Oracle® Application Server

MapViewer User's Guide

Release 10.1.3.1

B14036-03

November 2006

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

ORACLE®

Oracle Application Server MapViewer User's Guide, Release 10.1.3.1

B14036-03

Primary Author: Chuck Murray

Contributors: Joao Paiva, L.J. Qian, Ji Yang

Contents

2 OracleAS MapViewer Concepts

3 OracleAS MapViewer Map Request XML API

4 OracleAS MapViewer JavaBean-Based API

5 OracleAS MapViewer JSP Tag Library

6 OracleAS MapViewer PL/SQL API

7 OracleAS MapViewer Administrative Requests

8 Oracle Maps

Index

List of Examples

List of Figures

List of Tables

Preface

Oracle Application Server MapViewer User's Guide describes how to install and use Oracle Application Server MapViewer (OracleAS 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 User's Guide and Reference.*

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

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

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

- *Oracle Spatial User's Guide and Reference*
- *Oracle Spatial GeoRaster*
- *Oracle Spatial Topology and Network Data Models*
- *Oracle Database Concepts*
- *Oracle Database SQL Reference*

See also the following document in the Oracle Application Server documentation set:

- *Oracle Application Server 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 OracleAS MapViewer, which was included in Oracle10*g* Release 2 (10.1.2).

Oracle Maps Feature

Oracle Maps refers to a suite of technologies provided as part of MapViewer. It consists of automated caching of pregenerated base map image tiles; an easy to use Ajax-based Web mapping interface, and powerful feature of interest (FOI) layers driven by Oracle Spatial database queries that are also accessible through Ajax.For information about Oracle Maps, see Chapter 8.

Oracle Map Builder Tool

Oracle Map Builder is a standalone graphical tool for creating and editing styles, themes, and base maps. It replaces the Map Definition Utility that was provided in previous releases. The main features of Oracle Map Builder include WYSIWYG editing and previewing of style, theme and base-map definitions; wizards to simplify workflow; and import and export of style, theme, and base map definitions from and into text files.

For information about Oracle Map Builder, see Chapter 9.

Enhanced Map Labeling

The following new features provide enhance map labeling capabilities:

- MapViewer can automatically break a long text label into multiple lines on the map, when necessary.
- New placements and alignment hints can be specified for a text style for use with oriented points or for labeling a line string feature (text along a path).
- A text style can now be specified as *sticky*, which means that any feature that uses it as a label style will always have its text label drawn on a map. For more information and an example, see Section 2.2.3.
- MapViewer now supports bidirectional language text labeling, to accommodate text in languages with right-to-left text flow, such as Arabic and Hebrew.

PL/SQL API: SDO_MVCLIENT Package

MapViewer now has a PL/SQL application programming interface (API). This API consists of the PL/SQL package SDO_MVCLIENT, which is intended for Oracle Database users who want to access OracleAS MapViewer services from inside an Oracle database. This package exposes most capabilities of OracleAS MapViewer, and it handles all communication with the actual OracleAS MapViewer server running on a middle tier on behalf of a user making map requests For information about the PL/SQL API, see Chapter 6.

New Advanced Style Types: Dot Density, Bar Chart, Collection

MapViewer supports the following new advanced styles: dot density (see Section A.6.4), bar chart (see Section A.6.5), and collection (see Section A.6.6). A collection style combines and uses multiple styles in a single rendering pass.

TrueType Font Symbol Support

MapViewer can now manage TrueType font files, and turn any symbol in a TrueType font into a MapViewer marker. For information about TrueType font-based marker styles, see Section A.2.3.

Custom (External) Attribute Data Provider Support

MapViewer now supports middle-tier joins of external attribute data (such as sales) with geometries stored in Oracle Spatial, providing even more thematic mapping possibilities. For information about thematic mapping using external attribute data, see Section 2.3.3.1.

Map Scale Bars

MapViewer now enables you to provide easy to use, configurable scale bars on maps. To add a scale bar, use the `setScaleBar(ScaleBarDef sbar)` method of the `oracle.lbs.mapclient.MapViewer` class of the Java API, or use the addScaleBar method of the Oracle Maps JavaScript API. The Oracle Maps feature is described in Chapter 8, and its JavaScript API is described in Section 8.4.

Theme-Wide Transparency

Any theme, even an external WMS layer, can now have a specified transparency when a map is rendered.

Embedded Geometries and Label Text

MapViewer now supports geometries and label text that are embedded fields inside user-defined Oracle SQL object types.

WMS Service Enhancements

MapViewer now supports a customizable SDO to EPSG SRID mapping table (file). This enables the WMS user to use EPSG SRIDs when interacting with MapViewer's WMS service.

Image Processing Operations with GeoRaster Themes

MapViewer now supports image processing operations on GeoRaster themes in a map request. For more informations, see the description of the `<operation>` element in Section 3.2.19.

Client-Side Style Creation

When developing a MapViewer application, you can now create dynamic styles on the client side and add them to the map request using one method:
`MapViewer.addStyle(name, StyleModel)`

The `addStyle` method is described in Section 4.3.4.

Theme Modifiers for Base Map Themes

You can now specify dynamic theme modifiers for base map themes by using the new `<theme_modifiers>` element, which is described in Section 3.2.17.

Sample Image of Any Style

You can now issue a simple HTTP request to a MapViewer server and get back a sample image showing how a specified MapViewer style will appear. For example, the URL `http://www.mycorp.com/mapviewer/omserver?sty=m.star` will return a small image displaying the `M.STAR` style, if MapViewer has been deployed at that location. For more information, see Section 2.2.4.

1

Introduction to OracleAS MapViewer

Oracle Application Server MapViewer (OracleAS MapViewer) is a programmable tool for rendering maps using spatial data managed by Oracle Spatial or Oracle Locator (also referred to as Locator). OracleAS 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:

1.1 Overview of OracleAS MapViewer

OracleAS MapViewer is shipped as part of Oracle Application Server. Its main deliverable is a J2EE application that can be deployed to a J2EE container, such as that for Oracle Application Server. OracleAS 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, PL/SQL, 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 symbologies 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 OracleAS MapViewer and retrieving the map response. The JavaBean-based API and the PL/SQL API provide 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 7.

Oracle Maps, built on core MapViewer features, uses a Map Cache 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 OracleAS MapViewer is its integration with Oracle Spatial, Oracle Locator, and the Oracle Fusion middleware. The current release of OracleAS MapViewer supports two-dimensional vector geometries stored in Oracle Spatial, as well as data in the Oracle Spatial 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 OracleAS MapViewer

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

For a map request:

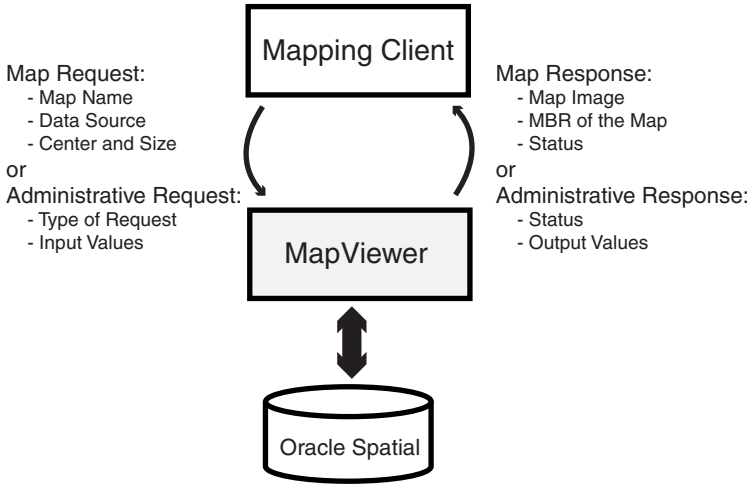
1. The client requests a map, passing in the map name, data source, center location, map size, and, optionally, other data to be plotted on top of a map.
2. The server returns the map image (or a URL for the image) and the minimum bounding rectangle (MBR) of the map, and the status of the request.

For an OracleAS MapViewer administrative request:

1. The client requests an OracleAS 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 OracleAS MapViewer.

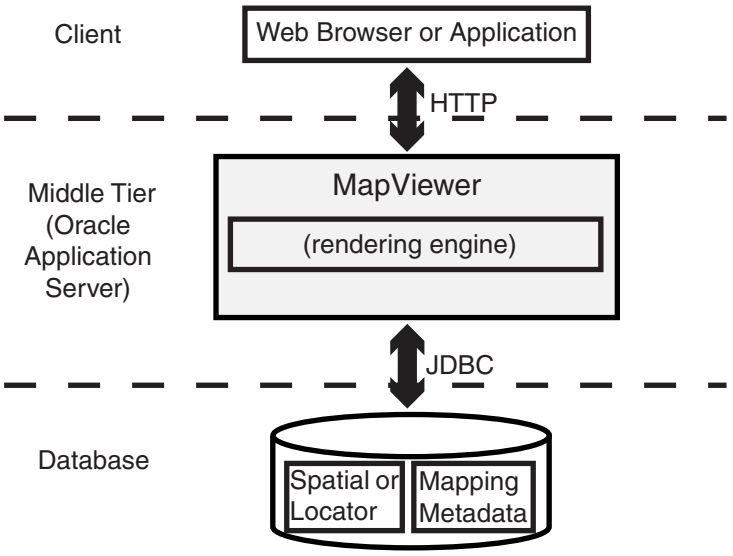
Figure 1–1 Basic Flow of Action with OracleAS MapViewer



1.1.2 OracleAS MapViewer Architecture

Figure 1–2 illustrates the architecture of OracleAS MapViewer.

Figure 1–2 OracleAS MapViewer Architecture



As shown in Figure 1–2:

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

1.2 Getting Started with OracleAS MapViewer

To get started using OracleAS MapViewer, follow these steps:

1. Either before or after you install and deploy OracleAS 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 OracleAS MapViewer (see Section 1.4).
4. Use OracleAS 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 7) to familiarize yourself with styles, themes, and maps, and the options for each, and optionally to preview spatial data.

1.3 Prerequisite Software for OracleAS MapViewer

To use OracleAS MapViewer, you must have the following Java packages and Oracle products, with the release number listed or a later release:

- Oracle Application Server 10*g* Release 3 (10.1.3) or later, or a standalone version of Oracle Application Server Containers for J2EE (OC4J) Release 10.1.3 or later, which is available from the Oracle Technology Network at

  `http://www.oracle.com/technology/`
- Oracle Database with Spatial or Locator (Release 9*i* or later)
- Oracle Client (Release 9*i* or later), if you need to use JDBC Oracle Call Interface (OCI) features. Note that in general, the JDBC thin driver is recommended for use with MapViewer, in which case Oracle Client is not required.
- Java SDK 1.5 or later

OracleAS MapViewer also supports the headless AWT mechanism in J2SE SDK, which enables OracleAS 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 OracleAS MapViewer:

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

1.4 Installing and Deploying OracleAS MapViewer

This section describes how to install (if necessary) and deploy OracleAS MapViewer to run in the middle tier. As mentioned previously, OracleAS MapViewer runs as a J2EE Web application and listens for incoming map requests on the container's HTTP port.

You can deploy OracleAS MapViewer either in a full Oracle Application Server environment or to a standalone installation of OC4J. Choose the procedure that applies to your needs:

- If you have already installed Oracle Application Server 10.1.3 or later, and want to deploy OracleAS MapViewer to that instance, follow the instructions in Section 1.4.1.
- If you have not installed Oracle Application Server but have installed the OC4J standalone kit and now want to install and deploy OracleAS MapViewer, follow the instructions in Section 1.4.2. OC4J standalone is a small footprint J2EE container and Web server provided by Oracle.

- Alternatively, you can download the latest MapViewer Quick Start kit from the MapViewer page on the Oracle Technology Network (OTN). This kit includes a standalone OC4J 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, the application server (or standalone OC4J) will create a home directory for MapViewer during deployment. This directory is typically located under the following directory:

```
$ORACLE_HOME/j2ee/<oc4j_instance_name>/applications
```

$ORACLE_HOME is the top directory of either the Application Server or standalone OC4J install. The value for <oc4j_instance_name> is typically `home` if deployed to standalone OC4J, or the name of the target OC4J instance if deployed to a full OracleAS installation. This MapViewer directory is typically named `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
   sql/
   web/
       fsmc/
       WEB-INF/
            lib/
            conf/
            log/
            mapcache/
            classes/
            admin/
```

The `/mapviewer/sql` directory contains several SQL scripts that are necessary for installing the MapViewer PL/SQL API package into the database. The `/mapviewer/web/fsmc` directory contains the JavaScript API library and several tutorials for Oracle Maps. The `/mapviewer/web/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 Application Server or Oracle Database installation, usually under the directory for Oracle *inter*Media files. You can copy them into the MapViewer `WEB-INF/lib` directory.

1.4.1 Deploying OracleAS MapViewer in an Oracle Application Server Environment

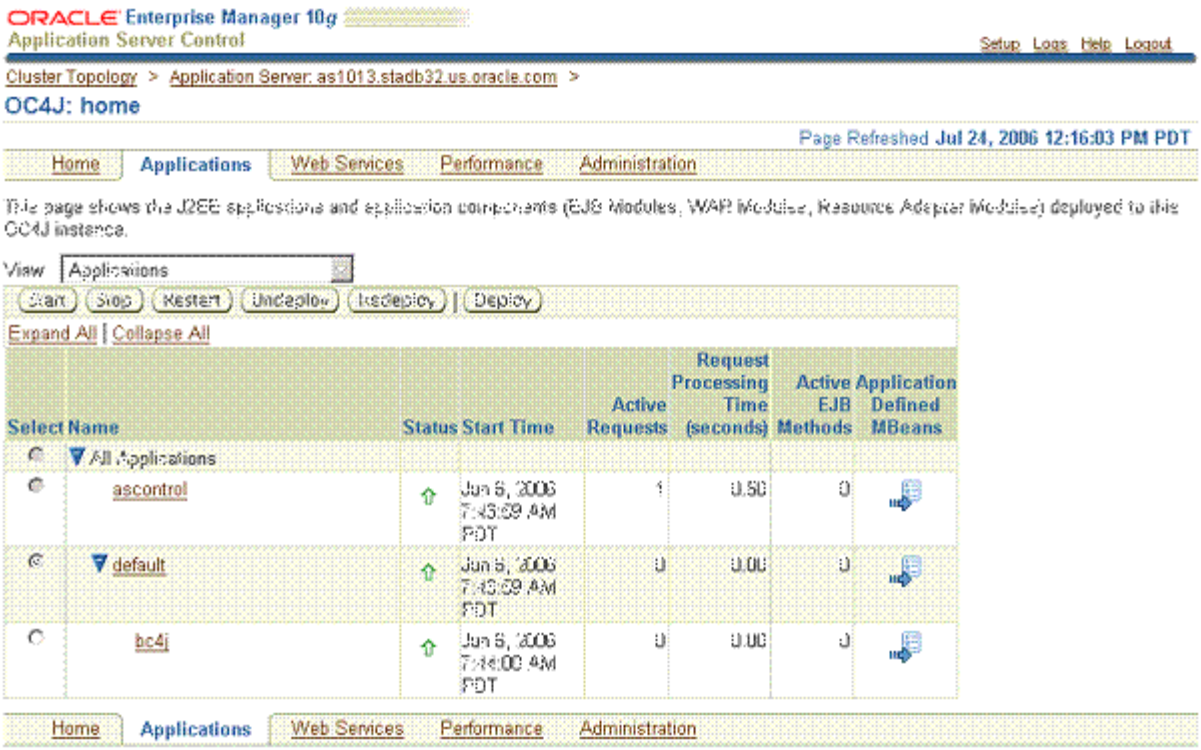
If you have already successfully installed Oracle Application Server version 10.1.3 or later, you can deploy the OracleAS MapViewer using the Oracle Enterprise Manager Server Control web interface. The main steps are the following:

1. Select an OC4J instance as the target for deploying MapViewer. You can select an existing OC4J instance, or create a new instance specifically for MapViewer. It is suggested that you create a new instance for MapViewer, but it is not required.
2. Locate the `mapviewer.ear` file. This file is either shipped with the Oracle Application Server software or, for version 10.1.3.1, can be downloaded from OTN.

3. Deploy the `mapviewer.ear` file to the selected OC4J instance using the Server Control web interface, or use Oracle Application Server command-line `admin` tool to deploy MapViewer (or any other J2EE application). For information about using the `admin` tool, see the *Oracle Application Server Administration Guide*.

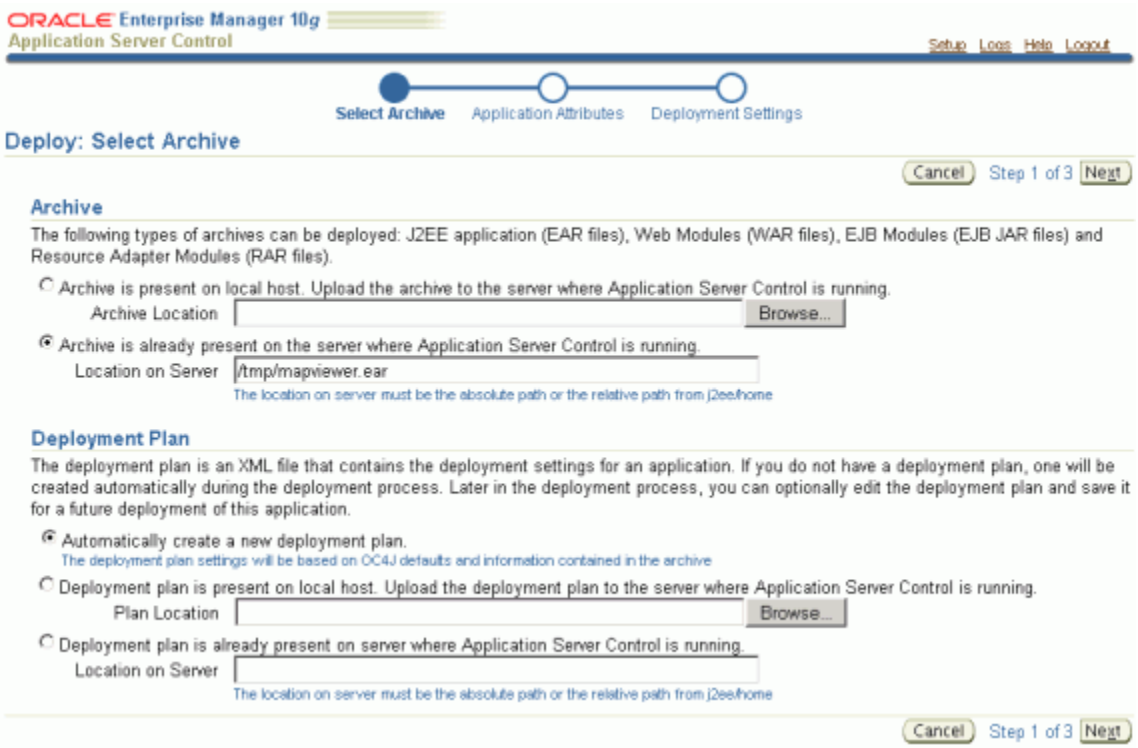
To start deploying MapViewer, navigate to the OracleAS Server Control page and select the desired OC4J instance, as shown in Figure 1–3, where the default home OC4J instance is selected.

Figure 1–3 Starting MapViewer Deployment



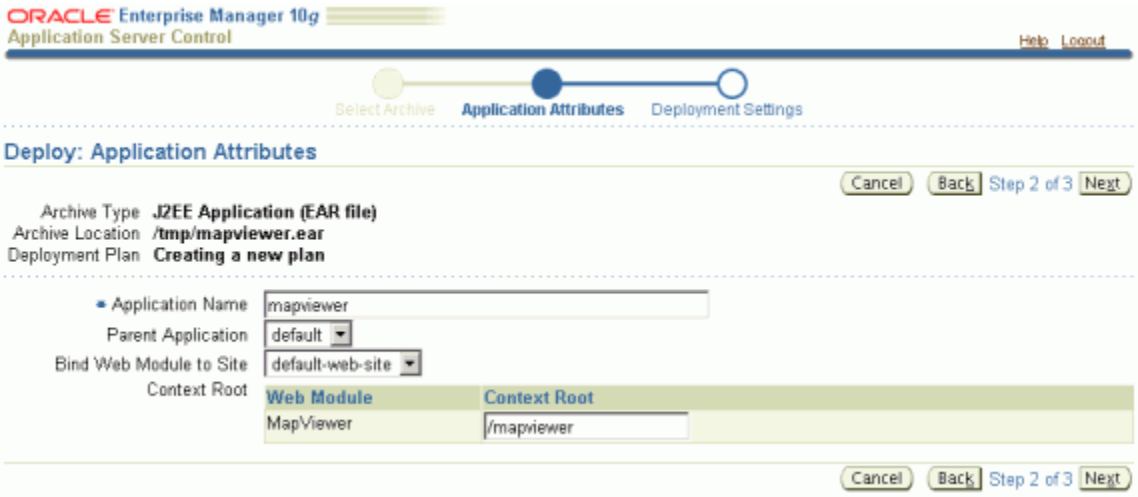
Click **Deploy** to display a page (shown in Figure 1–4) in which you enter the location of the `mapviewer.ear` file (a directory named `tmp` in this figure).

Figure 1–4 Specifying the mapviewer.ear Location



Click **Next** to display a page (shown in Figure 1–5) in which you specify the name of the application.

Figure 1–5 Specifying the Application Name



For **Application Name**, specify `mapviewer`. The Context Root will be set to `/mapviewer` by default. Do not change the context root value unless absolutely necessary.

Click **Next** to display the Deployment Setting page. You usually do not need to change any of the settings on this page.

Click **Deploy** on the Deployment Setting page to start the deployment of MapViewer. If the deployment is successful, the Confirmation page is displayed indicating that deployment of the application was successful.

After you complete the deployment, see Section 1.4.3.

1.4.2 Installing OracleAS MapViewer with a Standalone Installation of OC4J

To install and deploy OracleAS MapViewer with a standalone installation of OC4J, you must have installed OC4J on your system. The standalone OC4J installation kit is a single zip file that you can download from OTN. It contains the Oracle Container for J2EE and also a lightweight Web server. After you unzip this file, you can start the OC4J instance up by entering the command `java -jar oc4j.jar` from the *$OC4J_HOME*`/j2ee/home` directory, where *$OC4J_HOME* is the top directory into which you unzipped the installation file.

Note that you must have the Java 1.5 SDK installation, not the JRE installation, in your environment path in order for OC4J to start up and function properly.

Because standalone OC4J version 10.1.3 (or later) comes with its own Server Control Web interface, the deployment of MapViewer is almost exactly as described in Section 1.4.1 once you log into its Server Control Web page. The only difference is that you will not be able to choose a different OC4J instance, because you are running in a single standalone OC4J instance.

After you complete the deployment, see Section 1.4.3.

1.4.3 After Deploying MapViewer

After successfully deploying MapViewer to Oracle Application Server or standalone OC4J, you may want to verify whether it is actually working, as described in Section 1.4.3.1. It's also a good idea to become familiar with its web interface, particularly its 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 OracleAS MapViewer server has started correctly, point your browser to that OC4J instance. For example, if OracleAS MapViewer is installed on a system named www.xyzabc.com and the HTTP port is 8888, enter the following URL to invoke the OracleAS MapViewer server with a simple get-version request:

```
http://www.xyzabc.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:

```
Ver10131_B060225
```

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 OracleAS 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 Application Server 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 OracleAS 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 Application Server or standalone OC4J. Note that MapViewer 10.1.3.1 must be deployed to Application Server (or standalone OC4J) 10.1.3 or later. It will not work properly with earlier versions of Oracle Application Server or 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 or Oracle Locator (Release 8.1.6 or later) installed and from which you want OracleAS 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 MDSYS user:

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

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

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 cache instances 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 1–6 shows the MapViewer Welcome page

Figure 1–6 MapViewer Welcome Page

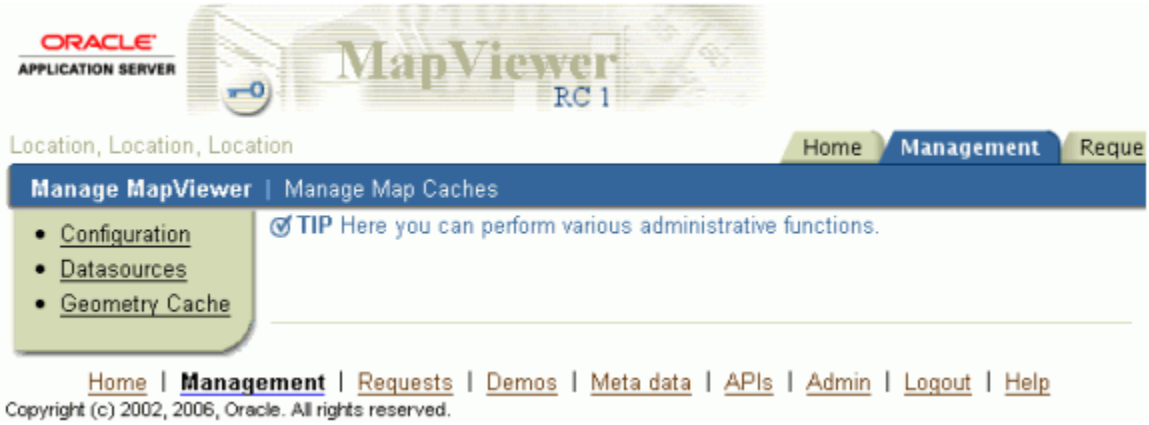
Click the **Admin** icon at the top right or text link at the bottom. A login prompt is displayed, asking for user name and password for the MapViewer administration page.

User Name: Enter `oc4jadmin`.

Password: Enter the password that you use to log into the Server Control page of the OracleAS or OC4J standalone installation.

After you log in, the MapViewer administration page is displayed, as shown in Figure 1–7.

Figure 1–7 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 cache instances. For detailed about configuration tasks, see Section 1.5.2; for information about administrative tasks, see Section 1.5.3.

1.5.2 Configuring OracleAS MapViewer

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

After you modify this file, you must restart OC4J to have the changes take effect; however, you can instead use the OracleAS MapViewer Admin page to restart only the OracleAS MapViewer servlet (instead of the entire OC4J instance, which may have other applications deployed and running) if either of the following applies:

- You installed OracleAS MapViewer with a standalone OC4J instance.
- The OracleAS MapViewer OC4J instance with Oracle Application Server is configured to have only one OC4J process running (the default) and not to be clustered (that is, not to be in an island).

  If you deployed OracleAS MapViewer to an OC4J instance with multiple processes (thus with multiple physical JVMs on the same host), or if you deployed to an OC4J instance that is in a clustered island (with multiple OC4J instances running on multiple hosts), you must restart the OC4J instance itself for the changes to the OracleAS MapViewer configuration file to take effect in all OracleAS MapViewer servers. In the latter case (clustered OC4J instances), you may also need to modify the OracleAS MapViewer configuration file in the OracleAS MapViewer directory hierarchy for each host's OC4J instance in the cluster. For more information about repository-based middle-tier clustering, see *Oracle Application Server High Availability Guide*.

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

- Logging information, defined 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 1.5.2.12)
- 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 cache server configurations, defined in the `<map_cache_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 OracleAS MapViewer Configuration File

```
<?xml version="1.0" ?>
<!-- This is the configuration file for OracleAS 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 2004, 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 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>
-->

<!-- ******************************************************************* -->
<!-- **************** 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 Cache Server Setting  ******************* -->
  <!-- ******************************************************************* -->
  <!-- Uncomment and modify the following to customize the map cache server.

       <cache_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/mapcache

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

       <logging> specifies the logging options for map cache server.
  -->

  <!--
     <map_cache_server>
       <cache_storage default_root_path="/scratch/mapcachetest/"/>
       <logging log_level="finest" log_thread_name="false" log_time="true">
          <log_output name="System.err"/>
          <log_output name="../log/mapcacheserver.log"/>
       </logging>
    </map_cache_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="!tiger"
                   jdbc_mode="thin"
                   number_of_mappers="3"
   />
   -->

</MapperConfig>
```

1.5.2.1 Specifying Logging Information

Logging information is specified in the `<logging>` element.

OracleAS MapViewer provides a flexible logging mechanism to record run-time information and events. You can configure the volume, format, and destination of the log output.

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 OracleAS 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, OracleAS MapViewer saves the requested map image as a file on the host system where OracleAS 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 OracleAS 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.abcxyz.com:1521/mapviewer/images`, the map image URL

for `omsmap1.gif` will be `http://dev04.abcxyz.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 OracleAS 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.
- 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 OracleAS MapViewer checks for possible files to delete.

  Default: OracleAS MapViewer never deletes the generated map images.
- The `recycle_interval` attribute specifies the number of minutes between times when OracleAS 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, OracleAS 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 OracleAS MapViewer users are permitted to make administrative requests.

However, if you want to restrict the ability to submit administrative requests, you can edit the OracleAS 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 OracleAS 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 OracleAS 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) 2003, XYZ Corporation"
              font="sans serif"
              position="SOUTH_EAST"/>
    <title  text="Map Courtesy of XYZ 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 OracleAS 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 OracleAS MapViewer is running.
- The `position` attribute provides a positioning hint to OracleAS 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 OracleAS MapViewer is running.
- The `position` attribute provides a positioning hint to OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS MapViewer automatically apply a globular map projection (that is, a map projection suitable for viewing the world, and specifically the azimuthal equidistant projection for OracleAS 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 OracleAS 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 OracleAS 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 2.3.1.2.

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 written in JSP 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 s[pecify 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 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>
  <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 `<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 (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 D.1.3.

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 2.3.3.1.

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 Configuring the Map Cache Server for Oracle Maps

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

```
<map_cache_server>
   <cache_storage default_root_path="/scratch/mapcache/" />
   <logging log_level="finest" log_thread_name="false" log_time="true">
      <log_output name="System.err"/>
      <log_output name="../log/mapcacheserver.log"/>
```

```
   </logging>
</map_cache_server>
```

The `<cache_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 cache server.

1.5.2.12 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="!tiger"
                 jdbc_mode="thin"
                 number_of_mappers="5"
                 max_connections="100"
                 allow_jdbc_theme_based_foi="true"
/>
```

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

- The `name` attribute specifies a unique data source name to OracleAS MapViewer. You must specify the data source name in all map requests that identify a data source.
- 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 OracleAS MapViewer next restarts, it will automatically obfuscate and replace the clear text password.
- The `jdbc_mode` attribute tells OracleAS 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 OracleAS MapViewer is running.
- The `number_of_mappers` attribute identifies the maximum number of map renderers available (and thus the maximum number of map requests that OracleAS 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, OracleAS 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 OracleAS MapViewer has finished processing one of the current requests.

  Specifying a large `number_of_mappers` value (such as 30 or 50) does not cause additional static memory to be used, and it does not affect the total number of

database connections that will remain open. However, specifying a large value does cause some additional overhead operations, which might affect server performance at times of peak loads. The maximum number of mappers for a single data source is 64.

- The `max_connections` attribute specifies the maximum number of database connections or sessions open for the data source at any given time. In most cases you should not specify this attribute, and accept the default value of 100.

  If you specify a value that is too small, the effect on performance can be significant. For example, if you specify `max_connections="5"` for a map request with 12 predefined themes, 12 connections will still be created temporarily to meet the demand, but 7 of them will be closed immediately upon the completion of the request (leaving only 5 open connections). OracleAS MapViewer will then dynamically create database connections whenever it needs more than 5 to meet the demand when processing map requests, because the number of permanently open database connections will never exceed the specified `max_connections` attribute value. Specifying a value that is too small will almost certainly increase the time it takes to process a map request, because opening a new database connection involves significant processing overhead.

- The `container_ds` attribute lets you specify the J2EE 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 OracleAS MapViewer data source as follows:

  ```
  <map_data_source name="mvdemo"
                   container_ds="jdbc/OracleDS"
                   number_of_mappers="5"
                   max_connections="100"
  />
  ```

  To use the `container_ds` attribute in the OracleAS MapViewer configuration file, you must start the OC4J instance with the `-userThreads` option. OracleAS 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 OracleAS MapViewer Admin page, you can use the `container_ds` attribute regardless of whether you started the OC4J instance with the `-userThreads` option.

- 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

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 cache instances 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 cache instance. 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 cache instance. You can also define the map cache properties in XML by clicking XML.

- Managing map cache instances 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 cache instance, under **Existing Map Cache Instances**, 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 High Availability and OracleAS MapViewer

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

With the current release of Oracle Application Server, OracleAS MapViewer users can benefit from the high availability features more effectively than in previous releases.

1.6.1 Deploying OracleAS MapViewer on a Multiprocess OC4J Instance

You can safely deploy OracleAS MapViewer in an OC4J instance of Oracle Application Server that has multiple processes. Oracle Application Server 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 OracleAS 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 OracleAS MapViewer is deployed to an OC4J instance with multiple processes, each process has an OracleAS MapViewer server running inside it. These OracleAS 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 OracleAS MapViewer servers. Each OracleAS MapViewer server generates map image files according to a unique naming scheme, with the names coordinated when the different OracleAS MapViewer servers are first started (that is, when the containing OC4J instance is started). This avoids the possibility of two OracleAS MapViewer servers generating map files in the same sequence with the same file names.

1.6.2 Deploying OracleAS MapViewer on a Middle-Tier Cluster

OC4J instances in different Oracle Application Server installations can be clustered into an island. This provides a middle-tier fail-safe option. OracleAS 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 OracleAS MapViewer server running on host A. OracleAS 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 OracleAS MapViewer servers running on different hosts. You must configure the map file storage information (see Section 1.5.2.2) for each OracleAS 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 OracleAS 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 OracleAS MapViewer instance (as explained in the preceding paragraph), so that each OracleAS 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.

2

OracleAS MapViewer Concepts

This chapter explains concepts that you should be familiar with before using OracleAS 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 OracleAS 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:

2.1 Overview of OracleAS MapViewer

When an application uses OracleAS 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.

OracleAS 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 OracleAS MapViewer metadata views described in Section 2.8 (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 Map Builder tool, described in Chapter 7.

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.
- **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 2.3.3. 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.

  The `AdvancedStyle` class is extended by `BucketStyle`, which is in turn extended by `ColorSchemeStyle` and `VariableMarkerStyle`. (Additional advanced styles, such as for charts, are planned for a future release.)

  Other advanced styles are dot density (described in Section A.6.4), bar chart (described in Section A.6.5), and collection (described in Section A.6.6).

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 2.8 and Section 2.8.3.

You can also create **dynamically defined styles** (that is, temporary styles) of any style type as part of a map request. To create a dynamically defined style, define the style using its XML elements within the `<map_request>` element. (You can also use the JavaBean API to add a dynamically defined style to a map request, as explained in Section 4.3.4.) OracleAS MapViewer creates dynamically defined styles from these definitions when it processes the map request, and it discards the dynamically defined 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.

2.2.1 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 2–1 creates an advanced style named `V.RB1` in which each bucket is assigned a text label style (using the `label_style` attribute), with some styles being used for several buckets.

Example 2–1 Advanced Style with Text Label Style for Each Bucket

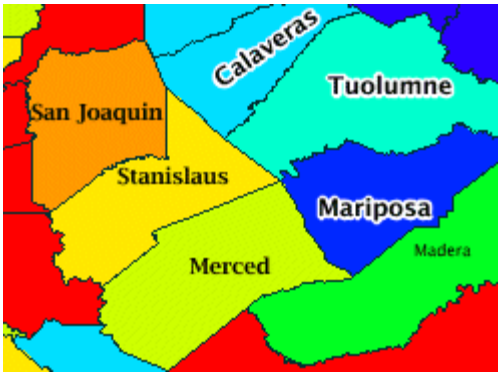
```
<?xml version="1.0" ?>
<AdvancedStyle>
  <BucketStyle>
     <Buckets>
      <RangedBucket seq="0"  label="10k or less"  high="10000"
        style="c.rb13_1"  label_style="T.AIRPORT NAME"/>
      <RangedBucket seq="1"  label="10k - 20k" low="10000" high="20000"
        style="c.rb13_2"  label_style="T.CITY NAME"/>
      <RangedBucket seq="2"  label="20k - 30k" low="20000" high="30000"
        style="c.rb13_3"  label_style="T.CITY NAME"/>
      <RangedBucket seq="4"  label="30k - 40k" low="30000" high="40000"
        style="c.rb13_4"  label_style="T.CITY NAME"/>
      <RangedBucket seq="5"  label="40k - 50k" low="40000" high="50000"
        style="c.rb13_5"  label_style="T.CITY NAME"/>
      <RangedBucket seq="6"  label="50k - 75k" low="50000" high="75000"
        style="c.rb13_6"  label_style="T.ROAD NAME"/>
      <RangedBucket seq="7"  label="75k - 100k" low="75000" high="100000"
        style="c.rb13_7"  label_style="T.PARK NAME"/>
      <RangedBucket seq="8"  label="100k - 125k" low="100000" high="125000"
        style="c.rb13_8"  label_style="T.RED STREET"/>
      <RangedBucket seq="9"  label="125k - 250k" low="125000" high="250000"
        style="c.rb13_9"  label_style="T.ROAD NAME"/>
```

```
      <RangedBucket seq="10"  label="250k - 450k" low="250000" high="450000"
        style="c.rb13_10"  label_style="T.ROAD NAME"/>
      <RangedBucket seq="11"  label="450k - 650k" low="450000" high="650000"
        style="c.rb13_11"  label_style="T.ROAD NAME"/>
      <RangedBucket seq="12"  label="650k up"   low="650000" style="c.rb13_13"/>
      </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.RB1` style in Example 2–1 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 2–1, 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.RB1`), 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.RB1">
    </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.RB1`, 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.RB1">
    </features>
    <label column="county" style="V.RB1">
        1
    </label>
  </rule>
</styling_rules>
```

2.2.2 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 introduced in Oracle Spatial for Oracle Database 10*g* Release 1 (10.1). 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 User's Guide and Reference*.

2.2.2.1 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 oriented point in the map request. When OracleAS 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 2–2 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 2–2 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 2–2. (The label is the phrase *Oriented Point*.)

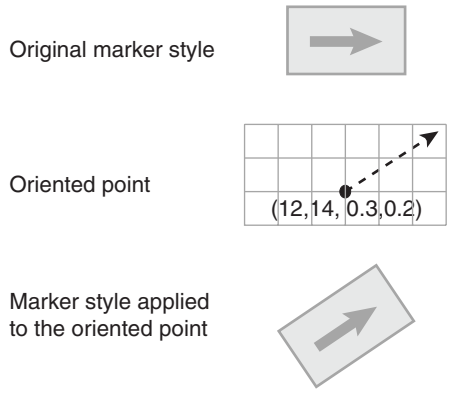
Figure 2–2 Map Display of the Label for an Oriented Point

2.2.2.2 Controlling Marker Orientation

When a marker style is applied to an oriented point, OracleAS 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 2–2 in Section 2.2.2.1, the marker style is rotated accordingly (in this case about 34 degrees counterclockwise) to reflect the orientation vector.

Figure 2–3 Oriented Marker



2.2.3 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 2–3 shows an XML definition of a style with the `sticky` attribute set to `true`.

Example 2–3 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>
```

2.2.4 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.

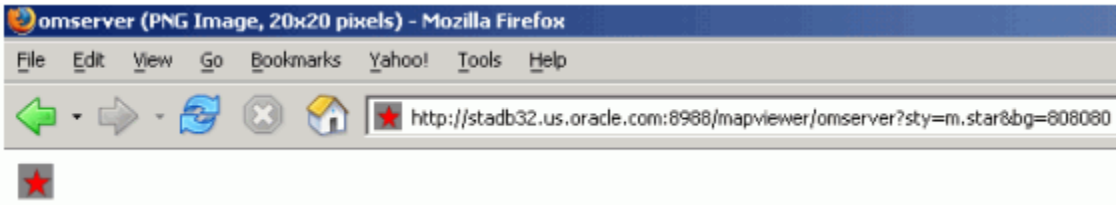
- bg specifies the background color of the sample image. The format must be *rrggbb*, such as 808080 for a gray color. The default value is ffffff (white).
- aa specifies whether the sample image should be rendered in antialiasing mode. The default value is the string true. Specify the string false if 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

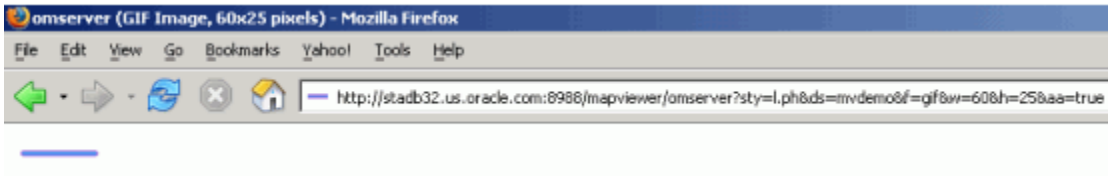


The following example generates an antialiased GIF image with the default white background, a width of 60 pixels, and a height of 25 pixels, displaying the line style named L.PH from the MapViewer data source named mvdemo:

```
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

A **theme** is a visual representation of a particular data layer. Typically, a theme is associated with a spatial geometry layer, that is, with a column of type SDO_GEOMETRY in a table or view. For example, a theme named US_STATES might be associated with a column named GEOMETRY in a STATES table. Other types of themes include the following:

- Image themes, which are associated with georeferenced images
- GeoRaster themes, which are associated with Oracle Spatial GeoRaster data
- Network themes, which are associated with networks in the Oracle Spatial network data model

- Topology themes, which are associated with topologies in the Oracle Spatial topology data model

When you define a theme, you must specify a base table or view, a spatial data column in that table or view, and a set of styling rules. For a predefined theme (described in Section 2.3.1), the definition is permanently stored in the database. However, you can also dynamically define a theme (that is, create a JDBC theme, described in Section 2.3.2) by supplying the definition within a map request.

If a theme is associated with a spatial layer geometry in a view, the view can be based on one or more tables. However, if the view is based on two or more tables (that is, if it is a join view), you must specify the `key_column` attribute (described in Section A.7) in the STYLING_RULES column definition in the USER_SDO_THEMES view. The following example specifies the column named GID in the join view named VIEW_THEME for the `key_column` attribute:

```
UPDATE user_sdo_themes SET styling_rules=
'<?xml version="1.0" standalone="yes"?>
<styling_rules key_column="gid">
<rule>
   <features style="L.PH"/>
   <label column="label" style="M.FLASH_SHIELD1">1</label>
</rule>
</styling_rules>' WHERE name='VIEW_THEME';
```

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 2.8, especially Section 2.8.2). When you specify a predefined theme in a map request, you need to specify only the theme name. OracleAS MapViewer automatically finds the theme's definition, constructs a query based on it, retrieves the relevant spatial and attribute data, and renders the theme 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 User's Guide and Reference*) 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 OracleAS 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 many themes (but not for GeoRaster, network, or topology themes), you can use the graphical Map Builder tool to create predefined themes of varying complexity. For information about the Map Builder tool, see Chapter 7.

2.3.1.1 Styling Rules in Predefined Spatial Geometry Themes

Each predefined theme is 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 2–4. 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 2–4 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 attribute value, and the style to be used for the selected features. 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 2–4, 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`. Any conditions that are valid in a SQL WHERE clause can be used as the value of a `<features>` element. 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, is used to determine whether or not a feature will be annotated. If the 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.

If two or more rules are specified, 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 to fetch qualified features contains the spatial column, the attribute column or columns, the name of the feature style, the label information, the WHERE clause, and the feature query. Based on the value of the attribute column or columns and the definition of the specified feature style, each feature is associated with a style.

2.3.1.2 Caching of Predefined Themes

By default, OracleAS MapViewer automatically caches the spatial data for a predefined theme when it is fetched from the database for processing by the OracleAS MapViewer rendering engine. By contrast, data for dynamic (JDBC) themes is never cached in OracleAS 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 OracleAS 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 OracleAS 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, OracleAS 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 OracleAS MapViewer spatial data cache is global, all predefined themes that are accessed by OracleAS 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 OracleAS MapViewer does not requery the database for these themes.

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, OracleAS 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 OracleAS 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 2–5 is a map request that includes a JDBC theme.

Example 2–5 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 OracleAS MapViewer executes for the JDBC theme in Example 2–5 is:

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

For this request, OracleAS 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 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 2–6 is a sample `<styling_rules>` element of a predefined theme with a complex SQL query.

Example 2–6 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 2–6 is defined as a predefined theme, OracleAS MapViewer still treats it as a JDBC theme at run time when a user requests a map that includes this theme. As with a normal JDBC theme, OracleAS 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 2–6. (For information about the `asis` attribute, see Section 3.2.9.)

2.3.3 Thematic Mapping

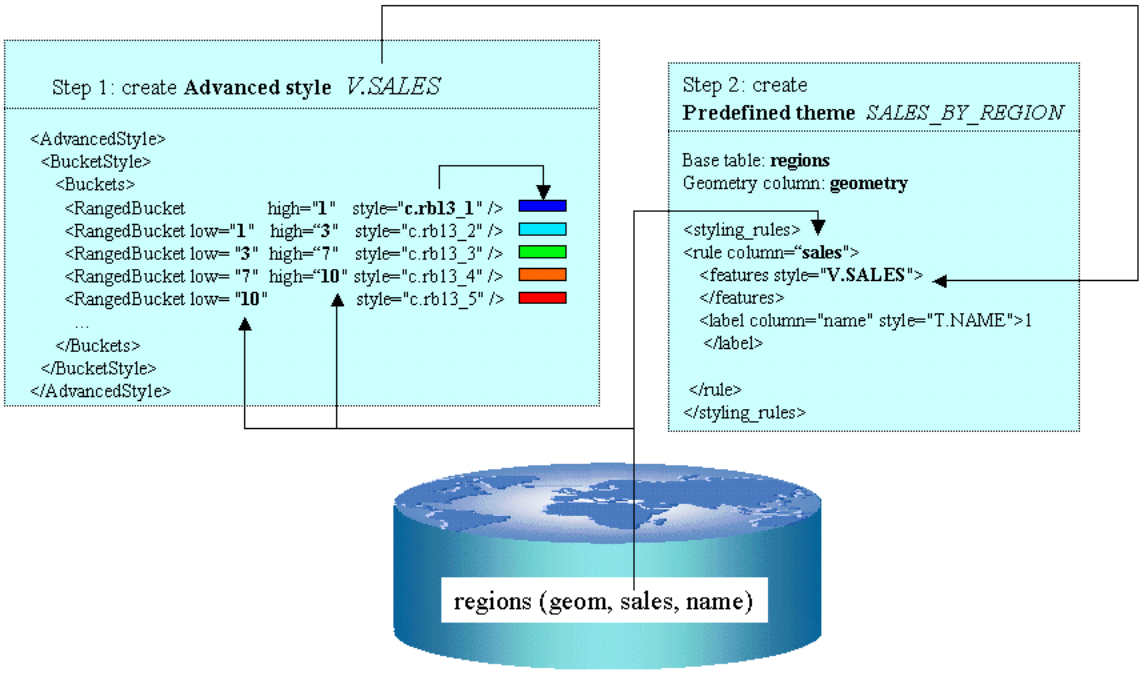
Thematic mapping refers to the drawing of spatial features based on their attribute values. OracleAS MapViewer uses thematic mapping to create maps in which colors or symbols are applied to features to indicate their attributes. For example, a `Counties` theme can be drawn using colors with different 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 2–6 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 2.2.1.)

Figure 2–6 Thematic Mapping: Advanced Style and Theme Relationship



In addition to the individual range-based buckets shown in Figure 2–6, OracleAS 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 2–7 is the XML definition for an `Earthquakes` theme.

Example 2–7 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 2–7 has only one rule. The `<rule>` element includes an attribute named `column` that does not appear in the `Airport` theme in Example 2–4. 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 2.8.2.) The RICHTER_SCALE column must be of type NUMBER. To understand why, look at the advanced style definition in Example 2–8.

Example 2–8 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 OracleAS MapViewer, which determines sequence only by the order in which elements appear in a definition.

Example 2–8 used the `<VariableMarkerStyle>` tag. The following examples use the `<ColorSchemeStyle>` tag in creating thematic maps of census blocks in California. Example 2–9 illustrates the use of a graduated color scale for a thematic mapping of population density. Example 2–10 is a thematic mapping of average household income using a graduated color scale. Example 2–11 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 2–9 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 2–9). 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 2–9, Example 2–10 illustrates the use of a base color and a graduated color scheme, this time to show household income.

Example 2–10 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 2–10 and Example 2–11). 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 2–11 uses specific color styles for each average household income range.

Example 2–11 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 2–12.

Example 2–12 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 2–12:

- `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 2–13.

Example 2–13 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

Table 2–2 (Cont.) Table Used with Gasoline Stations Theme

Column	Data Type
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 2–12) 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`.

2.3.3.1 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. Note that 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 Application Server listens for incoming requests. For example: `http://www.mycorp.com:8888/mapviewer/mapclient`

Example 2–14 shows a simple nonspatial data provider implementation. This implementation is also supplied with MapViewer as a default nonspatial data provider.

Example 2–14 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);

  }
}
```

2.3.4 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.

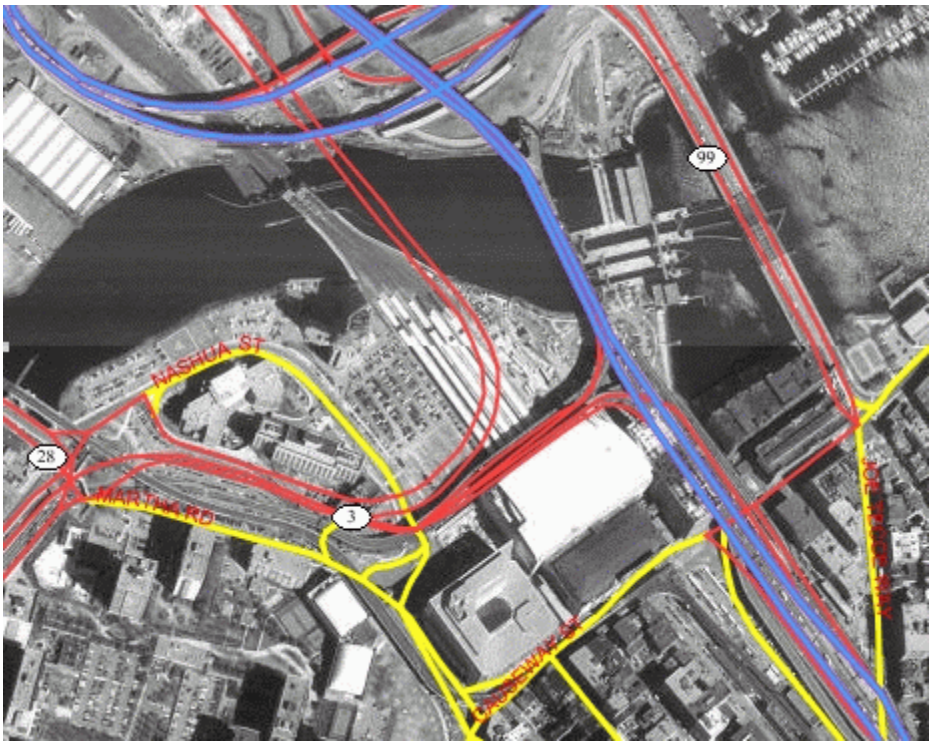
To specify any appearance-related attributes, use the `<theme>` element (described in Section 3.2.15) with the XML API or the JavaBean-based API (see especially Section 4.3).

2.3.5 Image Themes

An **image theme** is a special kind of OracleAS 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 *inter*Media object (ORDSYS.ORDImage) that points to the original image file. For information about creating an ORDSYS.ORDImage object, see Oracle interMedia 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 2.3.5.1. 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.
- OracleAS 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, OracleAS MapViewer supports only GIF, JPEG, PNG, and TIFF image formats. To enable OracleAS 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 OracleAS 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 OracleAS MapViewer processes the request, see Example 3–6 in Section 3.1.6.

2.3.5.1 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 2.8.2). Example 2–15 stores the definition of an image theme.

Example 2–15 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 2–15 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`, OracleAS 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 OracleAS 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 2–15, the image resolution is 2 meters per pixel.

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

2.3.6 GeoRaster Themes

A **GeoRaster theme** is a special kind of OracleAS MapViewer theme useful for visualizing GeoRaster objects. GeoRaster is a feature of Oracle Spatial 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 GeoRaster*.

Before you can use OracleAS 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 OracleAS 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 2–19 in Section 2.3.6.1 prepares GeoRaster data for use and stores a GeoRaster theme in the database.

OracleAS 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, OracleAS 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, OracleAS 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.

Example 2–16 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, OracleAS MapViewer calculates the best pyramid level by using the spatial resolution values set during or after the georeferencing process. (Note that in Example 2–16, `georid=1` in the WHERE clause refers to a column named GEORID in the GeoRaster table named PCI_IMAGE.)

Example 2–16 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 2–17 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 2–17 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>
```

2.3.6.1 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 2.8.2). Example 2–18 stores the definition of a GeoRaster theme.

Example 2–18 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 2–18 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 2.3.6, including `raster_bands`, `raster_pyramid`, `raster_id`, and `raster_table`, as shown in Example 2–18. 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 2–18 with the following:

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

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

Example 2–19 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 GeoRaster*.

Example 2–19 Preparing GeoRaster Data for Use with a GeoRaster Theme

```
connect scott/tiger

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.
--
-- This is REQUIRED for all GeoRaster tables.
-- It is used to manage the GeoRaster sysdata table.
---------------------------------------------------------------------
```

```
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/manager;

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/tiger;

 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/manager;

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/tiger;

--------------------------
-- Change the GeoRaster format (optional).
--------------------------

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 OracleAS 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;
```

2.3.7 Network Themes

A **network theme** is a special kind of OracleAS MapViewer theme useful for visualizing networks defined using the Oracle Spatial 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. OracleAS 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 Topology and Network Data Models*.

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 2.3.7.2.

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

Example 2–20 defines a network theme that specifies attributes for the display of links and nodes in the network named `NYC_NET`.

Example 2–20 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>
```

2.3.7.1 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 2.8.2). Example 2–21 stores the definition of a network theme.

Example 2–21 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 2–21 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 2–21:

- 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 2–21 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.

2.3.7.2 Using OracleAS MapViewer for Network Analysis

The network model Java API provides several network analysis capabilities. You can define OracleAS 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 2.3.7 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 2–22 defines a network theme that can be used for shortest-path analysis.

Example 2–22 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 2–23 defines a network theme that can be used for within-cost analysis.

Example 2–23 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>
```

2.3.8 Topology Themes

A **topology theme** is a special kind of OracleAS MapViewer theme useful for visualizing topologies defined using the Oracle Spatial 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 Topology and Network Data Models*.

OracleAS 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, OracleAS 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 2–24 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 2–24 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>
```

OracleAS 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 2–25 defines a debug-mode topology theme for rendering features, edges, nodes, and faces from all feature tables in the CITY_DATA topology.

Example 2–25 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>
```

2.3.8.1 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 2.8.2). Example 2–26 stores the definition of a topology theme.

Example 2–26 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 2–26 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 2–26 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.

2.4 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 2.8 and Section 2.8.1. The DEFINITION column in the USER_SDO_MAPS view contains an XML definition of a base map.

Example 2–27 shows a base map definition.

Example 2–27 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 2–27, 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 2–27, 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`.

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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 2–27 in Section 2.4 includes `min_scale` and `max_scale` values.

To determine the current map scale for a map returned by OracleAS 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. OracleAS 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 OracleAS MapViewer style, including advanced styles

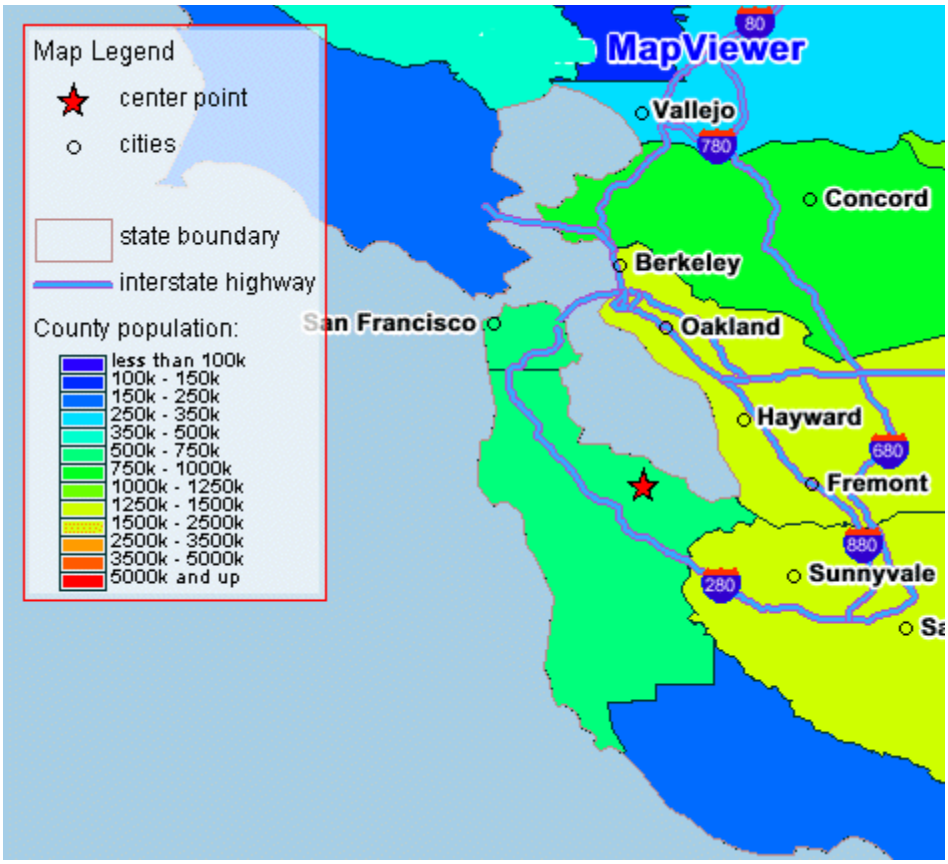
Example 2–28 is an excerpt from a request that includes a legend.

Example 2–28 Legend Included in a Map Request

```
<?xml version="1.0" standalone="yes"?>
<map_request
             basemap="density_map"
             datasource = "mvdemo">
  <center size="1.5">
     . . .
  </center>

  <legend bgstyle="fill:#ffffff;fill-opacity:128;stroke:#ff0000"
          position="NORTH_WEST" font="Dialog">
    <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>

  <themes>
     . . .
  </themes>

</map_request>
```

Figure 2–8 shows a map with the legend specified in Example 2–28.

Figure 2–8 Map with Legend



Notes on Example 2–28 and Figure 2–8:

- 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).

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 OracleAS MapViewer configuration file (`mapViewerConfig.xml`).
- You can define or modify a data source dynamically using the OracleAS 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 and the `max_connections` attribute (both described in Section 1.5.2.12) affect 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 OracleAS 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 OracleAS MapViewer server, the server picks a free renderer associated with the master data source in the request. This section describes the process that the OracleAS MapViewer server follows to generate a map. In brief, OracleAS 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 OracleAS 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 OracleAS MapViewer server.) The XML map request is parsed and its content is validated. OracleAS 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 OracleAS 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, OracleAS 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. OracleAS 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, OracleAS 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 OracleAS 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, OracleAS 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, OracleAS MapViewer plots any additional map features (such as a legend) on the internal map image. OracleAS 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, OracleAS 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 Workspace Manager Support in OracleAS 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 Application Developer's Guide - Workspace Manager*.

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 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 OracleAS MapViewer.

If you specify neither the `workspace_savepoint` nor `workspace_date` attribute, OracleAS MapViewer goes to the latest version of the workspace defined. If you specify both the `workspace_savepoint` and `workspace_date` attributes, OracleAS MapViewer uses the specified date instead of the savepoint name.

Example 2–29 shows the definition of a dynamic theme that uses attributes (shown in bold) related to Workspace Manager support. In this example, OracleAS MapViewer will render the data related to workspace `wsp_1` at the savepoint `sp1`.

Example 2–29 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 OracleAS MapViewer caching of predefined themes (explained in Section 2.3.1.2) and the use of Workspace Manager-related OracleAS 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 OracleAS 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 OracleAS 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 2–30 uses the `<list_workspace_name>` element in an administrative request.

Example 2–30 <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 2–30 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 2–30 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 2–30 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 OracleAS 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 2–31 uses the `<list_workspace_session>` element in an administrative request.

Example 2–31 <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 2–31 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 2–31 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 2–31 will be:

```
<?xml version="1.0" ?>
<non_map_response>
  <workspace_session succeed="false"/>
</non_map_response>
```

2.8 OracleAS 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 OracleAS 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.
- USER_SDO_THEMES and ALL_SDO_THEMES contain information about themes.
- USER_SDO_STYLES and ALL_SDO_STYLES contain information about styles.

> **Note:** You can use the Map Builder tool (described in Chapter 7) to manage most mapping metadata. However, for some features you must use SQL statements to update the OracleAS 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 OracleAS MapViewer metadata views are defined in the following file:

```
$ORACLE_HOME/lbs/admin/mapdefinition.sql
```

The following sections describe each set of views.

2.8.1 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

2.8.2 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

2.8.3 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 OracleAS 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 2–32 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 2–32 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

OracleAS MapViewer Map Request XML API

This chapter explains how to submit map requests in XML format to OracleAS 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 OracleAS 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 OracleAS MapViewer is that a client locates a remote OracleAS MapViewer instance, binds to it, sends a map request, and processes the map response returned by the OracleAS MapViewer instance.

A request to the OracleAS 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 OracleAS MapViewer is running.
- *`port`* is the port on which the Web server listens.
- *`MapViewer-servlet-path`* is the OracleAS 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 7.

For the current release, OracleAS 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
- OracleAS 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 3.1.11, "Java Program Using OracleAS MapViewer"
- Section 3.1.12, "PL/SQL Program Using OracleAS 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 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, OracleAS 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 OracleAS 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="tiger"
                           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 3.2.15 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 2.3.5.)

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>
```

OracleAS MapViewer processes the request in Example 3–6 as follows:

1. OracleAS 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. OracleAS MapViewer checks its internal list of all registered image renderers to see if one supports the ECW format (`image_format="ECW"`). Because OracleAS 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. OracleAS 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. OracleAS 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, OracleAS MapViewer converts all geometries from the data SRID to the map request SRID.

If no coordinate system is associated with the theme data, OracleAS 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 OracleAS 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.

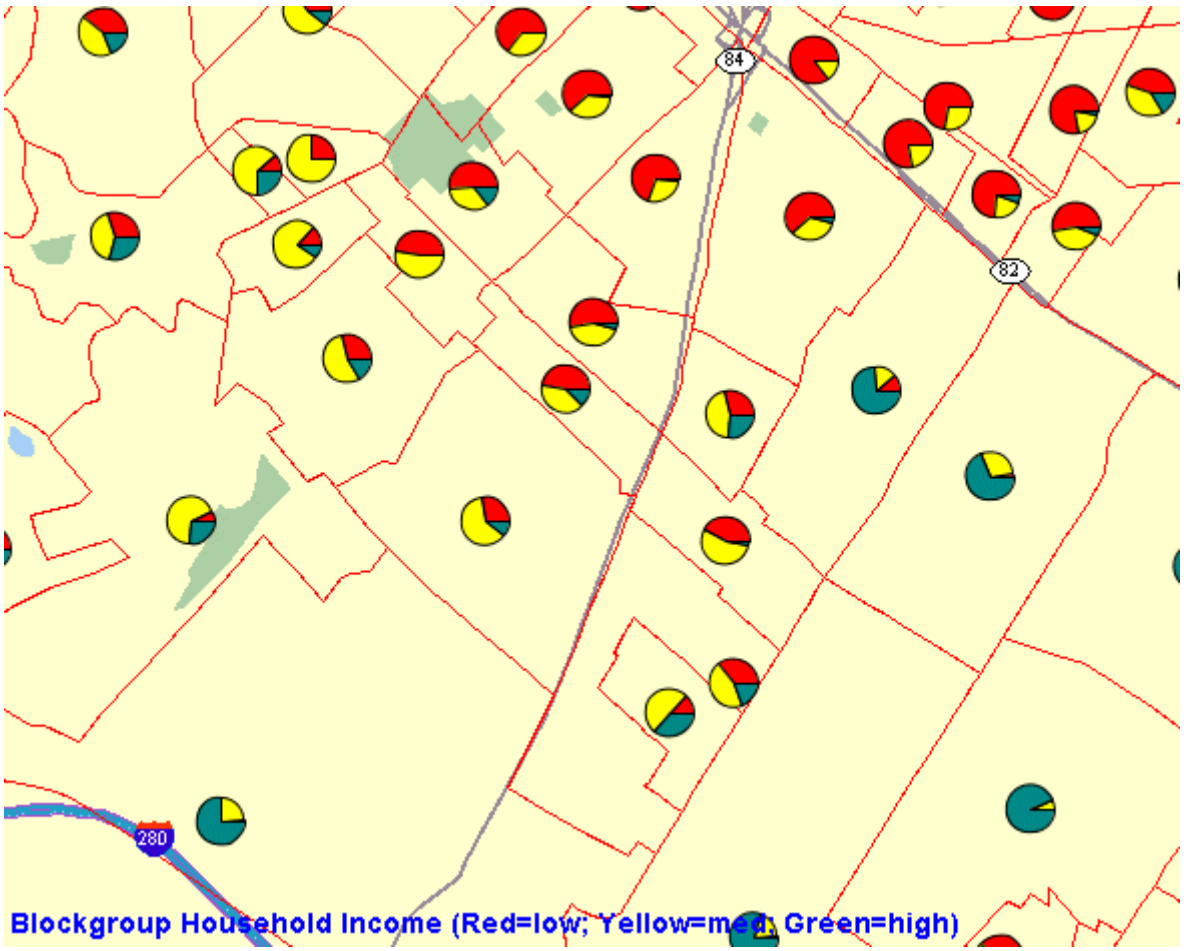
3. Issue a map request that uses the theme that you created. Example 3–9 requests a map that uses the `THEME_PIE_CHART` theme that was created in Step 2.

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 Java Program Using OracleAS MapViewer

Example 3–12 uses the `java.net` package to send an XML request to OracleAS MapViewer and to receive the response from OracleAS 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 3–12 Java Program That Interacts with OracleAS MapViewer

```
import java.net.*;
import java.io.*;

/**
 * A sample program that shows how to interact with OracleAS MapViewer
 */
public class MapViewerDemo
{
    private HttpURLConnection mapViewer = null;

    /**
     * Initializes this demo with the URL to the OracleAS 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 OracleAS MapViewer.
     * @param xmlreq   the XML document that is an OracleAS MapViewer request
     */
    public void submitRequest(String xmlreq)
    {
        try
        {
            mapViewer.setRequestMethod("POST");  //Use HTTP POST method.
            OutputStream os = mapViewer.getOutputStream();
            //OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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);
    }
```

```
}
```

3.1.12 PL/SQL Program Using OracleAS MapViewer

Example 3–13 is a sample PL/SQL program that sends an XML request to the OracleAS MapViewer server. This example works only on Oracle9*i* Release 9.0.1 and later releases.

Example 3–13 PL/SQL Program That Interacts with OracleAS 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?, 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|
                 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"
 >
<!ELEMENT center (geoFeature)>
<!ATTLIST center
    size CDATA #REQUIRED
>
<!ELEMENT box (coordinates) >
<!ATTLIST box
    ID CDATA #IMPLIED
    srsName CDATA #REQUIRED
>
<!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_cache_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
    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_cache_theme (#PCDATA)>
<!ATTLIST map_cache_theme
    map_cache_name       CDATA # REQUIRED
    snap_to_cache_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 styles (style+) >
<!ELEMENT style (svg | AdvancedStyle)?>
<!ATTLIST style
    name  CDATA #REQUIRED
>
```

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 3.2.13, "style Element"
- Section 3.2.14, "styles Element"
- Section 3.2.15, "theme Element"
- Section 3.2.16, "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*)>
```

The root element of a map request to OracleAS 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 OracleAS MapViewer.

The optional `<srs>` child element is ignored by the current version of OracleAS 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 3.2.16) specifies predefined or dynamically defined themes.

The optional `<styles>` element (see Section 3.2.14) 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.

OracleAS 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|
                 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
  keepthemesorder CDATE #IMPLIED
  use_cached_basemap  (TRUE|FALSE) "FALSE"
  snap_to_cache_scale (TRUE|FALSE) "FALSE"
>
```

`datasource` is a required attribute that specifies a data source. A data source provides information to OracleAS 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, OracleAS 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, OracleAS 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 2.8.1. 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.

`antialiasing` is an optional attribute. When its value is `TRUE`, OracleAS 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), OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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`, OracleAS 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.

`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 run time 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`, OracleAS 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, OracleAS 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 cache instance caches the same base map specified by the basemap attribute, MapViewer tries to use the map images cached by the map cache server to render the map specified by the map request. For information about the map cache 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.

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. OracleAS 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 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, OracleAS 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 extends 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 percent (`border_margin="0.02"`) 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 themes"
             datasource = "tilsmenv"
             basemap="qa_map"
             width="400"
             height="400"
             bgcolor="#a6cae0"
             antialiase="false"
             mapfilename="tilsmq202"
             format="PNG_STREAM">

  <bounding_themes border_margin="0.02">theme1, theme3</bounding_themes>

  <themes>
    <theme name="theme1" min_scale="5.0E7" max_scale="0.0">
       <jdbc_query
         datasource="tilsmenv"
         jdbc_srid="8265"
         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="tilsmenv"
         jdbc_srid="8265"
         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:#0000cc;fill:#3300ff;fill-opacity:128;line-style:L.DPH">
       </g>
     </svg>
```

```
    </style>
    <style name="myText">
      <svg width="1in" height="1in">
         <g class="text" float-width="3.0"
            style="font-style:bold;font-family:Serif;font-size:18pt;fill:#000000">
           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 OracleAS 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
>
```

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.

3.2.4 center Element

The element has the following definition:

```
<!ELEMENT center (geoFeature)>
<!ATTLIST center
   size CDATA #REQUIRED
>
```

The 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, OracleAS 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 OracleAS 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 OracleAS 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`, OracleAS MapViewer labels the features even if two or more labels will overlap in the display of a theme. (OracleAS MapViewer always tries to avoid overlapping labels.) If `label_always_on` is `FALSE` (the default), when it is impossible to avoid overlapping labels, OracleAS 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 3.2.15). 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 OracleAS MapViewer in rendering this feature. This provides a mechanism to override the default value specified for a marker style.

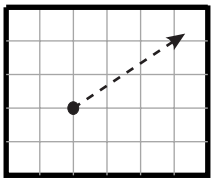
`radius` is an optional attribute. If it is supplied, it specifies a number or a comma-delimited list of numbers, with each number representing the radius of a circle to be drawn centered on this feature. For geodetic data, the unit is meters; for non-geodetic data, the unit is the unit of measurement 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 OracleAS 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 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
    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 2.3.6.

3.2.7 jdbc_image_query Element

The `<jdbc_image_query>` element, which is used to define an image theme (described in Section 2.3.5), 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`, OracleAS MapViewer does not attempt to modify the supplied query string. If `asis` is `FALSE` (the default), OracleAS 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 OracleAS 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`, OracleAS 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 OracleAS 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 2–15 in Section 2.3.5.1, 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 2.3.7.

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
    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.

`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`, OracleAS MapViewer does not attempt to modify the supplied query string. If `asis` is `FALSE` (the default), OracleAS 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 OracleAS 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`, OracleAS 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 2.3.8.

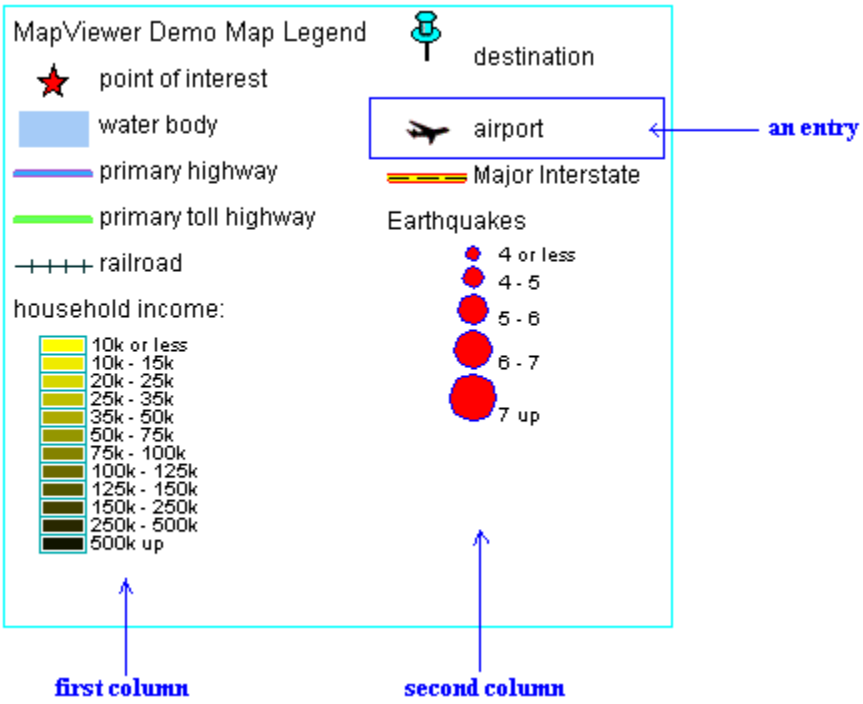
3.2.11 legend Element

The <legend> element has the following definition:

```
<!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
    width         CDATA  #implied
    height        CDATA  #implied
>
```

<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. 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 2–8 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 OracleAS 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 OracleAS 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.

The following example shows the `<legend>` element specification for the legend in Figure 2–8 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_cache_theme Element

The `<map_cache_theme>` element is used to define a map cache theme, which produces a map image layer rendered by the map cache server with pre-generated 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_cache_theme (#PCDATA)>
<!ATTLIST map_cache_theme
    map_cache_name       CDATA # REQUIRED
    snap_to_cache_scale  (TRUE|FALSE) "FALSE"
>
```

`map_cache_name` specifies the name of the map cache instance that caches and serves the map image tiles that are to used to render the map cache theme.

`snap_to_cache_scale` is an optional attribute that specifies whether to adjust the map scale to fit that of one of the cached predefined 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 cached predefined zoom levels, the cached map 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 cached predefined zoom levels when the request map scale does not fit any of the cached predefined zoom levels.

3.2.13 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>
```

3.2.14 styles Element

The `<styles>` element has the following definition:

```
<!ELEMENT styles (style+) >
```

The `<styles>` element specifies one or more `<style>` elements (described in Section 3.2.13).

3.2.15 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_cache_theme)?,
                 operations? >
<!ATTLIST theme
    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
    generalized_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
>
```

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 2.3.6), `<jdbc_network_query>` (described in Section 2.3.7), or `<jdbc_topology_query>` (described in Section 2.3.8).
- For a GeoRaster theme, you can define some image processing options (described in Section 3.2.19).

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 2.8.2). 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`.)

`label_always_on` is an optional attribute. If it is set to `TRUE`, OracleAS MapViewer labels all features of the theme even if two or more labels will overlap in the display. (OracleAS MapViewer always tries to avoid overlapping labels.) If `label_always_on` is `FALSE` (the default), when it is impossible to avoid overlapping labels, OracleAS 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), OracleAS 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 OracleAS 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`, OracleAS MapViewer uses the generic JDBC conversion algorithm. This process is slower than OracleAS 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 2.3.8. 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`, OracleAS 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 theme during rendering. The value can be from 0 to 1, with 0 meaning completely transparent and 1 (the default) meaning completely opaque (no transparency).

generalized_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 documentation about the `min_resolution` and `max_resolution` options for the SDO_FILTER operator.) The unit for `generalized_pixels` is screen pixels. For example, `generalized_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 2.7.

3.2.16 themes Element

The `<themes>` element has the following definition:

```
<!ELEMENT themes (theme+) >
```

The `<themes>` element specifies one or more `<theme>` elements (described in Section 3.2.15). 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.

3.2.17 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 3.2.15).

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>
```

3.2.18 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 3.2.19.) 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.

3.2.19 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 2.3.6). 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 3–14 specifies the normalization operation with a GeoRaster theme.

Example 3–14 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)
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)

3.2.20 operations Element

The `<operations>` element specifies one or more `<operation>` elements (described in Section 3.2.19). 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 3–14 shows the styling rules for a GeoRaster theme that uses the normalization operation.

Example 3–15 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>
```

3.3 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 OracleAS 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 an OracleAS 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 3–16 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 3–16 OracleAS 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 3–16 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 3–17 shows a map response.

Example 3–17 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 OracleAS 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)

OracleAS MapViewer supports the Geometry DTD as defined in the Open Geospatial Consortium (OGC) GML v1.0 specification. This specification and other, more recent, versions are available at the following URL:

http://www.opengeospatial.org/specs/

This specification has the following copyright information:

```
Copyright  ©  2000 OGC  All Rights Reserved.
```

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:

```
THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS
```

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

OracleAS MapViewer JavaBean-Based API

This chapter describes the JavaBean-based OracleAS MapViewer API. This API exposes all capabilities of OracleAS MapViewer through a single JavaBean, `oracle.lbs.mapclient.MapViewer`. This bean is a lightweight client that handles all communications with the actual OracleAS MapViewer service running on the middle tier on behalf of a user making map requests.

All communications between the bean and the actual OracleAS 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 OracleAS MapViewer bean is easier than manipulating XML documents for forming and sending OracleAS MapViewer requests, as well as for extracting information from the responses.

The bean delegates most of map data processing and rendering to the OracleAS MapViewer service. All the bean does is formulate user requests into valid OracleAS MapViewer XML requests and send them to an OracleAS MapViewer service for processing.

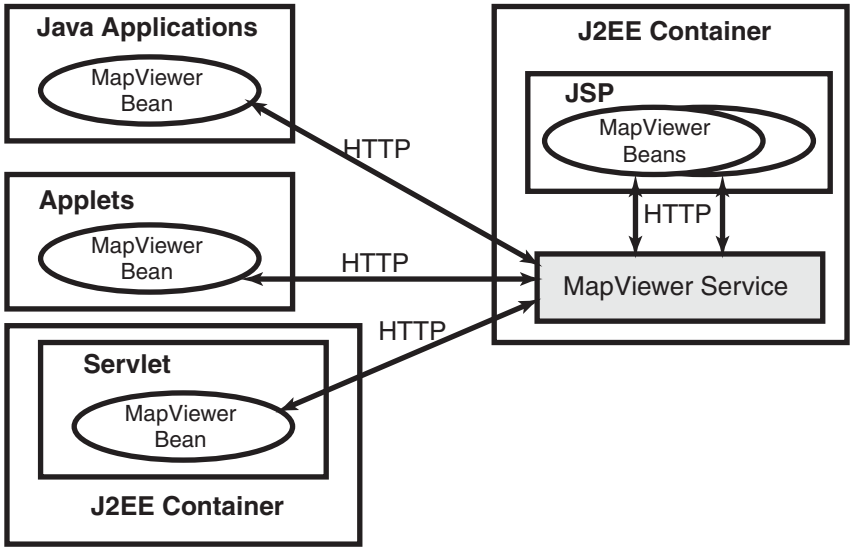
This chapter contains the following major sections:

4.1 Usage Model for the OracleAS MapViewer JavaBean-Based API

The OracleAS 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 OracleAS MapViewer service and the current map request and map response objects. In most cases, you will create only one OracleAS 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 OracleAS 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 OracleAS MapViewer bean.

Figure 4–1 OracleAS MapViewer Bean Usage Scenarios



The OracleAS MapViewer bean can communicate through the HTTP protocol with the OracleAS 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 2 Enterprise Edition (J2EE) container different from the J2EE container that contains the OracleAS MapViewer service
- In JavaServer Pages (JSP) code within the J2EE container that contains the OracleAS 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 an OracleAS 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 OracleAS 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 OracleAS MapViewer JavaBean-Based API

Before you can use the OracleAS MapViewer JavaBean, the OracleAS 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 Application Server, 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 *$OC4J_HOME*`/j2ee/home/applications`, the base directory for unpacked OracleAS MapViewer is *$OC4J_HOME*`/j2ee/home/applications/mapviewer`.)

Before you use the OracleAS MapViewer JavaBean, you should examine the Javadoc-generated API documentation and try the JSP demo:

- Javadoc documentation for the OracleAS 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 Application Server listens for incoming requests.

- A demo supplied with OracleAS MapViewer shows how to use the bean. After you have set up the OracleAS 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 Application Server listens for incoming requests. This JSP page confirms the OracleAS MapViewer service URL and then proceeds to the real demo page, `map.jsp`.

4.3 Using the OracleAS MapViewer Bean

To use the OracleAS 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 OracleAS 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 OracleAS MapViewer Bean

The first step in any planned use of the OracleAS 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 OracleAS MapViewer service. Unless you change it to something else using `setServiceURL`, the OracleAS MapViewer service at this URL will receive all subsequent requests from this bean. When an OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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), OracleAS 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), OracleAS 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 OracleAS 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 OracleAS 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 2.3.6 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 2.3.5 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 2.3.7 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 2.3.8 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 2.3.8 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 OracleAS 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 OracleAS 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 OracleAS MapViewer client.
- `addImageMarkerStyleFromURL(java.lang.String styleName, java.lang.String imgURL)` adds a GIF image as a marker symbol to the OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS MapViewer labels all features in a theme even if two or more labels will overlap in the display of a theme. (OracleAS MapViewer always tries to avoid overlapping labels.) If `labelAlwaysOn` is `TRUE`, OracleAS 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, OracleAS 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 OracleAS 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 OracleAS MapViewer. The OracleAS 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 OracleAS 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 OracleAS 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 OracleAS MapViewer service with too many rapid requests from the OracleAS MapViewer bean or beans. The OracleAS 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, OracleAS 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 OracleAS MapViewer service actually receives the request and processes it.

The methods for sending a map request to the OracleAS MapViewer service include the following:

- `run` sends the current map request to the OracleAS MapViewer service, and obtains a map response as sent back by the OracleAS 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 OracleAS MapViewer service. After the OracleAS MapViewer bean receives the response from the OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 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 operator to perform the filtering; however, you can use the OracleAS 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 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 OracleAS MapViewer bean in a JavaServer Page in an HTML file, a second round-trip to the OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS MapViewer service with a clickable theme, OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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

OracleAS MapViewer JSP Tag Library

This chapter explains how to submit requests to OracleAS 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) OracleAS MapViewer capabilities, such as setting up a map request, zooming, and panning, as well as identifying nonspatial attributes of user-clicked features.

Note: The OracleAS 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 OracleAS MapViewer JSP tags in the same session scope share access to a single OracleAS MapViewer bean.

This chapter contains the following major sections:

- Section 5.1, "Using OracleAS MapViewer JSP Tags"
- Section 5.2, "OracleAS MapViewer JSP Tag Reference Information"
- Section 5.3, "JSP Example (Several Tags) for OracleAS MapViewer"

5.1 Using OracleAS MapViewer JSP Tags

Before you can use OracleAS 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 OracleAS MapViewer.

1. If the Web application will be deployed in the same OC4J instance that is running OracleAS 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 OracleAS 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 OracleAS MapViewer tag library, and its value must be `http://xmlns.oracle.com/spatial/mvtaglib`, because it is so defined in the OracleAS MapViewer `web.xml` initialization file.
   - `prefix` identifies the prefix for tags on the page that belong to the OracleAS 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 OracleAS 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 OracleAS MapViewer operations:

- To create the OracleAS MapViewer bean and place it in the current session, use the init tag, which must come before any other OracleAS 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 OracleAS MapViewer JSP Tag Reference Information

This section provides detailed information about the Oracle-supplied JSP tags that you can use to communicate with OracleAS MapViewer. Table 5–1 lists each tag and briefly describes the information specified by the tag.

Table 5–1 JSP Tags for OracleAS MapViewer

Tag Name	Explanation
init	Creates the OracleAS MapViewer bean and places it in the current session. Must come before any other OracleAS 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 OracleAS 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 OracleAS 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 run time, 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 OracleAS 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`, OracleAS MapViewer does not attempt to modify the supplied query string. If `FALSE` (the default), OracleAS 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 3.2.15.) The predefined theme is added at the end of the theme list maintained in the associated OracleAS 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 OracleAS 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 OracleAS MapViewer service. This map image URL is kept in the associated OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS MapViewer bean and places it in the current session. This bean is then shared by all other OracleAS MapViewer JSP tags in the same session. The `init` tag must come before any other OracleAS 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 OracleAS 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 Application Server or OC4J listens.	Yes
`datasource`	Name of the OracleAS 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 1.5.2.12.)	Yes
`id`	Name that can be used to refer to the OracleAS 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 OracleAS MapViewer service, and a URL for that image is returned to the associated OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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`, OracleAS 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 OracleAS MapViewer. The definition of a base map is stored in the user's USER_SDO_MAPS view, as described in Section 2.8.1. 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 run time 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), OracleAS 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 OracleAS MapViewer

This section presents an example of using JSP code to perform several OracleAS MapViewer operations.

Example 5–1 initializes an OracleAS MapViewer bean, sets up map request parameters, issues a request, and displays the resulting map image. It also obtains the associated OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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

OracleAS MapViewer PL/SQL API

This chapter describes the PL/SQL application programming interface (API) to OracleAS MapViewer. This API consists of the PL/SQL package SDO_MVCLIENT, which is intended for Oracle Database users who want to access OracleAS MapViewer services from inside an Oracle database. This package exposes most capabilities of OracleAS MapViewer, and it handles all communication with the actual OracleAS 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/manager mvclient.jar
```

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 OracleAS 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/tiger
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

OracleAS MapViewer Administrative Requests

The main use of OracleAS MapViewer is for processing various map requests. However, OracleAS MapViewer also accepts various administrative (non-map) requests, such as to add a data source, through its XML API. For all OracleAS MapViewer administrative requests except for those that list base maps, themes, or styles, you must log in to the OracleAS MapViewer administration (Admin) page, for which there is a link on the main OracleAS MapViewer page. This section describes the format for each administrative request and its response.

All administrative requests are embedded in a `<non_map_request>` element, while all administrative responses are embedded in a `<non_map_response>` element, unless an exception is thrown by OracleAS MapViewer, in which case the response is an `<oms_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
- Listing All Maps
- Listing Themes
- Listing Styles
- Managing Cache
- Editing the OracleAS MapViewer Configuration File
- Restarting the OracleAS MapViewer Server

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 1.5.2.12.)

7.1.1 Adding a 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 OracleAS 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 J2EE 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.

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 OracleAS 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 OracleAS 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, OracleAS 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 OracleAS 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="tiger"
        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 OracleAS 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

The `<remove_data_source>` 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 OracleAS MapViewer).

Removing a data source only affects the ability of OracleAS 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="tiger"/>
</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

For convenience, OracleAS 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

The `<list_data_sources>` element lists all data sources known to the currently running OracleAS 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, all data source information except the password for the database user is returned.

The following example is a response that includes information about two data sources.

```
<?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>
```

7.1.5 Checking the Existence of a Data Source

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 Listing All Maps

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>
```

7.3 Listing Themes

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.

7.4 Listing Styles

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.

7.5 Managing Cache

OracleAS MapViewer uses two types of cache:

- Metadata cache for mapping metadata, such as style, theme, and base map definitions
- Spatial data cache for theme data (the geometric and image data used in generating maps)

The use of these caches improves performance by preventing OracleAS MapViewer from accessing the database for the cached information; however, the OracleAS MapViewer displays might reflect outdated information if that information has changed since it was placed in the cache.

If you want to use the current information without restarting OracleAS MapViewer, you can clear (invalidate) the content of either or both of these caches. If a cache is cleared, the next OracleAS MapViewer request will retrieve the necessary information from the database, and will also store it in the appropriate cache.

7.5.1 Clearing Metadata Cache for a Data Source

As users request maps from a data source, OracleAS MapViewer caches such mapping metadata as style, theme, and base map definitions for that data source. This prevents OracleAS MapViewer from unnecessarily accessing the database to fetch the mapping metadata. However, modifications to the mapping metadata do not take effect until OracleAS MapViewer is restarted.

If you want to use the changed definitions without restarting OracleAS MapViewer, you can request that OracleAS MapViewer clear (that is, remove from the cache) all cached mapping metadata for a specified data source. Clearing the metadata cache forces OracleAS MapViewer to access the database for the current mapping metadata.

The `<clear_cache>` element clears the OracleAS 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 OracleAS MapViewer metadata cache.

The following example clears the metadata for the `mvdemo` data source from the OracleAS 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>
```

7.5.2 Clearing Spatial Data Cache for a Theme

OracleAS 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. (OracleAS MapViewer does not cache 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 OracleAS MapViewer is still using copies of the data from the cache. To view the modified theme data without having to restart OracleAS 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>
```

7.6 Editing the OracleAS MapViewer Configuration File

The `<edit_config_file>` element lets you edit the OracleAS 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 OracleAS 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 OracleAS 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 OracleAS 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 OracleAS MapViewer server (see Section 7.7).

7.7 Restarting the OracleAS MapViewer Server

In general, the safest method for restarting the OracleAS MapViewer server is to restart its containing OC4J instance. However, if you are running OracleAS 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 OracleAS 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 OracleAS MapViewer.

This chapter contains the following major sections:

8.1 Overview of Oracle Maps

Oracle Maps consists of the following main components:

- A map cache 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
- 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 cache 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 base map. Because the map image tiles are pre-generated 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, 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 base map enable you to build Web mapping applications.

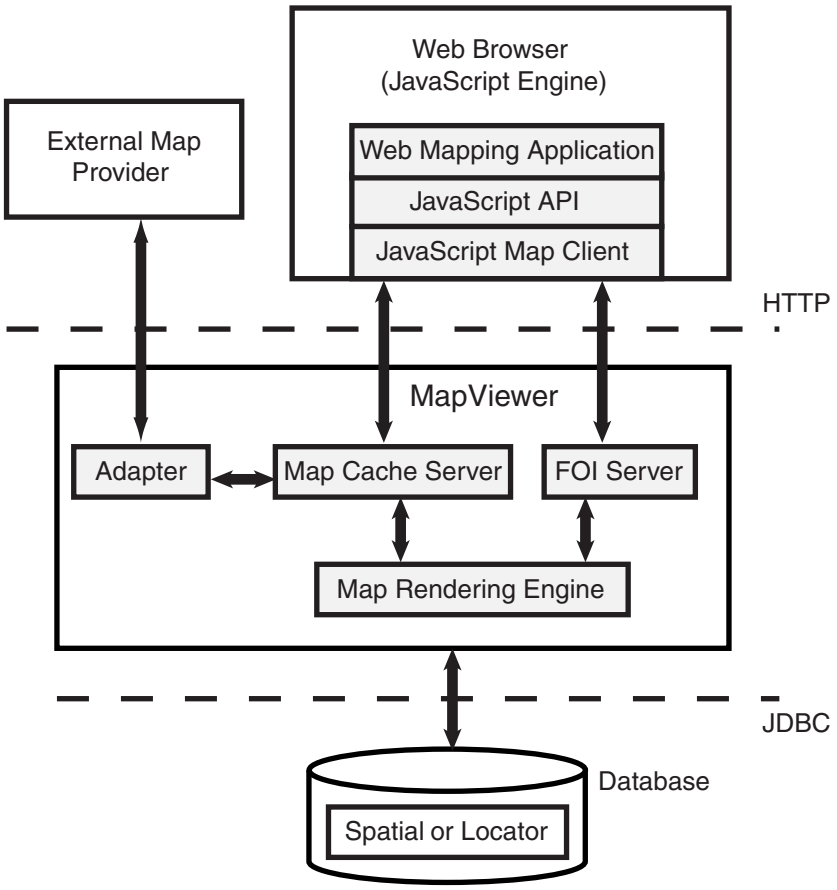
The JavaScript mapping client is a browser side map display engine that fetches map content from the servers and presents it to client applications. It also provides customizable map-related user interaction control, such as map dragging and clicking,

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 cache 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 base map.
- The JavaScript map client controls map-related user interaction for the application.
- When the map cache 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 cache server fetches the tile into the cache and returns it to the client. Tiles can be fetched either directly from

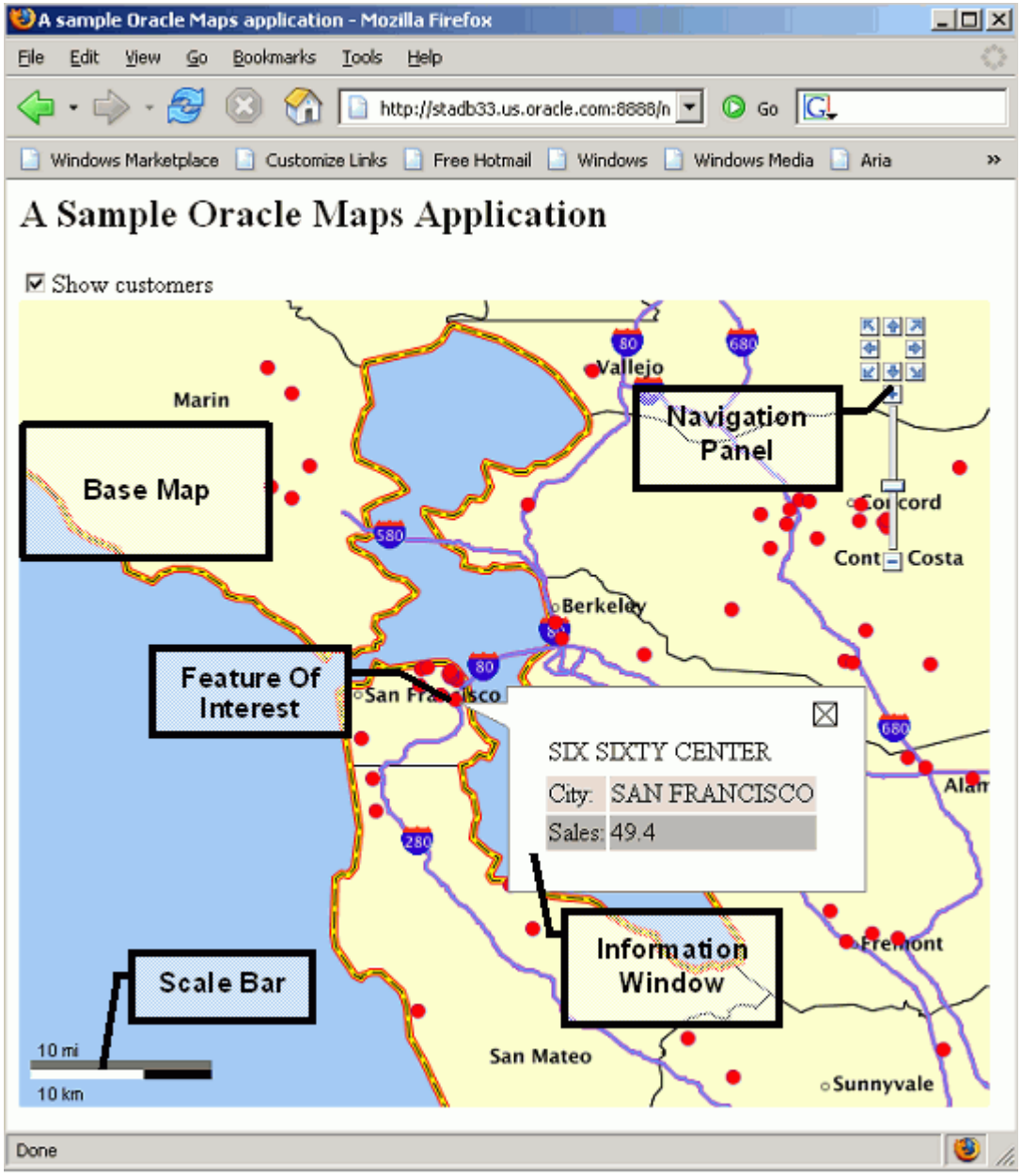
the OracleAS MapViewer map rendering engine or from an external Web map services provider.

- When the FOI server receives a request, it uses the OracleAS 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 Oracle Maps. This example is shipped with MapViewer, and can be accessed at `http://host:port/mapviewer/fsmc/sampleApp.html`. To run this application, follow the instructions in `http://host:port/mapviewer/fsmc/tutorial/setup.html` to set up the database schema and the necessary map cache instances.

Figure 8–2 Application Created Using Oracle Maps



The application shown in Figure 8–2 displays customers on the map. The map consists of two layers:

- The base map layer displays the ocean, county boundaries, cities, and highways. The whole base map displayed in the Web browser consists of multiple map image tiles that are rendered by the map cache server.
- The FOI layer displays customers as red dot markers on top of the base map. 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

```
<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/loadscript.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 base map layer as background.
  mapview.addBaseMapLayer(new MVBaseMap("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>
```

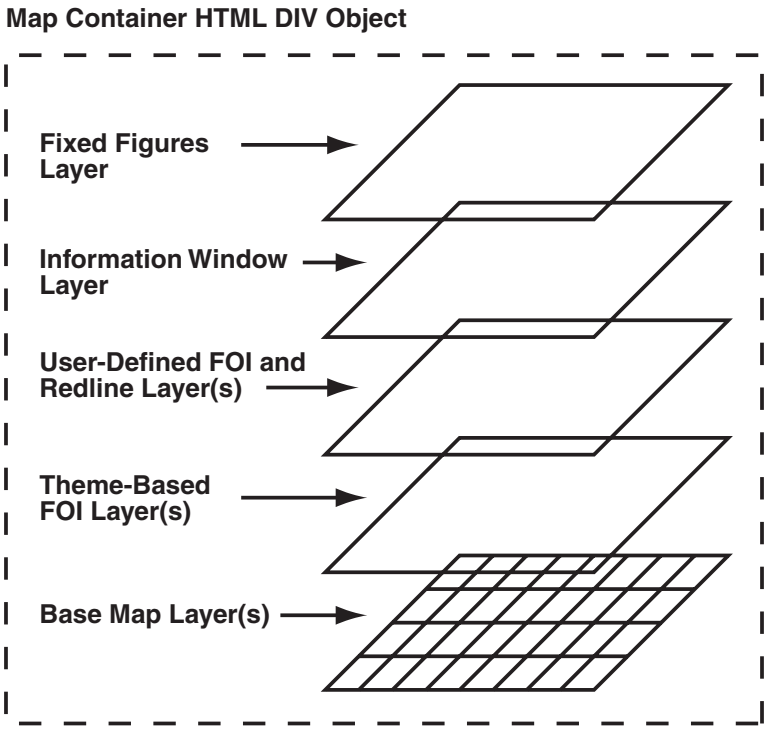
The components of this sample application and the process for creating a client application are described in Section 8.5.3.

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 8–3 shows the layout of the map layers.

Figure 8–3 Layers in a Map



As shown in Figure 8–3, there are five different types of map content layers: base 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 Base Map Layers

A typical Oracle Maps application has at least one base map layer, which assembles and displays pregenerated map image tiles from a map cache instance. The base map 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 base map layer. Only limited user interaction, such as map dragging, can be performed with a base map layer.

A map cache instance is usually associated with a MapViewer base map, and is managed by the MapViewer server; however, you can configure a map cache instance to cache map image tiles from an external (non-MapViewer) map provider.

Base map layers are always placed at the bottom of the layer hierarchy. These layers display static and background map contents. When multiple such layers from different map cache instances 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

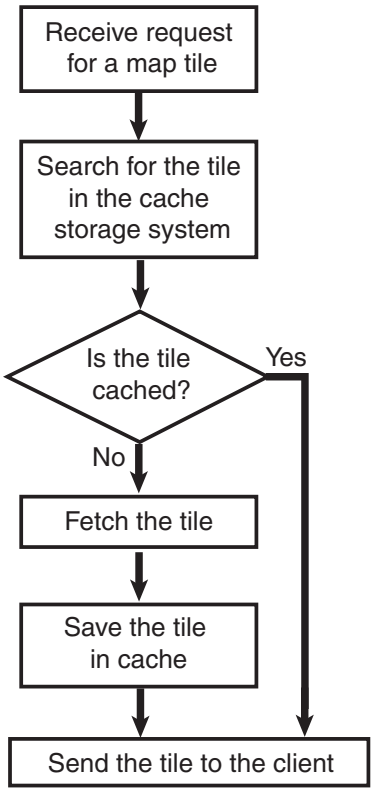
The topmost layer contains any fixed figures, which are immovable elements such as copyright notes, a scale bar, a navigation panel, and user-defined map decoration features. (A user-defined map decoration feature is an application defined element that can contain any custom HTML content, such as a map 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 Cache Server

The map cache 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 OracleAS MapViewer server. The map cache 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 8–4 shows the basic workflow of the map cache server.

Figure 8–4 Workflow of the Map Cache Server



As shown in Figure 8–4, when the map cache 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 cache server sends the tile to the client. If the tile is not cached, the map cache 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 cache server.

8.2.1 Map Cache Server Concepts

This section explains map cache server concepts that you need to know to be able to use Oracle Maps effectively.

8.2.1.1 Map Cache Instances and Base Map Sources

Each base map is managed by a map cache instance inside the map cache server. The **map cache instance** fetches and stores the map image tiles that belong to the base map and returns map image tiles to the client. The map cache server can have multiple map cache instances, each of which manages a different base map.

Each base map 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 base 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 base map 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_USER_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 cache instance can fetch and cache map tiles generated by the external map services provider. (A MapViewer instance other than the MapViewer inside which the Map Cache Server is running is also considered an external map services provider.)

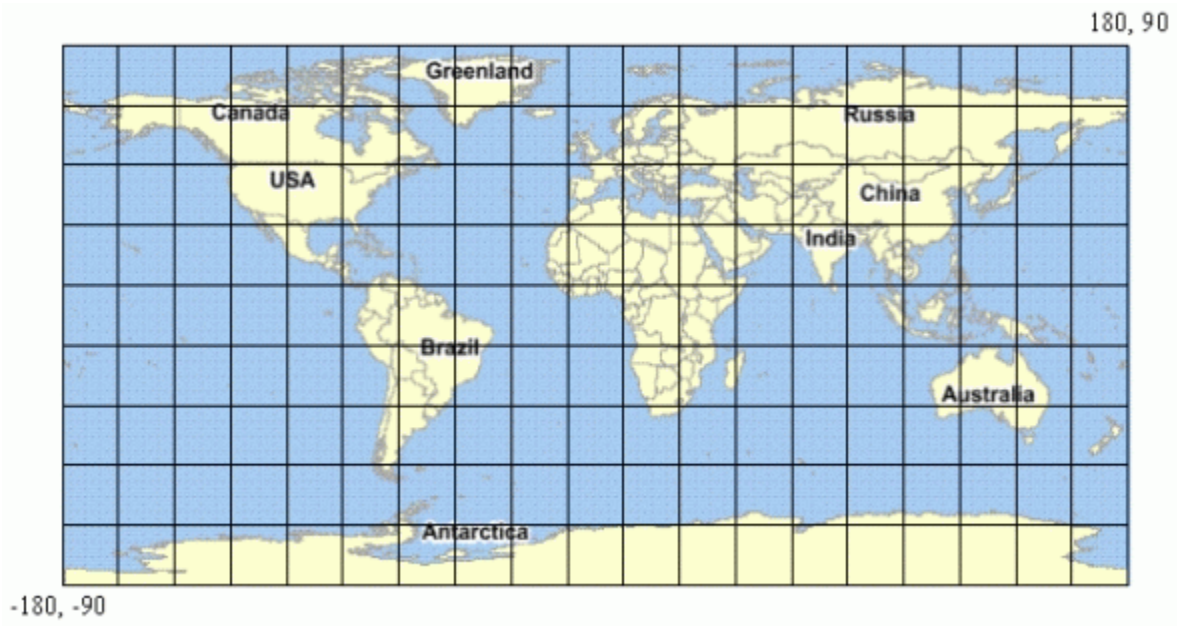
8.2.1.2 Storage of Map Image Tiles

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 cache server configuration settings.

8.2.1.3 Coordinate System for Map Tiles

Map images are cached and managed by Map Cache 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 8–5.

Figure 8–5 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 8–5 represent latitude and longitude). The map cache 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 cache instance configuration settings.

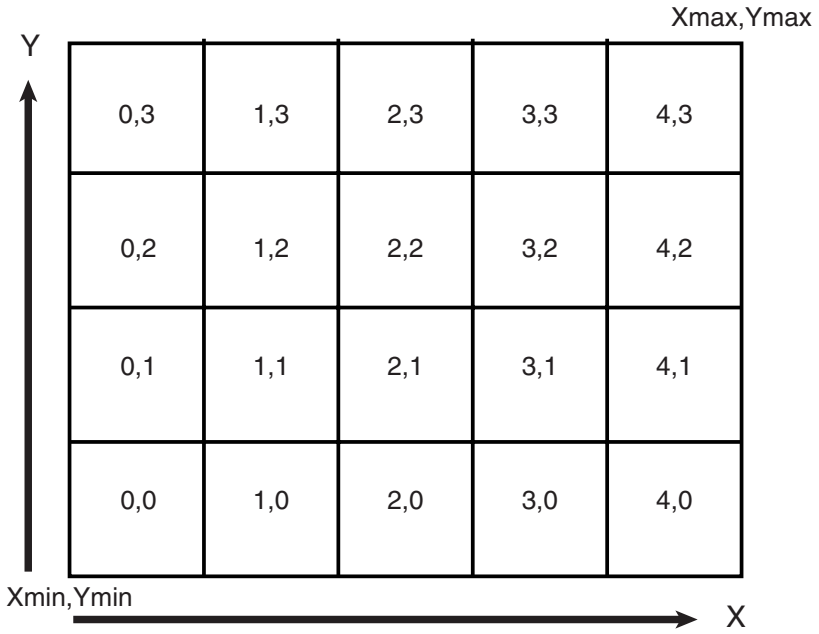
The whole map coordinate system can be represented by a rectangle, and its boundary is specified by (Xmin, Ymin) and (Xmax, Ymax), where Xmin is the minimum X value allowed in the coordinate system, Ymin is the minimum Y value allowed, Xmax is the maximum X value allowed and Ymax is the maximum Y value allowed. In Figure 8–5, 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 cache 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 8–6 shows the mesh codes of the tiles on a map.

Figure 8–6 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 cache 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 cache 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 cache server, as described in Section 8.2.2.

8.2.2 Map Cache Server Configuration

Map cache server configuration settings are stored in local configuration files and in database views. You can customize these settings.

8.2.2.1 Global Map Cache Server Configuration

Global map cache 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 cache server configuration settings are defined in element `<map_cache_server>` inside the top-level `<mapperConfig>` element, as shown in the following example:

```
<map_cache_server>
   <logging
      log_level="info"
      log_thread_name="false"
      log_time="true">
     <log_output name="System.err"/>
    <log_output name="../log/mapcacheserver.log" />
   </logging>
   <cache_storage default_root_path="/scratch/mapcache/"/>
</map_cache_server>
```

The <logging> element specifies the logging options. It has the same syntax and options as the `<logging>` element for MapViewer in the `mapViewerConfig.xml` file, as described in Section 1.5.2.1. However, because the logging information generated by the map cache server is not written to the MapViewer log file, if you want this information to be logged, you must specify the `<logging>` element in the `<map_cache_server>` element.

The `<cache_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/mapcache`. A subdirectory under this directory will be created and used for a map cache instance if the cache instance 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 cache instance.

8.2.2.2 Map Cache Instance Configuration

The configuration settings for a cache instance 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 cache instances.

Each database user (schema) has its own USER_SDO_CACHED_MAPS view. Each entry in this view stores the configuration settings for one map cache instance. If the cache instance is an internal MapViewer cache instance, the base map associated with the cache instance must be defined in the same database schema where the cache instance configuration settings are stored.

The map cache 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 cache 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_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 cache instance is online, or `NO` if the map cache instance is offline. When a tile is missing from the cache and the map cache instance is online, the map cache server will fetch the tile and return the fetched tile to the client. When a tile is missing and the map cache instance is offline, the map cache 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 cache instance, 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:

```
<cache_instance
   image_format ="tile-image-format">
  <internal_map_source
    base_map="name-of-MapViewer-base-map"
    bgcolor="base-map-background-color"/>
  </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>
  <cache_storage
     root_path="disk-path-of-cache-root-directory"
  </cache_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>
    <cache_bound>
       <coordinates> … </coordinates>
    </cache_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-
                       tile-width-specified-in-map-data-units-at-"
       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">
        <cache_bound>
          <coordinates> … </coordinates>
        </cache_bound>
      </zoom_level>
      …
    </zoom_levels>
  </cache_instance>
```

Example 8–2 shows the XML definition of an internal map cache instance, and Example 8–3 shows the XML definition of an external map cache instance. Explanations of the `<cache_instance>` element and its subelements follow these examples.

Example 8–2 XML Definition of an Internal Map Cache Instance

```
<?xml version = '1.0'?>
<!-- XML definition of an internal map cache instance.
-->
<mapcache_config>
   <cache_instance image_format="PNG">
      <internal_map_source base_map="demo_map"/>
      <cache_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>
   </cache_instance>
</mapcache_config>
```

Example 8–3 XML Definition of an External Map Cache Instance

```
<?xml version = '1.0'?>
```

```
<!-- XML definition of an external map cache instance.
-->
<mapcache_config>
   <cache_instance 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>
      <cache_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 cache server will
         automatically generate the <zoom_level> elements for the 10
         zoom levels.
      -->
      <zoom_levels levels="10" min_scale="5000" max_scale="10000000" />
   </cache_instance>
</mapcache_config>
```

The top-level element is `<cache_instance>`. 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 base 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 cache instance manages map tiles 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 cache instance; 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 base 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 cache instance manages map tiles 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 cache 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 base maps. 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 <cache_storage> element specifies cache storage settings for the cache instance. The optional root_path attribute specifies the file system directory to be used as the root directory of the cache instance 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 cache instance, the map coordinate system can be different from the data coordinate system. If the two are different, the map cache 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 <cache_bound> element specifies the bounding box of the cached map tiles. Map cache instance only fetches tiles inside this box and will return 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 cache instance. 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 cache 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 cache 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 cache 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 cache 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 cache 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 `<cache_bound>` element within the `<zoom_level>` element optionally specifies the bounding box of the cached map tiles for the zoom level. The map cache instance 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 `<cache_bound>` element within the `<zoom_level>`

element, it overrides the overall cache bounding box settings specified by the `<cache_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 cache instance and an external map services provider. When a map image tile needs to be fetched from the external map services provider, the map cache instance 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 cache instance.

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 cache 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 cache instance configuration settings.

- `public Properties getProperties()`

This method returns the provider-specific parameters defined in the map cache instance 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 8–4 shows an external map source adapter.

Example 8–4 External Map Source Adapter

```
/**
 * This is a sample map source adapter that can be used to fetch map
 * tiles from an OracleAS 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 8–5 shows the implementation of the `MapSourceAdapter.getTileImageBytes` method.

Example 8–5 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 base map. 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 8–6.

Example 8–6 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 8–6 specify the following. To see how these specifications affect the map display, see Figure 8–2, "Application Created Using Oracle Maps" 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 8–7 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 8–7 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 8–7, 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 8–8 shows some JavaScript client code to create an FOI layer that displays a buffer around each customer location.

Example 8–8 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 cache server and the FOI server. It implements the following functions:

- Fetching map tiles from the map cache server and displaying them as a base map in the Web browser.
- Sending FOI requests to the FOI server, and overlaying user-defined features and Oracle Spatial query-based features on top of the base map.
- 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, including the following:

- The `MVMapView` class is the main entry point of the API. It implements most of the map control interfaces.
- The `MVBaseMap` class defines a base map layer that displays map tiles rendered by a map cache instance.
- 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.
- 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 base map 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.

For detailed information about all classes in the Oracle Maps JavaScript API, see the Javadoc-style reference documentation, which is included with MapViewer and is available at the following location:

`http://`*`host:port`*`/mapviewer/fsmc/apidoc`

8.5 Developing Oracle Maps Applications

If you have all your map data stored in an Oracle database and have MapViewer deployed in Oracle Application Server, you can develop a Web-based mapping application using Oracle Maps by following the instructions in this section.

8.5.1 Creating One or More Map Cache Instances

An Oracle Maps application usually displays at least one base map layer as the static map background. You must create a map cache instance for each base map. For example, for the sample application described in Section 8.1.2, you must create a map cache instance for the base map that displays oceans, county boundaries, cities and highways.

Before you can create a map cache instance, 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 cache server.

When the map source is ready, you can create the map cache instance using the MapViewer administration page, as described in Section 1.5.3. When you create the map cache instance, 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 cache instance, 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 cache instance defined with the MapViewer instance.

If your application needs to display more than one base map layer, you must create a map cache instances for each base map.

8.5.2 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.

8.5.3 Creating the Client Application

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/loadscript.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 (base map, FOI layers, and so on), sets the initial map center and zoom level, implements application-specific logic, and displays the map, sets up initial map contents (base map, FOI layers, and so on), 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 base map layer as background.
    mapview.addBaseMapLayer(new MVBaseMap("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
```

9

Oracle Map Builder Tool

This chapter briefly describes the OracleAS 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 OracleAS 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 OracleAS 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:

`-help` displays information about the available options.

`-noconnect` disables the automatic opening of previously created database connections. If you specify `-noconnect`, data sources specified in the `oasmapbuilder.xml` preferences file or the file specified with the `-config` option will not be used; and to use any objects in these data sources, you will need to connect to the data sources. If you do not specify `-noconnect`, data sources specified in the `oasmapbuilder.xml` preferences file or the file specified with the `-config` option will be used.

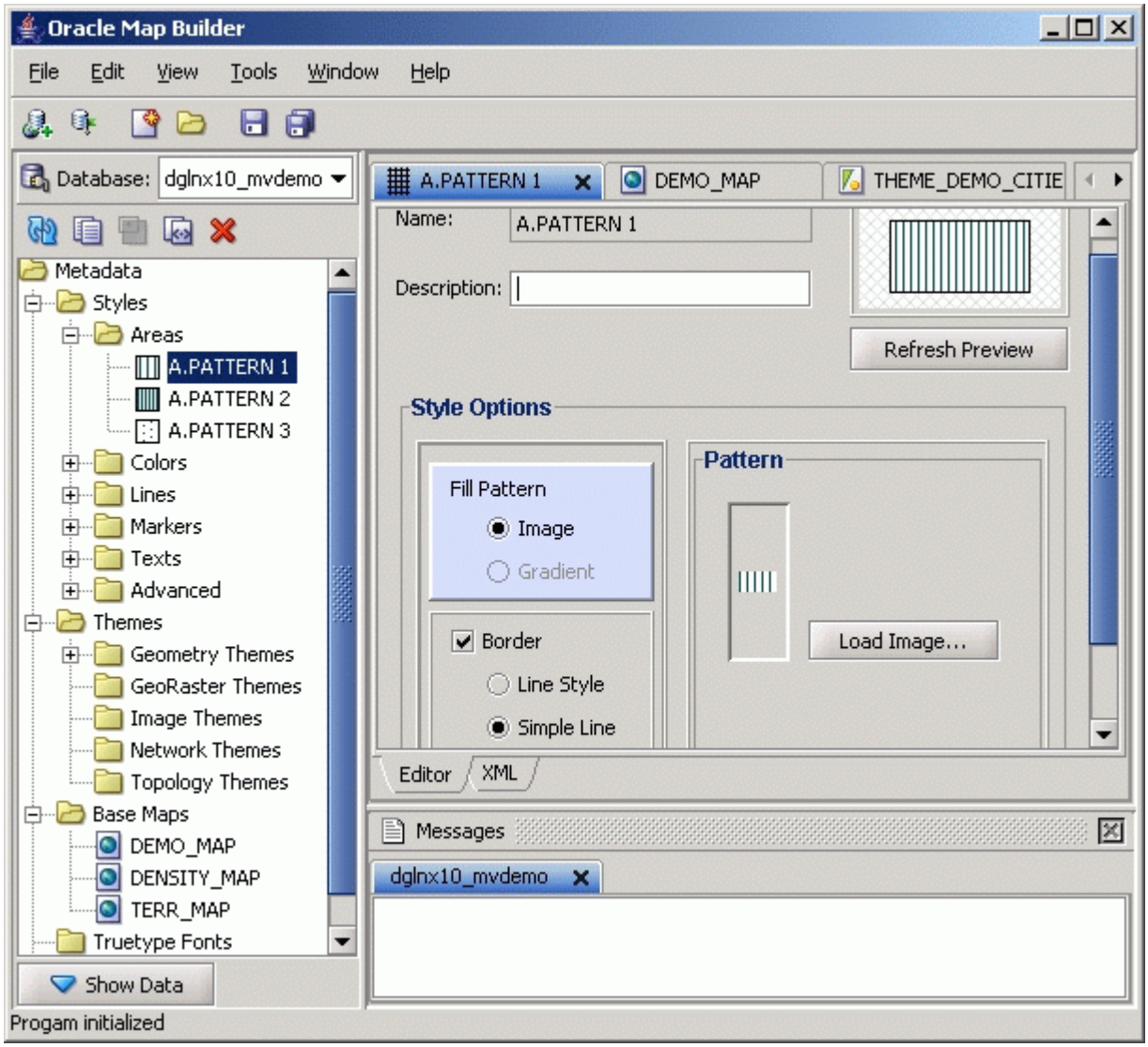
`-config <config-file>` specifies the location of file containing Oracle Map Builder configuration and preference information. If you do not specify this option, Oracle Map Builder looks for a file named `oasmapbuilder.xml` in your home directory.

`-cache <cache-size>` specifies the size of the in-memory geometry cache. For example: `-cache 64M`

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.

A

XML Format for Styles, Themes, and Base Maps

This appendix describes the XML format for defining style, themes, and base maps using the OracleAS MapViewer metadata views described in Section 2.8.

The metadata views for OracleAS 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 OracleAS 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>
```

Note that the OracleAS 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.

The metadata views for OracleAS 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 OracleAS 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"

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.

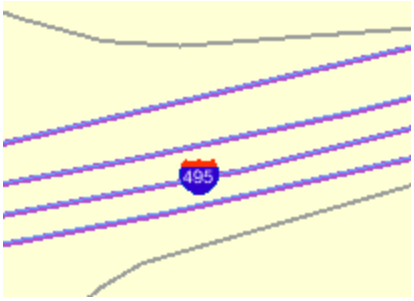
The following example shows the use of a TrueType font-based marker (with TrueType-specific material in bold):

```
<?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>
```

OracleAS 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 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 User's Guide and Reference*.)

When OracleAS 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>
```

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 2.8.3).

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 OracleAS 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 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), and collection (described in Section A.6.6).

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; OracleAS 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 2.2.1.

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>
```

Note that 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 2.2.1.

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.7 Themes: Styling Rules

A theme consists of one or more styling rules. These styling rules are specified in the STYLING_RULES column of the USER_SDO_THEMES metadata view, using the following DTD:

```
<!ELEMENT styling_rules (rule+, hidden_info?)>
<!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
                        network_name     CDATA #IMPLIED
                        network_level    CDATA #IMPLIED
                        topology_name    CDATA #IMPLIED>

<!ELEMENT rule (features, label?)>
<!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
>
```

The `<styling_rules>` element contains one or more `<rule>` elements and an optional `<hidden_info>` element.

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:

- `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 2.3.5.
- `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`, and `polygon_mask` attributes. GeoRaster themes are explained in Section 2.3.6.
- `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 2.3.7.
- `topology` specifies a topology theme. You must also specify the `topology_name` attribute. Topology themes are explained in Section 2.3.8.

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, OracleAS 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 OracleAS 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 OracleAS 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 2.3.1.2.

Each `<rule>` element must have a `<features>` element and can have a `<label>` element.

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 OracleAS 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 2.3.7.) 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 `<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.)

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
               min_scale         CDATA #IMPLIED
               max_scale         CDATA #IMPLIED
               label_always_on   (TRUE|FALSE) "FALSE"
               visible_in_svg    (TRUE|FALSE) "TRUE"
               selectable_in_svg (TRUE|FALSE) "FALSE"
               onclick           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.

Each `<theme>` element must have a `<name>` element, and it can have a scale range (`<min_scale>` and `<max_scale>` elements) and a requirement to display labels even if some labels overlap. Each theme name must be unique. If both the `<min_scale>` and the `<max_scale>` elements are specified for a theme, the `<min_scale>` value must be greater than the `<max_scale>` value. The default for the `<min_scale>` element is positive infinity, and the default for the `<max_scale>` element is negative infinity. If no scale values are specified for a theme, the theme will always be rendered.

`label_always_on` is an optional attribute. If it is set to `TRUE`, OracleAS MapViewer labels all features of the theme even if two or more labels will overlap in the display. (OracleAS MapViewer always tries to avoid overlapping labels.) If `label_always_on` is `FALSE` (the default), when it is impossible to avoid overlapping labels, OracleAS 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.

`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.

`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 only two parameters, 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.

See Section 2.4 for more information about defining base maps and for an example.

B

JavaScript Functions for SVG Maps

This appendix describes the OracleAS 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:

B.1 Navigation Control Functions

The OracleAS 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

OracleAS 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

OracleAS 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

OracleAS 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 3.2.15) 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

OracleAS 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 2.3.5.)

If you want to create a map request specifying an image theme with an image format that is not supported by OracleAS 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 OracleAS 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 *$ORACLE_HOME*`/lbs/lib` directory in an Oracle Application Server environment. If you performed a standalone installation of OC4J, `sdovis.jar` is unpacked into *$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
 * OracleAS MapViewer to support a custom image format.
 */
public interface CustomImageRenderer
{
  /**
   * The method is called by OracleAS 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. OracleAS 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 OracleAS 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 OracleAS 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 OracleAS MapViewer.

After you place your custom implementation classes and any required libraries in the OracleAS MapViewer CLASSPATH, you must register your class with OracleAS 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 OracleAS 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
     OracleAS 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, OracleAS 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 OracleAS 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 OracleAS 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

OGC WMS Support in OracleAS MapViewer

OracleAS MapViewer supports the rendering of data delivered using the Open GIS Consortium (OGC) Web Map Service (WMS) protocol, specifically the WMS 1.1.1 implementation specification. OracleAS MapViewer supports the GetMap, GetFeatureInfo, and GetCapabilities requests as defined in the OGC document 01-068r3.

OracleAS MapViewer does not currently support the optional Styled Layer Descriptor capability, and OracleAS MapViewer will not function as a Cascading Map Server in this release.

This appendix contains the following major sections:

D.1 Setting Up the WMS Interface for OracleAS 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
```

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.

D.1.1 Required Files

The following files are required for OracleAS 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 Application Server 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.

D.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 1.5.2.9.)

D.1.3 SDO to EPSG SRID Mapping File

By default, MapViewer uses the Oracle Spatial (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 4362, 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 1.5.2.9.

D.2 WMS Specification and Corresponding OracleAS MapViewer Concepts

This section describes the association between, or interpretation of, terms and concepts used in the WMS 1.1.1 specification and OracleAS MapViewer. It also includes some parameters that are specific to OracleAS MapViewer but that are not in the WMS 1.1.1 specification.

D.2.1 Supported GetMap Request Parameters

This section describes the supported GetMap request parameters and their interpretation by OracleAS MapViewer. (Parameters that are specific to OracleAS MapViewer and not mentioned in the WMS 1.1.1 specification are labeled OracleAS MapViewer-Only.) The supported parameters are in alphabetical order, with each in a separate subsection. Example D–1 shows a GetMap request.

Example D–1 GetMap Request

```
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
```

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 an OracleAS MapViewer administrative request.

The following optional GetMap parameters are not supported in the current release of OracleAS MapViewer:

- `TIME` (time dimension)
- `ELEVATION` (elevation dimension)
- `SLD` and `WFS` URLs

The OracleAS 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>&
```

D.2.1.1 BASEMAP Parameter (OracleAS 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 example:

```
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
```

D.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`

D.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).

D.2.1.4 DATASOURCE Parameter (OracleAS 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> OracleAS MapViewer administrative request or defined in the OracleAS MapViewer configuration file (mapViewerConfig.xml).

D.2.1.5 DYNAMIC_STYLES Parameter (OracleAS MapViewer-Only)

The DYNAMIC_STYLES parameter specifies a <styles> element as part of the GetMap request. For information about the <styles> element, see Section 3.2.14.

D.2.1.6 EXCEPTIONS Parameter

For the EXCEPTIONS parameter, the only supported value is the default: EXCEPTIONS=application/vnd.ogc.se_xml. That is, only Service Exception XML is supported. The exception is reported as an XML document conforming to the Service Exception DTD available at

http://schemas.opengis.net/wms/1.1.1/WMS_exception_1_1_1.dtd

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.

D.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.

D.2.1.8 HEIGHT Parameter

The HEIGHT parameter specifies the height for the displayed map in pixels.

D.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 an OracleAS 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 D.2.1.1.

D.2.1.10 LEGEND_REQUEST Parameter (OracleAS 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.

D.2.1.11 MVTHEMES Parameter (OracleAS MapViewer-Only)

The `MVTHEMES` parameter specifies a `<themes>` element as part of the GetMap request. For information about the `<themes>` element, see Section 3.2.16. The primary purpose for the `MVTHEMES` parameter is to support JDBC themes in an OracleAS MapViewer request. The `MVTHEMES` parameter is not a substitute or synonym for the `LAYERS` parameter; you still must specify the `LAYERS` parameter.

D.2.1.12 REQUEST Parameter

The `REQUEST` parameter specifies the type of request. The value must be `GetMap`, `GetFeatureInfo`, or `GetCapabilities`.

D.2.1.13 SERVICE Parameter

The `SERVICE` parameter specifies the service name. The value must be `WMS`.

D.2.1.14 SRS Parameter

The `SRS` parameter specifies the spatial reference system (coordinate system) for OracleAS MapViewer to use. The value must be one of the following: `SDO:srid-value` (where `srid-value` is a numeric Oracle Spatial 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`, for projections that have an arbitrary center of projection, is not supported.

D.2.1.15 STYLES Parameter

The `STYLES` parameter is ignored. Instead, use the `LAYERS` parameter to specify predefined themes for the display.

D.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. OracleAS MapViewer does not support transparent GIF (GIF89) images.

D.2.1.17 VERSION Parameter

The `VERSION` parameter specifies the WMS version number. The value must be `1.1.1`.

D.2.1.18 WIDTH Parameter

The `WIDTH` parameter specifies the width for the displayed map in pixels.

D.2.2 Supported GetCapabilities Request and Response Features

A WMS GetCapabilities request to OracleAS MapViewer should specify only the following parameters:

- `REQUEST=GetCapabilities`

- VERSION=1.1.1
- SERVICE=WMS

For example:

```
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&VERSION=1.1.1&SERVICE=
WMS
```

The response is an XML document conforming to the WMS Capabilities DTD available at

http://schemas.opengis.net/wms/1.1.1/WMS_MS_Capabilities.dtd

However, the current release of OracleAS 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 OracleAS 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 OracleAS 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 D–2, which is an excerpt (reformatted for readability) from a GetCapabilities response.

Example D–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 D–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 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 D–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 OracleAS 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.

D.2.3 Supported GetFeatureInfo Request and Response Features

This section describes the supported GetFeatureInfo request parameters and their interpretation by OracleAS MapViewer. Example D–3 shows a GetFeatureInfo request.

Example D–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
```

The response is an XML document and the Content-Type of the response is `text/xml`. Example D–4 is a response to the GetFeatureInfo request in Example D–3.

Example D–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 OracleAS MapViewer and not mentioned in

the WMS 1.1.1 specification are labeled OracleAS MapViewer-Only.) Section D.2.3.10 explains how to query attributes in a GetFeatureInfo request.

D.2.3.1 GetMap Parameter Subset for GetFeatureInfo Requests

A GetFeatureInfo request contains a subset of a GetMap request (`BBOX`, `SRS`, `WIDTH`, `HEIGHT`, and optionally `LAYERS` parameters). These parameters are used to convert the X, Y 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.

D.2.3.2 EXCEPTIONS Parameter

The only supported value for the `EXCEPTIONS` parameter is the default: `application/vnd.ogc.se_xml`. That is, only Service Exception XML is supported. The exception is reported as an XML document conforming to the Service Exception DTD available at

http://schemas.opengis.net/wms/1.1.1/WMS_exception_1_1_1.dtd

D.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;
```

D.2.3.4 INFO_FORMAT Parameter

The value of the `INFO_FORMAT` parameter is always `text/xml`.

D.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.

D.2.3.6 QUERY_TYPE Parameter (OracleAS 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 D.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 D.2.3.7), up to the number matching the value of the `FEATURE_COUNT` parameter value (see Section D.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.

D.2.3.7 RADIUS Parameter (OracleAS 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 D.2.3.6). If you specify the `RADIUS` parameter, you must also specify the `UNIT` parameter (see Section D.2.3.8).

D.2.3.8 UNIT Parameter (OracleAS 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 D.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 D.2.3.7).

D.2.3.9 X and Y Parameters

The `X` and `Y` parameters specify the x-axis and y-axis coordinate values (in pixels), respectively, of the query point.

D.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 D–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>`.

D.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 OracleAS MapViewer.

To add a WMS map theme, use the WMS-specific features of either the XML API (see Section D.3.1) or the JavaBean-based API (see Section D.3.2).

D.3.1 XML API for Adding a WMS Map Theme

To add a WMS map theme to the current map request using the OracleAS 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 an OracleAS MapViewer instance. If you want to specify an OracleAS MapViewer instance (for example, specifying `<service_url>` with a value of `http://mapviewer.mycorp.com:8888/mapviewer/wms`), consider using an OracleAS 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`, or `1.1.1` (the default).
- 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 D–5 shows the `<wms_getmap_request>` element in a map request.

Example D–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>
```

D.3.2 JavaBean-Based API for Adding a WMS Map Theme

To add a WMS map theme to the current map request using the OracleAS 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 an OracleAS 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

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z