of the Rendering Properties panel will reflect the current style used by the layer. You can view and modify Color, Line, and Marker properties using the appropriate tabs.

- For **Color**, the attributes are fill color, stroke color, and color fill color transparency.
- For **Line**, the attributes are fill/stroke color, line width, and fill/stroke color transparency.
- For **Marker**, the attributes are marker type, fill color, stroke color, marker size, and fill color transparency.

10.2.2.4 Labeling Properties

Labeling properties affect the labeling of editable data. Select an editable layer row in the data layer panel, and the contents of the Labeling Properties panel will reflect the current style used by the layer.

For Labeling Properties, the attributes are text font, text color, text size, italics, and bold.

10.2.3 Map Canvas Area

The Map Canvas area in the middle of the window is where the map is displayed. The content of this area changes to reflect properties or preferences that you set, tools that you select, and data editing operations that you perform.

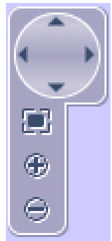
At the top of the Map Canvas area is a toolbar with icons for the following operations:

- **Save edited data in the target layer**
- **Save all edited data**
- **Undo** last change
- **Redo** last operation
- **Help** (question-mark icon)
- **Draw previous map** (not currently used)
- **Draw next map** (not currently used)
- **Cx** and **Cy** (map center X and Y coordinates in units of the spatial reference system, or SRID; for example, longitude and latitude for WGS 84 data)
- **Height** (in units of the SRID) or **Scale** (in meters for geodetic data) of the Y-axis for the area represented in the map canvas area. For example, at a particular map canvas size and zoom level, and with WGS 84 (longitude/latitude) data, the Height value might be 5.2657 (degrees of latitude) and the corresponding Scale value might be 5,750,861 (meters).
- **Refresh** the map preview

10.2.3.1 Navigation Panel

Below the Map Canvas area toolbar and on the left is a **navigation panel**, shown in Figure 10–7.

Figure 10–7 Navigation Panel

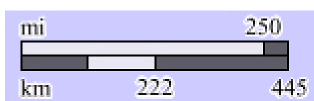


- Use the circle (spinning wheel) to pan the map (rotate and click an arrowhead to move the display in that direction).
- Use the rectangle to select (press, drag, release) an area to zoom to (sometimes called *marquee zoom*).
- Use the plus sign (**+**) to zoom in.
- Use the minus sign (**-**) to zoom out.

10.2.3.2 Map Scale Bar

If you have enabled the **Show scale bar** property for the Scale Bar Layer, then near the bottom of the map canvas area and on the right is a **map scale bar** that shows the distance in miles (mi) and kilometers (km) or meters (m) represented by various segments in the bar. Figure 10–8 shows an example.

Figure 10–8 Scale Bar



The distances in the scale bar reflect the current zoom level.

10.2.4 Tools Area

The Tools area on the right side of the MapViewer Editor window contains a set of collapsible panels, each containing a set of tools grouped according to their functions. Specific panels and tools are enabled (visible and usable) depending on the application context. For example, if there is any active editable layer, the selection tools are enabled.

The Tools groups are:

- Feature Tools (select, deselect, delete, duplicate, and edit feature's attributes)
- Drawing Tools (create new features by drawing their shapes on the map)
- Vertex Tools (manipulate individual vertices; split a polygon or perform void-related operations; break an existing line)
- Grouping Tools (group and ungroup features)
- Geometry Tools (validate, inspect, simplify, and add a buffer around a geometry)
- Transformation Tools (scale, rotate, and translate features)

10.2.4.1 Feature Tools

Icons are available for the following tools:

- **Select a feature**: Acts both as a generic pointer tool and a single selection tool (when clicking on a feature). When clicking on an area with no feature, it also deselects any currently selected features. When a feature is selected, its outlines become animated and (when the entire feature is within the map viewport) a set of **manipulators** (small squares) will be displayed around it. The Shift key can be used with the mouse click to deselect a feature if it is selected or to add it to selected list if it is not selected.

  The manipulators are a set of markers that can be used to move, scale, or rotate the feature. To move a feature, press and drag the center square mark. To scale feature, press and drag one of the outside square marks. To rotate feature, press and drag the circle mark. Multiple operations can be performed, and then click anywhere outside the feature to effectively apply the changes, or use the Transformation Tools tools to apply the changes.

- **Select multiple features**: Can be used to select multiple features by holding and dragging a rectangle on the map. Features that have any interaction with the rectangle are selected. The Shift key can be used with the mouse drag to deselect selected features or to add new selections to the existing selection set.

- **Select features by attribute values**: Allows the selection of features based on an attribute value. Displays the Feature Selection by Attribute dialog box, where you can specify the layer base table, the attribute column, the attribute value to be used in selection, and whether to add the result to the existing selections (if any) or to replace the existing selections with the result.

- **Unselect (deselect) features**: Deselects the currently selected features.

- **Delete selected features**: Deletes the currently selected features.

- **Duplicate selected features**: Duplicates selected features.

- **Edit feature attributes**: Edits the attributes of the selected feature.

10.2.4.2 Drawing Tools

To make the drawing tools active, check on a layer to be editable and check on a target layer. When creating a new feature using one of these tools, first deselect any selected features if you want the new geometry to be a separate new feature. If you use a feature creation tool while a feature is selected, the default behavior is to append the new geometry to the selected feature.

Icons are available for the following tools:

- **Draw a point**: Creates a new point feature. When you click on the map, a new point is created on the map, and a pop-up lets you enter the required key-column value, plus other attribute columns that you may want to populate. If a sequence name is assigned (can be changed in layers property panel), then the key value is automatically populated for the new feature.

- **Draw a line**: Creates a new line string feature (one or more linear segment). Use the left mouse button to digitize points, and the right button to end the line string.

- **Draw a polygon**: Creates a new polygon feature using linear segments. Use the left mouse button to digitize points, and the right button to end the polygon. The last point will be automatically linked to first point. To add holes (voids) to a polygon, use the corresponding tool under Grouping Tools.

- **Draw a rectangle**: Creates a new rectangle feature. Press and drag to the diagonally opposite corner, then release to generate an optimized rectangle.
- **Draw a circle or ellipse**: Creates a new circle or ellipse feature. Press and drag, then release to generate the circle or ellipse.

10.2.4.3 Vertex Tools

Vertex tools can be used to manipulate individual vertices. To make the vertex tools active, check on a layer to be editable and check on a target layer.

Icons are available for the following tools:

- **Add a vertex point to a line string:** Adds a new vertex to an existing line segment. As you move pointer close to the target line segment, it automatically snaps onto the segment; the snapped highlighted circle is a valid location to add the vertex. Then click to add the new vertex.
- **Remove a vertex point from a line string**: Removes an existing vertex. Move the mouse to the vertex (will be highlighted) and click to remove it. You also can press and drag to remove multiple vertices inside the dragging area.
- **Select and move vertex**: Moves an existing vertex. Move the mouse to the vertex (snaps will highlight the vertex) and then press and drag the vertex to the new position.
- **Add a void polygon**: Creates a new void in an existing polygon. Select a polygon feature and digitize the new void polygon inside the feature polygon. Use the left mouse button to digitize points, and the right button to end the polygon. The last point will be automatically linked to first point.
- **Remove a void from a polygon**: Removes an existing void from a polygon. To remove the void, click anywhere inside the void boundary of the selected feature, or press and drag a rectangle that encloses the void polygon.
- **Break a line**: Breaks an existing feature line string (simple line) into two lines and generates a new feature. Move the mouse over the line (snaps will highlight break points), and then click to break at the location. A dialog box is displayed for selecting which part will be a new feature and defining the new feature's attributes.
- **Split a polygon**: Splits a polygon feature into two features. Move the mouse over polygon boundary (split point will be highlighted), click to get the split point (it turns blue), then click to the next split line points until you click on another boundary split point to end the split line (snaps shows the boundary point highlighted). If a polygon has void elements that are also in the split line path, when you get a split point at the void boundary, the next split line point must be at the same void element boundary. After ending the split, a dialog box is displayed for selecting which part will be a new feature and defining the new feature's attributes.

10.2.4.4 Grouping Tools

Grouping tools are used to perform operations on multiple features and elements. With some of these tools, the target layer may be different from the selection layer. Grouping tools use Oracle Spatial and Graph operations such as union, difference, and intersections.

Icons are available for the following tools:

- **Group features:** Groups features. Select one or more features and click this button to group the selected features into a new feature. (A union operation is performed.)
- **Ungroup feature element**: (Not currently used.)
- **Remove feature element**: Removes one or more feature elements from within a features. To specify multiple elements, drag a rectangle that encloses them, and release.
- **Generate a feature from a union of features**: Generates a new feature or updates an existing feature (in the target layer) based on a union of selected features. The selected features list must have features from a layer that is not the current target layer.

  For example, assume that layer A is currently the target layer and you want to generate a new feature with the union of features from layer B.

  1. Change the selection layer in Data layers panel on the left to layer B.
  2. Select one or more features in layer B.
  3. Click the **Generate a feature from a union of features** icon to perform the union operation between the selected features in layer B. If layer A also has a selected feature, then this feature will also be used in the union operation, and the target feature geometry will be updated.
- **Generate a feature from an intersection of features**: Generates a new feature or updates an existing feature (in the target layer) based on an intersection of selected features. The selected features list must have features from a layer that is not the current target layer.

  For example, assume that layer A is currently the target layer and you want to generate a new feature with the intersection of features from layer B.

  1. Change the selection layer in Data layers panel on the left to layer B.
  2. Select one or more features in layer B.
  3. Click the **Generate a feature from an intersection of features** icon to perform the intersection operation between the selected features in layer B. If layer A also has a selected feature, then this feature will also be used in the intersection operation, and the target feature geometry will be updated.
- **Generate a feature from a difference of features**: Generates a new feature or updates an existing feature (in the target layer) based on a difference of selected features. The selected features list must have features from a layer that is not the current target layer.

  For example, assume that layer A is currently the target layer and you want to generate a new feature with the difference of features from layer B.

  1. Change the selection layer in Data layers panel on the left to layer B.
  2. Select one or more features in layer B.
  3. Click the **Generate a feature from a difference of features** icon to perform the difference operation between the selected features in layer B. If layer A also has a selected feature, then this feature will also be used in the difference operation, and the target feature geometry will be updated.
- **Update target layer geometries with union of features**: Updates features in the target layer with union of selected features from a different selection layer.

For example, assume there are two layers, one for States and another one for Counties, with each layer having its individual geometries. Each county feature has an attribute defining the State to which it belongs. You can make changes in county features, and then update the related state geometry with the union of the counties' changes. The results will include any necessary boundary adjustments in both geometry layers. Some sample steps:

1. Make Counties layer the target and the selection layer, and select all counties in State A.
2. Make some changes on these selected features (for example, scale all of them together).
3. Make the State layer the target layer and click the **Update target layer geometries with union of features** icon. A dialog box for defining the link attributes between the two layers is displayed, where you specify the target layer, target key attribute column, selection layer, and selection join attribute column.

   After you click **OK**, the geometries of the target layer State will be updated with the union of selected features based on the selection join attribute.

10.2.4.5 Geometry Tools

Geometry tools are used to perform Oracle Spatial and Graph operations on geometry objects.

Icons are available for the following tools:

- **Validate selected geometries:** Validates the selected feature geometries. If any invalid geometry is found, a dialog box is displayed with the invalid Oracle Spatial and Graph codes (For detailed information about geometry validation and error codes, see the reference material about the SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT function in *Oracle Spatial and Graph Developer's Guide*.)

  Selecting a table row in the dialog box highlights the geometry segment section related to the error. You may be able to use the available MapViewer Editor tools to fix any errors.

- **View geometry elements**: Displays a dialog box showing the geometry elements of the selected feature. You can click on a geometry node or one of the elements will highlight its border. You cannot edit the geometry information in this dialog box.

- **Simplify selected geometries**: Simplifies selected geometries. For polygon features, it first builds an internal topology to generate single edges between adjacent geometries. Depending on the number of selected geometries, this step may take a while for building the topology. A dialog box with two algorithm options is then shown, and you can simplify all edges or select a subset of edges to simplify. When selecting specific edges, you can also use the Shift key to deselect any previous selection. The available options depend on whether you select the Douglas-Peucker or Visvalingham-Whyatt algorithm.

  For Douglas-Peucker (uses the SDO_UTIL.SIMPLIFY function, described in *Oracle Spatial and Graph Developer's Guide*):

  – **Threshold**: Minimum distance between vertices to be considered. For geodetic data, this value is in meters; otherwise, it is in data units. Use the bar to define a range of points to be removed. (However, this does not necessarily mean that the algorithm will remove precisely this amount of data.)

- **Simplify**: Simplifies the selected geometries.
- **Reset**: Undoes the last simplification operation.
- **o% Points**: The percentage of original vertices retained after simplification.

For Visvalingham-Whyatt (uses the SDO_UTIL.SIMPLIFYVW function, described in *Oracle Spatial and Graph Developer's Guide*):

- **o% of points to remove**: use bar to define a range of points to be removed. (However, this does not necessarily mean that the algorithm will remove precisely this amount of data.)
- **Taller triangles**: Allows taller triangles when applying the flatness filter.
- **o% Points**: The percentage of original vertices retained after simplification.

- **Generate a buffer feature from the selected feature**: Generates a buffer feature around the selected feature. Enter the distance value to generate the buffer. If the data is geodetic, enter the value in meters; otherwise, enter the value in data units.

10.2.4.6 Transformation Tools

Icons are available for the following transformation tools:

- **Rotate selected geometries**: Rotates the selected feature geometries with a specified value in degrees.
- **Scale selected geometries in X**: Scales the selected feature geometry with a specified factor in the X direction.
- **Scale selected geometries in Y**: Scales the selected feature geometry with a specified factor in the Y direction.
- **Translate selected geometries in X**: Translates the selected geometries with a specified value in data units in X direction.
- **Translate selected geometries in Y**: Translates the selected geometries with a specified value in data units in Y direction.
- **Apply transformation to selected features in target layer**: Applies the current transformation to just the selected features of the target layer.
- **Apply transformation to all selected features**: Applies the current transformation to all selected features.
- **Reset transformation**: Clears the current transformation.

A

XML Format for Styles, Themes, Base Maps, and Map Tile Layers

This appendix describes the XML format for defining style, themes, and base maps using the MapViewer metadata views described in Section 2.9.

The metadata views for MapViewer styles (USER_SDO_STYLES and related views) contain a column named DEFINITION. For each style, the DEFINITION column contains an XML document that defines the style to the rendering engine.

Each style is defined using a syntax that is similar to SVG (scalable vector graphics). In the MapViewer syntax, each style's XML document must contain a single `<g>` element, which must have a `class` attribute that indicates the type or class of the style. For example, the following defines a color style with a filling color component:

```
<?xml version="1.0" standalone="yes"?>
   <svg width="1in" height="1in">
       <desc> red </desc>
           <g class="color" style="fill:#ff1100"/>
   </svg>
```

The MapViewer XML parser looks only for the `<g>` element in a style definition; other attributes such as the `<desc>` element are merely informational and are ignored.

> **Scalable Styles:** You can make the size of a style scalable by specifying a unit other than the default pixel (`px`) -- for example, `width:15.0km` or `stroke-width:10.0m`. For information about using scalable styles, see Section 2.2.1.

The metadata views for MapViewer themes (USER_SDO_THEMES and related views) contain a column named STYLING_RULES. For each theme in these views, the STYLING_RULES column contains an XML document (a CLOB value) that defines the styling rules of the theme.

The metadata views for MapViewer base maps (USER_SDO_MAPS and related views) contain a column named DEFINITION. For each base map in these views, the DEFINITION column contains an XML document (a CLOB value) that defines the base map.

The following sections describe the XML syntax for each type of mapping metadata:

- Section A.1, "Color Styles"
- Section A.2, "Marker Styles"
- Section A.3, "Line Styles"

- Section A.4, "Area Styles"
- Section A.5, "Text Styles"
- Section A.6, "Advanced Styles"
- Section A.7, "Themes: Styling Rules"
- Section A.8, "Base Maps"
- Section A.9, "Map Tile Layers"

A.1 Color Styles

A color style has a fill color, a stroke color, or both. When applied to a shape or geometry, the fill color (if present) is used to fill the interior of the shape, and the stroke color (if present) is used to draw the boundaries of the shape. Either color can also have an alpha value, which controls the transparency of that color.

For color styles, the `class` attribute of the `<g>` element must be set to `"color"`. The `<g>` element must have a `style` attribute, which specifies the color components and their optional alpha value. For example:

- `<g class="color" style="fill:#ff0000">` specifies a color style with only a fill color (whose RGB value is #ff0000).
- `<g class="color" style="fill:#ff0000;stroke:blue">` specifies a color style with a fill color and a stroke color (blue).

You can specify a color value using either a hexadecimal string (such as #00ff00) or a color name from the following list: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow.

To specify transparency for a color style, you can specify `fill-opacity` and `stroke-opacity` values from 0 (completely transparent) to 255 (opaque). The following example specifies a fill component with half transparency:

```
<g class="color" style="fill:#ff00ff;fill-opacity:128">
```

The following example specifies both stroke and fill opacity:

```
<g class="color" style= "stroke:red;stroke-opacity:70;
                         fill:#ff00aa;fill-opacity:129">
```

The syntax for the `style` attribute is a string composed of one or more `name:value` pairs delimited by semicolons. (This basic syntax is used in other types of styles as well.)

For stroke colors, you can define a stroke width. The default stroke width when drawing a shape boundary is 1 pixel. To change that, add a `stroke-width:`*`value`* pair to the `style` attribute string. The following example specifies a stroke width of 3 pixels:

```
<g class="color" style="stroke:red;stroke-width:3">
```

A.2 Marker Styles

A marker style represents a marker to be placed on point features or on label points of area and linear features. A marker can be either a vector marker or raster image marker. A marker can also have optional notational text. For a vector marker, the coordinates of the vector elements must be defined in its XML document. For a marker

based on a raster image, the XML document for the style indicates that the style is based on an external image.

The marker XML document specifies the preferred display size: the preferred width and height are defined by the `width:`*`value`*`;height:`*`value`* pairs in the `style` attribute of the `<g>` element. The `class` attribute must be set to `"marker"`. Some markers must be overlaid with some notational text, such as a U.S. interstate highway shield marker, which, when rendered, must also have a route number plotted on top of it. The style for such notational text is a style attribute with one or more of the following name-value pairs: `font-family:`*`value`*, `font-style:`*`value`*, `font-size:`*`value`*, and `font-weight:`*`value`*.

The following example defines an image-based marker that specifies font attributes (shown in bold) for any label text that may be drawn on top of the marker:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<desc></desc>
<g class="marker"
  style="width:20;height:18;font-family:sans-serif;font-size:9pt;fill:#ffffff">
    <image x="0" y="0" width="9999" height="9999" type="gif"
           href="dummy.gif"/>
</g>
</svg>
```

In the preceding example, when the marker is applied to a point feature with a labeling text, the label text is drawn centered on top of the marker, using the specified font family and size, and with the fill color (white in this case) as the text foreground. The label text (`495`) in Figure A–1 in Section A.2.4 has the text attributes specified in this example.

A.2.1 Vector Marker Styles

A vector marker can be a simple polygon, an optimized rectangle (defined using two points), a single polyline, or a circle, but not any combination of them. For each type of vector marker, its `<g>` element must contain a corresponding subelement that specifies the geometric information (coordinates for the polygon, optimized rectangle, or polyline, or radius for the circle):

- A polygon definition uses a `<polygon>` element with a `points` attribute that specifies a list of comma-delimited coordinates. For example:

```
<g class="marker">
   <polygon points="100,20,40,50,60,80,100,20"/>
</g>
```

- An optimized rectangle definition uses a `<rect>` element with a `points` attribute that specifies a list of comma-delimited coordinates. For example:

```
<g class="marker">
  <rect points="0,0, 120,120"/>
</g>
```

- A polyline definition uses a `<polyline>` element with a `points` attribute that specifies a list of comma-delimited coordinates. For example:

```
<g class="marker">
   <polyline points="100,20,40,50,60,80"/>
</g>
```

- A circle definition uses a `<circle>` element with an `r` attribute that specifies the radius of the circle. For example:

```
<g class="marker">
   <circle r="50"/>
</g>
```

You can specify a stroke or fill color, or both, for any vector-based marker. The syntax is the same as for the style attribute for a color style. The following example defines a triangle marker that has a black border and that is filled with a half-transparent yellow:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<g class="marker"  style="stroke:#000000;fill:#ffff00;fill-opacity:128">
     <polygon points="201.0,200.0, 0.0,200.0, 101.0,0.0"/>
</g>
</svg>
```

A.2.2 Image Marker Styles

For an image marker, its XML document contains an `<image>` element that identifies the marker as based on an image. The image must be in GIF format, and is stored in the IMAGE column in the styles metadata views.

The following example is an XML document for an image marker:

```
<?xml version="1.0" standalone="yes"?>
<svg>
   <g class="marker"
            style="width:20;height:18;font-family:sansserif;font-size:9pt">
     <image x="0" y="0" width="9999" height="9999" type="gif" href="dummy.gif"/>
   </g>
</svg>
```

Note that in the preceding example, it would be acceptable to leave the `<image>` element empty (that is, `<image/>`) to create a valid definition with the image to be specified later.

A.2.3 TrueType Font-Based Marker Styles

For a TrueType font-based marker, its marker symbol is stored in a TrueType font file, which has the .ttf file extension and which typically contains many individual symbols or glyphs. Many GIS software packages come with TrueType font files that contain symbols useful for mapping.

Before MapViewer can use a symbol in a TrueType font file, you must do the following:

1. Import the TrueType font file into the database, preferably by using the Map Builder tool (described in Chapter 9), which causes the symbols in the font file to be inserted into a single row in the system view USER_SDO_STYLES. In this new row, the TYPE column contains the string TTF, and the IMAGE column contains the contents of the TrueType font file. After the import operation, you can use the Map Builder tool to view all the glyphs or symbols contained inside the TrueType font file. Also, because the font file is now physically stored inside a database, it can be shared by all MapViewer users.
2. Create a MapViewer marker style based on a glyph or symbol inside an imported TrueType font, preferably using the Map Builder tool.

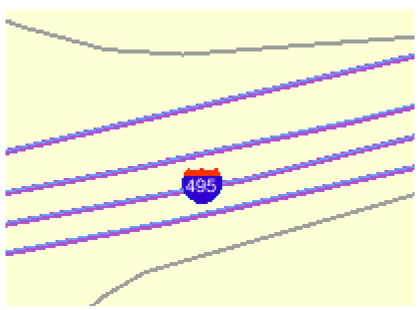
The following example shows the use of a TrueType font-based marker (with TrueType-specific material in bold):

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<g class="marker"  style="fill:#ff0000;width:25;height:25">
  <ttfSymbol fontName="ERS_INCIDENTS" charCode="118" />
</g>
</svg>
```

A.2.4 Using Marker Styles on Lines

Marker styles are usually applied to point features, in which case the marker style is rendered on the point location that represents the feature. However, with line (line string) features such as highways, the marker must be placed at some point along the line to denote some information about the feature, such as its route number. For example, on maps in the United States, a shield symbol is often placed on top of a highway, with a route number inside the symbol, as shown with Route 495 in Figure A–1.

Figure A–1 Shield Symbol Marker for a Highway



To achieve the result shown in Figure A–1, you must do the following:

1. Choose a marker style, and add a text style definition (font family, font size, fill color, and so on), as shown in the example in Section A.2.
2. Specify the marker style as the labeling style in the styling rules for the theme. The following example shows the XML document with the styling rules for a theme to show highways. A marker style (shown in bold in the example) is specified. The label text (`495` in Figure A–1) is a value from the label column, which is named LABEL in this example.

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="political">
<rule>
   <features style="L.PH"> (name_class = 'I' and TOLL=0) </features>
   <label column="label" style="M.SHIELD1">1</label>
</rule>
<styling_rules>
```

MapViewer automatically determines the optimal position on the line for placement of the marker style (the shield in this example).

A.3 Line Styles

A line style is applicable only to a linear feature, such as a road, railway track, or political boundary. In other words, line styles can be applied only to Oracle Spatial and Graph geometries with an SDO_GTYPE value ending in 2 (line) or 6 (multiline). (For information about the SDO_GEOMETRY object type and SDO_GTYPE values, see *Oracle Spatial and Graph Developer's Guide*.)

When MapViewer draws a linear feature, a line style tells the rendering engine the color, dash pattern, and stroke width to use. A line style can have a base line element which, if defined, coincides with the original linear geometry. It can also define two edges parallel to the base line. Parallel line elements can have their own color, dash pattern, and stroke width. If parallel lines are used, they must be located to each side of the base line, with equal offsets to it.

To draw railroad-like lines, you need to define a third type of line element in a line style called *hashmark*. For a `<line>` element of class `hashmark`, the first value in the dash array indicates the gap between two hash marks, and the second value indicates the length of the hash mark to either side of the line. The following example defines a hash mark line with a gap of 8.5 screen units and a length of 3 screen units at each side of the base line:

```
<line class="hashmark" style="fill:#003333"  dash="8.5,3.0"/>
```

The following example defines a complete line style.

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
   <g class="line" style="fill:#ffff00;stroke-width:5">
      <line class="parallel" style="fill:#ff0000;stroke-width:1.0"/>
      <line class="base" style="fill:black;stroke-width:1.0" dash="10.0,4.0"/>
   </g>
</svg>
```

In the preceding example, `class="line"` identifies the style as a line style. The overall fill color (#ffff00) is used to fill any space between the parallel lines and the base line. The overall line width (5 pixels) limits the maximum width that the style can occupy (including that of the parallel lines).

The line style in the preceding example has both base line and parallel line elements. The parallel line element (`class="parallel"`) is defined by the first `<line>` element, which defines its color and width. (Because the definition does not provide a dash pattern, the parallel lines or edges will be solid.) The base line element (`class="base"`) is defined by the second `<line>` element, which defines its color, width, and dash pattern.

A marker (such as a direction marker) can be defined for a line style. The `marker-name` parameter specifies the name of a marker style, the `marker-position` parameter specifies the proportion (from 0 to 1) of the distance along the line from the start point at which to place the marker, and the `marker-size` parameter specifies the number of display units for the marker size. The marker orientation follows the orientation of the line segment on which the marker is placed.

The following example defines a line style with direction marker:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
  <g class="line" style="fill:#33a9ff;stroke-width:4;
      marker-name:M.IMAGE105_BW;marker-position:0.15;marker-size=8">
    <line class="parallel" style="fill:red;stroke-width:1.0"/>
  </g>
```

```
</svg>
```

To get multiple markers, add the `multiple-marker` attribute to the style definition. In this case the marker-position will define the position for the first marker and the space in between markers. The following example defines a line style with a direction marker that starts at position 0.15 and that is repeated continually with a space of 0.15 between each occurrence.

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
 <g class="line" style="fill:#33a9ff;stroke-width:4;
     marker-name:M.IMAGE105_BW; marker-position:0.15;
     marker-size=8; multiple-marker=true">
   <line class="parallel" style="fill:red;stroke-width:1.0"/>
 </g>
</svg>
```

A.4 Area Styles

An area style defines a pattern to be used to fill an area feature. In the current release, area styles must be image-based. That is, when you apply an area style to a geometry, the image defining the style is plotted repeatedly until the geometry is completely filled.

The definition of an area style is similar to that of an image marker style, which is described in Section A.2.2.

The following example defines an area style:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
   <g class="area"  style="stroke:#000000">
      <image/>
   </g>
</svg>
```

In the preceding example, `class="area"` identifies the style as an area style. The stroke color (`style="stroke:#000000"`) is the color used to draw the geometry boundary. If no stroke color is defined, the geometry has no visible boundary, although its interior is filled with the pattern image.

You can also specify any line style to be used as the boundary for an area style. The following area style definition uses the `line-style` keyword (shown in bold in the example) to specify a line style to be used for the borders of features:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
  <g class="area" style="line-style:L.DPH">
    <image x="0" y="0" width="9999" height="9999" type="gif" href="dummy.gif"/>
  </g>
</svg>
```

As with the image marker style, the image for an area style must be stored in a separate column (identified in the IMAGE column in the USER_SDO_STYLES and ALL_SDO_STYLES metadata views, which are described in Section 2.9.3).

A.5 Text Styles

A text style defines the font and color to be used in labeling spatial features. The `class` attribute must have the value `"text"`. For the font, you can specify its style

(plain, italic, and so on), font family, size, and weight. To specify the foreground color, you use the `fill` attribute.

The following example defines a text style:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
   <g class="text" style="font-style:plain; font-family:Dialog; font-size:14pt;
             font-weight:bold; fill:#0000ff">
    Hello World!
   </g>
</svg>
```

In the preceding example, the text "Hello World!" is displayed only when the style itself is being previewed in a style creation tool, such as the Map Builder tool. When the style is applied to a map, it is always supplied with an actual text label that MapViewer obtains from a theme.

A text style can provide a floating white background around the rendered text, to make the labels easier to read on a map that has many features. Figure A–2 shows the label Vallejo with a white background wrapping tightly around the letters.

Figure A–2 Text Style with White Background

To achieve the result shown in Figure A–2, you must specify the `float-width` attribute in the `<g>` element of the text style definition. The following example uses the `float-width` attribute (shown in bold in the example) to specify a white background that extends 3.5 pixels from the boundary of each letter. (The Hello World! text is ignored when the style is applied to the display of labels.)

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<desc></desc>
<g class="text"  float-width="3.5"
   style="font-style:plain; font-family:Dialog; font-size:12pt; font-weight:bold;
     fill:#000000">
 Hello World!
</g>
</svg>
```

A.6 Advanced Styles

Advanced styles are structured styles made from simple styles. Advanced styles are used primarily for thematic mapping. The core advanced style is the bucket style (`BucketStyle`), and every advanced style is a form of bucket style. A bucket style is a one-to-one mapping between a set of primitive styles and a set of buckets. Each bucket contains one or more attribute values of features to be plotted. For each feature, one of its attributes is used to determine which bucket it falls into or is contained within, and then the style assigned to that bucket is applied to the feature.

Two special types of bucket styles are also provided: color scheme (described in Section A.6.2) and variable (graduated) marker (described in Section A.6.3).

Other advanced styles are dot density (described in Section A.6.4), bar chart (described in Section A.6.5), collection (described in Section A.6.6), and variable pie chart (described in Section A.6.7).

A.6.1 Bucket Styles

A bucket style defines a set of buckets, and assigns one primitive style to each bucket. The content of a bucket can be either of the following:

- A collection of discrete values (for example, a bucket for all counties with a hurricane risk code of 1 or 2, a bucket for all counties with a hurricane risk code of 3, and so on).
- A continuous range of values (for example, a bucket for all counties with average family income less than $30,000, a bucket for all counties with average family income from $30,000 through $39,999, and so on). In this case, the ranges of a series of buckets can be individually defined (each defined by an upper-bound value and lower-bound value) or equally divided among a master range.

The following code excerpt shows the basic format of a bucket style:

```
<?xml version="1.0" ?>
<AdvancedStyle>
  <BucketStyle>
    <Buckets>
    . . .
    </Buckets>
  </BucketStyle>
</AdvancedStyle>
```

In contrast with the other (primitive) styles, an advanced style always has a root element identified by the `<AdvancedStyle>` tag.

For bucket styles, a `<BucketStyle>` element is the only child of the `<AdvancedStyle>` element. Each `<BucketStyle>` element has one or more `<Buckets>` child elements, whose contents vary depending on the type of buckets.

A.6.1.1 Collection-Based Buckets with Discrete Values

If each bucket of a bucket style contains a collection of discrete values, use a `<CollectionBucket>` element to represent each bucket. Each bucket contains one or more values. The values for each bucket are listed as the content of the `<CollectionBucket>` element, with multiple values delimited by commas. The following example defines three buckets.

```
<?xml version="1.0" ?>
  <AdvancedStyle>
    <BucketStyle>
      <Buckets>
        <CollectionBucket  seq="0" label="commercial"
            style="10015">commercial</CollectionBucket>
        <CollectionBucket  seq="1" label="residential"
            style="10031">residential, rural</CollectionBucket>
        <CollectionBucket  seq="2" label="industrial"
            style="10045">industrial, mining, agriculture</CollectionBucket>
    </Buckets>
  </BucketStyle>
</AdvancedStyle>
```

In the preceding example:

- The values for each bucket are one or more strings; however, the values can also be numbers.
- The name of the style associated with each bucket is given.
- The label attribute for each `<CollectionBucket>` element (*commercial*, *residential*, or *industrial*) is used only in a label that is compiled for the advanced style.
- The order of the `<CollectionBucket>` elements is significant. However, the values in the `seq` (sequence) attributes are informational only; MapViewer determines sequence only by the order in which elements appear in a definition.

Although not shown in this example, if you want a bucket for all other values (if any other values are possible), you can create a `<CollectionBucket>` element with `#DEFAULT#` as its attribute value. It should be placed after all other `<CollectionBucket>` elements, so that its style will be rendered last.

To apply label styles to collection-based buckets with discrete values, see Section 2.2.2.

A.6.1.2 Individual Range-Based Buckets

If each bucket of a bucket style contains a value range that is defined by two values, use a `<RangedBucket>` element to represent each bucket. Each bucket contains a range of values. The following example defines four buckets.

```
<?xml version="1.0" ?>
 <AdvancedStyle>
   <BucketStyle>
     <Buckets>
          <RangedBucket high="10" style="10015"/>
          <RangedBucket low="10" high="40" style="10024"/>
          <RangedBucket low="40" high="50" style="10025"/>
          <RangedBucket low="50" style="10029"/>
     </Buckets>
   </BucketStyle>
</AdvancedStyle>
```

For individual range-based buckets, the lower-bound value is inclusive, while the upper-bound value is exclusive (except for the range that has values greater than any value in the other ranges; its upper-bound value is inclusive). No range is allowed to have a range of values that overlaps values in other ranges.

For example, the second bucket in this example (`low="10" high="40"`) will contain any values that are exactly 10, as well as values up to but not including 40 (such as 39 and 39.99). Any values that are exactly 40 will be included in the third bucket.

As with the `<CollectionBucket>` element, the style associated with each `<RangedBucket>` element is specified as an attribute.

To apply label styles to individual range-based buckets, see Section 2.2.2.

A.6.1.3 Equal-Ranged Buckets

If a bucket style contains a series of buckets that contain an equally divided range of a master range, you can omit the use of `<RangedBucket>` elements, and instead specify in the `<Buckets>` element the master upper-bound value and lower-bound value for the overall range, the number of buckets in which to divide the range, and a list of style names (with one for each bucket). The following example defines five buckets (`nbuckets=5`) of equal range between 0 and 29:

```
<?xml version="1.0" ?>
```

```
<AdvancedStyle>
   <BucketStyle>
      <Buckets low="0" high="29" nbuckets="5"
          styles="10015,10017,10019,10021,10023"/>
   </BucketStyle>
 </AdvancedStyle>
```

In the preceding example:

- If all values are integers, the five buckets hold values in the following ranges: 0 to 5, 6 to 11, 12 to 17, 18 to 23, and 24 to 29.
- The first bucket is associated with the style named `10015`, the second bucket is associated with the style named `10017`, and so on.

The number of style names specified must be the same as the value of the `nbuckets` attribute. The buckets are arranged in ascending order, and the styles are assigned in their specified order to each bucket.

A.6.2 Color Scheme Styles

A color scheme style automatically generates individual color styles of varying brightness for each bucket based on a base color. The brightness is equally spaced between full brightness and total darkness. Usually, the first bucket is assigned the brightest shade of the base color and the last bucket is assigned the darkest shade.

You can include a stroke color to be used by the color style for each bucket. The stroke color is not part of the brightness calculation. So, for example, if a set of polygonal features is rendered using a color scheme style, the interior of each polygon is filled with the color (shade of the base color) for each corresponding bucket, but the boundaries of all polygons are drawn using the same stroke color.

You can include an opacity value (0 to 255, for transparent to opaque) for the base color (using the `basecolor_opacity` attribute) and for the stroke color (using the `strokecolor_opacity` attribute).

The following example defines a color scheme style with a black stroke color and four buckets associated with varying shades of the base color of blue.

```
<?xml version="1.0" ?>
<AdvancedStyle>
 <ColorSchemeStyle basecolor="blue" strokecolor="black">
   <Buckets>
        <RangedBucket label="&lt;10"  high="10"/>
        <RangedBucket label="10 - 20" low="10" high="20"/>
        <RangedBucket label="20 - 30" low="20" high="30"/>
        <RangedBucket label="&gt;=30" low="30"/>
   </Buckets>
  </ColorSchemeStyle>
</AdvancedStyle>
```

Note: For the following special characters, use escape sequences instead.

For <, use: `&lt`

For >, use: `>`

For &, use: `&`

A.6.3 Variable Marker Styles

A variable marker style generates a series of marker styles of varying sizes for each bucket. You specify the number of buckets, the start (smallest) size for the marker, and the size increment between two consecutive markers.

Variable marker styles are conceptually similar to color scheme styles in that both base buckets on variations from a common object: with a color scheme style the brightness of the base color varies, and with a variable marker style the size of the marker varies.

The following example creates a variable marker style with four buckets, each associated with different sizes (in increments of 4) of a marker (`m.circle`). The marker for the first bucket has a radius of 10 display units, the marker for the second bucket has a radius of 14 display units, and so on. This example assumes that the marker named `m.circle` has already been defined.

```
 <?xml version="1.0" ?>
<AdvancedStyle>
  <VariableMarkerStyle basemarker="m.circle" startsize="10" increment="4">
     <Buckets>
         <RangedBucket label="&lt;10"  high="10"/>
         <RangedBucket label="10 - 20" low="10" high="20"/>
         <RangedBucket label="20 - 30" low="20" high="30"/>
         <RangedBucket label="&gt;=30"   low="30"/>
     </Buckets>
  </VariableMarkerStyle>
</AdvancedStyle>
```

A.6.4 Dot Density Marker Styles

A dot density advanced marker style, when applied to an area feature such as states or counties, randomly draws a set of dots inside the area. The number of dots drawn inside each area is determined by the count value associated with the area. When you define a dot density style, you must specify a marker style that will be used for each of the dots.

The following example shows the XML definition of a simple dot density style:

```
<?xml version="1.0" ?>
<AdvancedStyle>
        <DotDensityStyle MarkerStyle="M.STAR" DotWidth="8" DotHeight="8">
        </DotDensityStyle>
</AdvancedStyle>
```

In the preceding example, the marker style M.STAR is used for each dot, and the size of each dot is 8 pixels wide and high.

When you use a dot density style, you should "scale" the count value to a proper range. For example, if you want to apply a dot density style based on the population count for each county, you would not want to use the population count directly (one dot for each person), because this will result in an unacceptable number of drawn dots (for example, if a county has 15,000 people). Instead, supply a scaled down value or expression, such as `population/1000`, when you define the styling rules for the theme. (MapViewer does not perform any scaling-down internally, so you must do it at the SQL query level.)

A.6.5 Bar Chart Marker Styles

A bar chart advanced marker style is similar to a pie chart style, except that it draws a bar graph for each feature to which it is applied. The following example shows the XML definition of a bar chart style:

```
<?xml version="1.0" ?>
<AdvancedStyle>
   <BarChartStyle width="30" height="25" show_x_axis="true">
       <Bar name="1990" color="#FF0000" />
       <Bar name="1995" color="#FFC800" />
       <Bar name="1998" color="#0000FF" />
       <Bar name="2000" color="#00FF00" />
       <Bar name="2002" color="#00FFFF" />
   </BarChartStyle>
</AdvancedStyle>
```

In the preceding example, width and height specify the overall size of the bar chart, including all individuals bars within it.

When a bar chart is drawn on a feature based on a set of values associated with that feature, the height of each bar can be determined by either of two approaches: locally scaled or globally scaled. A locally scaled bar chart determines the height of each bar only from the associated values for that feature; and thus, for example, you cannot compare the second bar of one chart to the second bar on another chart on the same theme. A globally scaled bar chart uses the same bar scale for all charts on the map; and thus, for example, you can compare the second bar of one chart to the second bar on another chart on the same theme.

So, if you want to compare bars not only within the same chart, but also among all the charts showing on the map, you must use globally scaled bar chart style by specifying `share_scale="true"` in the definition of the bar chart style, as shown in the following example:

```
<?xml version="1.0" ?>
<AdvancedStyle>
   <BarChartStyle width="40" height="30" share_scale="true"
      min_value="0.0" max_value="100">
       <Bar name="1990" color="#FF0000" />
       <Bar name="1995" color="#FFC800" />
       <Bar name="1998" color="#0000FF" />
       <Bar name="2000" color="#00FF00" />
       <Bar name="2002" color="#00FFFF" />
   </BarChartStyle>
</AdvancedStyle>
```

When the bar chart style in the preceding example is applied to a theme, MapViewer considers the global range of values of all features in that theme, and then determines the height of each bar based on where a specific value falls in the global range from the minimum value to the maximum value.

A.6.6 Collection Styles

A collection advanced style is simply a collection of other types of styles that are applied together to a feature. This can result in faster rendering of a collection theme compared to using multiple themes based on different styles.

For example, a bar chart style, when applied to a county, draws only the bar chart somewhere inside the county, but the county itself (its boundary and interior area) is not drawn. However, you probably want to see the underlying boundaries of the

counties, to see which bar chart belongs to which county. To do this without a collection style, you would have to define a second theme in which each county is being associated with a color or area style. This approach would result in two rendering passes (because two themes are involved) for essentially the same group of features.

However, by using a collection style in this example, you can define a single style that refers to both the bar chart and the color or area style, and then apply the collection style to the theme for the counties. This theme, when rendered by MapViewer, will show both the bar charts and the boundaries on the map.

Another typical use of a collection style is for rendering collection type topology features, each of which can contain multiple types of geometries, such as polygons (areas), points, and lines. In such cases, a collection style can include styles that are most appropriate for each type of geometry in a collection topology feature.

The following example shows the XML definition of a collection style:

```
<?xml version="1.0" standalone="yes"?>
 <AdvancedStyle>
   <CollectionStyle>
     <style name="C.COUNTIES" shape="polygon" />
     <style name="L.PH" shape="line" />
     <style name="M.CIRCLE" shape="point" />
   </CollectionStyle>
  </AdvancedStyle>
```

A.6.7 Variable Pie Chart Styles

A variable pie chart generates a series of pie circles of varying sizes for each bucket. You specify the pie slice information, the start (smallest) radius size for a pie circle, and the radius size increment between two consecutive circles.

Variable pie chart styles are conceptually similar to variable marker styles. With a variable marker style the base marker size varies, whereas with the variable pie chart style the circle radius varies.

The following example creates a definition for a variable pie chart style with four buckets, each associated with different sizes (in increments of 4) of a circle with start radius of 5. The circle radius for the first bucket has a radius of 5 display units, the circle for the second bucket has a radius of 9 display units, and so on.

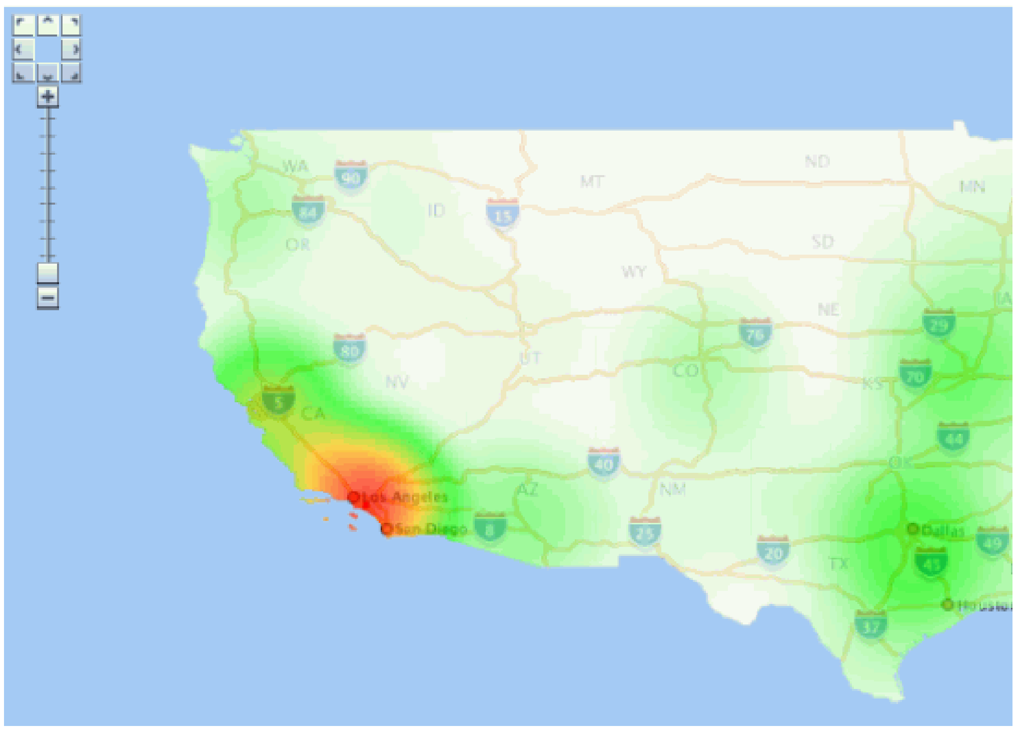
```
<?xml version="1.0" ?>
<AdvancedStyle>
  <VariablePieChartStyle startradius="5" increment="4">
    <PieSlice name="WHITE" color="#FFFFFF"/>
    <PieSlice name="BLACK" color="#000000"/>
    <PieSlice name="HISPANIC" color="#FF0000"/>
   <Buckets>
    <RangedBucket seq="0" label="0 - 6194757.2" low="0" high="6194757.2" />
    <RangedBucket seq="1" label="6194757.2 - 1.23895144E7" low="6194757.2"
high="1.23895144E7"/>
    <RangedBucket seq="2" label="1.23895144E7 - 1.85842716E7" low="1.23895144E7"
high="1.85842716E7"/>
    <RangedBucket seq="3" label="1.85842716E7 - 2.47790288E7" low="1.85842716E7"
high="2.47790288E7"/>
    <RangedBucket seq="4" label="2.47790288E7 - 3.0973786E7" low="2.47790288E7"
high="3.0973786E7"/>
   </Buckets>
  </VariablePieChartStyle>
</AdvancedStyle>
```

A.6.8 Heat Map Styles

A heat map style can be used to generate a two-dimensional (2D) color map of any point-type data set. The colors represent the distribution density or pattern of the points or events across the region. Internally, MapViewer creates a 2D matrix and assigns a value to each grid cell based on the result of a distance-weighted algorithm run against the point data set.

You can create a heat map style using the Map Builder tool, and assign it as the rendering style for a point-type geometry theme. You can then add this theme to a base map, or add it as a theme-based FOI layer to an interactive Oracle Maps application. Figure A–3 shows a map displayed using a theme based on a heat map style. This map shows the concentration of pizza restaurants: red areas have the highest concentration of pizza restaurants, with concentrations progressively lower for orange, yellow, dark green, lighter green, pale green, and white areas.

Figure A–3 Heat Map Showing Pizza Restaurant Concentration



The following example creates a definition for a heat map style.

```
<?xml version="1.0" ?>
<AdvancedStyle>
     <HeatMapStyle>
          <color_stops num_steps="200" alpha="128">
            FFFFFF,00FF00, FFC800,FF0000
          </color_stops>
          <spot_light_radius>75.0mile</spot_light_radius>
          <grid_sample_factor>2.5</grid_sample_factor>
          <container_theme>THEME_DEMO_STATES</container_theme>
    </HeatMapStyle>
</AdvancedStyle>
```

The preceding example defines these essential aspects of the heat map:

- **Color stops**. Color stops are used to generate a color gradient. In this example, the color gradient will go from white (maps to grid cells with a zero value) to green, to orange, and finally to full red (maps to grid cells with highest values). The gradient will have 200 colors that span these 4 color stops. All the colors will have an alpha value of 128 (half transparent, where 0 would be fully transparent and 255 would be opaque).
- **Spot light radius**. The spot light radius defines the radius around each grid cell where events or points within this radius will be contributing to the final aggregated value of that cell. The contribution of each point decreases as its distance from the cell center increases, and becomes zero beyond this radius.

  You can specify the radius in pixels or in a real ground unit such as `mile`. When you specify the radius in pixels (the default if you do not specify a unit), the mapping from the color gradient to the grid cells will vary as the user zooms in and out on the map. This occurs because the number of points fall within the radius is constantly changing as the user zooms in and out. To achieve a fixed heat map regardless of map scale, you must specify the spotlight radius in a ground unit such as `meter`, `km`, or `mile`. The preceding example uses `mile`.
- **Grid sample factor**. The grid sample factor is used to sample the current map window size when creating the internal matrix or grid for heat map calculation. For example, a sample factor of 4 means that the internal heat map grid will be one-fourth (0.25) the actual map window size. So, if the map is 1000x1000 pixels, the internal heat map grid is 250x250. Thus, the lower the grid sample factor value, the larger the internal heat map grid will be; and the higher the value, the smaller the internal heat map grid will be.

  The grid sample factor value probably has more effect on heat map rendering performance than any other attribute, because a large internal heat map grid (resulting from a low grid sample factor value) will significantly increase the overall computation time. A good general guideline is to specify a grid sample factor value high enough so that the internal heat map grid will be 512x512 pixels or smaller.
- **Container theme name**. The container theme name specifies the name of a theme (predefined geometry theme in the same database schema) that defines the boundary of the map for the heat map theme. For example, if you are generating a heat map for a point data set that scatters all over the entire United States of America, choose a theme that represents the US national boundary or all the states as its container theme.

  The specified container theme does not affect how the heat map itself is calculated (which is solely based on the point distribution and the spotlight radius). Instead, the container theme it masks out all colored cells that are outside the boundary of the study region. This helps to ensure a "clean" look for the heat map.

After you create a heat map style, you can create a theme for point data and assign the new heat map style as the rendering style for the theme.

Unlike other types of advanced styles, heat map styles do not require any attribute or value columns.

Labels are not supported for themes rendered using heat map styles.

A.7 Themes: Styling Rules

A theme definition contains one <styling_rules> element, which may have several other elements depending on the theme type. This <styling_rules> element is

specified in the STYLING_RULES column of the USER_SDO_THEMES metadata view, using the following DTD:

```
<!ELEMENT styling_rules (rule+, hidden_info?, operations?, bitmap_masks?,
parameters?)>
<!ATTLIST styling_rules theme_type         CDATA #IMPLIED
                        key_column         CDATA #IMPLIED
                        caching            CDATA #IMPLIED "NORMAL"
                        image_format       CDATA #IMPLIED
                        image_column       CDATA #IMPLIED
                        image_resolution   CDATA #IMPLIED
                        image_unit         CDATA #IMPLIED
                        raster_id          CDATA #IMPLIED
                        raster_table       CDATA #IMPLIED
                        raster_pyramid     CDATA #IMPLIED
                        raster_bands       CDATA #IMPLIED
                        polygon_mask       CDATA #IMPLIED
                        transparent_nodata CDATA #IMPLIED
                        network_name       CDATA #IMPLIED
                        network_level      CDATA #IMPLIED
                        topology_name      CDATA #IMPLIED
                        service_url        CDATA #IMPLIED
                        srs                CDATA #IMPLIED
                        feature_ids        CDATA #IMPLIED
                        provider_id        CDATA #IMPLIED
                        srid               CDATA #IMPLIED>

<!ELEMENT rule (features, label?, rendering?)>
<!ATTLIST rule column CDATA #IMPLIED>

<!ELEMENT features (#PCDATA?, link?, node?, path?)>
<!ATTLIST features style CDATA #REQUIRED>

<!ELEMENT label (#PCDATA?, link?, node?, path?)>
<!ATTLIST label column CDATA #REQUIRED
                style  CDATA #REQUIRED>

<!ELEMENT link (#PCDATA)>
<!ATTLIST link style               CDATA #REQUIRED
               direction_style     CDATA #IMPLIED
               direction_position  CDATA #IMPLIED
               direction_markersize CDATA #IMPLIED
               column              CDATA #REQUIRED>

<!ELEMENT node (#PCDATA)>
<!ATTLIST node style      CDATA #REQUIRED
               markersize CDATA #IMPLIED
               column     CDATA #REQUIRED>

<!ELEMENT path (#PCDATA)>
<!ATTLIST path ids    CDATA #REQUIRED
               styles CDATA #REQUIRED
               style  CDATA #REQUIRED
               column CDATA #REQUIRED>

<!ELEMENT hidden_info (field+)>

<!ELEMENT field (#PCDATA)>
<!ATTLIST field column CDATA #REQUIRED
                name   CDATA #IMPLIED>
```

```
<!ELEMENT rendering (style+)>

<!ELEMENT style (substyle?)>
<!ATTLIST style name          CDATA #REQUIRED
                value_columns CDATA #IMPLIED>

<!ELEMENT substyle (#PCDATA)>
<!ATTLIST substyle name          CDATA #REQUIRED
                   value_columns CDATA #REQUIRED
                   changes       CDATA #IMPLIED>

<!ELEMENT operations (operation?)>

<!ELEMENT operation (parameter?)>
<!ATTLIST operation name CDATA #REQUIRED>

<!ELEMENT parameters (parameter?)>

<!ELEMENT parameter (#PCDATA)>
<!ATTLIST parameter name  CDATA #REQUIRED
                    value DATA #REQUIRED>

<!ELEMENT bitmap_masks (mask+)>

<!ELEMENT mask (#PCDATA)>
<!ATTLIST mask raster_id    CDATA #REQUIRED
               raster_table CDATA #REQUIRED
               layers       CDATA #REQUIRED
               zeromapping  CDATA #IMPLIED
               onemapping   CDATA #IMPLIED>
```

The <styling_rules> element can have a `theme_type` attribute, which is used mainly for certain types of predefined themes. (The default `theme_type` attribute value is `geometry`, which indicates that the theme is based on spatial geometries.) The `theme_type` attribute values for these special types of predefined themes are as follows:

- `annotation` specifies an annotation text theme. Annotation text themes are explained in Section 2.3.10.
- `geom_custom` specifies a custom geometry theme. You must also specify the `provider_id` and `srid` attributes. Custom geometry themes are explained in Section 2.3.9.
- `georaster` specifies a GeoRaster theme. To use specified GeoRaster data (but not if you use a query condition to retrieve the GeoRaster data), you must also specify the `raster_id` and `raster_table` attributes. You can also specify the `raster_pyramid`, `raster_bands`, `polygon_mask`, and `transparent_nodata` attributes. GeoRaster themes are explained in Section 2.3.4.
- `image` specifies an image theme. You must also specify the `image_format` and `image_column` attributes, and you can specify the `image_resolution` and `image_unit` attributes. Image themes are explained in Section 2.3.3.
- `network` specifies a network theme. You must also specify the `network_name` attribute. You can specify the `network_level` attribute, but the default value (1) is the only value currently supported. Network themes are explained in Section 2.3.5.
- `topology` specifies a topology theme. You must also specify the `topology_name` attribute. Topology themes are explained in Section 2.3.6.

- wfs specifies a WFS theme. You must also specify the service_url and srs attributes. WFS themes are explained in Section 2.3.7.

The <styling_rules> element can have a key_column attribute. This attribute is needed only if the theme is defined on a join view (a view created from multiple tables). In such a case, you must specify a column in the view that will serve as the key column to uniquely identify the geometries or images in that view. Without this key column information, MapViewer will not be able to cache geometries or images in a join view.

The <styling_rules> element can have a caching attribute, which specifies the caching scheme for each predefined theme. The caching attribute can have one of the following values: NORMAL (the default), NONE, or ALL.

- NORMAL causes MapViewer to try to cache the geometry data that was just viewed, to avoid repeating the costly unpickling process when it needs to reuse the geometries. Geometries are always fetched from the database, but they are not used if unpickled versions are already in the cache.
- NONE means that no geometries from this theme will be cached. This value is useful when you are frequently editing the data for a theme and you need to display the data as you make edits.
- ALL causes MapViewer to pin all geometry data of this theme entirely in the cache before any viewing request. In contrast to the default value of NORMAL, a value of ALL caches all geometries from the base table the first time the theme is viewed, and the geometries are not subsequently fetched from the database.

For detailed information about the caching of predefined themes, see Section 2.3.1.5.

Each <rule> element must have a <features> element and can have a <label> element and a <rendering> element. The <rendering> element can be used to define multiple render styles, and in this case the render style in the <features> element may be undefined. If the render style in the <features> element is defined and <rendering> element is also defined, MapViewer will first render the style in the <features> element and then render the styles in <rendering> element. (The <rendering> element is explained later in this section.)

The optional column attribute of a <rule> element specifies one or more attribute columns (in a comma-delimited list) from the base table to be put in the SELECT list of the query generated by MapViewer. The values from such columns are usually processed by an advanced style for this theme. The following example shows the use of the column attribute:

```
<?xml version="1.0" standalone="yes"?>
<styling_rules >
  <rule column="TOTPOP">
    <features style="V.COUNTY_POP_DENSITY">  </features>
  </rule>
</styling_rules>
```

In the preceding example, the theme's geometry features will be rendered using an advanced style named V.COUNTY_POP_DENSITY. This style will determine the color for filling a county geometry by looking up numeric values in the column named TOTPOP in the base table for this theme.

Each <features> element for a network theme must have a <link>, <node>, or <path> element, or some combination of them. (The <link>, <node>, and <path> elements apply only to network themes, which are explained in Section 2.3.5.) The following example shows the styling rules for a network theme to render links and nodes.

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="network"
           network_name="LRS_TEST" network_level="1">
 <rule>
   <features>
     <link style="C.RED"
           direction_style="M.IMAGE105_BW"
           direction_position="0.85"
           direction_markersize="8"></link>
     <node style="M.CIRCLE" markersize="5"></node>
   </features>
 </rule>
</styling_rules>
```

A <label> element must have a SQL expression as its element value for determining whether or not a label will be applied to a feature. The column attribute specifies a SQL expression for text values to label features, and the style attribute specifies a text style for rendering labels.

The <rendering> element can be used to define multiple rendering styles. The styles are rendered in the order that they appear. Each style in a <rendering> element is defined by a <style> element, which must specify the name attribute and can specify the value_columns attribute. (The value_columns attribute is used with advanced styles, and the column names are added to the list of attributes defined in the column attribute of <rule> element.)

In the <rendering> element, each <style> element can have a <substyle> element that defines the attributes for filling the feature. A <substyle> element must specify the name attribute and can specify the value_columns and changes attributes. For the changes attribute, only the FILL_COLOR value is supported.

The following example shows the styling rules for a geometry theme using the <rendering> element. It defines an advanced style named V.POIVMK to render the feature shape and an advanced substyle named V.POIBKT to fill the feature shape.

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule>
    <features> </features>
    <label column="NAME" style="T.STREET2"> 1 </label>
    <rendering>
      <style name="V.POIVMK" value_columns="FEATURE_CODE">
        <substyle name="V.POIVBKT" value_columns="POINT_ID" changes="FILL_COLOR"/>
      </style>
    </rendering>
  </rule>
</styling_rules>
```

For more information about using the <rendering> element to apply multiple rendering styles in a single styling rule, see Section 2.3.1.4.

The <hidden_info> element specifies the list of attributes from the base table to be displayed when the user moves the mouse over the theme's features. The attributes are specified by a list of <field> elements.

Each <field> element must have a column attribute, which specifies the name of the column from the base table, and it can have a name attribute, which specifies the display name of the column. (The name attribute is useful if you want a text string other than the column name to be displayed.)

The <operations> element specifies the list of image processing operations to be applied on a GeoRaster theme. The operations are specified by a list of <operation> elements.

The <operation> element specifies the image processing operator and its parameters to be applied on a GeoRaster theme. Each <operation> element may have a list of <parameters> elements.

The <parameters> element defines a list of parameters to be used on a specific task. The parameters are specified by a list of <parameter> elements.

The <parameter> element must have the `name` and `value` attributes defined.

The <bitmap_masks> element defines the image mask attributes to be used with a GeoRaster theme. The bitmap masks are specified by a list of <mask> elements.

The <mask> element specifies a bitmap mask to be applied on a GeoRaster object. The `raster_id`, `raster_table`, and `layers` attributes must be defined, while the `zeromapping` and `onemapping` attributes are optional.

See Section 2.3.1.1 for more information about styling rules and for an example.

A.8 Base Maps

A base map definition consists of one or more themes. The XML definition of a base map is specified in the DEFINITION column of the USER_SDO_MAPS metadata view, using the following DTD:

```
<!ELEMENT map_definition (theme+)>

<!ELEMENT theme EMPTY>
<!ATTLIST theme name CDATA #REQUIRED
    name                  CDATA #REQUIRED
    datasource            CDATA #IMPLIED
    template_theme        CDATA #IMPLIED
    max_scale             CDATA #IMPLIED
    min_scale             CDATA #IMPLIED
    label_always_on       (TRUE|FALSE) "FALSE"
    fast_unpickle         (TRUE|FALSE) "TRUE"
    mode                  CDATA #IMPLIED
    min_dist              CDATA #IMPLIED
    fixed_svglabel        (TRUE|FALSE) "FALSE"
    visible_in_svg        (TRUE|FALSE) "TRUE"
    selectable_in_svg     (TRUE|FALSE) "FALSE"
    part_of_basemap       (TRUE|FALSE) "FALSE"
    simplify_shapes       (TRUE|FALSE) "TRUE"
    transparency          CDATA #IMPLIED
    minimum_pixels        CDATA #IMPLIED
    onclick               CDATA #IMPLIED
    onmousemove           CDATA #IMPLIED
    onmouseover           CDATA #IMPLIED
    onmouseout            CDATA #IMPLIED
    workspace_name        CDATA #IMPLIED
    workspace_savepoint   CDATA #IMPLIED
    workspace_date        CDATA #IMPLIED
    workspace_date_format CDATA #IMPLIED
    fetch_size            CDATA #IMPLIED
    timeout               CDATA #IMPLIED
>
```

The <map_definition> element contains one or more <theme> elements. Themes are rendered on a map on top of each other, in the order in which they are specified in the definition.

The <theme> element and its attributes are described in Section 3.2.20

See Section 2.4 for more information about defining base maps and for an example.

A.9 Map Tile Layers

An Oracle Maps map tile layer which assembles and displays pregenerated map image tiles from the map tile server, as described in Section 8.2.2.2. The XML configuration settings of a map tile layer is defined using the following DTD:

```
<!ELEMENT map_tile_layer ((internal_map_source|external_map_source), tile_storage,
coordinate_system, tile_image, tile_bound, zoom_levels)>
<!ATTLIST map_tile_layer
          name CDATA #REQUIRED
          image_format CDATA #IMPLIED>

<!ELEMENT internal_map_source EMPTY>
<!ATTLIST internal_map_source
          data_source CDATA #REQUIRED
          base_map CDATA #REQUIRED
          bgcolor CDATA #IMPLIED
          antialias (TRUE|FALSE) "TRUE">

<!ELEMENT external_map_source (properties?)>
<!ATTLIST external_map_source
          url CDATA #REQUIRED
          request_method CDATA #REQUIRED
          timeout CDATA #IMPLIED
          adapter_class CDATA #REQUIRED
          proxy_host CDATA #IMPLIED
          proxy_port CDATA #IMPLIED
          clipping_buffer CDATA #IMPLIED>

<!ELEMENT properties (property+) >

<!ELEMENT property EMPTY >
<!ATTLIST property
          name CDATA #REQUIRED
          value CDATA #REQUIRED>

<!ELEMENT tile_storage EMPTY >
<!ATTLIST tile_storage
          root_path CDATA #REQUIRED >

<!ELEMENT coordinate_system EMPTY >
<!ATTLIST coordinate_system
          srid CDATA #REQUIRED
          minX CDATA #REQUIRED
          minY CDATA #REQUIRED
          maxX CDATA #REQUIRED
          maxY CDATA #REQUIRED>

<!ELEMENT tile_bound (coordinates)>
<!ELEMENT coordinates (#PCDATA)>

<!ELEMENT tile_image EMPTY >
```

```
<!ATTLIST tile_image
          width CDATA #REQUIRED
          height CDATA #REQUIRED>

<!ELEMENT zoom_levels (zoom_level+)>
<!ATTLIST zoom_levels
          levels CDATA #REQUIRED
          min_scale CDATA #IMPLIED
          max_scale CDATA #IMPLIED
          min_tile_width CDATA #IMPLIED
          min_tile_height CDATA #IMPLIED>

<!ELEMENT zoom_level (tile_bound?)>
<!ATTLIST zoom_level
          level CDATA #REQUIRED
          level_name CDATA #IMPLIED
          description CDATA #IMPLIED
          scale CDATA #REQUIRED
          tile_width CDATA #REQUIRED
          tile_height CDATA #REQUIRED>
```

B

JavaScript Functions for SVG Maps

This appendix describes the MapViewer JavaScript application programming interface (API) for SVG maps. This API contains predefined functions that can be called from outside the SVG map, typically from the HTML document in which the SVG map is embedded. In addition, you can create JavaScript functions to be called when certain mouse-click actions occur. The predefined and user-defined functions can be used to implement sophisticated client-side interactive features, such as customized navigation.

If you use any of the JavaScript functions described in this appendix, end users must use Microsoft Internet Explorer to view the SVG maps, and Adobe SVG Viewer 3.0 or a later release must be installed on their systems.

This appendix contains the following major sections:

- Section B.1, "Navigation Control Functions"
- Section B.2, "Display Control Functions"
- Section B.3, "Mouse-Click Event Control Functions"
- Section B.4, "Other Control Functions"

B.1 Navigation Control Functions

The MapViewer JavaScript functions for controlling navigation include the following:

- `recenter(x, y)` sets the center point of the current SVG map.

  The input `x` and `y` values specify the coordinates (in pixels) of the new center point, which is the point inside the SVG map to be displayed at the center of the SVG viewer window. The SVG viewer window is the graphical area in the web browser displayed by the SVG viewer. The coordinates of the center point are defined in the SVG map screen coordinate system, which starts from (0, 0) at the upper-left corner of the map and ends at (width, height) at the lower-right corner.

- `setZoomRatio(zratio)` sets the current map display zoom ratio.

  This function can be used to zoom in or zoom out in the SVG map. (It does not change the center point of the map.) The original map zoom ratio without any zooming is 1, and higher zoom ratio values show the SVG map zoomed in. The map zoom ratio should be set to those values that fit predefined zoom levels. For example, if the `zoomlevels` value is 4 and `zoomfactor` value is 2, map zoom ratios at zoom level 0, 1, 2, and 3 will be 1, 2, 4, and 8, respectively; thus, in this example the `zratio` parameter value should be 1, 2, 4, or 8. For more information about predefined zoom levels, see the descriptions of the `zoomlevels`, `zoomfactor`, and `zoomratio` attributes in Section 3.2.1.1.

B.2 Display Control Functions

MapViewer provides functions to enable and disable the display of informational tips, the map legend, hidden themes, and the animated loading bar. The display control functions include the following:

- `switchInfoStatus()` enables or disables the display of informational tips. (Each call to the function reverses the previous setting.)

  You can control the initial display of informational tips by using the `<hidden_info>` element in theme styling rule definition (see Section A.7) and the `infoon` attribute in a map request (see Section 3.2.1.1). The `switchInfoStatus()` function toggles (reverses) the current setting for the display of informational tips.

- `switchLegendStatus()` enables or disables the display of the map legend. (Each call to the function reverses the previous setting.) The legend is initially hidden when the map is displayed.

- `showTheme(theme)` sets the specified theme to be visible on the map, and `hideTheme(theme)` sets the specified theme to be invisible on the map.

- `showLoadingBar()` displays the animated loading bar. The animated loading bar provides a visible indication that the loading of a new map is in progress. The bar is removed from the display when the loading is complete.

B.3 Mouse-Click Event Control Functions

MapViewer provides several predefined mouse-click event control functions, which are explained in Section B.3.1. You can also create user-defined mouse event control functions, as explained in Section B.3.2.

B.3.1 Predefined Mouse-Click Control Functions

MapViewer provides functions to enable and disable theme feature, rectangle, and polygon selection in SVG maps. It also provides functions to get information about selections and to toggle the selection status on and off. The functions for customizing mouse-click event control on an SVG map include the following:

- `enableFeatureSelect()` enables theme feature selection, and `disableFeatureSelect()` disables theme feature selection.

  Theme feature selection can be enabled if the `selectable_in_svg` attribute in the `<theme>` element is `TRUE` either in the map request (see Section 3.2.20) or in the base map (see Section A.8) definition. If the theme is selectable and theme feature selection is enabled, each feature of the theme displayed on the SVG map can be selected by clicking on it. If the feature is selected, its color is changed and its ID (rowid by default) is recorded. Clicking on an already selected feature deselects the feature. The list of IDs of all selected features can be obtained by calling the `getSelectedIdList()` function, described in this section.

  When theme feature selection is enabled, polygon selection and rectangle selection are automatically disabled.

- `enablePolygonSelect()` enables polygon selection, and `disablePolygonSelect()` disables polygon selection.

  If polygon selection is enabled, a polygon selection area can be defined by clicking and moving the mouse on the SVG map. Each click creates a shape point for the polygon. The coordinates of the polygon are recorded, and can be obtained by calling the `getSelectPolygon()` function, described in this section.

When polygon selection is enabled, theme feature selection and rectangle selection are automatically disabled.

- `enableRectangleSelect()` enables rectangle selection, and `disableRectangleSelect()` disables rectangle selection.

  If rectangle selection is enabled, a rectangular selection window can be defined by clicking and dragging the mouse on the SVG map. The coordinates of the rectangle are recorded, and can be obtained by calling the `getSelectRectangle()` function, described in this section.

  When rectangle selection is enabled, theme feature selection and polygon selection are automatically disabled.

- `getInfo(theme, key)` returns the informational note or tip string of the feature identified by theme name and key.
- `getSelectedIdList(theme)` returns an array of all feature IDs that are selected on the SVG map.
- `getSelectPolygon()` returns an array of the coordinates of all shape points of the selection polygon, using the coordinate system associated with the original user data.
- `getSelectRectangle()` returns an array of the coordinates of the upper-left corner and the lower-right corner of the selection rectangle, using the coordinate system associated with the original user data.
- `selectFeature(theme, key)` toggles the selection status of a feature (identified by its key value) in a specified theme.
- `setSelectPolygon(poly)` sets the coordinates of all shape points of the selection polygon, using the coordinate system associated with the original user data. The coordinates are stored in the array `poly`. Calling this function after `enablePolygonSelect()` draws a polygon on the SVG map.
- `setSelectRectangle(rect)` sets the coordinates of the upper-left corner and the lower-right corner of the selection rectangle, using the coordinate system associated with the original user data. The coordinates are stored in the array `rect`. Calling this function after `enableRectangleSelect()` draws a rectangle on the SVG map.

B.3.2 User-Defined Mouse Event Control Functions

User-defined JavaScript mouse-event control functions can be combined with predefined JavaScript functions (described in Section B.3.1) to implement further interactive customization. You can create map-level, theme-level, and selection event control functions.

B.3.2.1 Map-Level Functions

Map-level mouse event control functions can be defined for mouse-click events and mouse-move events.

A mouse-click event function is called whenever a click occurs anywhere in the SVG map, if both theme feature selection and window selection are disabled. The name of the function is defined by the `onclick` attribute in the map request (see Section 3.2.1.1).

A mouse-move event function is called whenever the mouse moves anywhere in the SVG map. The name of the function is defined by the `onmousemove` attribute in the map request (see Section 3.2.1.1).

These JavaScript functions must be defined in the web page that has the SVG map embedded. Mouse-click and mouse-move event functions must accept two parameters, `x` and `y`, which specify the coordinates inside the SVG viewer window where the mouse click or move occurred. The coordinate is defined in the local SVG viewer window coordinate system, which starts from (0,0) at the upper-left corner and ends at (width, height) at the lower-right corner.

B.3.2.2 Theme-Level Functions

Theme-level mouse event control functions can be defined for mouse-click, mouse-move, mouse-over, and mouse-out events.

A mouse-click event control function is called when theme feature selection is enabled and a feature of the theme is clicked. Each theme in the map can have its own mouse-click event control function. A theme-level mouse-click event control function is specified by the `onclick` attribute in the `<theme>` element in the map request or base map definition.

A mouse-move event control function is called whenever the mouse moves inside any feature of the theme. Each theme in the map can have its own mouse-move event control function. A theme-level mouse-move event control function is specified by the `onmousemove` attribute in the `<theme>` element in the map request or base map definition.

A mouse-over event control function is called whenever the mouse moves from outside a feature of the theme to inside a feature of the theme. Each theme in the map can have its own mouse-over event control function. A theme-level mouse-over event control function is specified by the `onmouseover` attribute in the `<theme>` element in the map request or base map definition.

A mouse-out event control function is called whenever the mouse moves out of a feature of the theme. Each theme in the map can have its own mouse-out event control function. A theme-level mouse-out event control function is specified by the `onmouseout` attribute in the `<theme>` element in the map request or base map definition.

These JavaScript functions must be defined in the web page that has the SVG map embedded. They take the following parameters:

- Theme name
- Key of the feature
- X-axis value of the point in the SVG viewer window where the mouse click occurred
- Y-axis value of the point in the SVG viewer window where the mouse click occurred

The key of the feature is the value of the key column from the base table, which is specified by the `key_column` attribute of the `<theme>` element in the map request or base map definition. ROWID is used as the default key column. For example, if the `onclick` attribute is set to `selectCounty` for the COUNTY theme, the following JavaScript function call is executed if the feature with rowid AAAHQDAABAAALk6Abm of the COUNTY theme is clicked on the SVG map at (100,120): `selectCounty('COUNTY', 'AAAHQDAABAAALk6Abm', 100, 120).`

The x-axis and y-axis values specify the coordinates inside the SVG viewer window where the mouse event occurred. The coordinate is defined in the local SVG viewer window coordinate system, which starts from (0,0) at the upper-left corner and ends at (width, height) at the lower-right corner.

B.3.2.3 Selection Event Control Functions

You can define a selection event control function for rectangle selection or polygon selection, or for both.

A rectangle selection event control function is called whenever rectangle selection is enabled and a rectangular selection area has been created by clicking and dragging the mouse (to indicate two diagonally opposite corners) on an SVG map. The function is called immediately after the selection of the rectangle is completed and the mouse key is released. The function name is specified with the `onrectselect` attribute in the map request (see Section 3.2.1.1).

A polygon selection event control function is called whenever polygon selection is enabled and a polygon-shaped selection area has been created by clicking and dragging the mouse at least four times on an SVG map, with the last click on the same point as the first click to complete the polygon. The function is called immediately after the selection of the polygon is completed. The function name is specified with the `onpolyselect` attribute in the map request (see Section 3.2.1.1).

B.4 Other Control Functions

MapViewer provides other useful functions for working with SVG maps. These functions include the following:

- `getUserCoordinate(x,y)` converts the screen coordinates into the original map data coordinates. This function returns the converted result in an array. The first element of the array is the converted X coordinate, and the second element of the array is the converted Y coordinate.
- `getScreenCoordinate(x,y)` converts the original map data coordinates into the screen coordinates. This function returns the converted result in an array. The first element of the array is the converted X coordinate, and the second element of the array is the converted Y coordinate.

C

Creating and Registering a Custom Image Renderer

This appendix explains how to implement and register a custom image renderer for use with an image theme. (Image themes are described in Section 2.3.3.)

If you want to create a map request specifying an image theme with an image format that is not supported by MapViewer, you must first implement and register a custom image renderer for that format. For example, the ECW format in Example 3–6 in Section 3.1.6 is not supported by MapViewer; therefore, for that example to work, you must first implement and register an image renderer for ECW format images.

The interface `oracle.sdovis.CustomImageRenderer` is defined in the package `sdovis.jar`, which is located in the *$ORACLE_HOME*`/lbs/lib` directory in an Oracle Fusion Middleware environment. If you performed a standalone installation of OC4J, `sdovis.jar` is unpacked into *$MAPVIEWER*`/web/WEB-INF/lib`. The following is the source code of this interface.

```
/**
 * An interface for a custom image painter that supports user-defined image
 * formats. An implementation of this interface can be registered with
 * MapViewer to support a custom image format.
 */
public interface CustomImageRenderer
{
  /**
   * The method is called by MapViewer to find out the image format
   * supported by this renderer. <br>
   * This format string must match the one specified in a custom image renderer
   * element defined in the configuration file (mapViewerConfig.xml).
   */
  public String  getSupportedFormat() ;

  /**
   * Renders the given images. MapViewer calls this method
   * to tell the implementor the images to render, the current map
   * window in user space, and the MBR (in the same user space) for each
   * image.
   * <br>
   * The implementation should not retain any reference to the parameters
   * permanently.
   * @param g2  the graphics context to draw the images onto.
   * @param images  an array of image data stored in byte array.
   * @param mbrs an array of double[4] arrays containing one MBR for each
   *        image in the images array.
   * @param dataWindow the data space window covered by the current map.
   * @param deviceView the device size and offset.
```

```
   * @param at  the AffineTransform using which you can transform a point
   *           in the user data space to the device coordinate space. You can
   *           ignore this parameter if you opt to do the transformation
   *           yourself based on the dataWindow and deviceView information.
   * @param scaleImage a flag passed from MapViewer to indicate whether
   *            the images should be scaled to fit the current device window.
   *            If it is set to false, render the image as-is without
   *            scaling it.
   */
  public void   renderImages(Graphics2D g2, byte[][] images, double[][] mbrs,
                             Rectangle2D dataWindow, Rectangle2D deviceView,
                             AffineTransform at, boolean scaleImage) ;
}
```

After you implement this interface, you must place your implementation class in a directory that is part of the MapViewer CLASSPATH definition, such as the `$MAPVIEWER/web/WEB-INF/lib` directory. If you use any native libraries to perform the actual rendering, you must ensure that any other required files (such as `.dll` and `.so` files) for these libraries are accessible to the Java virtual machine (JVM) that is running MapViewer.

After you place your custom implementation classes and any required libraries in the MapViewer CLASSPATH, you must register your class with MapViewer in its configuration file, `mapViewerConfig.xml` (described in Section 1.5.2). Examine, and edit as appropriate, the following section of the file, which tells MapViewer which class to load if it encounters a specific image format that it does not already support.

```
  <!-- ****************************************************************** -->
  <!-- ******************* Custom Image Renderers ********************** -->
  <!-- ****************************************************************** -->
  <!-- Uncomment and add as many custom image renderers as needed here,
       each in its own  <custom_image_renderer> element. The "image_format"
       attribute specifies the format of images that are to be custom
       rendered using the class with the full name specified in "impl_class".
       You are responsible for placing the implementation classes in the
       MapViewer classpath.
  -->
  <!--
  <custom_image_renderer image_format="ECW"
                         impl_class="com.my_corp.image.ECWRenderer"/>
  -->
```

In this example, for any ECW formatted image data loaded through the <jdbc_image_query> element of an image theme, MapViewer will load the class `com.my_corp.image.ECWRenderer` to perform the rendering.

Example C–1 is an example implementation of the `oracle.sdovis.CustomImageRenderer` interface. This example implements a custom renderer for the ECW image format. Note that this example is for illustration purposes only, and the code shown is not necessarily optimal or even correct for all system environments. This implementation uses the ECW Java SDK, which in turn uses a native C library that comes with it. For MapViewer to be able to locate the native dynamic library, you may need to use the command-line option `-Djava.library.path` when starting the OC4J instance that contains MapViewer.

Example C–1 Custom Image Renderer for ECW Image Format

```
package com.my_corp.image;
import java.io.*;
import java.util.Random;
```

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.BufferedImage;

import oracle.sdovis.CustomImageRenderer;
import com.ermapper.ecw.JNCSFile; // from ECW Java SDK

public class ECWRenderer implements CustomImageRenderer
{
  String tempDir = null;
  Random random = null;

  public ECWRenderer()
  {
    tempDir = System.getProperty("java.io.tmpdir");
    random = new Random(System.currentTimeMillis());
  }

  public String  getSupportedFormat()
  {
    return "ECW";
  }

  public void   renderImages(Graphics2D g2, byte[][] images,
                             double[][] mbrs,
                             Rectangle2D dataWindow,
                             Rectangle2D deviceView,
                             AffineTransform at)
  {
    // Taking the easy way here; you should try to stitch the images
    // together here.
    for(int i=0; i<images.length; i++)
    {
      String tempFile = writeECWToFile(images[i]);
      paintECWFile(tempFile, g2, mbrs[i], dataWindow, deviceView,at);
    }
  }

  private String writeECWToFile(byte[] image)
  {
    long l = Math.abs(random.nextLong());
    String file = tempDir + "ecw"+l+".ecw";
    try{
      FileOutputStream fos = new FileOutputStream(file);
      fos.write(image);
      fos.close();
      return file;
    }catch(Exception e)
    {
      System.err.println("cannot write ecw bytes to temp file: "+file);
      return null;
    }
  }

  private void  paintECWFile(String fileName, Graphics2D g,
                             double[] mbr,
                             Rectangle2D dataWindow,
                             Rectangle2D deviceView,
                             AffineTransform at)
  {
```

```
JNCSFile ecwFile = null;
boolean bErrorOnOpen = false;
BufferedImage ecwImage = null;
String errorMessage = null;

try {
  double dFileAspect, dWindowAspect;
  double dWorldTLX, dWorldTLY, dWorldBRX, dWorldBRY;
  int bandlist[];
  int width = (int)deviceView.getWidth(),
      height = (int)deviceView.getHeight();
  int line, pRGBArray[] = null;

  ecwFile = new JNCSFile(fileName, false);

  // Work out the correct aspect for the setView call.
  dFileAspect = (double)ecwFile.width/(double)ecwFile.height;
  dWindowAspect = deviceView.getWidth()/deviceView.getHeight();

  if (dFileAspect > dWindowAspect) {
    height =(int)((double)width/dFileAspect);
  } else {
    width = (int)((double)height*dFileAspect);
  }

  // Create an image of the ecw file.
  ecwImage = new BufferedImage(width, height,
                               BufferedImage.TYPE_INT_RGB);
  pRGBArray = new int[width];

  // Set up the view parameters for the ecw file.
  bandlist = new int[ecwFile.numBands];
  for (int i=0; i< ecwFile.numBands; i++) {
    bandlist[i] = i;
  }
  dWorldTLX = ecwFile.originX;
  dWorldTLY = ecwFile.originY;
  dWorldBRX = ecwFile.originX +
            (double)(ecwFile.width-1)*ecwFile.cellIncrementX;
  dWorldBRY = ecwFile.originY +
            (double)(ecwFile.height-1)*ecwFile.cellIncrementY;

  dWorldTLX = Math.max(dWorldTLX, dataWindow.getMinX());
  dWorldTLY = Math.max(dWorldTLY, dataWindow.getMinY());
  dWorldBRX = Math.min(dWorldBRX, dataWindow.getMaxX());
  dWorldBRY = Math.min(dWorldBRY, dataWindow.getMaxY());

  // Set the view.
  ecwFile.setView(ecwFile.numBands, bandlist, dWorldTLX,
                dWorldTLY, dWorldBRX, dWorldBRY, width, height);

  // Read the scan lines.
  for (line=0; line < height; line++) {
    ecwFile.readLineRGBA(pRGBArray);
    ecwImage.setRGB(0, line, width, 1, pRGBArray, 0, width);
  }

} catch(Exception e) {
  e.printStackTrace(System.err);
  bErrorOnOpen = true;
```

```
        errorMessage = e.getMessage();
        g.drawString(errorMessage, 0, 50);
      }

      // Draw the image (unscaled) to the graphics context.
      if (!bErrorOnOpen) {
        g.drawImage(ecwImage, 0, 0, null);
      }

    }
  }
```

D

Creating and Registering a Custom Spatial Data Provider

This appendix shows a sample implementation of a spatial data provider, and explains how to register this provider to be used with MapViewer. The complete implementation can be found under the MapViewer `web/demo/spatialprovider` directory. The implementation uses then following files:

- `us_bigcities.xml`: sample XML file that the provider parses
- `customSpatialProviderSample.java`: Java implementation of the spatial data provider
- `spatialprovider.jar`: jar file with the compiled version of the customSpatialProviderSample.java source file

The `us_bigcities.xml` file has sections to define the data attributes, the data extents, and the feature information, including the geometry (in GML format) and the attribute values. This file includes the following:

```
<?xml version="1.0" standalone="yes"?>
<spatial_data>

<data_attributes>
  <attribute name="city" type="string" />
  <attribute name="state_abrv" type="string" />
  <attribute name="pop90" type="double" />
</data_attributes>

<data_extents>
   <xmin> -122.49586 </xmin>
   <ymin> 29.45765 </ymin>
   <xmax> -73.943849 </xmax>
   <ymax> 42.3831 </ymax>
</data_extents>

<geoFeature>
  <attributes> New York,NY,7322564 </attributes>
  <geometricProperty>
    <Point>
      <coordinates>-73.943849, 40.6698</coordinates>
    </Point>
   </geometricProperty>
 </geoFeature>

. . .
</spatial_data>
```

This appendix contains the following major sections:

- Section D.1, "Implementing the Spatial Provider Class"
- Section D.2, "Registering the Spatial Provider with MapViewer"
- Section D.3, "Rendering the External Spatial Data"

D.1 Implementing the Spatial Provider Class

The provider must implement the class interface shown in Section 2.3.9. Example D–1 contains the partial code for the spatial provider in the supplied demo. Note that this sample code is deliberately simplified; it is not optimized, and the provider does not create any spatial indexing mechanism.

Example D–1 Implementing the Spatial Provider Class

```
package spatialprovider.samples;

import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.io.File;
import java.util.ArrayList;
import java.util.Properties;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import oracle.mapviewer.share.Field;
import oracle.mapviewer.share.ext.SDataProvider;
import oracle.mapviewer.share.ext.SDataSet;
import oracle.mapviewer.share.ext.SObject;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import oracle.spatial.geometry.JGeometry;
import oracle.spatial.util.GML;

public class CustomSpatialProviderSample implements SDataProvider
{
 ...

 /**
  * Constructor.
  */
 public CustomSpatialProviderSample()
 {
   ...
 }

 /**
  * Returns the initialization parameters for the provider.
  * The "datadir" parameter should be registered on MapViewer
  * configuration file and can be used to access the data.
  * @return
  */
 public String[] getInitParameterNames()
 {
   return new String[]{ "datadir" };
```

```
}

/**
 * Returns runtime parameter names. Runtime parameters are additional parameters
 * that the provider may use when retrieving the data objects.
 * @return
 */
public String[] getRuntimeParameterNames()
{
  return new String[]{ "filename" };
}

/**
 * Initializes the provider
 * @param params  init properties
 * @return
 */
public boolean init(Properties params)
{
  dataDirectory = null;
    if(params == null)
    return true;
    dataDirectory = params.getProperty("datadir");
  if(dataDirectory == null || dataDirectory.trim().length() == 0)
  {
    // try upper case
    dataDirectory = params.getProperty("DATADIR");
    if(dataDirectory == null || dataDirectory.trim().length() == 0)
      System.out.println("FINE: Init properties does not define \"datadir\" parameter.");
  }
      return true;
}

/**
 * Returns the data set (geometries plus attributes) that intersects the
 * query window. In this sample the data is parsed just once and
 * there is no spatial index for searching. The search is sequential.
 * @param queryWin  search area
 * @param nonSpatialColumns   attribute columns
 * @param params      runtime properties
 * @return
 */
public SDataSet buildDataSet(Rectangle2D queryWin,
                             String []nonSpatialColumns,
                             Properties params)
{
  if(!dataParsed)
  {
    dataParsed = parseData(params);
    if(!dataParsed)
      return null;
  }
    if(geometries.size() == 0)
    return null;

  SDataSet dataset = new SDataSet();
    boolean fullExtent = isFullExtent(queryWin);
    if(fullExtent)
  {
    for(int i=0;i<geometries.size();i++)
```

```
    {
      JGeometry geom = (JGeometry)geometries.get(i);
      SObject obj = new SObject(geom,getGeometryAttributes(nonSpatialColumns,i));
      dataset.addObject(obj);      }
  }
  else
  {
    for(int i=0;i<geometries.size();i++)
    {
      JGeometry geom = (JGeometry)geometries.get(i);
      double []mbr = geom.getMBR();
      if(mbr == null)
        continue;
      Rectangle2D.Double rect = new Rectangle2D.Double(mbr[0],mbr[1],
                              mbr[2]-mbr[0],
                              mbr[3]-mbr[1]);
            if(rect.getWidth() == 0. && rect.getHeight() == 0.)
      {
        Point2D.Double pt = new Point2D.Double(mbr[0],mbr[1]);
        if(queryWin.contains(pt))
        {
          SObject obj = new SObject(geom,getGeometryAttributes(nonSpatialColumns,i));
          dataset.addObject(obj);                    }
      }
      else if(queryWin.contains(rect) || queryWin.intersects(rect))
      {
        SObject obj = new SObject(geom,getGeometryAttributes(nonSpatialColumns,i));
        dataset.addObject(obj);
      }
    }        }
    if(dataset.getSize() == 0)
    return null;
      return dataset;
}

/**
 * Returns the data provider attribute list.
 * @return
 */
public Field[] getAttributeList(Properties params)
{
  if(!dataParsed)
  {
    dataParsed = parseData(params);
    if(!dataParsed)
      return null;
  }
    if(attributes.size() == 0)
    return null;

  return (Field[])attributes.toArray(new Field[attributes.size()]);
}

/**
 * Returns the data extents.
 * @return
 */
public Rectangle2D getDataExtents(Properties params)
{
  if(!dataParsed)
```

```
    {
      dataParsed = parseData(params);
      if(!dataParsed)
        return null;
    }
      if(extents == null || extents.length < 4)
      return null;
    else
      return new Rectangle2D.Double(extents[0],extents[1],
                                    extents[2]-extents[0],
                                    extents[3]-extents[1]);
  }

  /**
   * Builds a spatial index for the data. In this sample there is no
   * spatial index. The data is loaded into memory when data is parsed.
   * @return
   */
  public boolean buildSpatialIndex(Properties params)
  {
    return true;
  }

}
```

After you have implemented the provider code, compile it and generate a jar file with this compiled class. The file `spatialprovider.jar` in the demo directory contains the compiled version of this sample code, and you can use it directly. Copy this jar file to a directory that is part of MapViewer's CLASSPATH definition, such as the `web/WB-INF/lib` directory.

D.2 Registering the Spatial Provider with MapViewer

To register the spatial provider with MapViewer, add the following in the spatial provider section of the MapViewer configuration file, and then restart MapViewer:

```
<s_data_provider
  id="xmlProvider"
  class="spatialprovider.samples.CustomSpatialProviderSample"
  >
  <parameters>
    <parameter name="datadir" value="/temp/data" />
  </parameters>
</s_data_provider>
```

When you restart MapViewer, you should see a console message that the spatial provider has been registered. For example:

```
2007-10-01 14:30:31.109 NOTIFICATION Spatial Provider xmlProvider has been
registered.
```

D.3 Rendering the External Spatial Data

To enable you to render the sample external spatial data that comes with MapViewer kit., create a data source pointing to this data Example D–2 is an XML request that contains a dynamic custom geometry theme.

Example D–2 Map Request to Render External Spatial Data

```
<?xml version="1.0" standalone="yes"?>
<map_request
 title="Custom Geometry Theme"
 datasource="mvdemo"
 width="500"
 height="400"
 bgcolor="#a6caf0"
 antialiase="true"
 format="PNG_STREAM"
>
 <center size="40">
   <geoFeature>
     <geometricProperty typeName="center">
       <Point>
         <coordinates>-90,32</coordinates>
       </Point>
     </geometricProperty>
   </geoFeature>
 </center>

 <themes>
   <theme name="custom_theme" >
      <custom_geom_theme
        provider_id="xmlProvider"
        srid="8307"
        render_style="M.CIRCLE"
        label_column="city"
        label_style="T.CITY NAME"
        datasource="mvdemo">
      <parameters>
       <parameter name="filename" value="/lbs/demo/spatialprovider/us_bigcities.xml"/>
      </parameters>
     </custom_geom_theme>
   </theme>
 </themes>
</map_request>
```

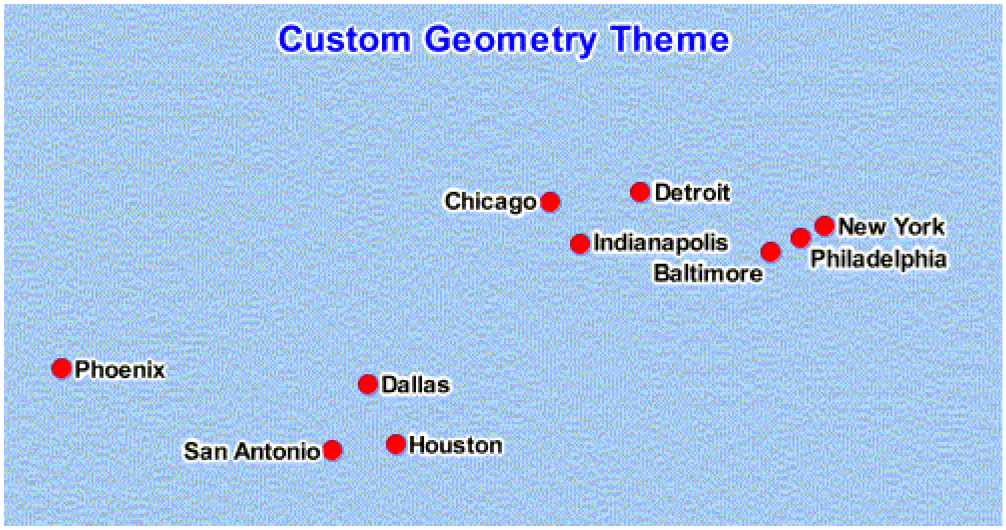
In Example D–2, the file name in the `<parameter>` element points to `/lbs/demo/spatialprovider/us_bigcities.xml`. If the file path is not accessible to MapViewer, the map request may generate error messages in the log file, such as the following:

```
07/09/28 10:26:47 ParseData: Can not access file: /lbs/demo/spatialprovider/us_
bigcities.xml
07/09/28 10:26:47 ParseData: File to be parsed: /temp/data\us_bigcities.xml
07/09/28 10:26:47 ParseData: File can not be accessed on provider data directory.
Copy files there.
```

When MapViewer searches for the file, it first tries to access the file using the original theme definition parameter; and if that fails, it tries the data directory defined in the MapViewer configuration file (`/temp/data` in the preceding example error messages). Therefore, if the original theme definition data path is not accessible to MapViewer, copy the data files to the directory defined in the configuration file before you issue the map request.

If MapViewer can find the data file, the map request inExample D–2 should generate an image like the one in Figure D–1.

Figure D–1 Map Image Using Custom Geometry Theme and External Spatial Data



E

OGC WMS Support in MapViewer

MapViewer supports the rendering of data delivered using the Open GIS Consortium (OGC) Web Map Service (WMS) protocol, specifically the WMS 1.1.1 and 1.3.0 implementation specifications. MapViewer supports the GetMap, GetFeatureInfo, and GetCapabilities requests as defined in the OGC document 01-068r3 and 06-042.

MapViewer does not currently support the optional Styled Layer Descriptor capability, and MapViewer will not function as a Cascading Map Server in this release.

This appendix contains the following major sections:

- Section E.1, "Setting Up the WMS Interface for MapViewer"
- Section E.2, "WMS Specification and Corresponding MapViewer Concepts"
- Section E.3, "Adding a WMS Map Theme"

E.1 Setting Up the WMS Interface for MapViewer

MapViewer is preconfigured to run as a WMS service. Internally, MapViewer translates all incoming WMS requests into proper XML requests to the MapViewer server. For example, the following HTTP request invokes the GetCapabilities service of a MapViewer server:

```
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&SERVICE=WMS&VERSION=1.1.1
or
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&SERVICE=WMS&VERSION=1.3.0
```

As shown in this example, the URL for the MapViewer WMS service is typically `http://`*`host:port`*`/mapviewer/wms?`, where *`host`* and *`port`* refer to the host and HTTP port of the MapViewer server. The context path `/mapviewer/wms` refers to the WMS interface of MapViewer.

Note: All WMS requests must be on a single line, so ignore any line breaks that might appear in WMS request examples in this chapter.

E.1.1 Required Files

The following files are required for MapViewer WMS support: `WMSFilter.jar` and `classgen.jar`.

- The servlet filter and its required classes are packaged in `WMSFilter.jar`. This should be located in the *`$MAPVIEWER_HOME`*`/web/WEB-INF/lib` directory.
- The servlet filter also requires `classgen.jar`, which is part of the XML Developer's Kit (XDK) for Java. A standalone OC4J installation usually does not

have this file; however, an Oracle Database or full Oracle Fusion Middleware installation will already have this file.

If your system does not already have the `classgen.jar` file, use a `classgen.jar` file from the same XDK for Java version as the one that ships with your standalone OC4J version. Place this file in the *$MAPVIEWER_HOME*`/web/WEB-INF/lib` directory or in a directory that is in the library path for OC4J.

The `classgen.jar` and `xmlparserv2.jar` files must be from the same XDK release, because the `classgen.jar` file depends on the `xmlparserv2.jar` file. Also, the XDK release for both files must be OC4J 10.0.0.3 or later, and preferably 10.1.2 or later.

E.1.2 Data Source Named wms

You must define a MapViewer data source named `wms`, unless every incoming WMS request explicitly specifies a `datasource` CGI parameter. All requests that do not specify the `datasource` parameter are by default directed to the data source named `wms`. For example, the `GetCapabilities` request will by default list all the available themes that are in the `wms` data source. (To configure the information returned by a GetCapabilities request, see Section 1.5.2.12.)

E.1.3 SDO to EPSG SRID Mapping File

By default, MapViewer uses the Oracle Spatial and Graph (SDO) native SRID (spatial reference ID) values when such information is requested in a WMS request such as GetCapabilities. The EPSG SRID values, however, are more widely used in WMS applications. To have MapViewer use EPSG SRID values when processing WMS requests and generating responses, specify a mapping file. This mapping file is a text file that tells MapViewer which SDO SRID values map to which EPSG SRID values. (Each pair of matching SRID values refers to the same spatial reference system.)

The mapping file contains lines where each line defines one pair of equivalent SRID values in the following format:

sdo_srid=epsg_srid

For example, the following lines define SDO SRID 8307 as equivalent to EPSG SRID 4326, and SDO SRID 81922 as equivalent to EPSG SRID 20248:

```
8307=4326
81922=20248
```

After you have created an SDO to EPSG mapping file, you can save it on the server where MapViewer is running, and specify its location in the MapViewer configuration file using the <sdo_epsg_mapfile> element in the <wms_config> element, as explained in Section 1.5.2.12.

E.2 WMS Specification and Corresponding MapViewer Concepts

This section describes the association between, or interpretation of, terms and concepts used in the WMS 1.1.1 and 1.3.0 specifications and MapViewer. It also includes some parameters that are specific to MapViewer but that are not in the WMS 1.1.1 and 1.3.0 specifications.

E.2.1 Supported GetMap Request Parameters

This section describes the supported GetMap request parameters and their interpretation by MapViewer. (Parameters that are specific to MapViewer and not mentioned in the WMS 1.1.1 and 1.3.0 specifications are labeled MapViewer-Only.) The supported parameters are in alphabetical order, with each in a separate subsection. Example E–1 shows some GetMap requests. (Each URL should be entered as a single string.)

Example E–1 GetMap Requests

```
http://localhost:8888/mapviewer/wms?REQUEST=GetMap&VERSION=1.1.1&FORMAT=image/gif&
SERVICE=WMS&BBOX=-121,37,-119,35&SRS=EPSG:4326&LAYERS=theme_demo_states,theme_
demo_counties,theme_demo_highways,theme_demo_cities&WIDTH=580&HEIGHT=500

http://localhost:8888/mapviewer/wms?REQUEST=GetMap&VERSION=1.3.0&FORMAT=image/gif&
SERVICE=WMS&BBOX=-121,37,-119,35&CRS=EPSG:4326&LAYERS=theme_demo_states,theme_
demo_counties,theme_demo_highways,theme_demo_cities&WIDTH=580&HEIGHT=500

http://localhost:8888/mapviewer/wms?request=GetMap&version=1.3.0&crs=none
&bbox=-122,36,-121,37&width=600&height=400&format=image/png&layers=theme_us_
states&mvthemes=<themes><theme%20name="theme_us_counties"/><theme%20name="theme_
us_road1"/></themes>&legend_
request=<legend%20bgstyle="fill:%23ffffff;stroke:%23ff0000"%20profile="medium"%20p
osition="SOUTH_EAST"><column><entry%20style="v.rb1"%20tab="1"/></column></legend>&
```

The default data source for a GetMap request is `WMS`. That is, if you do not specify the `DATASOURCE` parameter in a GetMap request, it is assumed that a data source named `WMS` was previously created using the <add_data_source> element (described in Section 7.1.1) in a MapViewer administrative request.

The following optional GetMap parameters are not supported in the current release of MapViewer:

- `TIME` (time dimension)
- `ELEVATION` (elevation dimension)
- `SLD` and `WFS` URLs

The MapViewer-only parameters must contain valid XML fragments. Because these are supplied in an HTTP GET request, they must be appropriately encoded using a URL encoding mechanism. For example, replace each space ( ) with `%20` and each pound sign (#) with `%23`. The following example shows the use of such encoding:

```
http://localhost:8888/mapviewer/wms?request=GetMap&version=1.1.1&srs=none&bbox=-12
2,36,-121,37&width=600&height=400&format=image/png&layers=theme_us_
states&mvthemes=<themes><theme%20name="theme_us_counties"/><theme%20name="theme_
us_road1"/></themes>&legend_
request=<legend%20bgstyle="fill:%23ffffff;stroke:%23ff0000"%20profile="medium"%20p
osition="SOUTH_EAST"><column><entry%20style="v.rb1"%20tab="1"/></column></legend>&
```

E.2.1.1 BASEMAP Parameter (MapViewer-Only)

The `BASEMAP` parameter specifies a named base map for the specified (or default) data source. If you specify both the `BASEMAP` and `LAYERS` parameters, all themes specified in the `LAYERS` parameters are added to the base map. Therefore, if you just want to get a map using a named base map, specify the `BASEMAP` parameter but specify an empty `LAYERS` parameter, as in the following examples:

```
REQUEST=GetMap&VERSION=1.1.1&BASEMAP=demo_
map&LAYERS=&WIDTH=500&HEIGHT=560&SRS=SDO:8307&BBOX=-122,36,-120,38.5&FORMAT=image/
```

```
png

REQUEST=GetMap&VERSION=1.3.0&BASEMAP=demo_
map&LAYERS=&WIDTH=500&HEIGHT=560&CRS=SDO:8307&BBOX=-122,36,-120,38.5&FORMAT=image/
png
```

E.2.1.2 BBOX Parameter

The BBOX parameter specifies the lower-left and upper-right coordinates of the bounding box for the data from the data source to be displayed. It has the format BBOX=minX,minY,maxX,maxY. For example: BBOX=-122,36,-120,38.5

E.2.1.3 BGCOLOR Parameter

The BGCOLOR parameter specifies background color for the map display using the RBG color value. It has the format 0xHHHHHH (where each H is a hexadecimal value from 0 to F). For example: BGCOLOR=0xF5F5DC (beige).

E.2.1.4 DATASOURCE Parameter (MapViewer-Only)

The DATASOURCE parameter specifies the name of the data source for the GetMap or GetFeatureInfo request. The default value is WMS. The specified data source must exist prior to the GetMap or GetFeatureInfo request. That is, it must have been created using the <add_data_source> MapViewer administrative request or defined in the MapViewer configuration file (mapViewerConfig.xml).

E.2.1.5 DYNAMIC_STYLES Parameter (MapViewer-Only)

The DYNAMIC_STYLES parameter specifies a <styles> element as part of the GetMap request. For information about the <styles> element, see Section 3.2.19.

E.2.1.6 EXCEPTIONS Parameter

For the EXCEPTIONS parameter, the only supported value is the default: EXCEPTIONS=application/vnd.ogc.se_xml for WMS 1.1.1 and EXCEPTIONS=XML for WMS 1.3.0. The exception is reported as an XML document conforming to the Service Exception DTD available at the following URLs:

http://schemas.opengis.net/wms/1.1.1/WMS_exception_1_1_1.dtd

http://schemas.opengis.net/wms/1.3.0/exceptions_1_3_0.xsd

The application/vnd.ogc.se_inimage (image overwritten with Exception message), and application/vnd.ogc.se_blank (blank image because Exception occurred) options are not supported.

E.2.1.7 FORMAT Parameter

The FORMAT parameter specifies the image format. The supported values are image/gif, image/jpeg, image/png, image/png8, and image/svg+xml.

The default value is image/png.

E.2.1.8 HEIGHT Parameter

The HEIGHT parameter specifies the height for the displayed map in pixels.

E.2.1.9 LAYERS Parameter

The LAYERS parameter specifies a comma-delimited list of predefined theme names to be used for the display. The specified values are considered to a be a case-sensitive,

ordered, comma-delimited list of predefined theme names in a default data source (named WMS) or in a named data source specified by the parameter DATASOURCE=<name>. For example, LAYERS=THEME_DEMO_STATES,theme_demo_counties,THEME_demo_HIGHWAYS translates to the following <themes> element in a MapViewer map request:

```
<themes>
<theme name="THEME_DEMO_STATES"/>
<theme name="theme_demo_counties"/>
<theme name="THEME_demo_HIGHWAYS"/>
</themes>
```

If you want to specify both a base map and one or more LAYERS values, see the information about the BASEMAP parameter in Section E.2.1.1.

E.2.1.10 LEGEND_REQUEST Parameter (MapViewer-Only)

The LEGEND_REQUEST parameter specifies a <legend> element as part of the GetMap request. For information about the <legend> element, see Section 3.2.11.

E.2.1.11 MVTHEMES Parameter (MapViewer-Only)

The MVTHEMES parameter specifies a <themes> element as part of the GetMap request. For information about the <themes> element, see Section 3.2.21. The primary purpose for the MVTHEMES parameter is to support JDBC themes in a MapViewer request. The MVTHEMES parameter is not a substitute or synonym for the LAYERS parameter; you still must specify the LAYERS parameter.

E.2.1.12 REQUEST Parameter

The REQUEST parameter specifies the type of request. The value must be GetMap, GetFeatureInfo, or GetCapabilities.

E.2.1.13 SERVICE Parameter

The SERVICE parameter specifies the service name. The value must be WMS.

E.2.1.14 SRS (1.1.1) or CRS (1.3.0) Parameter

The SRS parameter (WMS 1.1.1) or the CRS parameter (WMS 1.3.0) specifies the spatial reference system (coordinate system) for MapViewer to use. The value must be one of the following: SDO:srid-value (where srid-value is a numeric Oracle Spatial and Graph SRID value), EPSG:4326 (equivalent to SDO:8307), or none (equivalent to SDO:0).

Except for EPSG:4326 (the standard WGS 84 longitude/latitude coordinate system), EPSG numeric identifiers are not supported. The namespace AUTO (WMS 1.1.1) or AUTO2 (WMS 1.3.0), for projections that have an arbitrary center of projection, is not supported.

E.2.1.15 STYLES Parameter

The STYLES parameter is ignored. Instead, use the LAYERS parameter to specify predefined themes for the display.

E.2.1.16 TRANSPARENT Parameter

The TRANSPARENT=TRUE parameter (for a transparent image) is supported for PNG images, that is, with FORMAT=image/png, or FORMAT=image/png8 for indexed (8-bit) PNG format. MapViewer does not support transparent GIF (GIF89) images.

E.2.1.17 VERSION Parameter

The VERSION parameter specifies the WMS version number. The value must be 1.1.1 or 1.3.0.

E.2.1.18 WIDTH Parameter

The WIDTH parameter specifies the width for the displayed map in pixels.

E.2.2 Supported GetCapabilities Request and Response Features

A WMS GetCapabilities request to MapViewer should specify only the following parameters:

- REQUEST=GetCapabilities
- VERSION=1.1.1 or VERSION=1.3.0
- SERVICE=WMS

For example:

```
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&VERSION=1.1.1&SERVICE=
WMS
or
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&VERSION=1.3.0&SERVICE=
WMS
```

The response is an XML document conforming to the WMS Capabilities DTD available at the following, depending on the value of the VERSION parameter (1.1.1 or 1.3.0):

http://schemas.opengis.net/wms/1.1.1/WMS_MS_Capabilities.dtd

http://schemas.opengis.net/wms/1.3.0/capabilities_1_3_0.xsd

However, the current release of MapViewer returns an XML document containing the <Service> and <Capability> elements with the following information:

- The <Service> element is mostly empty, with just the required value of OGC:WMS for the <Service.Name> element. Support for more informative service metadata is planned for a future release of MapViewer.
- The <Capability> element has <Request>, <Exception>, and <Layer> elements.
- The <Request> element contains the GetCapabilities and GetMap elements that describe the supported formats and URL for an HTTP GET or POST operation.
- The <Exception> element defines the exception format. The Service Exception XML is the only supported format in this release. The <Exception> element returns an XML document compliant with the Service Exception DTD, but it does not report exceptions as specified in the implementation specification. The current release simply uses the CDATA section of a <ServiceException> element to return the OMSException returned by the MapViewer server.
- The <Layer> element contains a nested set of <Layer> elements. The first (outermost) layer contains a name (WMS), a title (Oracle WebMapServer Layers by data source), and one <Layer> element for each defined data source. Each data source layer contains a <Layer> element for each defined base map and one entry for each valid theme (layer) not listed in any base map. Each base map layer contains a <Layer> element for each predefined theme in the base map.

Themes that are defined in the USER_SDO_THEMES view, that have valid entries in the USER_SDO_GEOM_METADATA view for the base table and geometry column, and that are not used in any base map will be listed after the base maps for a data source. These themes will have no `<ScaleHint>` element. They will have their own `<LatLonBoundingBox>` and `<BoundingBox>` elements.

The Content-Type of the response is set to `application/vnd.ogc.wms_xml`, as required by the WMS implementation specification.

Because the list of layers is output by base map, a given layer or theme can appear multiple times in the GetCapabilities response. For example, the theme `THEME_DEMO_STATES`, which is part of the base maps named `DEMO_MAP` and `DENSITY_MAP`, appears twice in Example E–2, which is an excerpt (reformatted for readability) from a GetCapabilities response.

Example E–2 GetCapabilities Response (Excerpt)

```
<Title>Oracle WebMapServer Layers by data source</Title>
<Layer>
 <Name>mvdemo</Name>
 <Title>Datasource mvdemo</Title>
   <Layer>
     <Name>DEMO_MAP</Name>
     <Title>Basemap DEMO_MAP</Title>
     <SRS>SDO:8307</SRS>
     <LatLonBoundingBox>-180,-90,180,90</LatLonBoundingBox>
. . .
   <Layer>
     <Name>DENSITY_MAP</Name>
     <Title>Basemap DENSITY_MAP</Title>
     <SRS>SDO:8307</SRS>
     <LatLonBoundingBox>-180,-90,180,90</LatLonBoundingBox>
       <Layer>
         <Name>THEME_DEMO_STATES</Name>
         <Title>THEME_DEMO_STATES</Title>
         <SRS>SDO:8307</SRS>
         <BoundingBox SRS="SDO:8307" minx="-180" miny="-90" maxx="180"
                      maxy="90" resx="0.5" resy="0.5"/>
         <ScaleHint min="50.0" max="4.0"/>
       </Layer>
. . .
   </Layer>
   <Layer>
     <Name>IMAGE_MAP</Name>
     <Title>Basemap IMAGE_MAP</Title>
     <SRS>SDO:41052</SRS>
     <LatLonBoundingBox>-180,-90,180,90</ LatLonBoundingBox>
       <Layer>
         <Name>IMAGE_LEVEL_2</Name>
         <Title>IMAGE_LEVEL_2</Title>
         <SRS>SDO:41052</SRS>
         <BoundingBox SRS="SDO:41052" minx="200000" miny="500000" maxx="750000"
                    maxy="950000" resx="0.5" resy="0.5"/>
         <ScaleHint min="1000.0" max="0.0"/>
       </Layer>
. . .
   </Layer>
```

In Example E–2, the innermost layer describes the `IMAGE_LEVEL_2` theme. The `<ScaleHint>` element lists the `min_scale` and `max_scale` values, if any, for that

theme in the base map definition. For example, the base map definition for IMAGE_MAP is as follows:

```
SQL> select definition from user_sdo_maps where name='IMAGE_MAP';

DEFINITION
------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?>
<map_definition>
  <theme name="IMAGE_LEVEL_2" min_scale="1000.0" max_scale="0.0"/>
  <theme name="IMAGE_LEVEL_8" min_scale="5000.0" max_scale="1000.0"/>
  <theme name="MA_ROAD3"/>
  <theme name="MA_ROAD2"/>
  <theme name="MA_ROAD1"/>
  <theme name="MA_ROAD0"/>
</map_definition>
```

In the innermost layer, the <SRS> and <BoundingBox> elements identify the SRID and the DIMINFO information for that theme's base table, as shown in the following Spatial and Graph metadata query:

```
SQL> select srid, diminfo from user_sdo_geom_metadata, user_sdo_themes
  2  where name='IMAGE_LEVEL_2' and
  3  base_table=table_name and
  4  geometry_column=column_name ;

      SRID
----------
DIMINFO(SDO_DIMNAME, SDO_LB, SDO_UB, SDO_TOLERANCE)
--------------------------------------------------------------------------------
     41052
SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X', 200000, 500000, .5), SDO_DIM_ELEMENT('Y', 750
000, 950000, .5))
```

In Example E–2, the <Layer> element for a base map has an <SRS> element and a <LatLonBoundingBox> element. The <SRS> element is empty if all layers in the base map definition do not have the same SRID value specified in the USER_SDO_GEOM_METADATA view. If they all have the same SRID value (for example, 41052), the SRS element contains that value (for example, SDO:41052). The required <LatLonBoundingBox> element currently has default values (-180,-90,180,90). When this feature is supported by MapViewer, this element will actually be the bounds specified in the DIMINFO column of the USER_SDO_GEOM_METADATA view for that layer, converted to geodetic coordinates if necessary and possible.

All layers are currently considered to be opaque and queryable. That is, all layers are assumed to be vector layers, and not GeoRaster, logical network, or image layers.

E.2.3 Supported GetFeatureInfo Request and Response Features

This section describes the supported GetFeatureInfo request parameters and their interpretation by MapViewer. Example E–3 shows some GetFeatureInfo requests.

Example E–3 GetFeatureInfo Request

```
http://localhost:8888/mapviewer/wms?REQUEST=GetFeatureInfo&VERSION=1.1.1&BBOX=0,-0
.0020,0.0040&SRS=EPSG:4326&LAYERS=cite:Lakes,cite:Forests&WIDTH=200&HEIGHT=100&INF
O_FORMAT=text/xml&QUERY_LAYERS=cite:Lakes,cite:Forests&X=60&Y=60

http://localhost:8888/mapviewer/wms?REQUEST=GetFeatureInfo&VERSION=1.3.0
&BBOX=0,-0.0020,0.0040&CRS=EPSG:4326&LAYERS=cite:Lakes,cite:Forests&WIDTH=200&HEIG
```

```
HT=100
&INFO_FORMAT=text/xml&QUERY_LAYERS=cite:Lakes,cite:Forests&I=60&J=60
```

The response is an XML document and the Content-Type of the response is `text/xml`. Example E–4 is a response to a GetFeatureInfo request in Example E–3.

Example E–4 GetFeatureInfo Response

```
<?xml version="1.0" encoding="UTF-8" ?>
<GetFeatureInfo_Result>
  <ROWSET name="cite:Lakes">
    <ROW num="1">
      <ROWID>AAAK22AAGAAACUiAAA</ROWID>
    </ROW>
  </ROWSET>
  <ROWSET name="cite:Forests">
    <ROW num="1">
      <FEATUREID>109</FEATUREID>
    </ROW>
  </ROWSET>
</GetFeatureInfo_Result>
```

Most of the following sections describe parameters supported for a GetFeatureInfo request. (Parameters that are specific to MapViewer and not mentioned in the WMS 1.1.1 specification are labeled MapViewer-Only.) Section E.2.3.10 explains how to query attributes in a GetFeatureInfo request.

E.2.3.1 GetMap Parameter Subset for GetFeatureInfo Requests

A GetFeatureInfo request contains a subset of a GetMap request (`BBOX`, `SRS` [1.1.1] or `CRS` [1.3.0], `WIDTH`, `HEIGHT`, and optionally `LAYERS` parameters). These parameters are used to convert the X, Y (1.1.1) or I, J (1.3.0) point from screen coordinates to a point in the coordinate system for the layers being queried. It is assumed all layers are in the same coordinate system, the one specified by the `SRS` parameter.

E.2.3.2 EXCEPTIONS Parameter

The only supported value for the `EXCEPTIONS` parameter is the default: `application/vnd.ogc.se_xml` for WMS 1.1.1 or `xml` for WMS 1.3.0. That is, only Service Exception XML is supported. The exception is reported as an XML document conforming to the Service Exception DTD available at the following, depending on the version (1.1.1 or 1.3.0):

`http://schemas.opengis.net/wms/1.1.1/WMS_exception_1_1_1.dtd`

`http://schemas.opengis.net/wms/1.3.0/exceptions_1_3_0.xsd`

E.2.3.3 FEATURE_COUNT Parameter

The `FEATURE_COUNT` parameter specifies the maximum number of features in the result set. The default value is 1. If more features than the parameter's value interact with the query point (X, Y), then an arbitrary subset (of the size of the parameter's value) of the features is returned in the result set. That is, a GetFeatureInfo call translates into a query of the following general form:

```
SELECT <info_columns> FROM <layer_table>
  WHERE SDO_RELATE(<geom_column>,
    <query_point>, 'mask=ANYINTERACT')='TRUE'
  AND ROWNUM <= FEATURE_COUNT;
```

E.2.3.4 INFO_FORMAT Parameter

The value of the INFO_FORMAT parameter is always text/xml.

E.2.3.5 QUERY_LAYERS Parameter

The QUERY_LAYERS parameter specifies a comma-delimited list of layers to be queried. If the LAYERS parameter is specified, the QUERY_LAYERS specification must be a subset of the list specified in the LAYERS parameter.

If the QUERY_LAYERS parameter is specified, any BASEMAP parameter value is ignored.

E.2.3.6 QUERY_TYPE Parameter (MapViewer-Only)

The QUERY_TYPE parameter limits the result set to a subset of possibly qualifying features by specifying one of the following values:

- at_point: returns only the feature at the specified point.
- nn: returns only the nearest neighbor features, with the number of results depending on the value of the FEATURE_COUNT parameter value (see Section E.2.3.3). The result set is not ordered by distance.
- within_radius (or within_distance, which is a synonym): returns only results within the distance specified by the RADIUS parameter value (see Section E.2.3.7), up to the number matching the value of the FEATURE_COUNT parameter value (see Section E.2.3.3). The result set is an arbitrary subset of the answer set of potential features within the specified radius. The result set is not ordered by distance.

E.2.3.7 RADIUS Parameter (MapViewer-Only)

The RADIUS parameter specifies the radius of the circular search area for a query in which the QUERY_TYPE parameter value is within_radius (see Section E.2.3.6). If you specify the RADIUS parameter, you must also specify the UNIT parameter (see Section E.2.3.8).

E.2.3.8 UNIT Parameter (MapViewer-Only)

The UNIT parameter specifies the unit of measurement for the radius of the circular search area for a query in which the QUERY_TYPE parameter value is within_radius (see Section E.2.3.6). The value must be a valid linear measure value from the SHORT_NAME column of the SDO_UNITS_OF_MEASURE table, for example: meter, km, or mile.

If you specify the UNIT parameter, you must also specify the RADIUS parameter (see Section E.2.3.7).

E.2.3.9 X and Y or I and J Parameters

The X and Y (WMS 1.1.1) or I and J (WMS 1.3.0) parameters specify the x-axis and y-axis coordinate values (in pixels), respectively, of the query point.

E.2.3.10 Specifying Attributes to Be Queried for a GetFeatureInfo Request

In a GetFeatureInfo request, the styling rule for each queryable layer (theme) must contain a <hidden_info> element that specifies which attributes are queried and returned in the XML response. The <hidden_info> element is the same as the one used for determining the attributes returned in an SVG map request.

An example of such a styling rule as follows:

```
SQL> select styling_rules from user_sdo_themes where name='cite:Forests';

STYLING_RULES
--------------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
          <field column="FID" name="FeatureId"/>
       </hidden_info>
  <rule>
    <features style="C.PARK FOREST">  </features>
    <label column="NAME" style="T.PARK NAME"> 1 </label>
  </rule>
</styling_rules>
```

This styling rule specifies that if `cite:Forests` is one of the `QUERY_LAYERS` parameter values in a GetFeatureInfo request, the column named FID is queried, and its tag in the response document will be `<FEATUREID>`. The tag is always in uppercase. If no `<hidden_info>` element is specified in the styling rules for the theme's query layer, then the rowid is returned. In Example E–4, the styling rule for the `cite:Lakes` layer has no `<hidden_info>` element; therefore, the default attribute ROWID is returned in the XML response. The `cite:Forests` layer, however, does have a `<hidden_info>` element, which specifies that the attribute column is FID, and that its tag name, in the response document, should be `<FEATUREID>`.

E.3 Adding a WMS Map Theme

You can add a WMS map theme to the current map request. The WMS map theme is the result of a GetMap request, and it becomes an image layer in the set of layers (themes) rendered by MapViewer.

To add a WMS map theme, use the WMS-specific features of either the XML API (see Section E.3.1) or the JavaBean-based API (see Section E.3.4).

E.3.1 XML API for Adding a WMS Map Theme

To add a WMS map theme to the current map request using the MapViewer XML API, use the `<wms_getmap_request>` element in a `<theme>` element.

For better performance, the `<wms_getmap_request>` element should be used only to request a map image from a Web Map Server (WMS) implementation. That is, the `<service_url>` element in a `<wms_getmap_request>` element should specify a WMS implementation, not a MapViewer instance. If you want to specify a MapViewer instance (for example, specifying `<service_url>` with a value of `http://mapviewer.mycorp.com:8888/mapviewer/wms`), consider using a MapViewer predefined theme or a JDBC theme in the `<themes>` element instead of using a `<wms_getmap_request>` element.

The following example shows the general format of the `<wms_getmap_request>` element within a `<theme>` element, and it includes some sample element values and descriptive comments:

```
<themes>
  <theme>
     <wms_getmap_request isBackgroundTheme="true">
           <!-- The wms_getmap_request theme is rendered in the order it
                 appears in the theme list unless isBackgroundTheme is "true".
           -->
```

```
            <service_url> http://wms.mapsrus.com/mapserver </service_url>
            <version> 1.1.1 </version>
              <!-- version is optional. Default value is "1.1.1".
              -->
            <layers> Administrative+Boundaries,Topography,Hydrography </layers>
             <!— layers is a comma-delimited list of names.
                If layer names contain spaces, use '+' instead of a space -->
             <!— styles is optional. It is a comma-delimited list, and it must
                have the same number of  names as the layer list, if specified.
                If style names contain spaces, use '+' instead of a space -->
            <styles/>
            <srs> EPSG:4326 </srs>
            <format> image/png </format>
            <transparent> true </transparent>
            <bgcolor> 0xffffff </bgcolor>
            <exceptions> application/vnd.ogc.se_inimage </exceptions>
            <vendor_specific_parameters>
                 <!-- one or more <vsp> elements each containing
                       a <name> <value> pair  -->
                 <vsp>
                      <name> datasource </name>
                      <value> mvdemo </value>
                   </vsp>
            <vendor_specific_parameters>
       <wms_getmap_request>
    </theme>
  </themes>
```

The following attribute and elements are available with the `<wms_getmap_request>` element:

- The `isBackgroundTheme` attribute specifies whether or not this theme should be rendered before the vector layers. The default value is `false`.
- The `<service_url>` element specifies the URL (without the service parameters) for the WMS service. Example: `http://my.webmapserver.com/wms`
- The `<version>` element specifies the WMS version number. The value must be one of the following: `1.0.0`, `1.1.0`, `1.1.1` (the default), or `1.3.0`.
- The `<layers>` element specifies a comma-delimited list of layer names to be included in the map request.
- The `<styles>` element specifies a comma-delimited list of style names to be applied to the layer names in `layers`.
- The `<srs>` element specifies the coordinate system (spatial reference system) name. The default value is `EPSG:4326`.
- The `<format>` element specifies the format for the resulting map image. The default value is `image/png`.
- The `<transparent>` element specifies whether or not the layer or layers being added should be transparent in the resulting map image. The default value is `false`. To make the layer or layers transparent, specify `true`.
- The `<bgcolor>` element specifies the RGB value for the map background color. Use hexadecimal notation for the value, for example, `0xAE75B1`. The default value is `0xFFFFFF` (that is, white).
- The `<exceptions>` element specifies the format for server exceptions. The default value is `application/vnd.ogc.se_inimage`.

- The <vendor_specific_parameters> element contains one or more <vsp> elements, each of which contains a <name> element specifying the parameter name and a <value> element specifying the parameter value.

Example E–5 shows the <wms_getmap_request> element in a map request.

Example E–5 Adding a WMS Map Theme (XML API)

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="Raster WMS Theme and Vector Data"
  datasource="mvdemo" srid="0"
  width="500"
  height="375"
  bgcolor="#a6caf0"
  antialiase="true"
  mapfilename="wms_georaster" format="PNG_URL">
  <center size="185340.0">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>596082.0,8881079.0</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
  <themes>
    <theme name="WMS_TOPOGRAPHY" user_clickable="false"  >
       <wms_getmap_request isBackgroundTheme="true">
         <service_url> http://wms.mapservers.com:8888/mapserver/wms </service_url>
         <layers> TOPOGRAPHY </layers>
         <srs> EPSG:29190 </srs>
         <format> image/png </format>
         <bgcolor> 0xa6caf0 </bgcolor>
         <transparent> true </transparent>
         <vendor_specific_parameters>
           <vsp>
              <name> ServiceType </name>
              <value> mapserver </value>
           </vsp>
         </vendor_specific_parameters>
       </wms_getmap_request>
    </theme>
    <theme name="cl_theme" user_clickable="false">
       <jdbc_query  spatial_column="geom" render_style="ltblue"
         jdbc_srid="82279" datasource="mvdemo"
         asis="false">select geom from classes where vegetation_type = 'forests'
       </jdbc_query>
     </theme>
  </themes>
  <styles>
    <style name="ltblue">
      <svg width="1in" height="1in">
       <g class="color"
          style="stroke:#000000;stroke-opacity:250;fill:#33ffff;fill-opacity:100">
          <rect width="50" height="50"/>
       </g>
      </svg>
    </style>
  </styles>
</map_request>
```

E.3.2 Predefined WMS Map Theme Definition

The predefined XML definition for a WMS theme uses the same structure of the parameters in Section E.3.1, and adds the optional `capabilities_url` attribute, which is used by Map Builder when editing a WMS theme. If the `capabilities_url` attribute is defined, Map Builder will issue a `GetCapabilities` request to populate some UI elements in the editor page.

Example E–6 shows how to create a predefined WMS theme in the metadata. The base table and base column names can be any values, and in this example `'WMS'` is used for both.

Example E–6 Creating a Predefined WMS Theme

```
INSERT INTO user_sdo_themes  VALUES (
  'PRED_WMS_THEME',
  'WMS data',
  'WMS',
  'WMS',   '<?xml version="1.0" standalone="yes"?>
   <styling_rules theme_type="wms">
     <service_url>
http://sampleserver1b.arcgisonline.com/arcgis/services/Specialty/ESRI_
StateCityHighway_USA/MapServer/WMSServer </service_url>
     <layers> 0,1,2 </layers>
     <version> 1.3.0 </version>
     <srs> CRS:84 </srs>
     <format> image/png </format>
     <bgcolor> 0xA6CAF0 </bgcolor>
     <transparent> false </transparent>
     <styles> +,+,+ </styles>
     <exceptions> xml </exceptions>
     <capabilities_url>
http://sampleserver1.arcgisonline.com/ArcGIS/services/Specialty/ESRI_
StateCityHighway_USA/MapServer/WMSServer? </capabilities_url>
   </styling_rules>');
```

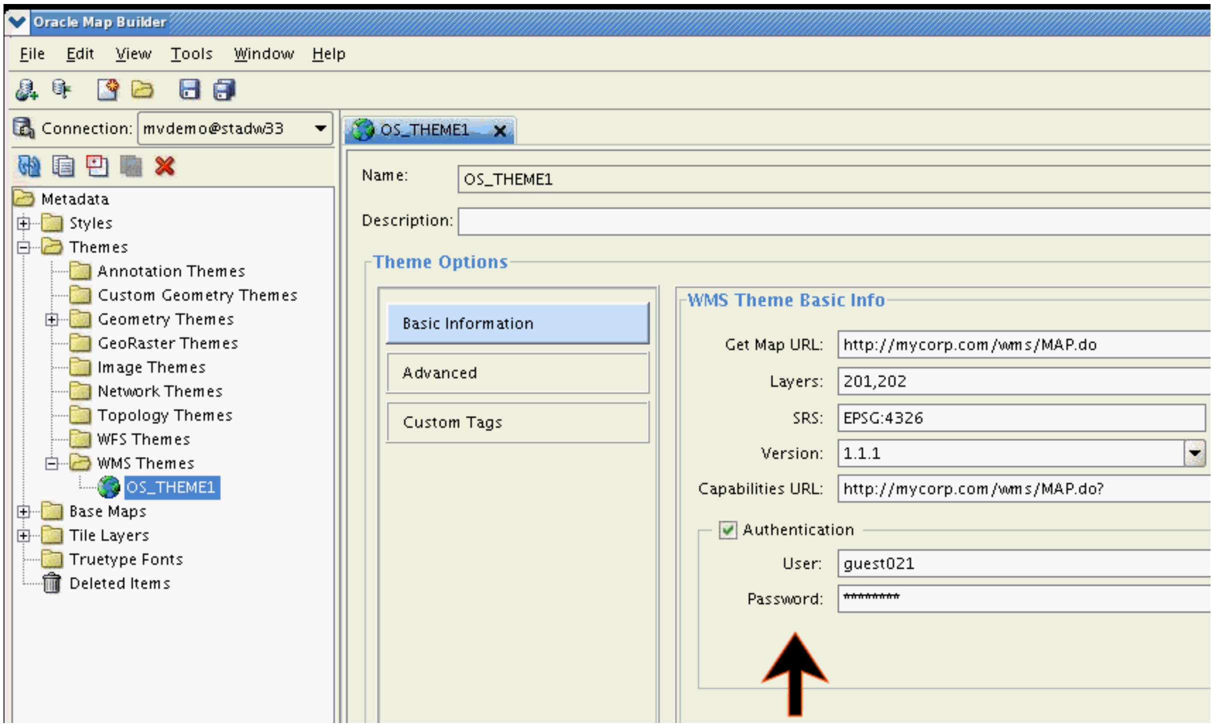
E.3.3 Authentication with WMS Map Themes

For a WMS server that requires authentication for access to the WMS data, the following must be included in the theme definition:

- `<user>` element specifying the user name
- `<password>` element specifying the user password

  If you use the Map Builder tool to create a WMS map theme, the password value will be automatically encrypted. Figure E–1 shows the use of the Map Builder tool to create a WMS theme with authentication information. In this figure, the Authentication option is checked (enabled), and User and Password are specified.

Figure E–1 Using Map Builder to Specify Authentication with a WMS Theme



Example E–7 shows how to create a WMS theme that includes authentication information.

Example E–7 WMS Theme with Authentication Specified

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="wms">
    <service_url> http://localhost:7001/mapviewer/wms </service_url>
    <user> wmsuser </user>
    <password> ******* </password>
    <layers> THEME_DEMO_STATES </layers>
    <version> 1.1.1 </version>
    <srs> EPSG:4326 </srs>
    <format> image/png </format>
    <bgcolor> 0xA6CAF0 </bgcolor>
    <transparent> true </transparent>
    <exceptions> application/vnd.ogc.se_xml </exceptions>
    <vendor_specific_parameters>
        <vsp>
            <name> datasource </name>
            <value> mvdemo </value>
    </vsp>
  </vendor_specific_parameters>
    <capabilities_url> http://localhost:7001/mapviewer/wms? </capabilities_url>
</styling_rules>
```

E.3.4 JavaBean-Based API for Adding a WMS Map Theme

To add a WMS map theme to the current map request using the MapViewer JavaBean-based API, use the `addWMSMapTheme` method.

This method should be used only to request a map image from a Web Map Server (WMS) implementation. That is, the `serviceURL` parameter should specify a WMS implementation, not a MapViewer instance.

The `addWMSMapTheme` method has the following format:

```
addWMSMapTheme(String name, String serviceURL, String isBackgroundTheme,
               String version, String[] layers, String[] styles,
               String srs, String format, String transparent,
               String bgcolor, String exceptions,
               Object[] vendor_specific_parameters
               );
```

The `name` parameter specifies the theme name.

The `serviceURL` parameter specifies the URL (without the service parameters) for the WMS service. Example: `http://my.webmapserver.com/wms`

The `isBackgroundTheme` parameter specifies whether or not this theme should be rendered before the vector layers. The default value is `false`.

The `version` parameter specifies the WMS version number. The value must be one of the following: `1.0.0`, `1.1.0`, or `1.1.1` (the default).

The `layers` parameter specifies a comma-delimited list of layer names to be included in the map request.

The `styles` parameter specifies a comma-delimited list of style names to applied to the layer names in `layers`.

The `srs` parameter specifies the coordinate system (spatial reference system) name. The default value is `EPSG:4326`.

The `format` parameter specifies the format for the resulting map image. The default value is `image/png`.

The `transparent` parameter specifies whether or not the layer or layers being added should be transparent in the resulting map image. The default value is `false`. To make the layer or layers transparent, specify `true`.

The `bgcolor` parameter specifies the RGB value for the map background color. Use hexadecimal notation for the value, for example, `0xAE75B1`. The default value is `0xFFFFFF` (that is, white).

The `exceptions` parameter specifies the format for server exceptions. The default value is `application/vnd.ogc.se_inimage`.

The `vendor_specific_parameters` parameter specifies a list of vendor-specific parameters. Each element in the object array is a String array with two strings: parameter name and value. Example: `vsp = new Object[]{new String[]{"DATASOURCE", "mvdemo"}, //param 1 new String[]{"antialiasing", "true"} //param 2`

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z