Oracle® Application Server 10*g*

MapViewer User’s Guide

10*g* (9.0.4)

Part No. B10559-01

September 2003

Oracle Application Server 10*g* MapViewer User’s Guide, 10*g* (9.0.4)

Part No. B10559-01

Primary Author: Chuck Murray

Contributors: Dan Abugov, Janet Blowney, Clarke Colombo, Dan Geringer, Albert Godfrind, Frank Lee, L.J. Qian, Vishal Rao, Jayant Sharma, Ji Yang

Contents

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Application Server 10*g* MapViewer User’s Guide, 10*g* (9.0.4)

Part No. B10559-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and enough information to identify the context. You can send comments to us in the following ways:

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825   Attn: MapViewer Documentation
- Postal service:

  Oracle Corporation
  MapViewer Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  USA

If you would like a reply, please include your name and contact information.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

Oracle Application Server 10g MapViewer User's Guide describes how to install and use MapViewer, a tool that renders maps showing different kinds of spatial data.

This preface contains these topics:

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. Standards will continue to evolve over time, and Oracle Corporation 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 additional information, visit the Oracle Accessibility Program Web site at
`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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 Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This guide contains the following elements:

Chapter 1, "Introduction to MapViewer"

Explains what MapViewer is, and how to install and configure it.

Chapter 2, "MapViewer Concepts"

Explains important concepts that you must understand to use MapViewer.

Chapter 3, "MapViewer Map Requests"

Explains how to submit XML requests to MapViewer, and it describes the XML document type definitions (DTDs) for the requests (input) and responses (output).

Chapter 4, "MapViewer JavaBean-Based API"

Explains how to use the JavaBean-based MapViewer API, which exposes all capabilities of MapViewer through a single JavaBean.

Chapter 5, "MapViewer JSP Tag Library"

Explains how to submit requests to MapViewer using JavaServer Pages (JSP) tags in an HTML file.

Chapter 6, "MapViewer Administrative Requests"

Explains how to submit various administrative (non-map) requests, such as to add a data source, through the MapViewer XML API.

Chapter 7, "Map Definition Tool"

Explains the console (Map Definition Tool) interface to MapViewer, which you can use to manage mapping metadata (styles, themes, and maps) used by MapViewer.

Appendix A, "XML Format for Styles, Themes, and Base Maps"

Explains the XML format for defining each type of style. (This is intended only for advanced users of MapViewer.)

Appendix B, "Creating and Registering a Custom Image Renderer"

Explains how to implement and register a custom image renderer for use with an image theme.

Appendix C, "Using the Flash Mapping Client"

Explains how to use the Macromedia Flash mapping client to allow users of Flash applications to interact with maps.

Appendix D, "Connection Pools and Java Object Cache in MapViewer"

Describes how MapViewer uses Java Database Connectivity (JDBC) connection pooling and caching of Java objects to provide efficient performance.

Related Documentation

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

- *Oracle Spatial User's Guide and Reference*
- *Oracle9i Database Concepts*

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/membership`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/documentation`

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are also used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface	Boldface type indicates a term defined in the text.
italic	Italic type is used for book titles, emphasis, and some special terms.
`monospace`	Monospace type is used for the names of parameters, elements, attributes, styles, themes, files, and directory paths. It is also used for code examples.
`monospace italic`	Monospace italic type is used to represent placeholders as variables.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

New and Changed Features

This section describes major features that are new or changed since the previous release of MapViewer, which was included in Oracle9*i*AS Release 2 (9.0.2 and 9.0.3).

Java Client API

A new Java client API provides convenient access to most MapViewer functions from a Java application, a Java applet, a servlet within a Java2 Enterprise Edition (J2EE) container different from the J2EE container that contains the MapViewer service, or JavaServer Pages (JSP) code within the J2EE container that contains the MapViewer service. For information about the JavaBean-based API, see Chapter 4.

JSP Tag Library

A set of custom JSP tags can be used to develop JSP applications using MapViewer features. The minimum version requirement for using this JSP tag library is Oracle9*i*AS release 9.0.3 or standalone OC4J release 9.0.3. For information about the MapViewer JSP tag library, see Chapter 5.

PNG Format Support

Maps can now be rendered into PNG format, which provides a richer representation media and faster creation time than the GIF format. The `format` attribute of the `<map_request>` element accepts the `PNG_STREAM` and `PNG_URL` values, as explained in Section 3.2.2.

Map Legend Support

Map legends are supported. The content and layout of a legend can be customized. For information about map legends, see Section 2.4.2.

Query Capabilities with Nonspatial Attributes

A set of methods is provided for querying nonspatial attributes based on locations. See Section 4.3.8 for details.

Basic Support for Georeferenced Images

You can define image themes that can be rendered along with the normal vector themes. By default, MapViewer supports JPEG formatted images, but other formats can be supported through a custom image renderer interface. For information about image themes, see Section 2.3.4. For information about creating and registering a custom image renderer, see Appendix B.

JDK 1.4 and AWT Headless Mode Support

MapViewer can now be run in the JDK 1.4 Java virtual machine (JVM). MapViewer also supports the new headless AWT mechanism in JDK 1.4, which enables MapViewer to run on Linux or UNIX systems without setting any `X11 DISPLAY` variable. To enable AWT headless mode on Linux or UNIX systems, specify the following in the command line to start MapViewer:

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

Performance Improvements

The performance of MapViewer has been improved, particularly in querying and retrieving spatial data.

You may also be able to improve performance further by customizing the spatial geometry data cache, as explained in Section 1.5.6.

1 Introduction to MapViewer

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

1.1 Overview of MapViewer

MapViewer includes the following main components:

- A rendering engine (Java class library) that provides cartographic rendering capabilities (map renderer)
- An Extensible Markup Language (XML) API that provides a programmable interface to MapViewer

The rendering engine connects to the Oracle database through Java Database Connectivity (JDBC). It also loads the map metadata (such as map definitions, styling rules, and symbology) from the database, and applies it to the retrieved spatial data.

The XML API provides high-level application developers with a convenient interface for submitting a map request to MapViewer and handling the map response.

In addition to these components, the Map Definition Tool, an unsupported tool available through the Oracle Technology Network, simplifies the process of creating and managing map, theme, and symbology metadata in a spatial database. For information about the Map Definition Tool, see Chapter 7.

The primary benefit of MapViewer is its integration with Oracle Spatial and Oracle Locator. The current release of MapViewer supports only two-dimensional vector geometries. MapViewer is not a full-featured Web map server or spatial application server.

1.1.1 Basic Flow of Action

With MapViewer, the basic flow of action involves two steps, regardless of whether or not the client requests a map or some MapViewer administrative action.

For a map request:

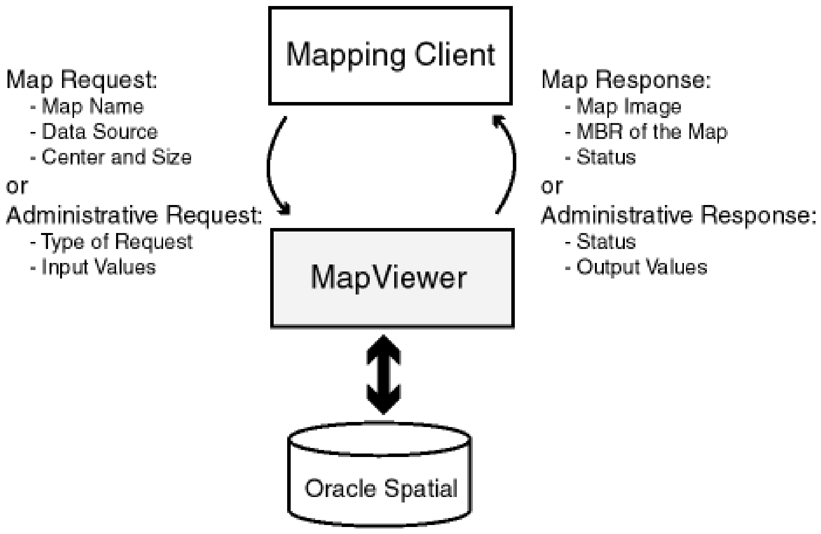
1. The client requests a map, passing in the map name, data source, center location, map size, and, optionally, other data to be plotted on top of a map.
2. The server returns the map image (or a URL for the image) and the minimum bounding rectangle (MBR) of the map, and the status of the request.

For a MapViewer administrative request:

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

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

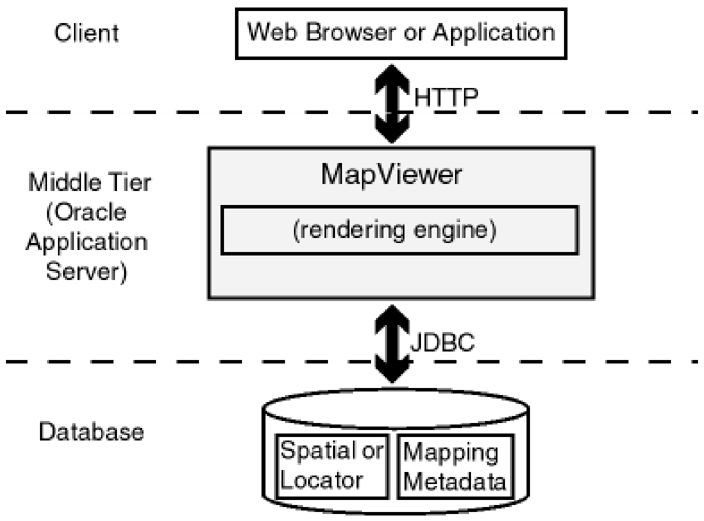
Figure 1–1 Basic Flow of Action



1.1.2 Architecture

Figure 1–2 illustrates the architecture of MapViewer.

Figure 1–2 Architecture



As shown in Figure 1–2:

- MapViewer is part of the Oracle Application Server middle tier.
- MapViewer includes a rendering engine.
- MapViewer can communicate with a client Web browser or application using the HTTP protocol.
- MapViewer performs spatial data access (reading and writing Oracle Spatial and 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 MapViewer

To get started using MapViewer, follow these steps:

1. Either before or after you install and deploy MapViewer, read Chapter 2 to be sure you understand important terms and concepts.
2. Ensure that you have the prerequisite software (see Section 1.3).
3. Install (if necessary) and deploy MapViewer (see Section 1.4).

4. Use MapViewer for some basic tasks, as described in Section 1.6.
5. Optionally, use the Map Definition Tool (described in Chapter 7) to familiarize yourself with styles, themes, and maps, and the options for each.

1.3 Prerequisite Software

To use 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* (9.0.4), or a standalone version of Oracle Application Server Containers for J2EE (OC4J) release 9.0.2 or later (available from the Oracle Technology Network at `http://otn.oracle.com`)
- Oracle Spatial or Oracle Locator (release 8.1.6 or later)
- Oracle Client (release 8.1.7 or later), if you need to use JDBC Oracle Call Interface (OCI) features
- Java JDK (or JRE) 1.2, 1.3, or 1.4.

MapViewer also supports the headless AWT mechanism in JDK 1.4, which enables MapViewer to run on Linux or UNIX systems without setting any `X11 DISPLAY` variable. To enable AWT headless mode on Linux or UNIX systems, specify the following in the command line to start MapViewer:

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

1.4 Installing and Deploying MapViewer

This section describes how to install (if necessary) and deploy MapViewer to run in the middle tier. MapViewer runs as an OC4J Web application and receives map requests from a client.

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

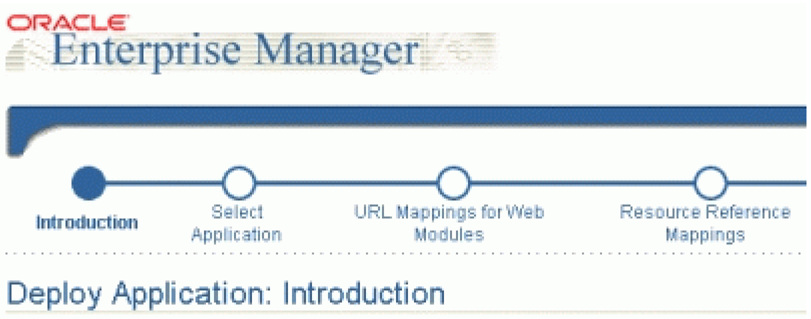
- If you have already installed Oracle Application Server and want to deploy MapViewer, follow the instructions in Section 1.4.1.
- If you have not installed Oracle Application Server but have installed OC4J and now want to install and deploy MapViewer, follow the instructions in Section 1.4.2.

1.4.1 Deploying MapViewer in an Oracle Application Server Environment

If you have already successfully installed Oracle Application Server, you can deploy the MapViewer application using the Oracle Enterprise Manager interface.

Start Oracle Enterprise Manager, navigate to the OC4J instance where you want to deploy MapViewer, and select **Deploy Application** to start a wizard that takes you through the deployment steps. Figure 1–3 shows part of the introductory page for this wizard.

Figure 1–3 Deploying MapViewer: Wizard Introduction Page


ORACLE
Enterprise Manager
Introduction
Select Application
URL Mappings for Web Modules
Resource Reference Mappings
Deploy Application: Introduction

1.4.1.1 Select Application Page

For **J2EE Application**, specify the complete path for the `mapviewer.ear` file.

For **Application**, specify: `mapviewer`

1.4.1.2 URL Mappings for Web Modules Page

For **URL Binding**, specify: `/mapviewer`

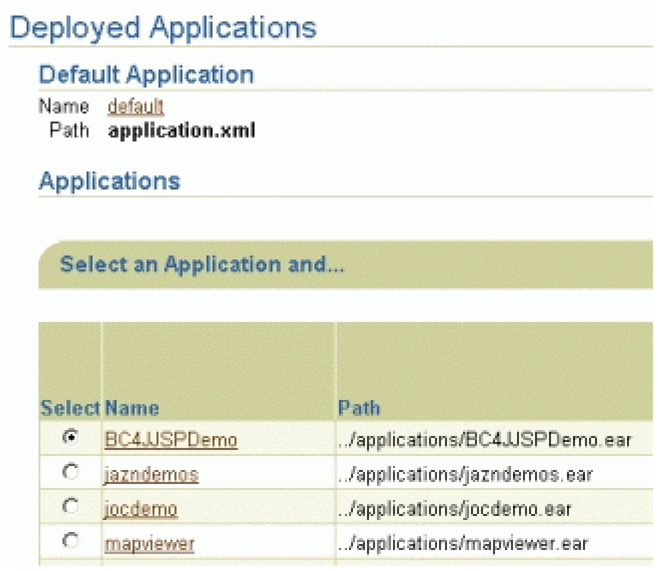
Click **Finish** to go directly to the Summary page.

1.4.1.3 Summary Page

Review the information on the Summary page. If you need to make any changes, go back to the appropriate screen. If the information is correct, click **Deploy**.

Oracle Enterprise Manager deploys `mapviewer.ear`, modifies some XML files, creates a URL binding in the Oracle HTTP listener, and displays a screen with information about deployed applications. Figure 1–4 shows part of this page.

Figure 1–4 Deployed Applications



Deployed Applications

Default Application

Name default

Path **application.xml**

Applications

Select an Application and...

Select	Name	Path
●	BC4JJSPDemo	../applications/BC4JJSPDemo.ear
○	jazndemos	../applications/jazndemos.ear
○	jocdemo	../applications/jocdemo.ear
○	mapviewer	../applications/mapviewer.ear

1.4.1.4 Pages for Completing the Deployment

After you click Deploy on the Summary page, you must perform some steps to associate the `sdovis.jar` file with MapViewer. This section presents these steps.

Note: The `sdovis.jar` file is the core rendering library for MapViewer. It is not packaged as part of the `mapviewer.ear` file, because some other Oracle Application Server components require its functions. Therefore, the `sdovis.jar` file must be accessible even if MapViewer is never deployed (that is, even if the `mapviewer.ear` file is never unpacked).

---

1. In the Deployed Applications section of the page shown in Figure 1–4, click the button (under the Select column) next to **mapviewer** (under the Name column).
2. Click **Edit**.
3. On the next page, in the Administration section, under Properties, click **General**.

4. On the next page, in the Library Paths section, click **Add Another Row**.
5. In the box for the added row, type the path for the `sdovis.jar` file, which is in the *`$ORACLE_HOME/`*`lbs/lib` directory.

   For example: `D:\oracle\ora_Bl\lbs\lib\sdovis.jar`
6. Click **Apply**.
7. Restart the OC4J instance by clicking **Restart** on the OC4J instance page.
8. If the release number of the target Oracle database is 9.0.1 or lower, run SQL scripts to create the MapViewer metadata views and predefined styles (see Section 1.4.2.3).
9. Verify that the deployment was successful (see Section 1.4.2.4).

1.4.2 Installing MapViewer with a Standalone Installation of OC4J

To install and deploy MapViewer with a standalone installation of OC4J, you must have installed OC4J on your system.

Follow these steps to install and deploy MapViewer with a standalone installation of OC4J:

1. If you have not already installed Oracle Application Server Wireless, download the `mapviewer.ear` file to the *`$ORACLE_HOME/`*`lbs` directory. If this directory does not exist, create it.

   You can put the `mapviewer.ear` file in another directory; however, the instructions in this guide assume that the `mapviewer.ear` file is in the *`$ORACLE_HOME/`*`lbs` directory.
2. Edit the OC4J configuration files (see Section 1.4.2.1).
3. Restart OC4J (see Section 1.4.2.2).
4. If the release number of the target Oracle database is 9.0.1 or lower, run SQL scripts to create the MapViewer metadata views and predefined styles (see Section 1.4.2.3).
5. Verify that the deployment was successful (see Section 1.4.2.4).

1.4.2.1 Editing the OC4J Configuration Files to Autostart MapViewer

To start MapViewer automatically each time OC4J is restarted, edit the OC4J configuration files, as follows.

1. Edit *$OC4J_HOME*/config/default-web-site.xml (or http-web-site.xml if you downloaded an OC4J kit from the Oracle Technology Network), where *$OC4J_HOME* should be *$ORACLE_HOME*/j2ee/home. Add a <web-app> element inside the <web-site> element. For example:

   ```
   <web-app application="mapviewer" name="web" root="/mapviewer"
     load-on-startup="true" />
   ```

   The following example shows a sample default-web-site.xml file after the modification.

   ```
   <?xml version="1.0"?>
   <!DOCTYPE web-site PUBLIC "Oracle Application Server XML Web site"
   "http://xmlns.oracle.com/ias/dtds/web-site.dtd">

   <!-- Change the host name below to your own host name. Localhost will -->
   <!-- not work with clustering. -->
   <!-- Also add cluster-island attribute as below.
   <web-site host="localhost" port="8888" display-name="Default Oracle
   Application Server Java WebSite" cluster-island="1" >
   -->

   <web-site port="8888" display-name="Default Oracle Application Server
   Containers for J2EE Web Site">
     <!-- Uncomment the following line when using clustering -->
     <!-- <frontend host="your_host_name" port="80" /> -->
     <!-- The default web-app for this site, bound to the root -->
     <default-web-app application="default" name="defaultWebApp" />

     <!-- Access Log, where requests are logged to -->

     <access-log path="../log/default-web-access.log" />
     <web-app application="mapviewer" name="web" root="/mapviewer"
      load-on-startup="true" />
   </web-site>
   ```

2. Modify $OC4J_HOME/config/server.xml. Add an <application> element inside the <application-server> element. For example:

   ```
   <application name="mapviewer" path="$MAPVIEWER_EAR_PATH" auto-start="true"/>
   ```

   $MAPVIEWER_EAR_PATH should be the full path of the mapviewer.ear file.

   The following example shows a sample server.xml file after the modification.

```
<?xml version="1.0"?>
<!DOCTYPE application-server PUBLIC "Orion Application Server Config"
"http://xmlns.oracle.com/ias/dtds/application-server.dtd">

<application-server application-directory="../applications"
                    deployment-directory="../application-deployments">
  <rmi-config path="./rmi.xml" />
  <!-- JMS-server config link, uncomment to activate the JMS service -->
  <jms-config path="./jms.xml" />
  <log>
    <file path="../log/server.log" />
  </log>

  <global-application name="default" path="application.xml" />

  <global-web-app-config path="global-web-application.xml" />
  <!-- <web-site path="./secure-web-site.xml" /> -->
  <web-site path="./default-web-site.xml" />

  <application name="mapviewer" path="D:\Oracle\Ora817\lbs\mapviewer.ear"
   auto-start="true" />
</application-server>
```

1.4.2.2 Restarting OC4J

If OC4J is already running, you should not need to restart it. Instead, after you save changes made to the OC4J configuration files, OC4J should automatically restart and "hot deploy" MapViewer. In this case, you should see messages such as the following:

```
Auto-unpacking D:\Oracle\Ora817\lbs\mapviewer.ear... done.
Auto-unpacking D:\Oracle\Ora817\lbs\mapviewer\web.war... done.
Installed mapviewer...
[oracle.spatial.mapserver.core.MapperPool, WARN] destroying all mapper
instances.
[oracle.spatial.mapserver.oms, INFO] *** Oracle Spatial MapViewer is
successfully started. ***
[oracle.spatial.mapserver.core.MapRecycleThread, Tue Oct 23 15:46:07 EDT
2001,#Thread-3, INFO] cleansing old maps
```

If OC4J is not running, start OC4J after saving the changes that you made to the OC4J configuration files. OC4J should start to deploy MapViewer.

While it is deploying MapViewer, OC4J extracts the whole MapViewer directory structure from `mapviewer.ear` into the *`$ORACLE_HOME/`*`lbs/mapviewer` directory.

1.4.2.3 Running SQL Scripts, If Necessary

If all target databases are running Oracle Database release 9.2 or a later release, skip this step and go to the next section. A *target database* is a database with Oracle Spatial or Oracle Locator (release 8.1.6 or later) installed and from which you want MapViewer to be able to render maps.

For each target database that is running Oracle Database release 9.0.1 or a previous release, run SQL scripts to create the MapViewer metadata views and predefined styles. While you are connected to the database as the MDSYS user, you must run the first of the following SQL scripts, and it is recommended that you run the second script:

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

The second script (`defaultstyles.sql`) inserts some styles and themes and a base map into the MapViewer metadata views. You can use these styles and themes in applications, and you can also use them as models when you create your own MapViewer metadata objects.

1.4.2.4 Verifying That the Deployment Was Successful

To test if the MapViewer servlet has started correctly, point your browser to that OC4J instance. For example, if MapViewer is installed on a system named `mapserver.xyzabc.com` and the HTTP port is 8888, enter the following URL to invoke the MapViewer servlet without sending it a request:

```
http://mapserver.xyzabc.com:8888/mapviewer/omserver
```

You should use an XML-enabled Web browser, such as Internet Explorer 5.0 or a later version, to see the XML response.

If the servlet has been started and initialized correctly, it generates a response, which will probably be a message such as the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<oms_error>Message:[oms] empty or null xml map request string. Wed Oct 24
12:22:03 EDT 2001 Machine Node Name: mapserver Severity: 0 Description: at
oracle.spatial.mapserver.oms.getXMLDocument(oms.java:379) at
oracle.spatial.mapserver.oms.doPost(oms.java:151) at
oracle.spatial.mapserver.oms.doGet(oms.java:119) at
javax.servlet.http.HttpServlet.service(HttpServlet.java:195) at
javax.servlet.http.HttpServlet.service(HttpServlet.java:309) at
javax.servlet.http.HttpServlet.service(HttpServlet.java:336) at
com.evermind.server.http.ServletRequestDispatcher.invoke(ServletRequestDispatche
r.java:501) at
com.evermind.server.http.ServletRequestDispatcher.forwardInternal(ServletRequest
Dispatcher.java:170) at
com.evermind.server.http.HttpRequestHandler.processRequest(HttpRequestHandler.ja
va:576) at
com.evermind.server.http.HttpRequestHandler.run(HttpRequestHandler.java:189) at
com.evermind.util.ThreadPoolThread.run(ThreadPoolThread.java:62)</oms_error>
```

The preceding display indicates that the servlet has been started and initialized correctly. The apparent errors in the display are normal at this point, because no request was specified in the URL.

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

1.5 Configuring MapViewer

Note: Most readers should skip this section, because after the installation MapViewer is configured to run using the default settings. This section is intended for advanced users who need to customize the MapViewer configuration.

If the default configuration settings for running MapViewer are not adequate, you can configure MapViewer by editing the MapViewer configuration file, `mapViewerConfig.xml`, which is located in the *`$ORACLE_HOME`*`/lbs/mapviewer/conf` directory. After you modify this file, you must restart OC4J to have the changes take effect.

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

- Logging information, defined in the `<logging>` element (see Section 1.5.1)
- Map image file information, defined in the `<save_images_at>` element (see Section 1.5.2)
- Administrative request restrictions, defined in the `<ip_monitor>` element (see Section 1.5.3)

- Web proxy information for accessing external information across a firewall, defined in the `<web_proxy>` element (see Section 1.5.4)
- Global map "look and feel" configuration, defined in the `<global_map_config>` element (see Section 1.5.5)
- Internal spatial data cache settings, defined in the `<spatial_data_cache>` element (see Section 1.5.6)
- Custom image renderer registration, defined in the `<custom_image_renderer>` element (see Appendix B)
- Permanent map data sources, defined in the `<map_data_source>` element (see Section 1.5.7)

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

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

Example 1–1 Sample MapViewer Configuration File

```
<?xml version="1.0" ?>
<!-- This is the configuration file for Oracle Application Server MapViewer. -->
<!-- Note: All paths are resolved relative to this directory (where this
           configuration 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"

       Image 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://system3.my_corp.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" />
-->


<!-- ****************************************************************** -->
<!-- ********************** 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 emptry string ("") for
          the text attribute of <note> and <title> elements.
       - 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 dynamically
       apply a globular projection 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 2003, 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 and 512 MB
     for disk spooling of spatial data. The disk cache path is determined by
     MapViewer by default.

     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).
     max_disk_cache_size: Maximum size of disk-based cache for MapViewer.
                      Size must be specified in megabytes (MB).
     disk_cache_path: Temporary disk path where the spooled objects will be
                      located. Default is the "../cache" directory.
 -->

<!--
  <spatial_data_cache   max_cache_size="64"
                        max_disk_cache_size="512"
                        disk_cache_path="../cache"
  />
-->


<!-- ****************************************************************** -->
<!-- ******************* 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" />
  -->


  <!-- ****************************************************************** -->
  <!-- ******************** 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>
```

1.5.1 Specifying Logging Information

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

MapViewer provides a flexible logging mechanism to record runtime 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 affects 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 fatal 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 Specifying Map File Storage and Life Cycle Information

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

A mapping client can request that MapViewer send back the URL for an image file instead of the actual map image data, by setting the `format` attribute of the `<map_request>` element (described in Section 3.2.2) to `GIF_URL` or `PNG_URL`. In this case, MapViewer saves the requested map image as a file on the host system where MapViewer is running and sends a response containing the URL of the image file back to the map client.

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

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

Default value: `file_prefix=omsmap`

- The `url` attribute identifies the map image base URL, which points to the directory under which all map image files are saved on the MapViewer host. The map image URL sent to the mapping client is the map image base URL plus the map image file name. For example, if the map image base URL is `http://dev04.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 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 MapViewer checks for possible files to delete.

  Default: MapViewer never deletes the generated map images.

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

  Default value: `480` (8 hours)

1.5.3 Restricting Administrative (Non-Map) Requests

In addition to map requests, MapViewer accepts administrative (non-map) requests, such as requests to list all data sources and to add and delete data sources. (Chapter 6 describes the administrative requests.) By default, all MapViewer users are permitted to make administrative requests.

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

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

Example 1–2 Restricting Administrative Requests

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

In Example 1–2:

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

Syntax notes for the `<ip_monitor>` element:

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

1.5.4 Specifying a Web Proxy for Background Image URLs

If a map request contains the `bgimage` (background image) attribute specifying a URL for an image, the image might be behind a firewall that MapViewer cannot directly access. To allow MapViewer to access background images in these cases, use the `<web_proxy>` element to identify the host name and port number for proxy access. For example:

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

1.5.5 Specifying Global Map Configuration Options

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

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

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

```
<global_map_config>
    <note text="Copyright (c) 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 MapViewer configuration file. Note the following information about the attributes of the `<title>` element:

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

  Default value: `NORTH`

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

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

  Default value: `SOUTH_EAST`

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

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

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

  Default value: `SOUTH_WEST`

If the logo image is obtained through a URL that is outside your firewall, you may need to set the Web proxy in order for MapViewer to retrieve the logo image. For information about specifying a Web proxy, see Section 1.5.4.

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 3.2.10. The default position for a map legend is `SOUTH_WEST`.)

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

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

1.5.6 Customizing the Spatial Data Cache

You can customize the memory cache and disk cache that MapViewer uses for spatial geometry objects by using the `<spatial_data_cache>` element. For example:

```
<spatial_data_cache   max_cache_size="64"
                      max_disk_cache_size="512"
                      disk_cache_path="/var/tmp"
/>
```

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 `max_disk_cache_size` attribute specifies the maximum number of megabytes (MB) of disk cache.

  Default value: `512`

- The `disk_cache_path` attribute specifies the temporary disk path where the spooled cache will be located.

  Default value: location specified for the Java environment variable `java.io.tmpdir`

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 need to specify the `max_cache_size` attribute value as `0` (zero).

1.5.7 Defining Permanent Map Data Sources

Every map request must have a `name` 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"
/>
```

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

- The `name` attribute specifies a unique data source name to MapViewer. You must specify the data source name in all map requests that identify a data source.
- The `jdbc_host`, `jdbc_sid`, `jdbc_port`, and `jdbc_user` parameters specify the database connection information and the database user name.
- The `jdbc_password` attribute specifies the database user's login password. It must be prefixed with an exclamation point (!) when you specify the password for the first time. When MapViewer next restarts, it will automatically obfuscate and replace the clear text password.
- The `jdbc_mode` attribute tells MapViewer which JDBC driver to use when connecting to the database. The default is `thin` (for the "thin" driver). The other possible value is `oci8`, which requires that you also have the Oracle database client installed on the same host on which MapViewer is running.
- The `number_of_mappers` attribute identifies the number of map renderers to be created (that is, the number of requests that MapViewer can process at the same time) for this data source. Each map renderer typically uses from 5 MB to 30 MB of memory, depending on the volume of spatial data retrieved and processed during any map generation. Any unprocessed map requests are

queued and eventually processed. For example, if the value is 3, MapViewer will be able to process at most three mapping requests concurrently. If a fourth map request comes while three requests are being processed, it will wait until MapViewer has finished processing one of the current requests. The maximum number of mappers for a single data source is 64.

1.6 Getting Started Using MapViewer

To get started using MapViewer quickly, you can load a supplied set of demonstration data to which styles have been applied. If you have downloaded the entire MapViewer kit from OTN (`http://otn.oracle.com`), you have a file named `mvdemo.zip`, which includes an Oracle 8.1.7 export file `mvdemo.dmp`. Follow these steps:

1. Import the `mvdemo.dmp` file into your Oracle database under the supplied user SCOTT. (Do not import it under any other database user. The demo may fail if you import the file under a different user). Use the following command (and include the directory path in the `FILE` parameter if `mvdemo.dmp` is not in the current directory):

   ```
   imp SCOTT/TIGER FILE=mvdemo.dmp FULL=Y
   ```

2. Run the SQL script `copymeta.sql` (included in the `mvdemo.zip` file) to set up the mapping metadata for the user SCOTT.
3. Define a data source for user SCOTT in MapViewer, as explained in Section 1.6.1.
4. Optionally, use the supplied example JSP file described in Section 1.6.2.

1.6.1 Dynamically Defining MapViewer Data Sources

Before you can use MapViewer to render a map, you must have at least one map data source defined. A data source can be permanently defined in the `mapViewerConfig.xml` file, or it can be dynamically defined using the MapViewer home page. The rest of this section explains how to define a data source dynamically.

To define a data source dynamically, follow these steps:

1. After starting MapViewer, go to a MapViewer page for submitting administrative and other requests by visiting a URL that has the following format:

   ```
   http://hostname:port/mapviewer
   ```

In the preceding format, *hostname:port* is the host name string and port number for MapViewer. For example:

```
http://mapserver.xyzabc.com:8888/mapviewer
```

2. Examine the **Add a data source** form, which contains the following or similar text:

```
<?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="3" />
</non_map_request>
```

3. Edit the name and the JDBC-related information to reflect your environment.
4. Click **Submit**.

This page contains forms that you can use for a variety of other tasks, such as:

- Removing a data source
- Redefining a data source
- Listing all existing data sources
- Listing all maps defined in a data source
- Listing all themes defined in a data source
- Listing all themes defined in a data source that belong to a specific map
- Adding a marker style
- Clearing the MapViewer metadata cache for a specific data source

1.6.2 Example JSP File That Uses MapViewer

The directory *$ORACLE_HOME/*`lbs/mapviewer/web/demo` contains a simple JavaServer Pages (JSP) file named `mapclient.jsp` that demonstrates how to interact with MapViewer. The `mapclient.jsp` file lets you submit map requests, and it displays the resulting map image. (This file is one of several JSP example files, as explained in Section 1.6.3.)

To run this example file, go to a URL that has the following format:

```
http://hostname:port/mapviewer/demo/mapclient.jsp
```

In the preceding format, *`hostname:port`* is the host name string and port number for MapViewer. For example:

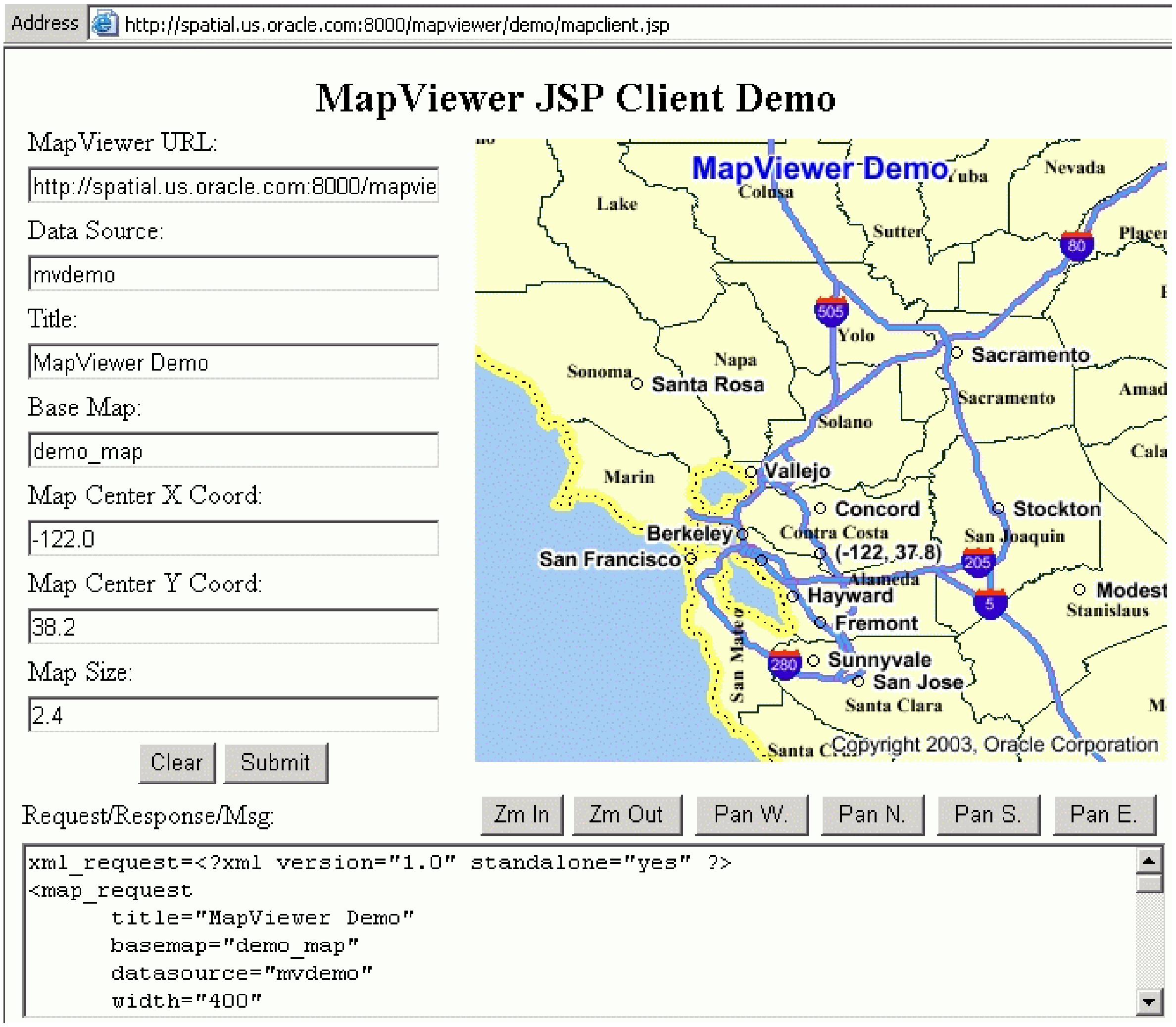
```
http://mapserver.xyzabc.com:8888/mapviewer/demo/mapclient.jsp
```

To submit a map request using this page, enter the necessary information in the text boxes above the Clear and Submit buttons (Title is optional), and click **Submit**.

A map is displayed reflecting the information you entered, and the Request / Response / Msg box contains the XML format of the map request and response. You can perform additional operations on the map display by clicking the other buttons on the page, such as **Zm In** and **Zm Out** for zoom operations.

Figure 1–5 shows this page displaying the result of a map request.

Figure 1–5 MapViewer Example JSP Display


Address http://spatial.us.oracle.com:8000/mapviewer/demo/mapclient.jsp

MapViewer JSP Client Demo

MapViewer URL: http://spatial.us.oracle.com:8000/mapvie

Data Source: mvdemo

Title: MapViewer Demo

Base Map: demo_map

Map Center X Coord: -122.0

Map Center Y Coord: 38.2

Map Size: 2.4

Request/Response/Msg:

```
xml_request=<?xml version="1.0" standalone="yes" ?>
<map_request
      title="MapViewer Demo"
      basemap="demo_map"
      datasource="mvdemo"
      width="400"
```

1.6.3 Additional JSP Example Files

The MapViewer home page (that is, the URL with the format `http://`*`hostname`*`:`*`port`*`/mapviewer`) contains a **Demos** link, which leads to JSP example files that can help you to develop applications that use MapViewer. In addition to a link to the `mapclient.jsp` file (described in Section 1.6.2), there are links to pages for the following files:

- `jview.jsp` visualizes the results of spatial queries issued against a specified data source. You can type in up to three separate queries that retrieve geometric data, and choose different styling for each query result.
- `mapinit.jsp` shows how to use the MapViewer client API to develop a simple interactive Web mapping application with a feature-identifying capability. That is, you can select **Identify** and then click the circle for a city to display data (from nonspatial columns) about that city.
- `tagmap.jsp` shows how to use the MapViewer JSP tag library and the client API together. It also shows how to generate a map legend and place it on the mapping page.

2 MapViewer Concepts

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

The fundamental concepts are *style*, *theme*, *base map*, *mapping metadata*, and *map*.

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

2.1 Overview

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

MapViewer lets you define styles, themes, and base maps, including the rules for applying one or more styles to each theme. These styles, themes, base maps, and associated rules are stored in the database in map definition tables under the MDSYS schema, and they are visible to you through metadata views. All styles in a database instance are shared by all users. The mapping metadata (the set of styles, themes, and base maps) that you can access is determined by the MapViewer metadata views described in Section 2.5 (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 Definition 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 core advanced style is `BucketStyle`, which defines the relationship between a set of simple styles and a set of buckets. For

each feature to be plotted, a designated value from that feature is used to determine which bucket it 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.)

Table 2–1 lists the Java class for creating styles of each type.

Table 2–1 Style Types and Associated Java Classes

Style Type	Java Class	Applicable Geometry Types
Color	oracle.sdovis.style.StyleColor	(any type)
Marker	oracle.sdovis.style.StyleMarker	point, line
Line	oracle.sdovis.style.StyleLine	line
Area	oracle.sdovis.style.StyleArea	polygon
Text	oracle.sdovis.style.StyleText	(any type)
Advanced	oracle.sdovis.stylex.AdvancedStyle and extensions	(any type)

All styles for a database user are stored in that user's USER_SDO_STYLES view, which is described in Section 2.5 and Section 2.5.3.

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

2.3 Themes

A **theme** is a visual representation of a particular data layer. Each theme (except for image themes) is associated with a specific spatial geometry layer, that is, with a column of type MDSYS.SDO_GEOMETRY in a table or view. For example, a theme named `US_States` might be associated with the STATE_SHAPE spatial geometry column in a STATES table.

A theme can have its definition, including styling rules, stored permanently in the database (a **predefined theme**), or a theme can be dynamically defined with a map request (a **JDBC theme**).

All predefined themes for a database user are stored in that user's USER_SDO_THEMES view, which is described in Section 2.5 and Section 2.5.2.

2.3.1 Styling Rules in Predefined Themes

Each predefined theme is associated with one or more **styling rules**. The styling rules for each predefined theme are expressed using XML, such as in Example 2–1 for an `Airport` theme.

Notes: 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.

In the content (character data) of an XML document, `&lt;` and `&gt;` 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 2–1 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 rows (features) in the table or view will be selected based on its attribute value, and the style to be used for those selected features. The `<label>` element specifies whether or not to annotate the selected feature, and if so, which column in the table or view to use for text labels.

In Example 2–1, there are two styling rules associated with the `Airport` theme:

- The first rule specifies that only those rows that satisfy the condition `runway_number &gt; 1` (that is, runway number greater than 1) will be selected, and these will be rendered using the style named `c.black gray`. Any valid SQL WHERE clause conditions 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 &gt; 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.

2.3.2 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 one or more geometries from the database. You can also specify a rendering style and labeling style, to be used if you also selected text values in the query that can be used for labeling the geometries.

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

Example 2–2 JDBC Theme in a Map Request

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

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

</map_request>
```

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

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

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

For more information about JDBC themes, see the information about the `<jdbc_query>` element in Section 3.2.7.

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 2–3 is a sample `<styling_rules>` element of a predefined theme with a complex SQL query.

Example 2–3 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,mdsys.sdo_geometry(2002,41124,null,
                    mdsys.sdo_elem_info_array(1,2,1),
                                       mdsys.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 2–3 is defined as a predefined theme, MapViewer still treats it as a JDBC theme at runtime when a user requests a map that includes this theme. As with a normal JDBC theme, MapViewer by default imposes a window filtering process (if a query window was included in the map request) on top of the SQL query. To override this default behavior and have the supplied query string executed without any modification, specify `asis="true"` in the `<features>` element, as shown in Example 2–3. (For information about the `asis` attribute, see Section 3.2.7.)

2.3.3 Thematic Mapping

Thematic mapping refers to the drawing of spatial features based on their attribute values. MapViewer uses thematic mapping to create maps in which colors or symbols are applied to features to indicate their attributes. For example, a `Counties` theme can be drawn using colors with different hues that map directly with 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, specify advanced styles in the styling rules associated with a theme. You must specify attribute columns in the table or view whose values will be used to determine exactly how a feature will be rendered thematically. Example 2–4 is the XML definition for an `Earthquakes` theme.

Example 2–4 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 2–4 has only one rule. The `<rule>` element includes an attribute named `column` that does not appear in the `Airport` theme in Example 2–1. The `column` attribute specifies one or more columns (comma-separated) 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 2.5.2.) The RICHTER_SCALE column must be of type NUMBER. To understand why, look at the advanced style definition in Example 2–5.

Example 2–5 Advanced Style Definition for Earthquakes Theme

```
<?xml version="1.0" ?>
<AdvancedStyle>
  <VariableMarkerStyle basemarker="m.circle" startsize="7" increment="4" >
     <Buckets>
          <RangedBucket seq="0"  label="less than 4"  high="4" />
          <RangedBucket seq="1"  label="4 - 5" low="4" high="5" />
          <RangedBucket seq="2"  label="5 - 6" low="5" high="6" />
          <RangedBucket seq="3"  label="6 - 7" low="6" high="7" />
          <RangedBucket seq="4"  label="7 and up" low="7" />
      </Buckets>
  </VariableMarkerStyle>
</AdvancedStyle>
```

This style specifies that the marker named `m.circle` is used to indicate the location of an earthquake. The size of the marker to be rendered for an earthquake depends on the numeric value of the RICHTER_SCALE column for that row. In this example there are five buckets, each covering a predetermined range of values. For example, if an earthquake is of magnitude 5.7 on the Richter scale, the marker size will be 15 pixels (7 + 4 + 4), because the value 5.7 falls in the third bucket (5 - 6) and the starting marker size is 7 pixels (`startsize="7"`) with an increment of 4 for each range (`increment="4"`).

Note: The `label` attribute value (for example, `label="less than 4"`) is not displayed on the map, but is used only in a label that is compiled for an advanced style.

The `seq` attribute value (for example, `seq="0"`) is ignored by MapViewer, which determines sequence only by the order in which elements appear in a definition.

Example 2–5 used the <VariableMarkerStyle> tag. The following examples use the <ColorSchemeStyle> tag in creating thematic maps of census blocks in California. Example 2–6 illustrates the use of a graduated color scale for a thematic mapping of population density. Example 2–7 is a thematic mapping of average household income using a graduated color scale. Example 2–8 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 2–6 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 columns named DENSITYCY (used in Example 2–6). 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 2–6, Example 2–7 illustrates the use of a base color and a graduated color scheme, this time to show household income.

Example 2–7 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 2–7 and Example 2–8). 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 2–8 uses specific color styles for each average household income range.

Example 2–8 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 2–9.

Example 2–9   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 2–9:

- `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, `t.Street` and `T.STREET` 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 2–10.

Example 2–10 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	MDSYS.SDO_GEOMETRY
MERK	VARCHAR2(40)

In this table, the GEOM column contains spatial geometries, and the MERK column contains company names (`Shell`, `Esso`, and so on).

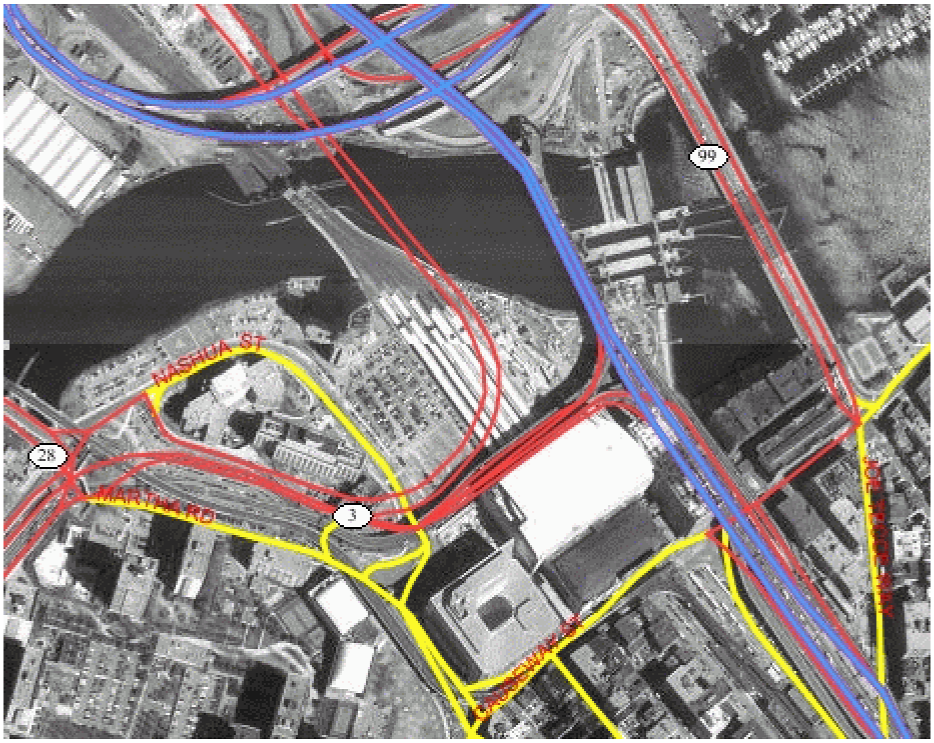
The styling rules for the `theme_gasstation` theme specify that the marker (style `v.gasstations`) at a location specified by the content of the GEOM column is determined by the value of the MERK column for that row. The style `v.gasstations` (see Example 2–9) specifies that if the column value is `Shell`, 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`.

2.3.4 Image Themes

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

You can define an image theme dynamically or permanently (as a predefined theme) in the database. You can use image themes with vector (non-image) themes in a map. Figure 2–1 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 2–1 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.
- 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 2.3.4.1. 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 3.2.7, but note the following additional considerations with dynamic image themes:

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

For any image theme definition, note the following considerations:

- You cannot use the Map Definition Tool to create an image theme. Instead, you must create an image theme and add it to the MapViewer instance programmatically, or you must predefine the theme as explained in Section 2.3.4.1.
- MapViewer supports only GIF and JPEG image formats. To enable MapViewer to visualize data in any other image format, you must implement a custom image renderer using the `oracle.sdovis.CustomImageRenderer` interface in Java, and then register your implementation class in the `mapViewerConfig.xml` file (to tell MapViewer which custom image renderer to use for image data in a specific format). For detailed information about implementing and registering a custom image renderer, see Appendix B.

For an example of a map request specifying an image theme, including an explanation of how MapViewer processes the request, see Example 3–5 in Section 3.1.5.

2.3.4.1 Storing Image Theme Definitions in the Database

To permanently store the definition of an image theme, you must insert a row into the USER_SDO_THEMES view. Example 2–11 stores the definition of an image theme.

Example 2–11 Storing an Image Theme in the Database

```
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 2–11 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 MBRs for the images are stored in the column named `IMAGE_MBR`. In the STYLING_RULES column of the USER_SDO_THEMES view, an XML document with one `<styling_rules>` element is inserted.

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

- `theme_type` must be `image` in order for this theme to be recognized as an image theme.
- `image_column` specifies the column in the base table or view that stores the actual image data.
- `image_format` is a string identifying the format of the image data. If you specify `GIF` or `JPEG`, MapViewer can always render the image data. If you specify any other value, such as `ECW`, you must have implemented a custom image renderer and registered it to MapViewer in order for the image to be

rendered properly. For information about implementing a custom image renderer, see Appendix B.

- `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 2–11, the image resolution is 2 meters per pixel.

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
- Themes (in addition to any in the base map)
- JDBC queries
- JDBC image queries

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 2.5 and Section 2.5.1. The DEFINITION column in the USER_SDO_MAPS view contains an XML description of a base map.

Example 2–12 shows a base map definition.

Example 2–12 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 2–12, 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 2–12, 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 MapViewer is actually the denominator value in a popular method of representing map scale as 1/*n*, where:

- 1, the numerator, is 1 unit (1 inch for MapViewer) on the displayed map.

- *n*, the denominator, is the number of units of measurement (for example, decimal degrees, meters, or miles) represented by 1 unit (1 inch for MapViewer) on the displayed map.

For example:

- If 1 inch on a computer display represents 0.5 decimal degree of user data, the fraction is 1/0.5. The decimal value of the fraction is 2.0, but the scale value for MapViewer is 0.5.
- If 1 inch on a computer display represents 2 miles of user data, the fraction is 1/2. The decimal value of the fraction is 0.5, but the scale value for MapViewer is 2.
- If 1 inch on a computer display represents 10 miles of user data, the fraction is 1/10. The decimal value of the fraction is 0.1, but the scale value for MapViewer is 10.

The `min_scale` and `max_scale` attributes in a `<theme>` element describe the visible scale range of a theme. These attributes control whether or not a theme is displayed, depending on the current map scale. The default scale value for `min_scale` is positive infinity, and the default value for `max_scale` is negative infinity (or in other words, by default display the theme for all map scales, if possible given the display characteristics).

- `min_scale` is the value to which the display must be zoomed in for the theme to be displayed. For example, if parks have a `min_scale` value of 5 and if the current map scale value is 5 or less but greater than the `max_scale` value, parks will be included in the display; however, if the display is zoomed out so that the map scale value is greater than 5, parks will not be included in the display.
- `max_scale` is the value beyond which the display must be zoomed in for the theme not to be displayed. For example, if counties have a `max_scale` value of 3 and if the current map scale value is 3 or less, counties will not be included in the display; however, if the display is zoomed out so that the map scale value is greater than 3, counties will be included in the display.

A high `min_scale` value is associated with less map detail and a smaller scale in cartographic terms, while a high `max_scale` value is associated with greater map detail and a larger scale in cartographic terms. (Note that the MapViewer meaning of map scale is different from the popular meaning of cartographic map scale.) The `min_scale` value for a theme should be larger than the `max_scale` value. Example 2–12 in Section 2.4 includes `min_scale` and `max_scale` values.

To determine the current map scale for a map returned by MapViewer, first find the map size, namely the height (vertical span) of the map in terms of the coordinate system associated with the map data. For example, assume that a map with a height of 10 (miles, meters, decimal degrees, or whatever unit of measurement is associated with the data) is requested, and that the map is drawn on a device with a size of 500 by 350 pixels, where 350 is the height. MapViewer assumes a typical screen resolution of 72 dpi. Because 72 pixels equals 1 inch, the height of the returned map is 4.86 inches (350/72 = 4.86). In this example, the size of the map is 10, and therefore the map scale is approximately 2.057 (10/4.86 = 2.057).

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 and border style
- Have one or more columns in the legend
- Add space to separate legend entries
- Indent legend entries
- Use any MapViewer style, including advanced styles

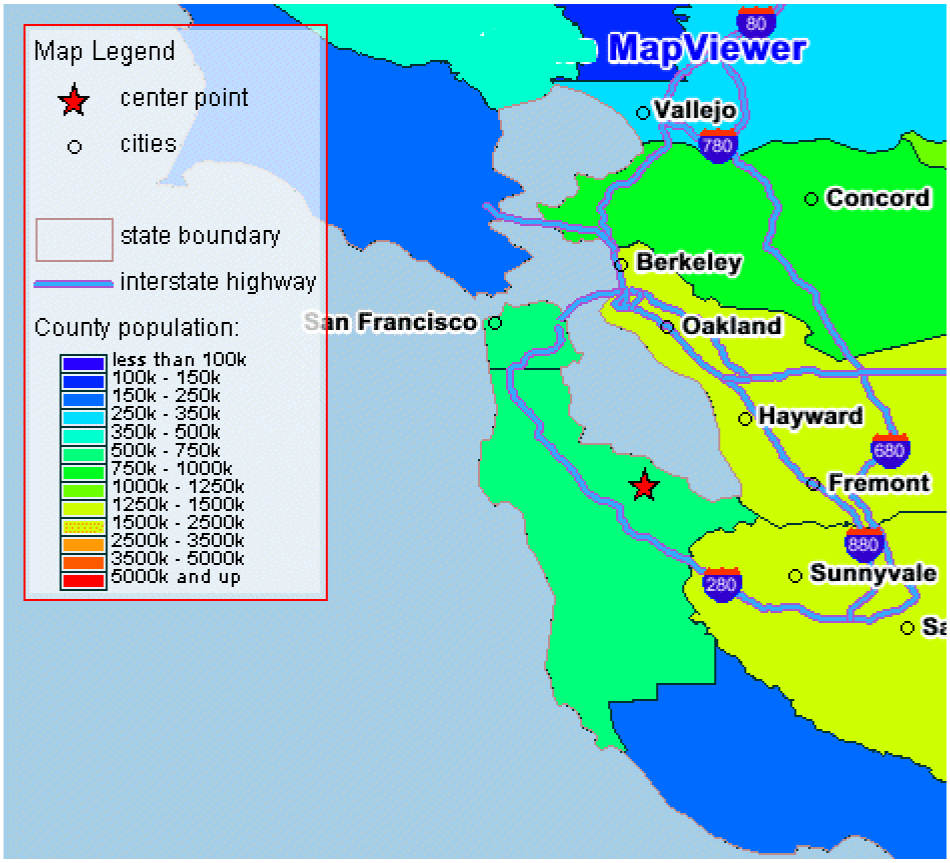
Example 2–13 is an excerpt from a request that includes a legend.

Example 2–13 Legend Included in 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">
    <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 2–2 shows a map with the legend specified in Example 2–13.

Figure 2–2 Map with Legend



Notes on Example 2–13 and Figure 2–2:

- 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 3.2.10.

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 1.5.5.) The default position for a legend is SOUTH_WEST.

2.4.3 Processing of Map Requests

A map specified in the XML map request includes a data source name, an optional base map name, and an optional array of theme descriptors for a set of predefined and dynamic themes. If a base map name is specified, the description of that base map (that is, the theme names, their display order, and their scale ranges) is determined by reading the DEFINITION column value associated with that base map in the ALL_SDO_MAPS view (described in Section 2.5.1). That is, MapViewer performs the following query from the user-specified data source:

```
SELECT DEFINITION FROM ALL_SDO_MAPS WHERE NAME=:basemap AND OWNER=:username;
```

2.5 MapViewer Metadata Views

The mapping metadata describing base maps, themes, and styles is stored in the global tables SDO_MAPS_TABLE, SDO_THEMES_TABLE, and SDO_STYLES_TABLE, which are owned by MDSYS. However, you should never directly update these tables. Each MapViewer user has the following views available in the schema associated with that user:

- USER_SDO_MAPS and ALL_SDO_MAPS contain information about base maps.
- USER_SDO_THEMES and ALL_SDO_THEMES contain information about themes.
- USER_SDO_STYLES and ALL_SDO_STYLES contain information about styles.

Note: You are encouraged to use the Map Definition Tool (described in Chapter 7) to manage the mapping metadata. Only advanced users should consider using SQL statements to update the MapViewer metadata views.

The USER_SDO_xxx views contain metadata information about mapping elements (styles, themes, base maps) owned by the user (schema), and the ALL_SDO_xxx views contain metadata information about mapping elements on which the user has SELECT permission.

The ALL_SDO_xxx views include an OWNER column that identifies the schema of the owner of the object. The USER_SDO_xxx views do not include an OWNER column.

All styles defined in the database can be referenced by any user to define that user's themes, markers with a text style, or advanced styles. However, themes and base maps are not shared among users; so, for example, you cannot reference another user's themes in a base map that you create.

The following rules apply for accessing the mapping metadata:

- If you need to add, delete, or modify any metadata, you must perform the operations using the USER_SDO_xxx views. The ALL_SDO_xxx views are automatically updated to reflect any changes that you make to USER_SDO_xxx views.
- If you need only read access to the metadata for all styles, you should use the ALL_SDO_STYLES view. Both the OWNER and NAME columns make up the primary key; therefore, when you specify a style, be sure to include both the OWNER and NAME.

The MapViewer metadata views are defined in the following file:

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

The following sections describe each set of views.

2.5.1 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

2.5.2 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 MDSYS.SDO_GEOMETRY)
STYLING_RULES	CLOB	XML definition of the styling rules to be associated with the theme

2.5.3 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	MDSYS.SDO_GEOMETRY	(Reserved for future use)

3 MapViewer Map Requests

This chapter explains how to submit map requests in XML format to MapViewer, and it describes the XML document type definitions (DTDs) for the map requests (input) and responses (output). XML is widely used for transmitting structured documents using the HTTP protocol. If an HTTP request (GET or POST method) is used, it is assumed the request has a parameter named `xml_request` whose value is a string containing the XML document for the request.

(In addition to map requests, the MapViewer XML API can be used for administrative requests, such as adding new data sources. Administrative requests are described in Chapter 6.)

As shown in Figure 1–1 in Section 1.1.1, the basic flow of action with MapViewer is that a client locates a remote MapViewer instance, binds to it, sends a map request, and processes the map response returned by the MapViewer instance.

A request to the MapViewer servlet has the following format:

```
http://hostname[:port]/MapViewer-servlet-path?xml_request=xml-request
```

In this format:

- *hostname* is the network path of the server on which MapViewer is running.
- *port* is the port on which the Web server listens.
- *MapViewer-servlet-path* is the MapViewer servlet path (for example, `mapviewer/omserver`).
- *xml-request is* the URL-encoded XML request submitted using the HTML GET or POST method.

The input XML is required for all requests. The output depends on the content of the request: the response can be either an XML document, or a binary object containing the (generated image) file requested by the user.

In an input request, you must specify a data source, and you can specify one or more of the following:

- Themes and styles.
- A center point or a box for the map display, and options such as highlight, label, and styles.
- A predefined base map, which can be reused and overlaid with custom data.
- A custom theme with the user data points (or any geometry) retrieved dynamically and plotted directly from an accessible database.
- Custom features (point, circles, or any geometry) specified in the XML request string to be plotted. These require that you provide the (dynamic) data in GeoFeature format, as defined in the DTD. The geometry portion of GeoFeature 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 Definition Tool, which is described in Chapter 7.

For the current release, MapViewer accepts only a coordinate pair to identify the location for a map request; it cannot take a postal address as direct input for a map.

This chapter first presents some examples of map requests (see Section 3.1), and then presents detailed explanations of the following XML DTDs for requests and other operations:

- Map Request DTD
- Information Request DTD
- Map Response DTD
- MapViewer Exception DTD
- Geometry DTD (OGC)

3.1 Map Request Examples

This section provides examples of map requests. It refers to concepts, elements, and attributes that are explained in detail in Section 3.2.

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 Dynamically Defined Theme

```
<?xml version="1.0" encoding="UTF-8" ?>
<map_request datasource="lbs_data">
  <themes>
    <theme name="t1">
      <jdbc_query spatial_column = "GEOM"
                  datasource = "lbs_data">
      SELECT geom FROM counties
      </jdbc_query>
    </theme>
  </themes >
</map_request>
```

3.1.3 Map Request with Base Map, Center, and Additional Predefined Theme

Example 3–3 requests a map with a specified center for the result map, and specifies a predefined theme (`poi_theme_us_restaurants`) to be rendered in addition to the predefined themes that are part of the base map (`basemap="us_base"`).

Example 3–3 Map Request with Base Map, Center, and Additional Predefined Theme

```
<?xml version="1.0" encoding="UTF-8" ?>
<map_request datasource="lbs_data" title="LBS CUSTOMER MAP"
      basemap="us_base" width="500" height="375"
      bgcolor="#a6cae0" format="GIF_URL">
   <center size="1">
```

```
      <geoFeature typeName="mapcenter" label="Motel 1" text_style="T.MOTEL"
         render_style="M.MOTEL" radius="300">
         <geometricProperty>
            <Point>
              <coordinates>-122.2615, 37.5266</coordinates>
            </Point>
         </geometricProperty>
      </geoFeature>
   </center>
   <srs>SDO:8265</srs>
   <themes>
      <theme name="poi_theme_us_restaurants" />
   </themes >
</map_request>
```

Notes on Example 3–3:

- Because `basemap` is specified, MapViewer first draws all predefined themes for that base map before drawing the specified theme (`poi_theme_us_restaurants`).
- The center will be drawn with a marker of the `M.MOTEL` style and the label `Motel 1` in the `T.MOTEL` style.
- A circle with a radius of 300 meters will be drawn around the center.

3.1.4 Map Request with Center, Base Map, Dynamically Defined Theme, and Other Features

Example 3–4 requests a map with a specified center, a predefined theme named `theme_lbs_customers`, a dynamically defined theme named `sales_by_region`, and all base themes in the base map `us_base_road`, plus two features: a polygon representing the top sales region, and a point. The requested map will be stored at the MapViewer host and a URL to that GIF image (`format="GIF_URL"`) will be returned to the requester.

Example 3–4 Map Request with Center, Base Map, Dynamically Defined Theme, Other Features

```
<?xml version="1.0" encoding="UTF-8" ?>
<map_request datasource="lbs_data2" title="LBS CUSTOMER MAP 2"
      width="400" height="300" format="GIF_URL" basemap="us_base_road">
   <center size="1.5">
      <geoFeature typeName="nil">
         <geometricProperty>
```

```
            <Point>
              <coordinates>-122.2615, 37.5266</coordinates>
            </Point>
         </geometricProperty>
      </geoFeature>
   </center>
   <themes>
      <theme name="theme_lbs_customers" />
      <theme name="sales_by_region">
         <jdbc_query spatial_column ="region"
                           label_column="manager"
                           render_style="V.SALES COLOR"
                           label_style="T.SMALL TEXT"
                           jdbc_host="data.my_corp.com"
                           jdbc_sid="orcl"
                           jdbc_port="1521"
                           jdbc_user="scott"
                           jdbc_password="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 3.2.6 and Section 3.2.7.

3.1.5 Map Request with Image Theme

Example 3–5 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 2.3.4.)

Example 3–5 Map Request with 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”
                          data_source=”lbs_data”>
        SELECT image, img_extent, image_id FROM my_images
      </jdbc_image_query>
    </theme>
  </themes >
</map_request>
```

MapViewer processes the request in Example 3–5 as follows:

1. MapViewer retrieves the image data by executing the user-supplied query (`SELECT image, img_extent, image_id FROM my_images`) in the current map window context.
2. MapViewer checks its internal list of all registered image renderers to see if one supports the ECW format (`image_format="ECW"`). Because MapViewer as

supplied by Oracle does not support the ECW format, you must implement and register a custom image renderer that supports the format, as explained in Appendix B.

3. MapViewer calls the `renderImages` method, and image data retrieved from the user-supplied query is passed to the method as one of its parameters.
4. MapViewer retrieves and renders any requested vector data on top of the rendered image.

3.1.6 Map Request for Image of Map Legend Only

Example 3–6 requests a map with just 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 3.2.10.)

Example 3–6 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 3–6, 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.

3.1.7 Map Request Using a Pie Chart Theme

This section shows how to use thematic mapping with a pie chart theme. The result is a map in which each county contains a pie chart in which the size of each slice reflects the proportion of the population in a specified household income level category (low, medium, or high) in the county.

The basic steps are as follows.

1. Create an advanced style that defines the characteristics of the pie charts to be used. The following example creates an advanced style named `V.PIECHART1`.

```
INSERT INTO user_sdo_styles VALUES (
'V.PIECHART1', 'ADVANCED', null,
'<?xml version="1.0" ?>
<AdvancedStyle>
        <PieChartStyle pieradius="10">
                <PieSlice name="low"   color="#ff0000" />
                <PieSlice name="medium"  color="#ffff00" />
                <PieSlice name="high" color="#00ff00" />
        </PieChartStyle>
</AdvancedStyle>',  null, null);
```

   When the style defined in the preceding example is applied to a geographic feature, a pie chart is created with three slices. The `pieradius` attribute specifies the size of each pie chart in pixels. Each slice (`<PieSlice>` element) has a color defined for it. The `name` attribute for each slice is ignored by MapViewer.

2. Create a new theme that uses the style that you created, as in the following example:

```
INSERT INTO user_sdo_themes VALUES (
'THEME_PIE_CHART', null, 'COUNTIES', 'GEOM',
'<?xml version="1.0" standalone="yes"?>
<styling_rules >
  <rule column="INC_LOW,INC_MED,INC_HIGH" >
    <features style="C.US MAP YELLOW">  </features>
    <label  column="''dummy''" style="V.PIECHART1"> 1 </label>
  </rule>
</styling_rules>');
```

   In the theme definition in the preceding example, the `<label>` element of the styling rule specifies `style="V.PIECHART1"`, to indicate that this pie chart style (the style created in step 1) is used to label each geometry displayed on the map.

The column attribute (`column="''dummy''"` in this example) is required, even though it has no effect on the resulting map. The `column` attribute value can be dummy or any other string, and the value must be enclosed on both sides by two single quotation marks.

Because the `V.PIECHART1` style is defined with three slices, the preceding example must specify the names of three columns from the COUNTIES table, and these columns must have a numeric data type. The column names are INC_LOW, INC_MED, and INC_HIGH. These columns will supply the value that will be used to determine the size of each pie slice.

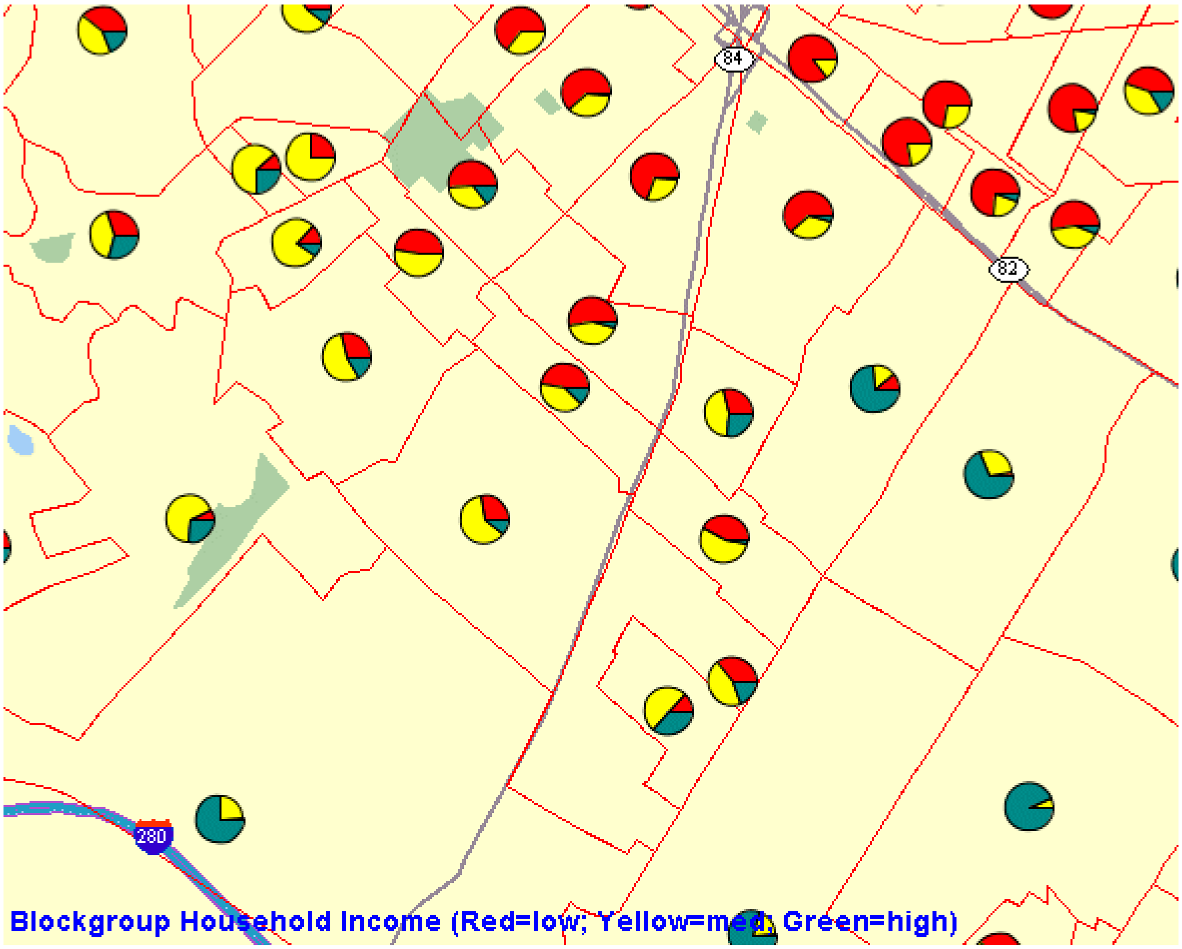
3. Issue a map request that uses the theme that you created. Example 3–7 requests a map that uses the `THEME_PIE_CHART` theme that was created in step 2.

Example 3–7 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 3–7.

Figure 3–1 Map Display Using a Pie Chart Theme


84
82
280
Blockgroup Household Income (Red=low; Yellow=med; Green=high)

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 3–8 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 3–8 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>
```

3.1.8 Java Program Using MapViewer

Example 3–9 uses the `java.net` package to send an XML request to MapViewer and to receive the response from MapViewer. (Note, however, most programmers will find it more convenient to use the JavaBean-based API, described in Chapter 4, or the JSP tag library, described in Chapter 5.)

Example 3–9 Java Program That Interacts with MapViewer

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

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

    /**
```

```
     * Initializes this demo with the url to the MapViewer server.
     * The URL is typically http://my_corp.com:8888/mapviewer/omserver.
     */
    public MapViewerDemo(String mapViewerURLString)
    {
        URL url;

        try
        {
            url = new URL(mapViewerURLString);
            mapViewer = (HttpURLConnection) url.openConnection();
            mapViewer.setDoOutput(true);
            mapViewer.setDoInput(true);
            mapViewer.setUseCaches(false);
        }
        catch (Exception e)
        {
            e.printStackTrace(System.err);
            System.exit(1);
        }
    }

    /**
     * Submits an XML request to MapViewer.
     * @param xmlreq   the xml document that is a MapViewer request.
     */
    public void submitRequest(String xmlreq)
    {
        try
        {
            mapViewer.setRequestMethod("POST");  //use HTTP POST method
            OutputStream os = mapViewer.getOutputStream();
            //MapViewer expects to find the request as a parameter
            //named "xml_request".
            xmlreq = "xml_request="+URLEncoder.encode(xmlreq);
            os.write(xmlreq.getBytes());
            os.flush();
            os.close();
        }
        catch (Exception e)
        {
            e.printStackTrace(System.err);
            System.exit(1);
        }
    }
    /**
     * Receives an XML response from MapViewer.
     */
    public String getResponse()
    {
        ByteArrayOutputStream content = new ByteArrayOutputStream();
        InputStream is = null;
        try
        {
            is = mapViewer.getInputStream();
            int c;
            while ((c = is.read()) != -1)
              content.write(c);
            is.close();
            content.flush();
            content.close();
            return content.toString();
        }
        catch (Exception e)
        {
            e.printStackTrace(System.err);
            return null;
        }
    }

    // A simple main program that sends a list_data_sources xml
    // request to MapViewer through HTTP POST.
    public static void main(String[] args)
    {
        if(args.length<1)
        {
          System.out.println("Usage: java  MapViewerDemo <mapviewer url>");
          System.out.println("Example: java MapViewerDemo http://my_
corp.com/mapviewer/omserver");
          System.exit(1);
        }

        // a sample xml request for MapViewer
        String
        listDataSources = "<?xml version=\"1.0\" standalone=\"yes\"?>" +
                          "  <non_map_request>" +
                          "    <list_data_sources />" +
                          "  </non_map_request>";
        MapViewerDemo tester = null;
        tester = new MapViewerDemo(args[0]);
        System.out.println("submitting request:\n"+listDataSources);
        tester.submitRequest(listDataSources);
        String response = tester.getResponse();
        System.out.println("response from MapViewer: \n" + response);
    }
}
```

3.1.9 PL/SQL Program Using MapViewer

Example 3–10 is a sample PL/SQL program that sends an XML request to the MapViewer server. This example works only on Oracle Database release 9.0.1 and higher.

Example 3–10 PL/SQL Program That Interacts with MapViewer

```
set serverout on size 1000000;

--
--  Author: Clarke Colombo
--
declare

    l_http_req   utl_http.req;
    l_http_resp  utl_http.resp;
    l_url       varchar2(4000):=  'http://my_corp.com:8888/mapviewer/omserver';


    l_value      varchar2(4000);
    img_url      varchar2(4000);
    response     sys.xmltype;

    output       varchar2(255);

    map_req      varchar2(4000);

begin

  utl_http.set_persistent_conn_support(TRUE);

  map_req := '<?xml version="1.0" standalone="yes"?>
  <map_request  title="MapViewer Demonstration"
                datasource="mvdemo"
                basemap="course_map"
                width="500"
                height="375"
                bgcolor="#a6cae0"
                antialiasing="false"
                format="GIF_URL">
    <center size="5" >
      <geoFeature>
        <geometricProperty>
          <Point>
            <coordinates>-122.2615, 37.5266</coordinates>
          </Point>
        </geometricProperty>
      </geoFeature>
    </center>
  </map_request>';

  l_http_req := utl_http.begin_request(l_url, 'POST', 'HTTP/1.0');

  --
  -- sets up proper HTTP headers
  --
  utl_http.set_header(l_http_req, 'Content-Type',
'application/x-www-form-urlencoded');
  utl_http.set_header(l_http_req, 'Content-Length', length('xml_request=' ||
map_req));
  utl_http.set_header(l_http_req, 'Host', 'my_corp.com');
  utl_http.set_header(l_http_req, 'Port', '8888');
  utl_http.write_text(l_http_req, 'xml_request=' || map_req);
  --
  l_http_resp := utl_http.get_response(l_http_req);

  utl_http.read_text(l_http_resp, l_value);

  response := sys.xmltype.createxml (l_value);

  utl_http.end_response(l_http_resp);

  img_url := response.extract('/map_response/map_image/map_
content/@url').getstringval();

  dbms_output.put_line(img_url);

end;
/
```

3.2 Map Request DTD

The following is the complete DTD for a map request. The main elements and attributes of the DTD are explained in sections that follow.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- <box> is defined in OGC GML v1.0 -->
<!ELEMENT map_request ((box | center)?, srs?, legend?, themes?, geoFeature*)>
<!ATTLIST map_request
  datasource    CDATA #REQUIRED
  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) "GIF_URL"
  title         CDATA #IMPLIED
  bgcolor       (CDATA) "#A6CAF0"
  bgimage       CDATA #IMPLIED
>
<!ELEMENT center (geoFeature)>
<!ATTLIST center
   size CDATA #REQUIRED
>

<!ELEMENT srs (#PCDATA) >
<!ELEMENT themes (theme+) >
<!ELEMENT theme (jdbc_query | jdbc_image_query)? >
<!ATTLIST theme
          name            CDATA #REQUIRED
          max_scale       CDATA #IMPLIED
          min_scale       CDATA #IMPLIED
          label_always_on (TRUE|FALSE) "FALSE"
>
<!ELEMENT jdbc_query (#PCDATA)>
<!ATTLIST jdbc_query
    asis              (TRUE|FALSE) "FALSE"
    spatial_column     CDATA #REQUIRED
    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 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 geoFeature (description?, property*,
 geometricProperty)>
<!ATTLIST geoFeature
   typeName        CDATA #IMPLIED
   id              CDATA #IMPLIED
   label           CDATA #IMPLIED
   render_style    CDATA #IMPLIED
   text_style      CDATA #IMPLIED
   label_always_on (TRUE|FALSE) "FALSE"
   marker_size     CDATA #IMPLIED
   radius          CDATA #IMPLIED
>
<!ELEMENT legend column+ >
<!ATTLIST legend
    bgstyle  CDATA #implied
    profile  (MEDIUM|SMALL|LARGE)  "MEDIUM"
>
<!ELEMENT column entry+ >
<!ATTLIST entry
    is_separator  (true|false) "false"
    tab           CDATA  "0"
    style         CDATA  #implied
    text          CDATA  #implied
```

```
>
```

3.2.1 map_request Element

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

```
<!ELEMENT map_request ((box | center)?, srs?, themes?, geoFeature*)>
```

The root element of a map request to MapViewer is always named `map_request`.

`<map_request>` can have a child element that is either `<box>` (see Section 3.2.3) or `<center>` (see Section 3.2.4), which specifies the range of the user data to be plotted on a map. If neither `box` nor `center` is specified, the result map is drawn using all data available to MapViewer.

The optional <SRS> child element is ignored by the current version of MapViewer.

The optional `<themes>` element (see Section 3.2.5) specifies predefined or dynamically defined themes.

The `<geoFeature>` element (see Section 3.2.9) can be used to specify any number of individual geometries and their rendering attributes.

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.

3.2.2 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
  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) "GIF_URL"
  title          CDATA #IMPLIED
  bgcolor        (CDATA) "#A6CAF0"
  bgimage        CDATA #IMPLIED
>
```

`datasource` is a required attribute that specifies a data source. A data source provides information to MapViewer about where to fetch the user data (and the mapping metadata) that are required to render a map.

`basemap` is an optional attribute. When it is specified, MapViewer renders all themes that are specified for this base map. The definition of a base map is stored in the user's USER_SDO_MAPS view, as described in Section 2.5.1. 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 required 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.

`antialiasing` is an optional attribute. When its value is `TRUE`, MapViewer renders the map image in an antialiased manner. This usually provides a map with better graphic quality, but it may take longer for the map to be generated. The default value is `FALSE` (for faster map generation). (For backward compatibility, `antialiase` is a synonym for `antialiasing`, but you are encouraged to use `antialiasing`.)

`imagescaling` is an optional attribute. When its value is `TRUE` (the default), MapViewer attempts to scale the images to fit the current querying window and the generated map image size. When its value is `FALSE`, and if an image theme is included directly or indirectly (such as through a base map), the images from the image theme are displayed in their original resolution. This attribute has no effect when no image theme is involved in a map request.

`format` is an optional attribute that specifies the file format of the returned map image. The default value is `GIF_URL`, which is a URL to a GIF image stored on the MapViewer host system. If you specify `GIF`, the generated GIF image data is embedded in a `MapResponse` object and returned to the client. If you specify `GIF_STREAM`, the generated image map content is returned directly to the client through the HTTP MIME type `image/gif`. If you specify `JAVA_IMAGE`, a Java 2D `BufferedImage` object with a color model of `TYPE_INT_RGB` is embedded in a `MapResponse` object and returned to the client. If you specify `PNG_STREAM`, the stream of the image in PNG format is returned directly; if you specify `PNG_URL`, a URL to a PNG image stored on the MapViewer host system is returned. (The PNG image format has some advantages over the GIF format, including faster image encoding and true color support.)

`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 runtime when a map request is being processed, and it is rendered before any other map features, except that any `bgcolor` value is rendered before the background image.

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 `<center>` element has the following definition:

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

The `<center>` element is used to specify the center of a resulting map. It has a required attribute named `size`, which specifies the vertical span of the map in terms of the original data unit. For example, if the user’s data is in decimal degrees, the `size` attribute specifies the number of decimal degrees in latitude. If the user’s data is projected with meter as its unit, MapViewer interprets the size in meters.

The center itself must embed a `<geoFeature>` element, which is specified in Section 3.2.9.

3.2.5 themes Element

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

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

The `<themes>` element specifies one or more `<theme>` elements (described in Section 3.2.6). 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.

3.2.6 theme Element

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

```
<!ELEMENT theme (jdbc_query | jdbc_image_query)? >
<!ATTLIST theme
          name            CDATA #REQUIRED
          max_scale       CDATA #IMPLIED
          min_scale       CDATA #IMPLIED
          label_always_on (TRUE|FALSE) "FALSE"
>
```

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 a `<jdbc_query>` element (described in Section 3.2.7) or a `<jdbc_image_query>` element (described in Section 3.2.8).

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 2.5.2). For a dynamically defined theme, this is just a temporary name for referencing the `jdbc_query`-based theme.

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, MapViewer labels all features of the theme even if two or more labels will overlap in the display. (MapViewer always tries to avoid overlapping labels.) If label_always_on is FALSE (the default), when it is impossible to avoid overlapping labels, MapViewer disables the display of one or more labels so that no overlapping occurs. The label_always_on attribute can also be specified for a map feature (geoFeature element, described in Section 3.2.9), 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.

3.2.7 jdbc_query Element

The <jdbc_query> element, which is used to define a theme dynamically, has the following definition:

```
<!ELEMENT jdbc_query (#PCDATA)>
<!ATTLIST jdbc_query
    asis               (TRUE|FALSE) "FALSE"
    spatial_column     CDATA #REQUIRED
    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"
>
```

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 MDSYS.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).

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, MapViewer does not attempt to modify the supplied query string. If asis is FALSE (the default), MapViewer embeds the SQL query as a subquery of its spatial filter query. For example, assume that you want a map centered at (-122, 37) with size 1, and the supplied query is:

```
SELECT geometry, sales FROM crm_sales WHERE sales < 100000;
```

If asis is FALSE, the actual query that MapViewer executes is similar to:

```
SELECT * FROM
   (SELECT geometry, sales FROM crm_sales WHERE sales < 100000)
WHERE sdo_filter(geometry, sdo_geometry(... -122.5, 36.5, -123.5, 37.5...)
='TRUE';
```

In other words, the original query is further refined by a spatial filter query using the current map window. However, if asis is TRUE, MapViewer executes the query as specified, namely:

```
SELECT geometry, sales FROM crm_sales WHERE sales < 100000;
```

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.

3.2.8 jdbc_image_query Element

The <jdbc_image_query> element, which is used to define an image theme (described in Section 2.3.4), has the following definition:

```
<!ELEMENT jdbc_image_query (#PCDATA) >
<!ATTLIST jdbc_image_query
    asis              (TRUE|FALSE) "FALSE"
    image_format      CDATA  #REQUIRED
    image_column      CDATA  #REQUIRED
    image_mbr_column  CDATA #REQUIRED
    image_resolution  CDATA #IMPLIED
    image_unit        CDATA #IMPLIED
    datasource        CDATA #IMPLIED
    jdbc_host         CDATA #IMPLIED
    jdbc_port         CDATA #IMPLIED
    jdbc_sid          CDATA #IMPLIED
    jdbc_user         CDATA #IMPLIED
    jdbc_password     CDATA #IMPLIED
    jdbc_srid         CDATA #IMPLIED
    jdbc_mode         (thin|oci8) "thin"
>
```

To define a theme dynamically, you must supply a valid SQL query as the content of the `<jdbc_image_query>` element. You must specify the JDBC connection information for an image theme (either `datasource` or the combination of `jdbc_host`, `jdbc_port`, `jdbc_sid`, `jdbc_user`, and `jdbc_password`).

`jdbc_srid` is an optional attribute that specifies the coordinate system (`SDO_SRID` value) of the data to be rendered.

`jdbc_mode` identifies the Oracle JDBC driver (`thin` or `oci8`) to use to connect to the database.

`asis` is an optional attribute. If it is set to `TRUE`, MapViewer does not attempt to modify the supplied query string. If `asis` is `FALSE` (the default), MapViewer embeds the SQL query as a subquery of its spatial filter query. For example, assume that you want a map centered at (-122, 37) with size 1, and the supplied query is:

```
SELECT geometry, sales FROM crm_sales WHERE sales < 100000;
```

If `asis` is `FALSE`, the actual query that MapViewer executes is similar to:

```
SELECT * FROM
   (SELECT geometry, sales FROM crm_sales WHERE sales < 100000)
WHERE sdo_filter(geometry, sdo_geometry(... -122.5, 36.5, -123.5, 37.5...)
='TRUE';
```

In other words, the original query is further refined by a spatial filter query for current map window. However, if `asis` is `TRUE`, MapViewer executes the query as specified, namely:

```
SELECT geometry, sales FROM crm_sales WHERE sales < 100000;
```

`image_format` identifies the format (such as GIF or JPEG) of the image data. If the image format is not supported by MapViewer, you must create and register a custom image renderer for the format, as explained in Appendix B.

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 2–11 in Section 2.3.4.1, 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 3–5 in Section 3.1.5.

3.2.9 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
>
```

<geoFeature> elements are used to provide individual geospatial entities to be rendered on a map. The main part of a <geoFeature> element is the geometry (<geometricProperty> element), which must be supplied in compliance with the OGC GML v1.0 Geometry DTD (described in Section 3.6).

typeName is an optional attribute that is ignored by the current release of MapViewer.

id is an optional attribute that is ignored by the current release of 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 geoFeature. 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 geoFeature.

label_always_on is an optional attribute. If it is set to TRUE, MapViewer labels the features even if two or more labels will overlap in the display of a theme. (MapViewer always tries to avoid overlapping labels.) If label_always_on is FALSE (the default), when it is impossible to avoid overlapping labels, MapViewer disables the display of one or more labels so that no overlapping occurs. The label_always_on attribute can also be specified for a theme (theme element, described in Section 3.2.6). 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 geoFeature, and if render_style is a marker-type style, the specified size is used by MapViewer in rendering this geoFeature. 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.

The following example shows a <geoFeature> element specification for a restaurant at longitude and latitude coordinates (-78.1234, 41.0346). In this case, the feature will be invisible because the render_style and text_style attributes are not specified.

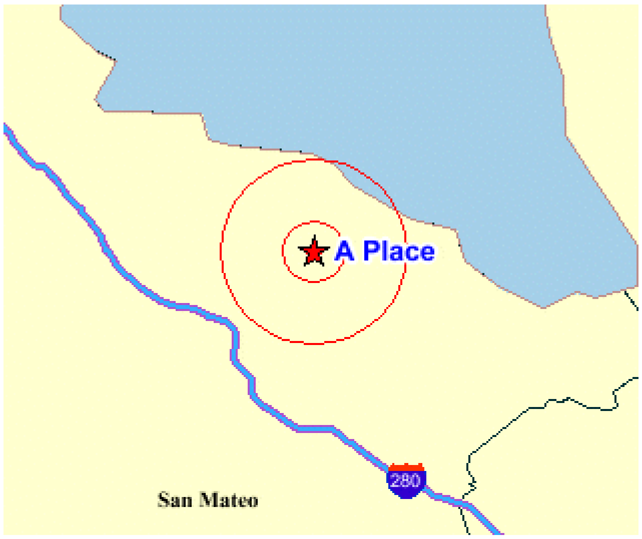
```
<geoFeature typeName="Customer" label="PizzaHut in Nashua">
   <geometricProperty>
      <Point srsName="SDO:8265">
         <coordinates>-78.1234,41.0346</coordinates>
      </Point>
   </geometricProperty>
</geoFeature>
```

The following example shows a `<geoFeature>` element specification for a point of interest at longitude and latitude coordinates (-122.2615, 37.5266). The feature will be rendered on the generated map because the `render_style` attribute is specified. The example specifies some label text (`A Place`) and a text style for drawing the label text. It also instructs MapViewer to draw two circles, centered on this feature, with radii of 1600 and 4800 meters. (In this case, the `srsName` attribute of the `<Point>` element must be present, and it must specify an Oracle Spatial 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 3–2 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 3–2 Map with <geoFeature> Element Showing Two Concentric Circles



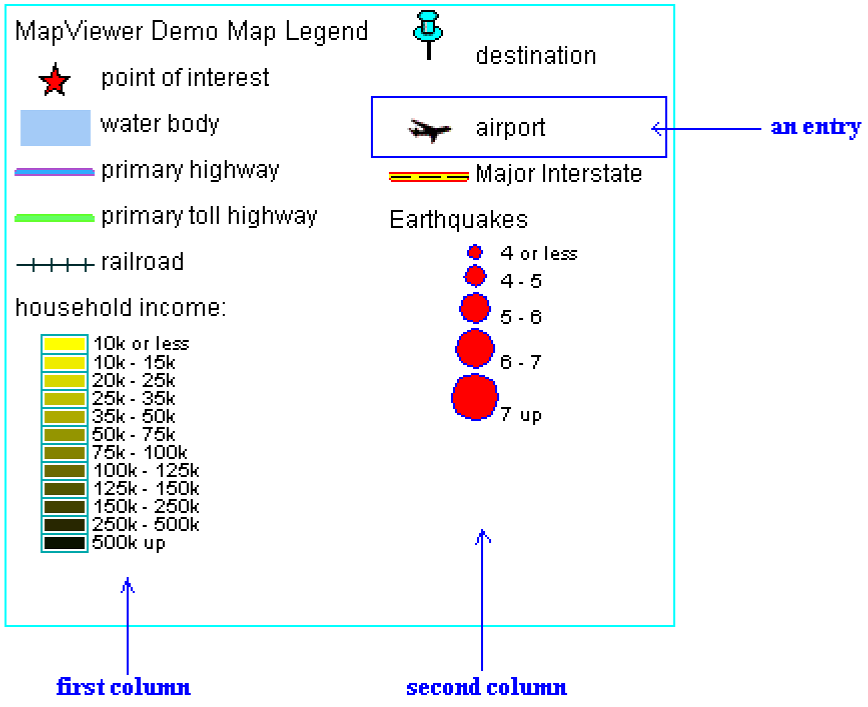
3.2.10 legend Element

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

```
<!ELEMENT legend column+ >
<!ATTLIST legend
    bgstyle  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
>
```

`<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 2–2 in Section 2.4.2 shows a one-column legend. Figure 3–3 shows a two-column legend.

Figure 3–3 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"
```

profile is an optional attribute that specifies a relative size of the legend on the map, using one of the following keywords: SMALL, MEDIUM (the default), or LARGE.

position is an optional attribute that specifies where the legend should be drawn on the map. The default is SOUTH_WEST, which draws the legend in the lower-left corner of the resulting map.

is_title is an optional attribute of the <entry> element. When its value is TRUE, the entry is used as the title for the column, which means that the description text appears in a more prominent font than regular legend text, and any other style attribute defined for the entry is ignored. The default is FALSE.

is_separator is an optional attribute of the <entry> element. When its value is TRUE, the entry is used to insert a blank line for vertical spacing in the column. The default is FALSE.

tab is an optional attribute of the <entry> element. It specifies the number of tab positions to indent the entry from the left margin of the column. The default is 0 (zero), which means no indentation.

style is an optional attribute of the <entry> element. It specifies the name of the MapViewer style (such as a color or an image) to be depicted as part of the entry.

text is an optional attribute of the <entry> element. It specifies the description text (for example, a short explanation of the associated color or image) to be included in the entry.

The following example shows the <legend> element specification for the legend in Figure 2–2 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.

3.3 Information Request DTD

In addition to issuing map requests (see Section 3.2) and administrative requests (see Chapter 6), you can issue information requests to MapViewer. An information request is an XML request string that you can use to execute SQL queries and obtain the result as an array of strings or an XML document. The SQL query must be a SELECT statement and must select only primitive SQL types (for example, not LOB types or user-defined object types).

The following is the DTD for a MapViewer information request.

```
<!ELEMENT info_request (#PCDATA) >
<!ATTLIST info_request
              datasource CDATA #REQUIRED
              format     (strict | non-strict)  "strict"
>
```

`datasource` is a required attribute that specifies the data source for which to get the information.

`format` is an optional attribute. If it is `strict` (the default), all rows are formatted and returned in an XML document. If `format` is set to `non-strict`, all rows plus a column heading list are returned in a comma-delimited text string.

Example 3–11 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 3–11 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 3–11 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, 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.

Example 3–12 shows a map response.

Example 3–12   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>
     <WMTException version="1.0.0" error_code="SUCCESS">
     </WMTException>
   </map_image>
</map_response>
```

3.5 MapViewer Exception DTD

The following DTD is used by the output XML when an exception or unrecoverable error is encountered while processing a map request:

```
<!ELEMENT oms_error (#PCDATA)>
```

The exception or error message is embedded in this element.

3.6 Geometry DTD (OGC)

This section contains the Geometry DTD as defined in the Open GIS Consortium (OGC) GML v1.0 specification, which is available at the following URL:

```
http://www.opengis.org/techno/specs/00-029/GML.html
```

The specification has the following copyright information:

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

The specification has the following status information:

```
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 the specification:

```
THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS
OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE;
THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE
IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS,
COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE
OR IMPLEMENTATION OF THE CONTENTS THEREOF.
```

The OGC Geometry DTD is as follows:

```
<!-- ============================================================== -->
<!--       G e o g r a p h y                                        -->
<!--       M a r k u p                                              -->
<!--       L a n g u a g e                                          -->
<!--                                                                -->
```

```
<!--       ( G M L )                                             -->
<!--                                                             -->
<!--       G E O M E T R Y   D T D                               -->
<!--                                                             -->
<!--  Copyright (c) 2000 OGC All Rights Reserved.                -->
<!-- ============================================================== -->

<!-- the coordinate element holds a list of coordinates as parsed character
data. Note that it does not reference a SRS and does not constitute a proper
geometry class. -->
<!ELEMENT coordinates (#PCDATA) >
<!ATTLIST coordinates
   decimal  CDATA    #IMPLIED
   cs       CDATA    #IMPLIED
   ts       CDATA    #IMPLIED >

<!-- the Box element defines an extent using a pair of coordinates and a SRS
name. -->
<!ELEMENT Box (coordinates) >
<!ATTLIST Box
   ID       CDATA    #IMPLIED
   srsName  CDATA    #REQUIRED >

<!-- ============================================================== -->
<!--  G E O M E T R Y   C L A S S   D e f i n i t i o n s          -->
<!-- ============================================================== -->

<!-- a Point is defined by a single coordinate. -->
<!ELEMENT Point (coordinates) >
<!ATTLIST Point
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >

<!-- a LineString is defined by two or more coordinates, with linear
interoplation between them. -->
<!ELEMENT LineString (coordinates) >
<!ATTLIST LineString
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >

<!-- a Polygon is defined by an outer boundary and zero or more inner
boundaries. These boundaries are themselves defined by LinerRings. -->
<!ELEMENT Polygon (outerBoundaryIs, innerBoundaryIs*) >
<!ATTLIST Polygon
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT outerBoundaryIs (LinearRing) >
<!ELEMENT innerBoundaryIs (LinearRing) >

<!-- a LinearRing is defined by four or more coordinates, with linear
interpolation between them. The first and last coordinates must be
coincident. -->
<!ELEMENT LinearRing (coordinates) >
<!ATTLIST LinearRing
   ID       CDATA    #IMPLIED >

<!-- a MultiPoint is defined by zero or more Points, referenced through a
pointMember element. -->
<!ELEMENT MultiPoint (pointMember+) >
<!ATTLIST MultiPoint
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT pointMember (Point) >

<!-- a MultiLineString is defined by zero or more LineStrings, referenced
through a lineStringMember element. -->
<!ELEMENT MultiLineString (lineStringMember+) >
<!ATTLIST MultiLineString
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT lineStringMember (LineString) >

<!-- a MultiPolygon is defined by zero or more Polygons, referenced through a
polygonMember element. -->
<!ELEMENT MultiPolygon (polygonMember+) >
<!ATTLIST MultiPolygon
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT polygonMember (Polygon) >

<!-- a GeometryCollection is defined by zero or more geometries, referenced
through a geometryMember element. A geometryMember element may be any one of
the geometry classes. -->
<!ENTITY % GeometryClasses "(
   Point | LineString | Polygon |
   MultiPoint | MultiLineString | MultiPolygon |
   GeometryCollection )" >

<!ELEMENT GeometryCollection (geometryMember+) >
<!ATTLIST GeometryCollection
```

```
   ID       CDATA    #IMPLIED
   srsName  CDATA    #IMPLIED >
<!ELEMENT geometryMember %GeometryClasses; >

<!-- ============================================================== -->
<!--   G E O M E T R Y   P R O P E R T Y   D e f i n i t i o n s    -->
<!-- ============================================================== -->

<!-- GML provides an 'endorsed' name to define the extent of a feature. The
extent is defined by a Box element, the name of the property is boundedBy. -->
<!ELEMENT boundedBy (Box) >

<!-- the generic geometryProperty can accept a geometry of any class. -->
<!ELEMENT geometryProperty (%GeometryClasses;) >

<!-- the pointProperty has three descriptive names: centerOf, location and
position. -->
<!ELEMENT pointProperty (Point) >
<!ELEMENT centerOf (Point) >
<!ELEMENT location (Point) >
<!ELEMENT position (Point) >

<!-- the lineStringProperty has two descriptive names: centerLineOf and
edgeOf. -->
<!ELEMENT lineStringProperty (LineString) >
<!ELEMENT centerLineOf (LineString)>
<!ELEMENT edgeOf (LineString)>

<!-- the polygonProperty has two descriptive names: coverage and extentOf. -->
<!ELEMENT polygonProperty (Polygon) >
<!ELEMENT coverage (Polygon)>
<!ELEMENT extentOf (Polygon)>

<!-- the multiPointProperty has three descriptive names: multiCenterOf,
multiLocation and multiPosition. -->
<!ELEMENT multiPointProperty (MultiPoint) >
<!ELEMENT multiCenterOf (MultiPoint) >
<!ELEMENT multiLocation (MultiPoint) >
<!ELEMENT multiPosition (MultiPoint) >

<!-- the multiLineStringProperty has two descriptive names: multiCenterLineOf
and multiEdgeOf. -->
<!ELEMENT multiLineStringProperty (MultiLineString) >
<!ELEMENT multiCenterLineOf (MultiLineString) >
<!ELEMENT multiEdgeOf (MultiLineString) >
```

```
<!-- the multiPolygonProperty has two descriptive names: multiCoverage and
multiExtentOf. -->
<!ELEMENT multiPolygonProperty (MultiPolygon) >
<!ELEMENT multiCoverage (MultiPolygon) >
<!ELEMENT multiExtentOf (MultiPolygon) >

<!ELEMENT geometryCollectionProperty (GeometryCollection) >

<!-- ============================================================== -->
<!--    F E A T U R E   M E T A D A T A   D e f i n i t i o n s    -->
<!-- ============================================================== -->

<!-- Feature metadata, included in GML Geometry DTD for convenience; name and
description are two 'standard' string properties defined by GML. -->

<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

4 MapViewer JavaBean-Based API

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

All communications between the bean and the actual MapViewer service follow a request/response model. Requests are always sent as XML documents to the service. Depending on the type and nature of a request, the response received by the bean is either an XML document or some binary data. However, using the MapViewer bean is easier than manipulating XML documents for forming and sending MapViewer requests, as well as for extracting information from the responses.

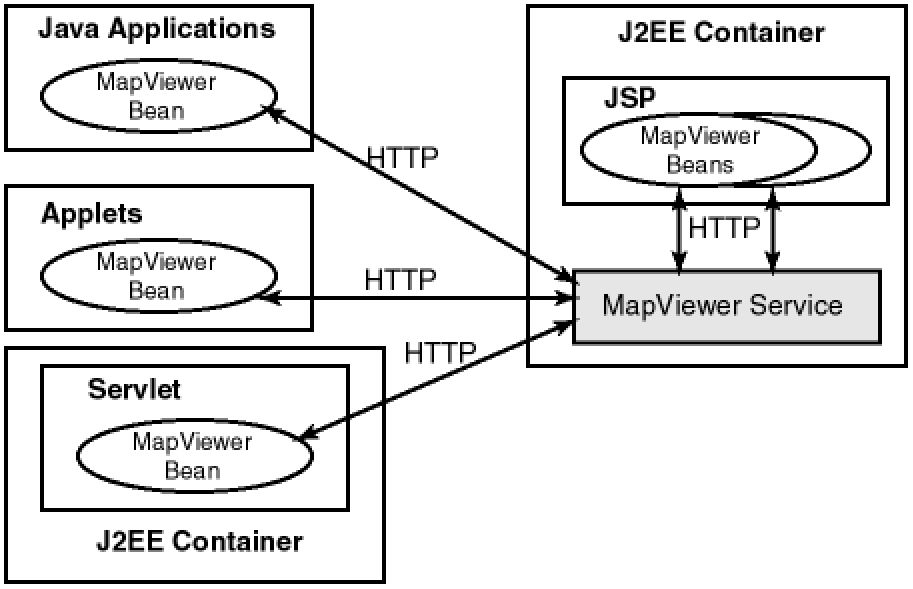
The bean delegates most of map data processing and rendering to the MapViewer service. All the bean does is formulate user requests into valid MapViewer XML requests and send them to a MapViewer service for processing.

4.1 Usage Model for MapViewer JavaBean-Based API

The MapViewer bean can be created and used in either server-side objects such as JavaServer Pages (JSP) and servlets, or in client-side objects such as Java applets or standalone Java applications. The bean is a lightweight class that maintains an active HTTP connection to the MapViewer service and the current map request and map response objects. In most cases, you will create only one MapViewer bean and use it for all subsequent tasks; however, you can create more than one bean and use these beans simultaneously. For example, you may need to create a Web page where a small overview map displays the whole world and a large map image displays a more detailed map of the region that is selected on the overview map. In this case, it is probably easier to create two MapViewer beans, one dedicated to the smaller overview map, and the other to the larger detail map.

Figure 4–1 shows some possible usage scenarios for the MapViewer bean.

Figure 4–1 MapViewer Bean Usage Scenarios



The MapViewer bean can communicate, through the HTTP protocol, with the MapViewer service, in several usage scenarios, the following of which are shown in Figure 4–1:

- In a Java application
- In a Java applet
- In a servlet within a Java2 Enterprise Edition (J2EE) container different from the J2EE container that contains the MapViewer service
- In JavaServer Pages (JSP) code within the J2EE container that contains the MapViewer service

In all usage models, the same JavaBean class is used, and most of its methods apply. However, some methods work or are useful only in a JSP HTML-based context, and other methods work or are useful only in an interactive standalone Java application or applet context (thick clients). For example, consider the following methods of the bean:

- `java.awt.Image getGeneratedMapImage`
- `String getGeneratedMapImageURL`

Both methods extract the generated map image information from a response received from a MapViewer service; however, the first method returns the actual binary image data that is a `java.awt.BufferedImage` class, and the second method returns an HTTP URL string to the generated map image that is stored in the host running the MapViewer service. Clearly, if your application is a JavaServer Page, you should use the second method, because otherwise the JSP page will not know how to handle the `BufferedImage`. However, if you are programming a standalone Java application where you have a Java panel or window for displaying the map, you can use the first method to get the actual image and render it inside your panel or window, plus any other features that you may have created locally and want to render on top of the map.

The set of methods that are only applicable in the thick client context, which are designed to achieve optimal performance for such clients, are described in more detail in Section 4.3.9.

4.2 Preparing to Use the MapViewer JavaBean-Based API

Before you can use the MapViewer JavaBean, the MapViewer `mvclient.jar` library must be in a directory that is included in the CLASSPATH definition. After you deploy the `mapviewer.ear` file in OC4J or Oracle 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 `$OC4J_HOME/j2ee/home/applications`, the base directory for unpacked MapViewer is `$OC4J_HOME/j2ee/home/applications/mapviewer`.)

Before you use the MapViewer JavaBean, you should examine the javadoc documentation and try the JSP demo:

- Javadoc documentation for the MapViewer bean API is available at a URL with the following format:

  ```
  http://host:port/mapviewer/mapclient
  ```

  In this format, *host* and *port* indicate where OC4J or Oracle Application Server listens for incoming requests.

- A demo supplied with MapViewer shows how to use the bean. After you have set up the MapViewer demo data set (which can be downloaded from the Oracle Technology Network) by importing it into a database and running all necessary scripts, you can try the JSP demo. The URL for the JSP demo has the following format:

  ```
  http://host:port/mapviewer/demo/mapinit.jsp
  ```

  In this format, *host* and *port* indicate where OC4J or Oracle Application Server listens for incoming requests. This JSP page confirms the MapViewer service URL and then proceeds to the real demo page, `map.jsp`.

4.3 Using the MapViewer Bean

To use the MapViewer bean, you must create the bean (see Section 4.3.1), after which you can invoke methods to do the following kinds of operations:

- Set up parameters of the current map request (see Section 4.3.2)
- Add themes or features to the current map request (see Section 4.3.3)
- Manipulate the themes in the current map request (see Section 4.3.4)
- Send a request to the MapViewer service (see Section 4.3.5)
- Extract information from the current map response (see Section 4.3.6)
- Use data source and mapping metadata methods (see Section 4.3.7)
- Query nonspatial attributes in the current map window (see Section 4.3.8)
- Use optimal methods for thick clients (see Section 4.3.9)

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 documentation (described in Section 4.2).

4.3.1 Creating the MapViewer Bean

The first step in any planned use of the MapViewer bean is to create the bean, as shown in the following example:

```
import oracle.spatial.mapclient.MapViewer;
MapViewer mv = new MapViewer(“http://my_corp.com:8888/mapviewer/omserver”);
```

The only parameter to the constructor is a URL to an actual MapViewer service. Unless you change it to something else using `setServiceURL`, the MapViewer service at this URL will receive all subsequent requests from this bean. When a MapViewer bean is created, it contains an empty current map request. There are a few parameters in the current request that are initialized with default values, such as the width and height of the map image and the background color for maps. These default values are explained in the XML API element and attribute descriptions in Chapter 3.

4.3.2 Setting Up Parameters of the Current Map Request

As explained in Chapter 3, a map request can have many parameters that affect the final look of the generated map image. When you use the MapViewer JavaBean, such parameters can be set through a group of methods whose names start with *set*. Each of these parameters has 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.
- `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.
- `setDefaultStyleForCenter(String defRenderStyleName, String defLabelStyleName, 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.

- `setDataSourceName(java.lang.String name)` sets the name of the data source to be used when loading data for the map.
- `setDeviceSize(java.awt.Dimension dsz)` sets the image dimension of the map to be generated.
- `setImageFormat(int f)` sets the image format that MapViewer should use when generating the map. For JSP pages, you should always set it to `PNG_URL` or `GIF_URL`.
- `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 3.2.10.
- `setMapTitle(java.lang.String title)` sets the map title for the map to be generated.
- `setServiceURL(java.lang.String url)` sets the MapViewer service URL.
- `setSize(double size)` sets the height (size) in the user data space for the map to be generated.
- `setWebProxy(java.lang.String proxyHost, java.lang.String proxyPort)` sets the Web proxy to be used when connecting to the MapViewer service. This is needed only if there is a firewall between the Web service and this bean.

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 two kinds of dynamic themes: one retrieves geometric data (JDBC theme) and the other retrieves image data (image 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:

- `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.
- `addJDBCTheme` and its variants add a JDBC theme, for which you must supply the query string for retrieving the geometric data.
- `addPredefinedTheme` and its variants add a predefined theme to the current map request.
- `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 4.3.4.
- `addPointFeature(double x, double y, java.lang.String styleName, java.lang.String label, java.lang.String labelStyleName, double[] radius)` adds 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 your point feature. 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. There is no limit to the number of point features you can add.
- `addLinearFeature(double[] coordinates, java.lang.String styleName, java.lang.String label, java.lang.String labelStyleName)` adds a single linear feature (line string) to the current map request. You must specify a rendering style, and you can specify the labeling text and text style for drawing the label. The coordinates must be in the user data space. There is no limit to the number of linear features you can add.

You can remove all added point and linear features by calling `removeAllPointFeatures` and `removeAllLinearFeatures`, respectively.

4.3.4 Manipulating Themes in the Current Map Request

After you add themes using any of the methods that start with *add*, you can manipulate them, performing such operations as listing their names, moving them up or down in rendering order for the current request, and even disabling themes and enabling themes that had been disabled. However, you cannot manipulate themes that are implicitly included when you set a base map (using the `setBaseMapName` method), because the list of themes in the base map is not actually included until the MapViewer service processes the request.

The methods for manipulating themes in the current map request include the following:

- `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`.
- `getThemeNames` returns an ordered list of names of themes that have been explicitly added to the current map request.
- `setThemeEnabled(boolean v, java.lang.String themeName)` sets a specified theme to be enabled or disabled in the current map request.
- `enableThemes(java.lang.String[] themes)` enables all themes whose names appear in the supplied list.
- `setAllThemesEnabled(boolean v)` sets all themes to be enabled or disabled.
- `getEnabledThemes` gets a list of all themes that are currently enabled.
- `getActiveTheme(double currentScale)` gets the name of the active theme, that is, the top theme on the current display map.
- `setThemeScale(java.lang.String name, double minScale, double maxScale)` sets the minimum and maximum scale values for displaying a theme.
- `deleteTheme(java.lang.String name)` deletes an explicitly added theme from the current map request.
- `getThemePosition(java.lang.String name)` returns the position in the rendering sequence of an explicitly added theme.
- `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.
- `setLabelAlwaysOn(java.lang.String name, Boolean labelalwaysOn)` controls whether or not MapViewer labels all features in a theme even if two or more labels will overlap in the display of a theme. (MapViewer always tries to avoid overlapping labels.) If `labelalwaysOn` is

TRUE, MapViewer displays the labels for all features even if two or more labels overlap. If labelalwaysOn is FALSE, when it is impossible to avoid overlapping labels, MapViewer disables the display of one or more labels so that no overlapping occurs.

4.3.5 Sending a Request to the MapViewer Service

As an application developer, you typically issue a new map request as a result of certain user input (such as a mouse click on the currently displayed map) or after you have modified some aspect of the map request (such as setting a new background color). In fact, you can issue a map request any time you want, as long as you do not overwhelm the middle-tier MapViewer service with too many rapid requests from the MapViewer bean or beans. The MapViewer service tries to process requests in the order in which they arrive; if you send a second request before receiving the response from the first one, MapViewer continues to process the first request completely before starting to process the second request.

Any modifications to the current map request, such as changing to a new background color or moving a theme down in the rendering sequence, do not take effect in the map display until you send the map request, at which time the MapViewer service actually receives the request and processes it.

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

- `run` sends the current map request to the MapViewer service, and obtains a map response as sent back by the MapViewer service.
- `pan(int x, int y)` pans to the specified device point. Each coordinate is in the screen / display unit, in this case, pixel.
- `zoomIn(double factor)` zooms in on the map without changing the other map request parameters.
- `zoomIn(int x, int y, double factor)` zooms in on the specified device point.
- `zoomIn(int x1, int y1, int x2, int y2)` zooms in on the specified device rectangle.
- `zoomOut(double factor)` zooms out on the current map without changing the other map request parameters.
- `zoomOut(int x, int y, double factor)` zooms out and recenters the current map.

Each of these methods assembles a single XML map request document based on all properties of the current map request, and then sends it to the MapViewer service. After the MapViewer bean receives the response from the MapViewer service, the bean does any the necessary postprocessing and makes the response ready for your use.

As an alternative to using these methods, you can formulate an XML request string outside the bean, and then use the `sendXMLRequest(java.lang.String req)` method to send the request to the MapViewer service. However, if you use this method, you are responsible for receiving and unpacking the response using the `getXMLResponse` method, and for parsing and interpreting the response string yourself. The state of the bean remains unchanged, because the methods are only making use of the bean's capability to open an HTTP connection to send and receive documents over the connection.

All methods described in this section throw an exception if any unrecoverable error occurs during the transmission of the request or response, or in the MapViewer service during processing. You are responsible for taking care of such exceptions in any way you consider appropriate, such as by trying the request again or by reporting the problem directly to the user.

4.3.6 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:

- `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.
- `getGeneratedMapImage` returns the actual map image data contained in the response from the MapViewer service. You must have set the image format to `FORMAT_RAW_COMPRESSED` using the `setImageFormat` method. The `getGeneratedMapImage` method is primarily used in thick clients, although you may also use it in a JavaServer Page or a servlet (for example, to save the image in a format that is not supported by MapViewer).
- `getMapMBR` returns the MBR (minimum bounding rectangle) for the currently generated map, in the user's data space.
- `getMapRequestString` returns the last submitted map request in XML format.

4.3.7 Using Data Source and Mapping Metadata Methods

The MapViewer bean has methods that you can use to obtain information about the data source for the MapViewer service to which the bean is connected, and to query the mapping metadata. The methods for obtaining information about the data source and the mapping metadata include the following:

- `dataSourceExists(java.lang.String dsrc)` checks if a given data source exists in (that is, is known to) the MapViewer service.
- `addDataSource(....)` adds a data source to the MapViewer service.
- `addImageMarkerFromURL(java.lang.String gifURL, java.lang.String styleName)` adds a GIF image as a marker symbol to the current style cache of the MapViewer service.

4.3.8 Querying Nonspatial Attributes in the Current Map Window

It is often necessary to query nonspatial attributes that are associated with the spatial features being displayed in the current map image. For example, assume that you just issued a map request to draw a map of all customer locations within a certain county or postal code. The next logical step is to find more information about each customer being displayed in the resulting map image. In the JSP or HTML environment, because you get only an image back from the MapViewer service, you will need another round-trip to the service to fetch the nonspatial information requested by the user. This section describes a set of methods that can help you do just that. (You can, however, obtain both the nonspatial attribute values of a certain theme and the resulting map image in a single request when the bean is used in a standalone Java application or applet environment, as described in Section 4.3.9.)

A typical situation is that the user clicks on a feature on the displayed map and wants to find out more (nonspatial attributes) about the feature being "identified." 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 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:

- `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.
- `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.
- `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.
- `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.

4.3.9 Using Optimal Methods for Thick Clients

When you use the MapViewer bean in a JavaServer Page in an HTML file, a second round-trip to the MapViewer service is needed to obtain nonspatial attributes of features being displayed. It is also true that with a JavaServer Page in an HTML file, even if most themes remain unchanged from one map request to the next (such as when zooming in to the center of a map), all themes must still be reprocessed each time the MapViewer service processes the page, which causes the data for each theme to be retrieved again from the database. (This is mainly due to the stateless nature of the MapViewer service and the insufficient mechanism provided in the JSP context for handling user interaction, which must be based on the request/response model.)

However, when you are working in a thick client environment, such as with J2SE (Java 2 Platform Standard Edition) applications and applets, you can reduce the processing and bandwidth overhead when using the bean. This is primarily because in such environments you have greater control of how content (including the map) should be displayed, you can better respond to the user’s interaction, and you can devote more resources to maintaining the states on the client side.

A key optimization available only to the thick client context is **live features**. Basically, a live feature is a spatial feature that originates from the MapViewer service but exists in the thick client. Each live feature contains the actual shape representing the geometry data, and a set of nonspatial attributes that the user might be interested in. To obtain live features, a thick client must set its parent theme to be "clickable." When a map request is sent to the MapViewer service with a clickable theme, MapViewer does not attempt to render features for that theme in the resulting map. Rather, the set of features that would have been drawn as part of the map is returned to the requesting client as an array of live feature objects. The rest of the map is still rendered and transmitted as a single image to the client. After the client has received both the live features and the base image, it must render the live features on top of the accompanying map image, using one of the methods described later in this section.

One of the benefits of using live features is that the thick client will not need to issue a request for the clickable theme every time a map request is sent. For example, if the request is to zoom in to the current map, the client can determine for each live feature if it should be displayed in the zoomed-in map image. Another, and probably more significant, advantage is that the nonspatial attributes for all features displayed in the map are now readily available to the user. For example, as the user moves the mouse over a range of features on the map, the thick client can immediately get the corresponding nonspatial attributes and display them in a pop-up window that follows the mouse trail. No round-trip to the MapViewer service is needed for this type of action, and the feedback to the user is more responsive.

The methods that are optimal for thick clients include the following:

- `drawLiveFeatures(java.awt.Graphics2D g2, java.awt.Color stroke, java.awt.Color fill, double pointRadius, double strokeWidth)` draws all live features that are returned to this client from MapViewer.
- `getLiveFeatureAttrs(int x, int y, int tol)` gets the nonspatial attributes of the feature being clicked on by the user.
- `hasLiveFeatures` checks if there are any live (clickable) features.
- `getNumLiveFeatures` returns the number of live features currently available.
- `highlightFeatures` and variants highlight all live features that are intersecting the user-specified rectangle. These methods also let you specify the style for highlighting features.
- `isClickable(java.lang.String themeName)` checks if the specified theme is clickable (that is, if users can click on the theme to get its attributes).
- `setClickable(boolean v, java.lang.String themeName)` sets the theme clickable (so that its features will be available to the client as live features that users can click on and get attributes of).

To obtain a set of features and keep them live at the thick client, you must first call `setClickable` to set the theme whose features you want to be live. Then, after you issue the current map request, the bean processes the response from the MapViewer service, which (if it succeeded) contains both a base map image and an array of `LiveFeature` instances. You can then call `getGeneratedMapImage` to get and draw the base image, and use `drawLiveFeatures` to render the set of live features on top of the base map. If the user clicks or moves the mouse over a certain position on the map, you can use the `highlightFeatures` method to highlight the touched features on the map. You can also use the `getLiveFeatureAttrs` method to obtain the associated nonspatial attributes of the features being highlighted. You do not have direct access to the `LiveFeature` instances themselves.

The behavior of calling the methods described in this section in the context of JSP pages is not defined.

5 MapViewer JSP Tag Library

This chapter explains how to submit requests to MapViewer using JavaServer Pages (JSP) tags in an HTML file. Through an XML-like syntax, the JSP tags provide a set of important (but not complete) MapViewer capabilities, such as setting up a map request, zooming, and panning, as well as identifying nonspatial attributes of user-clicked features.

Note: The MapViewer JSP tag library will not work with Oracle9*i*AS release 9.0.2 or the standalone OC4J release 9.0.2. The minimum version required is Oracle9*i*AS release 9.0.3 or the standalone OC4J release 9.0.3.

You can develop a location-based application by using any of the following approaches:

- Using the XML API (see Chapter 3)
- Using the JavaBean-based API (see Chapter 4)
- Using JSP files that contain XML or HTML tags, or both, and that include custom Oracle-supplied JSP tags (described in this chapter)

Creating JSP files is often easier and more convenient than using the XML or JavaBean-based API, although the latter two approaches give you greater flexibility and control over the program logic. However, you can include calls to the Java API methods within a JavaServer Page, as is done with the call to the `getMapTitle` method in Example 5–1 in Section 5.3.

All MapViewer JSP tags in the same session scope share access to a single MapViewer bean.

5.1 Using MapViewer JSP Tags

Before you can use MapViewer JSP tags, you must perform one or two steps, depending on whether or not the Web application that uses the tags will be deployed in the same OC4J instance that is running MapViewer.

1. If the Web application will be deployed in the same OC4J instance that is running MapViewer, skip this step and go to step 2.

   If the Web application will be deployed in a separate OC4J instance, you must copy the `mvclient.jar` file (located in the `$MAPVIEWER/web/WEB-INF/lib` directory) and the `mvtaglib.tld` file (located in the `$MAPVIEWER/web/WEB-INF` directory) to that OC4J instance's application deployment directory. Then you must define a `<taglib>` element in your application's `web.xml` file, as shown in the following example:

```
<taglib>
    <taglib-uri>
      http://xmlns.oracle.com/spatial/mvtaglib
    </taglib-uri>
    <taglib-location>
      /WEB-INF/mvtaglib.tld
    </taglib-location>
 </taglib>
```

2. Import the tag library (as you must do with any JSP page that uses custom tags), by using the `taglib` directive at the top of the JSP page and before any other MapViewer tags. For example:

```
<%@ taglib uri="http://xmlns.oracle.com/spatial/mvtaglib"
             prefix="mv" %>
```

   The `taglib` directive has two parameters:

   - `uri` is the unique name that identifies the MapViewer tag library, and its value must be `http://xmlns.oracle.com/spatial/mvtaglib`, because it is so defined in the MapViewer `web.xml` initialization file.
   - `prefix` identifies the prefix for tags on the page that belong to the MapViewer tag library. Although you can use any prefix you want as long as it is unique in the JSP page, `mv` is the recommended prefix for MapViewer, and it is used in examples in this guide.

     The following example shows the `mv` prefix used with the `setParam` tag:

```
<mv:setParam title="Hello World!" bgcolor="#ffffff"
   width="500" height="375" antialiasing="true" />
```

The tags enable you to perform several kinds of MapViewer operations:

- Create the MapViewer bean and place it in the current session: `init` tag, which must come before any other MapViewer JSP tags.
- Set parameters for the map display and optionally a base map: `setParam` tag.
- Add themes and a legend: `addPredefinedTheme`, `addJDBCTheme`, `importBaseMap`, and `makeLegend` tags.
- Get information: `getParam`, `getMapURL`, and `identify` tags.
- Submit the map request for processing: `run` tag.

5.2 MapViewer JSP Tag Reference Information

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

Table 5–1 JSP Tags for MapViewer

Tag Name	Explanation
`init`	Creates the MapViewer bean and places it in the current session. Must come before any other MapViewer JSP tags.
`setParam`	Specifies one or more parameters for the current map request.
`addPredefinedTheme`	Adds a predefined theme to the current map request.
`addJDBCTheme`	Adds a dynamically defined theme to the map request.
`importBaseMap`	Adds the predefined themes that are in the specified base map to the current map request.
`makeLegend`	Creates a legend (map inset illustration) drawn on top of the generated map.
`getParam`	Gets the value associated with a specified parameter for the current map request.
`getMapURL`	Gets the HTTP URL for the currently available map image, as generated by the MapViewer service.

Table 5–1 (Cont.) JSP Tags for MapViewer

Tag Name	Explanation
`identify`	Gets nonspatial attribute (column) values associated with spatial features that interact with a specified point or rectangle on the map display, and optionally uses a marker style to identify the point or rectangle.
`run`	Submits the current map request to the MapViewer service for processing. The processing can be to zoom in or out, to recenter the map, or perform a combination of these operations.

Except where noted, you can use JSP expressions to set tag attribute values at runtime, using the following format:

```
<mv:tag attribute="<%= jspExpression %>" >
```

The following sections (in alphabetical order by tag name) provide reference information for all parameters available for each tag: the parameter name, a description, and whether or not the parameter is required. If a parameter is required, it must be included with the tag. If a parameter is not required and you omit it, a default value is used.

Short examples are provided in the reference sections for JSP tags, and a more comprehensive example is provided in Section 5.3.

5.2.1 addJDBCTheme

The `addJDBCTheme` tag adds a dynamically defined theme to the map request. (It performs the same operation as the `<jdbc_query>` element, which is described in Section 3.2.7.)

Table 5–2 lists the `addJDBCTheme` tag parameters.

Table 5–2 addJDBCTheme Tag Parameters

Parameter Name	Description	Required
name	Name for the dynamically defined theme. Must be unique among all themes already added to the associated MapViewer bean.	Yes
min_scale	The value to which the display must be zoomed in for the theme to be displayed, as explained in Section 2.4.1. If min_scale and max_scale are not specified, the theme is displayed for all map scales, if possible given the display characteristics.	No
max_scale	The value beyond which the display must be zoomed in for the theme not to be displayed, as explained in Section 2.4.1. If min_scale and max_scale are not specified, the theme is displayed for all map scales, if possible given the display characteristics.	No
spatial_column	Column of type MDSYS.SDO_GEOMETRY containing geometry objects for the map display.	Yes
srid	Coordinate system (SDO_SRID value) of the data to be rendered. If not specified, a null coordinate system is assumed.	No
datasource	Name of the data source instance that contains information for connecting to the database.	Yes[1]
jdbc_host	Host name for connecting to the database.	Yes[1]
jdbc_port	Port name for connecting to the database	Yes[1]
jdbc_sid	SID for connecting to the database	Yes[1]
jdbc_user	User name for connecting to the database	Yes[1]
jdbc_password	Password for connecting to the database	Yes[1]
jdbc_mode	The Oracle JDBC driver (thin or oci8) to use to connect to the database. The default is thin.	No
asis	If set to TRUE, MapViewer does not attempt to modify the supplied query string. If FALSE (the default), MapViewer embeds the SQL query as a subquery of its spatial filter query. (For more information and an example, see Section 3.2.7.)	No

Table 5–2 (Cont.) addJDBCTheme Tag Parameters

Parameter Name	Description	Required
`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 3.2.6.) The predefined theme is added at the end of the theme list maintained in the associated MapViewer bean.

Table 5–3 lists the `addPredefinedTheme` tag parameters.

Table 5–3 addPredefinedTheme Tag Parameters

Parameter Name	Description	Required
`name`	Name of the predefined theme to be added to the current map request. This theme must exist in the USER_SDO_THEMES view of the data source used by the associated MapViewer bean.	Yes
`min_scale`	The value to which the display must be zoomed in for the theme to be displayed, as explained in Section 2.4.1. If `min_scale` and `max_scale` are not specified, the theme is displayed for all map scales, if possible given the display characteristics.	No
`max_scale`	The value beyond which the display must be zoomed in for the theme not to be displayed, as explained in Section 2.4.1. If `min_scale` and `max_scale` are not specified, the theme is displayed for all map scales, if possible given the display characteristics.	No

The following example adds the theme named `THEME_DEMO_CITIES` to the current Map request:

```
<mv:addPredefinedTheme name="THEME_DEMO_CITIES" />
```

5.2.3 getMapURL

The `getMapURL` tag gets the HTTP URL (uniform resource locator) for the currently available map image, as generated by the MapViewer service. This map image URL is kept in the associated MapViewer bean, and it does not change until after the `run` tag is used.

The `getMapURL` tag has no parameters.

The following example displays the currently available map image, using the `getMapURL` tag in specifying the source (`SRC` keyword value) for the image:

```
<IMG SRC="<mv:getMapURL />" ALIGN="top">
```

5.2.4 getParam

The `getParam` tag gets the value associated with a specified parameter for the current map request.

Table 5–4 lists the `getParam` tag parameter.

Table 5–4 getParam Tag Parameter

Parameter Name	Description	Required
`name`	Name of the parameter whose value is to be retrieved. It must be one of the valid parameter names for the `setParam` tag. The parameter names are case-sensitive. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes

The following example displays the value of the `title` parameter for the current map request:

```
<P> The current map title is: <mv:getParam name="title"/> </P>
```

5.2.5 identify

The `identify` tag gets nonspatial attribute (column) values associated with spatial features that interact with a specified point or rectangle on the map display, and it optionally uses a marker style to identify the point or rectangle. For example, if the user clicks on the map and you capture the X and Y coordinate values for the mouse pointer when the click occurs, you can retrieve values of nonspatial columns associated with spatial geometries that interact with the point. For example, if the user clicks on a point in Chicago, your application might display the city name, state abbreviation, and population of Chicago, and it might also display a "city" marker on the map near where the click occurred.

The attributes are returned in a `String[][]` array of string arrays, which is exposed by this tag as a scripting variable.

The list of nonspatial columns to fetch must be provided in the tag body, in a comma-delimited list, which the MapViewer bean uses to construct a SELECT list for its queries.

You can optionally associate a highlighting marker with each feature that is identified by using the style attribute and specifying a marker style. To display a new map that includes the highlighting markers, use the `getMapURL` tag.

Table 5–5 lists the `identify` tag parameters.

Table 5–5 identify Tag Parameters

Parameter Name	Description	Required
`id`	Name for the scripting variable through which the returned nonspatial attribute values will be exposed. The first array contains the column names. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes
`datasource`	Name of the MapViewer data source from which to retrieve the nonspatial information.	No
`table`	Name of the table containing the column identified in `spatial_column`. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes
`spatial_column`	Column of type MDSYS.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 not specified, a null coordinate system is assumed.	No
`x`	The X ordinate value of the point; or the X ordinate value of the lower-left corner of the rectangle if `x2` and `y2` are specified.	Yes
`y`	The Y ordinate value of the point; or the Y ordinate value of the lower-left corner of the rectangle if `x2` and `y2` are specified.	Yes
`x2`	The X ordinate value of the upper-right corner of the rectangle.	No
`y2`	The Y ordinate value of the upper-right corner of the rectangle.	No
`style`	Name of the marker style to be used to draw a marker on features that interact with the specified point or rectangle. To display a new map that includes the highlighting markers, use the `getMapURL` tag.	No

The following example creates an HTML table that contains a heading row and one row for each city that has any spatial interaction with a specified point (presumably, the city where the user clicked). Each row contains the following nonspatial data: city name, population, and state abbreviation. The `String[][]` array of string arrays that holds the nonspatial information about the associated city or cities is exposed through the scripting variable named `attrs`. The scriptlet after the tag loops through the array and outputs the HTML table (which in this case will contain information about one city).

```
<mv:identify id="attrs" style="M.CYAN PIN"
```

```
           table="cities" spatial_column="location"
           x="100" y="200" >
    City, Pop90 Population, State_abrv State
</mv:identify>

<%
    if(attrs!=null && attrs.length>0)
    {
      out.print("<CENTER> <TABLE border=\"1\">\n");
      for(int i=0; i<attrs.length; i++)
      {
        if(i==0) out.print("<TR BGCOLOR=\"#FFFF00\">");
        else out.print("<TR>\n");
        String[] row = attrs[i];
        for(int k=0; k<row.length; k++)
          out.print("<TD>"+row[k]+"</TD>");
          out.print("</TR>\n");
       }
      out.print("</TABLE></CENTER>");
    }
%>
```

5.2.6 importBaseMap

The `importBaseMap` tag adds the predefined themes that are in the specified base map to the current map request. (This has the same effect as using the `setParam` tag with the `basemap` attribute.)

Table 5–6 lists the `importBaseMap` tag parameter.

Table 5–6 importBaseMap Tag Parameter

Parameter Name	Description	Required
`name`	Name of the base map whose predefined themes are to be added at the end of the theme list for the current map request. This base map must exist in the USER_SDO_MAPS view of the data source used by the associated MapViewer bean.	Yes

The following example adds the predefined themes in the base map named `demo_map` at the end of the theme list for the current map request:

```
<mv:importBaseMap name="demo_map" />
```

5.2.7 init

The `init` tag creates the MapViewer bean and places it in the current session. This bean is then shared by all other MapViewer JSP tags in the same session. The `init` tag must come before any other MapViewer JSP tags.

Table 5–7 lists the `init` tag parameters.

Table 5–7 init Tag Parameters

Parameter Name	Description	Required
`url`	The uniform resource locator (URL) of the MapViewer service. It must be in the form `http://`*`host:port`*`/mapviewer/omserver`, where *host* and *port* identify the system name and port, respectively, on which Oracle Application Server or OC4J listens.	Yes
`datasource`	Name of the MapViewer data source to be used when requesting maps and retrieving mapping data. If you have not already created the data source, you must do so before using the `init` tag. (For information about creating a data source, see Section 1.6.1.)	Yes
`id`	Name that can be used to refer to the MapViewer bean created by this tag. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes

The following example creates a data source named `mvdemo` with an `id` value of `mvHandle`:

```
<mv:init url="http://mycompany.com:8888/mapviewer/omserver"
            datasource="mvdemo" id="mvHandle" />
```

5.2.8 makeLegend

The `makeLegend` tag accepts a user-supplied XML legend specification and creates a standalone map legend image. The legend image is generated by the MapViewer service, and a URL for that image is returned to the associated MapViewer bean. This tag exposes the URL as a scripting variable.

The body of the tag must contain a `<legend>` element. See Section 3.2.10 for detailed information about the `<legend>` element and its attributes.

Table 5–8 lists the `makeLegend` tag parameters.

Table 5–8 makeLegend Tag Parameters

Parameter Name	Description	Required
`id`	Name for the scripting variable that can be used to refer to the URL of the generated legend image. (This attribute must have a literal value; it cannot take a JSP expression value.)	Yes
`datasource`	Name of the MapViewer data source from which to retrieve information about styles specified in the legend request.	No
`format`	Format of the legend image to be created on the server. If specified, must be `GIF_URL` (the default) or `PNG_URL`.	No

The following example creates a single-column legend with the `id` of `myLegend`, and it displays the legend image.

```
<mv:makeLegend id="myLegend">
  <legend bgstyle="fill:#ffffff;stroke:#ff0000" profile="MEDIUM" >
    <column>
      <entry text="Legend:" is_title="true" />
      <entry style="M.STAR" text="center point" />
      <entry style="M.CITY HALL 3" text="cities" />
      <entry is_separator="true" />
      <entry style="C.ROSY BROWN STROKE" text="state boundary" />
      <entry style="L.PH" text="interstate highway" />
      <entry text="County population density:" />
      <entry style="V.COUNTY_POP_DENSITY" tab="1" />
    </column>
  </legend>
</mv:makeLegend>

<P> Here is the map legend: <IMG SRC="<%=myLegend%>"> </P>
```

5.2.9 run

The `run` tag submits the current map request to the MapViewer service for processing. The processing can be to zoom in or out, to recenter the map, or to perform a combination of these operations.

The `run` tag does not output anything to the JSP page. To display the map image that MapViewer generates as a result of the `run` tag, you must use the `getMapURL` tag.

Table 5–9 lists the `run` tag parameters.

Table 5–9 run Tag Parameters

Parameter Name	Description	Required
action	One of the following values to indicate the map navigation action to be taken: zoomin (zoom in), zoomout (zoom out), or recenter (recenter the map). For zoomin or zoomout, factor specifies the zoom factor; for all actions (including no specified action), x and y specify the new center point; for all actions (including no specified action), x2 and y2 specify (with x and y) the rectangular area to which to crop the resulting image. If you do not specify an action, the map request is submitted for processing with no zooming or recentering, and with cropping only if x, y, x2, and y2 are specified.	No
x	The X ordinate value of the point for recentering the map, or the X ordinate value of the lower-left corner of the rectangular area to which to crop the resulting image if x2 and y2 are specified.	No
y	The Y ordinate value of the point for recentering the map, or the Y ordinate value of the lower-left corner of the rectangular area to which to crop the resulting image if x2 and y2 are specified.	No
x2	The X ordinate value of the upper-right corner of the rectangular area to which to crop the resulting image.	No
y2	The Y ordinate value of the upper-right corner of the rectangular area to which to crop the resulting image.	No
factor	Zoom factor: a number by which the current map size is multiplied (for zoomin) or divided (for zoomout). The default is 2. This parameter is ignored if action is not zoomin or zoomout.	No

The following example requests a zooming in on the map display (with the default zoom factor of 2), and recentering of the map display at coordinates (100, 250) in the device space.

```
<mv:run action="zoomin" x="100" y="250" />
```

5.2.10 setParam

The setParam tag specifies one or more parameters for the current map request. You can set all desired parameters at one time with a single setParam tag, or you can set different parameters at different times with multiple setParam tags. Most of the parameters have the same names and functions as the attributes of the <map_

`request>` root element, which is described in Section 3.2.2. The parameter names are case-sensitive.

Table 5–10 lists the `setParam` tag parameters.

Table 5–10 setParam Tag Parameters

Parameter Name	Description	Required
`antialiasing`	When its value is TRUE, MapViewer renders the map image in an antialiased manner. This usually provides a map with better graphic quality, but it may take longer for the map to be generated. The default value is FALSE (for faster map generation).	No
`basemap`	Base map whose predefined themes are to be rendered by MapViewer. The definition of a base map is stored in the user's USER_SDO_MAPS view, as described in Section 2.5.1. Use this parameter if you will always need a background map on which to plot your own themes and geometry features.	No
`bgcolor`	The background color in the resulting map image. The default is water-blue (RGB value #A6CAF0). It must be specified as a hexadecimal value.	No
`bgimage`	The background image (GIF or JPEG format only) in the resulting map image. The image is retrieved at runtime when a map request is being processed, and it is rendered before any other map features, except that any `bgcolor` value is rendered before the background image.	No
`centerX`	X ordinate of the map center in the data coordinate space.	No
`centerY`	Y ordinate of the map center in the data coordinate space.	No
`height`	The height (in device units) of the resulting map image.	No
`imagescaling`	When its value is TRUE (the default), MapViewer attempts to scale the images to fit the current querying window and the generated map image size. When its value is FALSE, and if an image theme is included directly or indirectly (such as through a base map), the images from the image theme are displayed in their original resolution. This parameter has no effect when no image theme is involved in a map request.	No
`size`	Vertical span of the map in the data coordinate space.	No

Table 5–10 (Cont.) setParam Tag Parameters

Parameter Name	Description	Required
`title`	The map title to be displayed on the top of the resulting map image.	No
`width`	The width (in device units) of the resulting map image.	No

The following example uses two `setParam` tags. The first `setParam` tag sets the background color, width, height, and title for the map. The second `setParam` tag sets the center point and vertical span for the map.

```
<mv:setParam bgcolor="#ff0000" width="800" height="600"
                title="My Map!" />

<mv:setParam centerX="-122.35" centerY="37.85" size="1.5" />
```

5.3 JSP Example (Several Tags) for MapViewer

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

Example 5–1 initializes a MapViewer bean, sets up map request parameters, issues a request, and displays the resulting map image. It also obtains the associated MapViewer bean and places it in a scripting variable (`myHandle`), which is then accessed directly in the statement:

```
Displaying map:  <B> <%=myHandle.getMapTitle()%> </B>
```

Example 5–1 MapViewer Operations Using JSP Tags

```
<%@ page contentType="text/html" %>
<%@ page session="true" %>
<%@ page import="oracle.spatial.mapclient.MapViewer" %>

<%@ taglib uri="http://xmlns.oracle.com/spatial/mvtaglib"
             prefix="mv" %>
<HTML>
<BODY>
Initializing client MapViewer bean. Save the bean in the session
using key "mvHandle"....<P>
 <mv:init url="http://my_corp.com:8888/mapviewer/omserver"
            datasource="mvdemo" id="mvHandle" />
```

```
Setting MapViewer parameters...<P>
<mv:setParam title="Hello World!" bgcolor="#ffffff" width="500" height="375"
antialiasing="true" />

Adding themes from a base map...<P>
<mv:importBaseMap name="density_map"/>

Setting initial map center and size...<P>
<mv:setParam centerX="-122.0" centerY="37.8" size="1.5" />

Issuing a map request... <P>
<mv:run />

<%
  // Place the MapViewer bean in a Java variable.
  MapViewer myHandle = (MapViewer) session.getAttribute("mvHandle");
%>

Displaying map:  <B> <%=myHandle.getMapTitle()%> </B>
<IMG SRC="<mv:getMapURL />"  ALIGN="top" />
</BODY>
</HTML>
```

6 MapViewer Administrative Requests

The main use of MapViewer is for processing various map requests. However, MapViewer also accepts various administrative (non-map) requests, such as to add a data source, through its XML API. 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 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
- Managing Styles
- Managing Cache

6.1 Managing Data Sources

You can add, remove, redefine, and list data sources. (For information about data sources and how to define them, see Section 1.6.1.)

6.1.1 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
   jdbc_tns_name    CDATA #IMPLIED
   jdbc_host        CDATA #IMPLIED
   jdbc_port        CDATA #IMPLIED
   jdbc_sid         CDATA #IMPLIED
   jdbc_user        CDATA #REQUIRED
   jdbc_password    CDATA #REQUIRED
   jdbc_mode        (oci8 | thin) #IMPLIED
   number_of_mappers INTEGER #REQUIRED
               >
```

The `name` attribute identifies the data source name. The name must be unique among MapViewer data sources. (Data source names are not case-sensitive.)

You must specify either a net service name (TNS name) or all necessary connection information. That is, you must specify either of the following (but not both):

- `jdbc_tns_name`
- `jdbc_host`, `jdbc_port`, `jdbc_sid`, and `jdbc_mode`

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 MapViewer is running. You do not need Oracle Client if `thin` is used for all of your data sources.

The `number_of_mappers` attribute identifies the number of map renderers to be created (that is, the number of requests that MapViewer can process at the same time) for this data source. Each map renderer typically uses from 5 MB to 30 MB of memory, depending on the volume of spatial data retrieved and processed during any map generation. Any unprocessed map requests are queued and eventually processed. For example, if the value is 3, MapViewer will be able to process at most three mapping requests concurrently. If a fourth map request comes while three requests are being processed, it will wait until MapViewer has finished processing one of the current requests. The maximum number of mappers for a single data source is 64.

Example 6–1 adds a data source named `mvdemo`.

Example 6–1 Adding a Data Source

```
<?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="3" />
</non_map_request>
```

The DTD for the response to an `add_data_source` request has the following format:

```
<!ELEMENT non_map_response add_data_source>
<!ELEMENT add_data_source  EMPTY>
<!ATTLIST add_data_source
   succeed   (true | false) #REQUIRED
   comment   CDATA #IMPLIED
>
```

The `comment` attribute appears only if the request did not succeed, in which case the reason is in the `comment` attribute. In the following example, `succeed="true"` indicates that the user request has reached the server and been processed without any exception being raised regarding its validity. It does not indicate whether the user's intended action in the request was actually fulfilled by the MapViewer server. In this example, the appearance of the `comment` attribute indicates that the request failed, and the string associated with the `comment` attribute gives the reason for the failure (`"data source already exists"`).

```
<?xml version="1.0" ?>
 <non_map_response>
     <add_data_source succeed="true" comment="data source already exists" />
</non_map_response>
```

6.1.2 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 MapViewer).

Removing a data source only affects the ability of MapViewer to use the corresponding database schema; nothing in that schema is actually removed.

Example 6–2 removes a data source named mvdemo.

Example 6–2 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>
```

6.1.3 Redefining a Data Source

For convenience, MapViewer lets you redefine a data source. Specifically, if a data source with the same name already exists, it is removed and then added using the new definition. If no data source with the name exists, a new data source is added. If an existing data source has the same name, host, port, SID, user name, password, mode, and number of mappers as specified in the request, the request is ignored.

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

```
<!ELEMENT non_map_request redefine_data_source>
<!ELEMENT redefine_data_source  EMPTY>
<!ATTLIST redefine_data_source
   name              CDATA #REQUIRED
   jdbc_tns_name     CDATA #IMPLIED
   jdbc_host         CDATA #IMPLIED
   jdbc_port         CDATA #IMPLIED
   jdbc_sid          CDATA #IMPLIED
   jdbc_user         CDATA #REQUIRED
   jdbc_password     CDATA #REQUIRED
   jdbc_mode         (oci8 | thin) #IMPLIED
   number_of_mappers INTEGER #REQUIRED
>
```

The required attributes and their explanations are the same as the `<add_data_source>` element, which is described in Section 6.1.1.

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>
```

6.1.4 Listing All Data Sources

The `<list_data_sources>` element lists all data sources known to the currently running MapViewer. It has the following definition:

```
<!ELEMENT non_map_request list_data_sources>
<!ELEMENT list_data_sources  EMPTY>
```

For example:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <list_data_sources />
</non_map_request>
```

The DTD for the response to a `list_data_sources` request has the following format:

```
<!ELEMENT non_map_response map_data_source_list>
<!ELEMENT map_data_source_list  (map_data_source*) >
<!ATTLIST map_data_source_list
   succeed      (true|false) #REQUIRED
>
<!ELEMENT map_data_source  EMPTY>
<!ATTLIST map_data_source
   name       CDATA #REQUIRED
   host       CDATA #REQUIRED
   sid        CDATA #REQUIRED
   port       CDATA #REQUIRED
   user       CDATA #REQUIRED
   mode       CDATA #REQUIRED
   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>
```

6.1.5 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>
```

6.2 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>
```

6.3 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.

6.4 Managing Styles

You can list styles, add a non-image marker style, add an image marker style, or check if a specified style exists.

6.4.1 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.

6.4.2 Adding a Style (Not an Image Marker Style)

The `<add style>` element adds a style (other than an image marker style) to a specified data source by supplying the style definition using syntax similar to scalable vector graphics (SVG). It has a definition in the following general format:

```
<!ELEMENT non_map_request add_style>
```

```
<!ELEMENT add_style svg_definition_of_style>
<!ATTLIST add_style
   name  CDATA #REQUIRED
>
```

The actual content of the *svg_definition_of_style* element depends on the type of style you are adding. See Appendix A for detailed information about style formats.

Note that styles added in this way will not be available if MapViewer terminates and is restarted. If you want styles to be persistent, use the Map Definition Tool (described in Chapter 7) to store styles in the database permanently.

The following example adds a color style:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <add_style name="null000000" >
    <svg width="1in" height="1in" >
      <g class="color" style="stroke:#000000;stroke-width:1" >
        <rect width="50" height="50"/>
      </g>
    </svg>
  </add_style>
</non_map_request>
```

The DTD for the response to an `add_style` request has the following format:

```
<!ELEMENT non_map_response add_style>
<!ELEMENT add_style  EMPTY>
<!ATTLIST add_style
   succeed    (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
 <non_map_response>
 <add_style succeed="true" />
</non_map_response>
```

6.4.3 Adding an Image Marker Style

The `<add_image_marker>` element adds a marker style that is based on an external GIF image to a specified data source. It has the following definition:

```
<!ELEMENT  non_map_request  add_image_marker>
<!ELEMENT add_image_marker  EMPTY>
<!ATTLIST  add_image_marker
   image_marker  CDATA #REQUIRED
   url           CDATA #REQUIRED
>
```

The `image_marker` attribute specifies the name to be given to the resulting image marker style.

The `url` attribute specifies the location of the GIF image that is to be used for the style.

The following example adds the `m.tv` marker style based on the image at URL `/mapviewer/images/tv.gif`:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <add_image_marker image_marker="m.tv"
  url="/mapviewer/images/tv.gif"/>
</non_map_request>
```

The DTD for the response to an `add_image_marker` request has the following format:

```
<!ELEMENT non_map_response add_image_marker>
<!ELEMENT add_image_marker  EMPTY>
<!ATTLIST add_image_marker
   succeed   (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
<non_map_response>
 <add_image_marker succeed="true" />
</non_map_response>
```

6.4.4 Checking If a Style Exists

The `<marker_style_exists>` element checks if a specified style (image marker or other type) exists in a specified data source. It has the following definition:

```
<!ELEMENT non_map_request marker_style_exists>
<!ELEMENT marker_style_exists  EMPTY>
<!ATTLIST marker_style_exists
```

```
   marker_style  CDATA #REQUIRED
>
```

The `marker_style` attribute specifies the name of the style that you want to check. (It can be any type of style.)

For example:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <marker_style_exists marker_style="m.tv"/>
</non_map_request>
```

The DTD for the response to a `marker_style_exists` request has the following format:

```
<!ELEMENT non_map_response marker_style_exists>
<!ELEMENT marker_style_exists  EMPTY>
<!ATTLIST marker_style_exists
   succeed   (true | false) #REQUIRED
   exists    (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
<non_map_response>
 <marker_style_exists succeed="true" exists="true" />
</non_map_response>
```

6.5 Managing Cache

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 MapViewer from accessing the database for the cached information; however, the 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 MapViewer, you can clear (invalidate) the content of either or both of these caches. If a cache is cleared, the next MapViewer request will retrieve the necessary information from the database, and will also store it in the appropriate cache.

6.5.1 Clearing Metadata Cache for a Data Source

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

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

The `<clear_cache>` element clears the MapViewer metadata cache. It has the following definition:

```
<!ELEMENT non_map_request clear_cache>
<!ELEMENT clear_cache  EMPTY>
<!ATTLIST clear_cache
   data_source  CDATA #REQUIRED
>
```

The `data_source` attribute specifies the name of the data source whose metadata is to be removed from the MapViewer metadata cache.

The following example clears the metadata for the `mvdemo` data source from the MapViewer metadata cache:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <clear_cache data_source="mvdemo"/>
</non_map_request>
```

The DTD for the response to a `clear_cache` request has the following format:

```
<!ELEMENT non_map_response clear_cache>
<!ELEMENT clear_cache  EMPTY>
<!ATTLIST clear_cache
   succeed   (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
<non_map_response>
 <clear_cache succeed="true" />
</non_map_response>
```

6.5.2 Clearing Spatial Data Cache for a Theme

MapViewer caches spatial data (geometries or georeferenced images) for a predefined theme as it loads the data from the database into memory for rendering, unless it is told not to do so. (MapViewer does not cache dynamic or JDBC themes.) Thus, if a predefined theme has been frequently accessed, most of its data is probably in the cache. However, if the spatial data for the theme is modified in the database, the changes will not be visible on maps, because MapViewer is still using copies of the data from the cache. To view the modified theme data without having to restart MapViewer, you must first clear the cached data for that theme.

The `<clear_theme_cache>` element clears the cached data of a predefined theme. It has the following definition:

```
<!ELEMENT non_map_request clear_theme_cache>
<!ELEMENT clear_theme_cache  EMPTY>
<!ATTLIST clear_theme_cache
   data_source  CDATA #REQUIRED
    theme  CDATA #REQUIRED
>
```

The `data_source` attribute specifies the name of the data source. The `theme` attribute specifies the name of the predefined theme in that data source.

The following example clears the cached spatial data for the predefined theme named `STATES` in the `mvdemo` data source:

```
<?xml version="1.0" standalone="yes"?>
<non_map_request>
  <clear_theme_cache data_source="mvdemo" theme="STATES" />
</non_map_request>
```

The DTD for the response to a `clear_theme_cache` request has the following format:

```
<!ELEMENT non_map_response clear_theme_cache>
<!ELEMENT clear_theme_cache  EMPTY>
<!ATTLIST clear_theme_cache
   succeed   (true | false) #REQUIRED
>
```

For example:

```
<?xml version="1.0" ?>
<non_map_response>
 <clear_theme_cache succeed="true" />
</non_map_response>
```

7 Map Definition Tool

This chapter describes the graphical interface to the Oracle Map Definition Tool. This tool is a standalone application that lets you create and manage mapping metadata that is stored in the database. This mapping metadata can be used by applications that use MapViewer to generate customized maps.

Note: The Map Definition Tool is currently an unsupported tool, and to use it you must download the software from the Oracle Technology Network (`http://otn.oracle.com`).

The information in this chapter reflects the Map Definition Tool interface at the time this guide was published. The online help for the Map Definition Tool may contain additional or more recent information.

To use the Map Definition Tool effectively, you must understand the MapViewer concepts explained in Chapter 2 and the information about map requests in Chapter 3.

The Map Definition Tool is shipped as a JAR file (`mapdef.jar`). You can run it as a standalone Java application in a Java Development Kit (JDK) 1.2 or later environment, as follows (all on a single command line):

```
% java [-classpath <path>]  [-Dhost= <host> ] [-Dsid=<sid>] [-Dport=<port>]
[-Duser=<user>] [-Dpassword=<password>]  oracle.eLocation.console.GeneralManager
```

In the preceding command line format:

- `<path>` specifies the path that the Java interpreter uses to find the `mapdef.jar` file and the JDBC `classes12.zip` file. This overrides the default value of the CLASSPATH environment variable, if it is set. Files are separated by colons on UNIX systems and by semicolons on Windows systems.

- <host> specifies the name or IP address of the local computer that hosts the target database.
- <sid> specifies the database instance identifier.
- <port> specifies the listener port for client connections to the database listener.
- <user> specifies the user name for connecting to the database.
- <password> specifies the password for the specified user for connecting to the database.

If you include any of the connection options in your command line, their values will be used as the defaults for the corresponding fields in the connection box on the Connection page (described in Section 7.2); otherwise, you must specify their values in the connection box to connect to the database.

The following example (all on a single command line) starts the Map Definition Tool on a UNIX system. (Note the use of the colon to separate files in the CLASSPATH specification on UNIX systems.)

```
%java -classpath
/usr/lbs/lib/mapdef.jar:/private/oracle/ora90/jdbc/lib/classes12.zip
-Dhost="127.0.0.1" -Dsid="orcl"  -Dport="1521" -Duser="scott" -Dpassword="tiger"
oracle.eLocation.console.GeneralManager
```

7.1 Overview of the Map Definition Tool

The Map Definition Tool lets you create, modify, and delete styles, themes, and base maps. For example, you can enter the design information for a new line style, see a preview of the style, modify your design if you wish, and then click **Insert** to insert your style definition in XML format into the database. The tool uses the information that you entered to generate the XML document for the style definition.

The styles, themes, and base maps for a user are maintained in that user's USER_SDO_STYLES, USER_SDO_THEMES, and USER_SDO_MAPS views, respectively. These views (described in Section 2.5) are created by MDSYS for you to access your mapping metadata. You can create your new mapping metadata in these views. However, the styles that you create in your USER_SDO_STYLES view will be shared by all other database users.

Whenever possible, you should use the Map Definition Tool instead of directly modifying MapViewer metadata views to create, modify, and delete information about styles, themes, and maps. The Map Definition Tool always checks and maintains the referential integrity between objects. If you perform these operations by using SQL procedures or SQL*Plus statements, the referential integrity of the mapping metadata may become corrupted if you are not careful. For example, if you delete a style using SQL*Plus, a theme may still be referencing the name of that style.

The tool consists of pages grouped under the following categories:

- Connection: a page for connecting to the database
- Styles: a page for each type of style
- Themes: a page for themes
- Maps: a page for maps

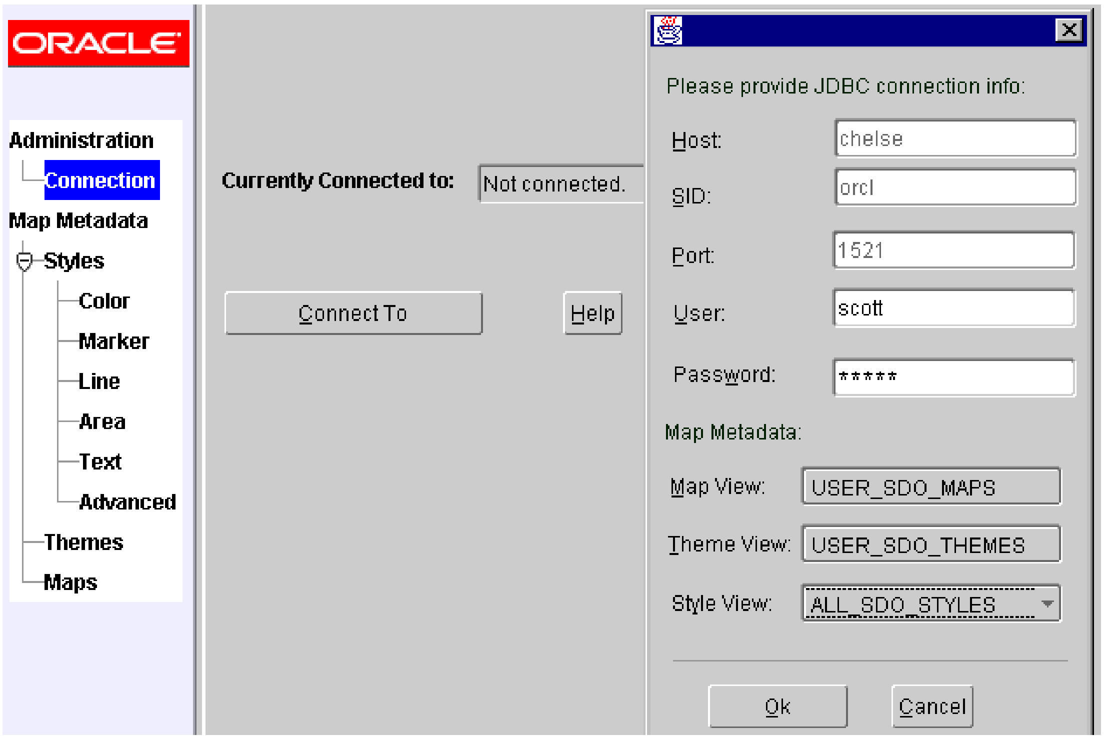
For detailed information about the options on each page, see later sections in this chapter or click **Help** on that page when using the Map Definition Tool.

Note: For all **Name** fields, any entry that you type is automatically converted to and stored in uppercase. (Names of mapping metadata objects are not case-sensitive.)

7.2 Connection Page

Figure 7–1 shows the Connection page after the user has clicked the **Connect To** button.

Figure 7–1 Connection Page



Currently Connected to: Not connected.

Please provide JDBC connection info:

Host: chelse
SID: orcl
Port: 1521
User: scott
Password: *****

Map Metadata:

Map View: USER_SDO_MAPS
Theme View: USER_SDO_THEMES
Style View: ALL_SDO_STYLES

Currently connected to: Contains information about your database connection, or *Not connected* if you are not currently connected to an Oracle database.

Connect To: Displays a JDBC database connection dialog box, in which you specify the Host, SID, Port, User, Password, and mapping metadata views. You can change your connection at any time; the old connection is disconnected when you click OK for a new connection.

Map Metadata: For maps and themes, you must use the USER_SDO_MAPS and USER_SDO_THEMES views, respectively.

For styles, if you select the ALL_SDO_STYLES view, you can see all styles that all users have created, *but you cannot create, modify, or delete* any styles. (The Insert, New, Update, and Delete buttons are disabled.) The ALL_SDO_xxx views are for read-only access. If you select the USER_SDO_STYLES view, you can see only the styles that you have created, but you can create, modify, and delete styles.

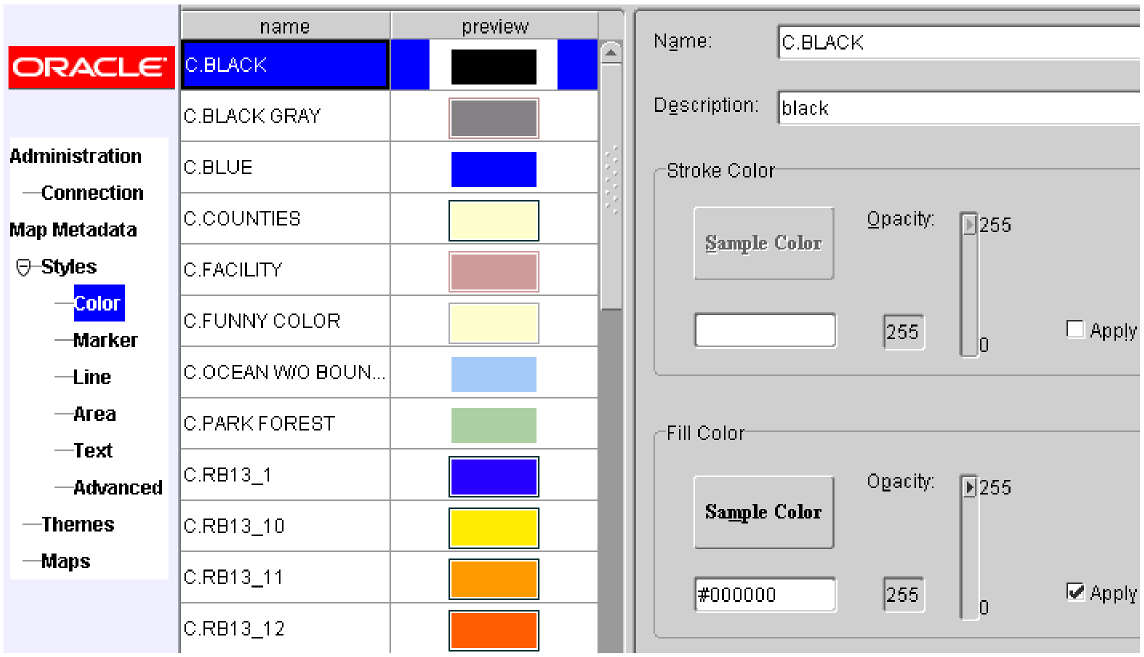
For example, you might connect using the ALL_SDO_STYLES view to see all available styles and get design ideas, and then connect again later using the USER_SDO_STYLES view to create and modify your own styles. However, with either styles view, you have access to all styles defined on your system when you create or edit themes.

To exit the Map Definition Tool: Select Close from the application window menu (upper-left corner), or click the "X" box (upper-right corner).

7.3 Styles: Color Page

Figure 7–2 shows the Color page under the Styles category.

Figure 7–2 Color Page



name	preview
C.BLACK	⬛
C.BLACK GRAY	⬛
C.BLUE	🟦
C.COUNTIES	🟨
C.FACILITY	🟥
C.FUNNY COLOR	🟨
C.OCEAN WO BOUN...	🟦
C.PARK FOREST	🟩
C.RB13_1	🟦
C.RB13_10	🟨
C.RB13_11	🟧
C.RB13_12	🟧

Name: C.BLACK

Description: black

Stroke Color

Sample Color | Opacity: 255 | 255 | 0 | ☐ Apply

Fill Color

Sample Color | #000000 | Opacity: 255 | 255 | 0 | ☑ Apply

Name and **Preview** columns: List currently defined color styles, with a preview of each. (The styles listed depend on whether you selected the ALL_SDO_STYLES or USER_SDO_STYLES view at connection time.)

Name: Name of the style. Must be unique within a schema.

Description: Optional descriptive text about the style.

Stroke Color (for the border) and **Fill Color:**

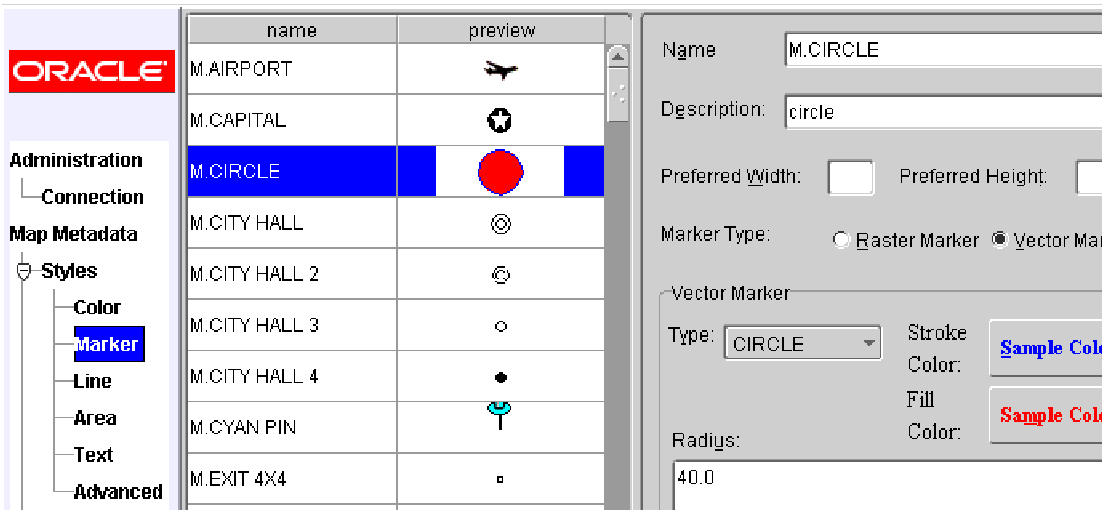
- The rectangle button displays the current color in its foreground text. You can click the rectangle button to display a dialog box to specify a new color by a swatch, HSB value, or RGB value.
- **Opacity:** A value from 0 (transparent) to 255 (solid, or completely opaque).
- **Apply:** If checked, the color is used; if not checked, the color is not used. For example, you might specify a fill color, but not use any border (stroke) color.

Buttons: New lets you enter information for a new style; **Insert** inserts a new style using the specified information; **Update** updates the style using the specified information; **Delete** removes the style; **Cancel** clears any information that you have entered for a new style.

7.4 Styles: Marker Page

Figure 7–3 shows the Marker page under the Styles category.

Figure 7–3 Marker Page



name	preview
M.AIRPORT	✈
M.CAPITAL	✪
M.CIRCLE	●
M.CITY HALL	◎
M.CITY HALL 2	◎
M.CITY HALL 3	○
M.CITY HALL 4	•
M.CYAN PIN	📍
M.EXIT 4X4	▫

Name: M.CIRCLE

Description: circle

Preferred Width:

Preferred Height:

Marker Type: Raster Marker [✓ Vector Mar]

Vector Marker

Type: CIRCLE

Stroke Color: Sample Col

Fill Color: Sample Col

Radius: 40.0

Name and **Preview** columns: List currently defined marker styles, with a preview of each. (The styles listed depend on whether you selected the ALL_SDO_STYLES or USER_SDO_STYLES view at connection time.)

Name: Name of the style. Must be unique within a schema.

Description: Optional descriptive text about the style.

Preferred Width: Number of screen pixels for the preferred width of the marker. If no value is specified, the actual width of the marker is used; no scaling is performed.

Preferred Height: Number of screen pixels for the preferred height of the marker. If no value is specified, the actual width of the marker is used; no scaling is performed.

Marker Type: Raster Marker for an image marker, or Vector Marker for a vector graphics marker.

Raster Marker (for image graphics):

- **Import Image:** Displays a dialog box for specifying the file for the image to be used for the marker.
- **Preview:** Shows a sample of the style as it would look with the imported image. However, no changes are made to the style until you click the New, Insert, or Update button.

Vector Marker (for vector graphics):

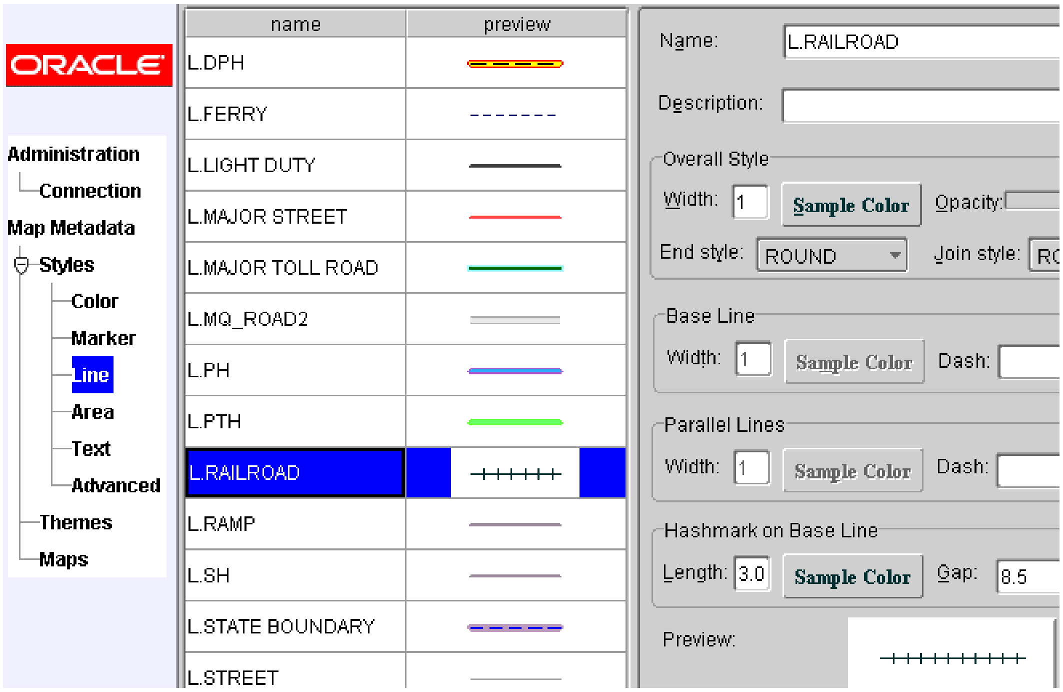
- **Type:** POLYGON (simple polygon only), POLYLINE (line string with one or more segments), CIRCLE, or RECTANGLE.
- **Stroke** (border) and **Fill** colors: The rectangle button displays the current color in its foreground text. You can click the rectangle button to display a dialog box to specify a new color by a swatch, HSB value, or RGB value.
- **Coordinates** or **Radius:** Coordinates for each vertex of a polygon or polyline, or for the upper-left corner, width, and height of a rectangle; or the number of screen pixels for the radius of a circle.

Buttons: New lets you enter information for a new style; **Insert** inserts a new style using the specified information; **Update** updates the style using the specified information; **Delete** removes the style; **Cancel** clears any information that you have entered for a new style.

7.5 Styles: Line Page

Figure 7–4 shows the Line page under the Styles category.

Figure 7–4 Line Page



name	preview
L.DPH	
L.FERRY	
L.LIGHT DUTY	
L.MAJOR STREET	
L.MAJOR TOLL ROAD	
L.MQ_ROAD2	
L.PH	
L.PTH	
L.RAILROAD	
L.RAMP	
L.SH	
L.STATE BOUNDARY	
L.STREET	

Name: L.RAILROAD

Description:

Overall Style
Width: 1 Opacity:
End style: ROUND Join style: RO

Base Line
Width: 1 Dash:

Parallel Lines
Width: 1 Dash:

Hashmark on Base Line
Length: 3.0 Gap: 8.5

Preview:

Name and **Preview** columns: List currently defined line styles, with a preview of each. (The styles listed depend on whether you selected the ALL_SDO_STYLES or USER_SDO_STYLES view at connection time.)

Name: Name of the style. Must be unique within a schema.

Description: Optional descriptive text about the style.

Overall Style:

- **Width:** Number of screen pixels for the width of the line.

- The rectangle button displays the current color in its foreground text. You can click the rectangle button to display a dialog box to specify a new color by a swatch, HSB value, or RGB value.
- **Opacity:** A value from 0 (transparent) to 255 (solid, or completely opaque).
- **End Style:** Style to be used at each end of the line: ROUND, BUTT, or SQUARE.
- **Join Style:** Style to be used at each vertex of the line: ROUND, BEVEL, or MITER.

Base Line: If applied, specifies attributes for the center line of the linear feature (for example, of a highway or river).

- **Width:** Number of screen pixels for the width of the center line.
- The rectangle button displays the current color in its foreground text. You can click the rectangle button to display a dialog box to specify a new color by a swatch, HSB value, or RGB value.
- **Dash:** Pattern to be used for drawing a dashed line, using the number of screen pixels for solid and the number of screen pixels for space (separated by a comma) for each segment. Example: 5.0,3.0 means a 5-pixel solid line followed by a 3-pixel space (gap).
- **Apply:** If checked, causes this feature to be applied to the style; if unchecked, causes the feature not to be applied to the style.

Parallel Lines: If applied, specifies attributes for the edges of the linear feature. Edges are two parallel lines, each an equal distance from the center line.

- **Width:** Number of screen pixels for the width of each edge.
- The rectangle button displays the current color in its foreground text. You can click the rectangle button to display a dialog box to specify a new color by a swatch, HSB value, or RGB value.
- **Dash:** Pattern to be used for drawing a dashed line, using the number of screen pixels for solid and the number of screen pixels for space (separated by a comma) for each segment. Example: 5.0,3.0 means a 5-pixel solid line followed by a 3-pixel space (gap).
- **Apply:** If checked, causes this feature to be applied to the style; if unchecked, causes the feature not to be applied to the style.

Hashmark on Base Line: If applied, specifies attributes for hash marks on each side of the center line of the linear feature.

- **Length:** Number of screen pixels for the length of each hash mark.

- The rectangle button displays the current color in its foreground text. You can click the rectangle button to display a dialog box to specify a new color by a swatch, HSB value, or RGB value.
- **Gap:** Number of screen pixels for the distance between each hash mark.
- **Apply:** If checked, causes this feature to be applied to the style; if unchecked, causes the feature not to be applied to the style.

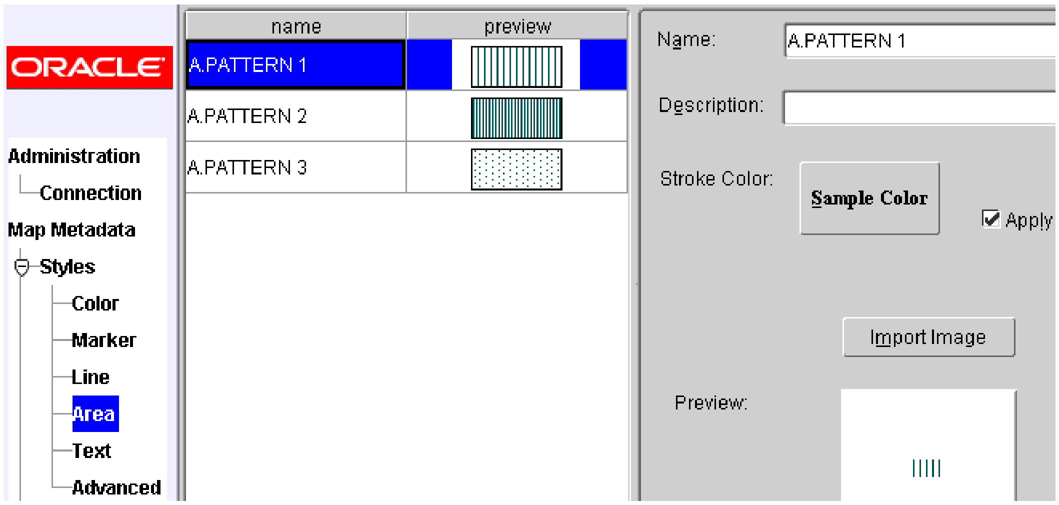
Preview: Shows a sample of the style as it would look with the current specifications. However, no changes are made to the style until you click the New, Insert, or Update button.

Buttons: New lets you enter information for a new style; **Insert** inserts a new style using the specified information; **Update** updates the style using the specified information; **Delete** removes the style; **Cancel** clears any information that you have entered for a new style.

7.6 Styles: Area Page

Figure 7–5 shows the Area page under the Styles category.

Figure 7–5 Area Page



name	preview
A.PATTERN 1	
A.PATTERN 2	
A.PATTERN 3	

Name: A.PATTERN 1

Description:

Stroke Color: Sample Color ☑ Apply

Name and **Preview** columns: List currently defined area styles, with a preview of each. (The styles listed depend on whether you selected the ALL_SDO_STYLES or USER_SDO_STYLES view at connection time.)

Name: Name of the style. Must be unique within a schema.

Description: Optional descriptive text about the style.

Stroke Color (for the border) and **Fill Color:**

- The rectangle button displays the current color in its foreground text. You can click the rectangle button to display a dialog box to specify a new color by a swatch, HSB value, or RGB value.
- **Apply:** If checked, the color is used; if unchecked, the color is not used. For example, you might specify an image, but not use any border (stroke) color.

Import Image: Displays a dialog box for specifying the file for the image to be used as a pattern for the area.

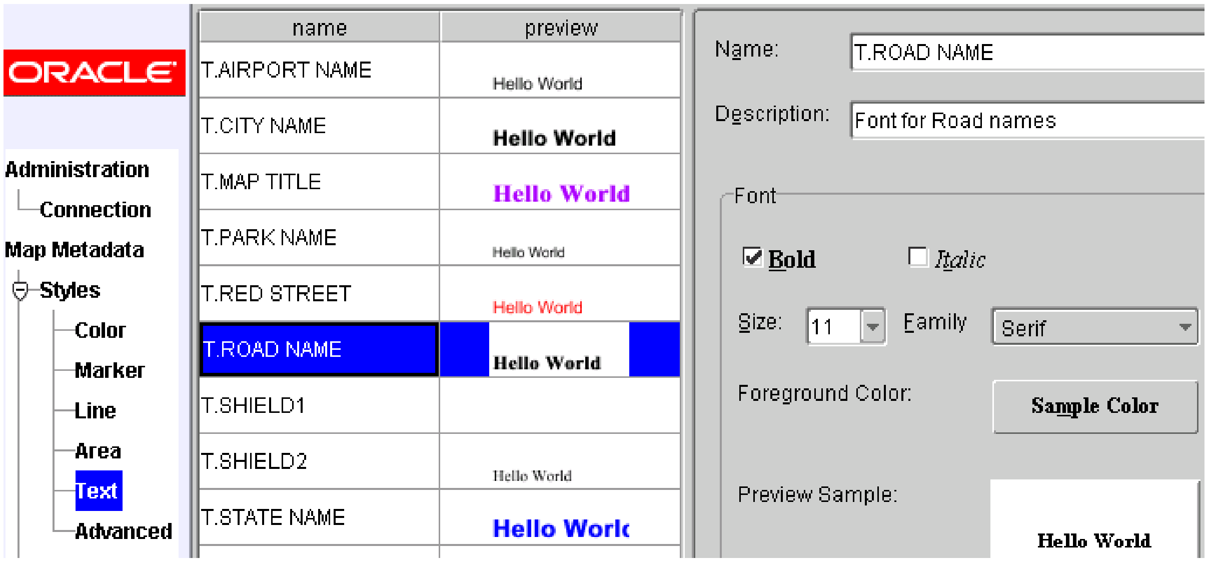
Preview: Shows a sample of the style as it would look with the imported image. However, no changes are made to the style until you click the New, Insert, or Update button.

Buttons: New lets you enter information for a new style; **Insert** inserts a new style using the specified information; **Update** updates the style using the specified information; **Delete** removes the style; **Cancel** clears any information that you have entered for a new style.

7.7 Styles: Text Page

Figure 7–6 shows the Text page under the Styles category.

Figure 7–6 Text Page



name	preview
T.AIRPORT NAME	Hello World
T.CITY NAME	Hello World
T.MAP TITLE	Hello World
T.PARK NAME	Hello World
T.RED STREET	Hello World
T.ROAD NAME	Hello World
T.SHIELD1	
T.SHIELD2	Hello World
T.STATE NAME	Hello Worlc

Name: T.ROAD NAME

Description: Font for Road names

Font

Bold ✓ Italic

Size: 11 Family Serif

Foreground Color:

Preview Sample: Hello World

Name and **Preview** columns: List currently defined text styles, with a preview of each. (The styles listed depend on whether you selected the ALL_SDO_STYLES or USER_SDO_STYLES view at connection time.)

Name: Name of the style. Must be unique within a schema.

Description: Optional descriptive text about the style.

Bold: If checked, displays the text in bold.

Italic: If checked, displays the text in italic.

Size: Font size.

Family: Font family. (Currently, only Java native font families are supported.)

Foreground Color: The rectangle button displays the current text foreground color in its foreground text. You can click the rectangle button to display a dialog box to specify a new color by a swatch, HSB value, or RGB value.

Preview Sample: Shows a sample of the style as it would look with the current information. However, no changes are made to the style until you click the New, Insert, or Update button.

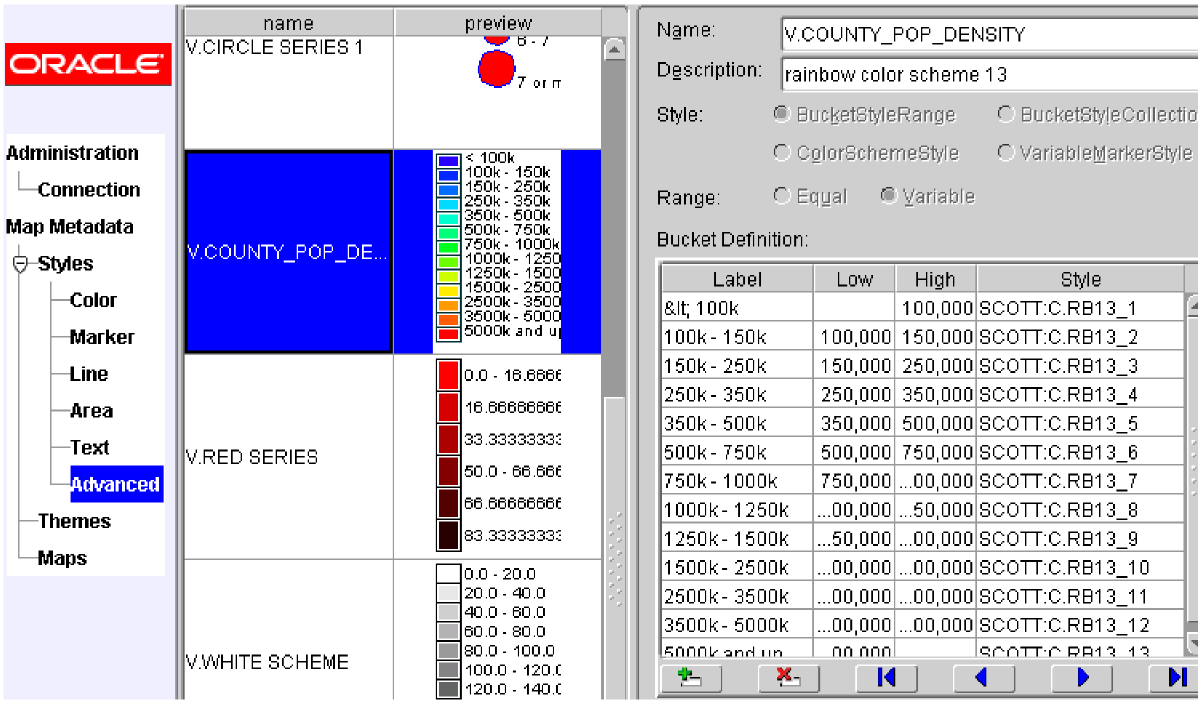
Buttons: New lets you enter information for a new style; **Insert** inserts a new style using the specified information; **Update** updates the style using the specified information; **Delete** removes the style; **Cancel** clears any information that you have entered for a new style.

7.8 Styles: Advanced Page

Figure 7–7 shows the Advanced page under the Styles category. (For a discussion of thematic mapping using advanced styles, including several examples, see Section 2.3.3.)

Note: To create and modify advanced styles, you must understand the types of advanced styles, which are explained in detail (with XML examples) in Section A.6.

Figure 7–7 Advanced Page



name	preview
V.CIRCLE SERIES 1	6 - 7 / 7 or m
V.COUNTY_POP_DE...	< 100k / 100k - 150k / 150k - 250k / 250k - 350k / 350k - 500k / 500k - 750k / 750k - 1000k / 1000k - 1250 / 1250k - 1500 / 1500k - 2500 / 2500k - 3500 / 3500k - 5000 / 5000k and u
V.RED SERIES	0.0 - 16.6666 / 16.66666666 / 33.3333333 / 50.0 - 66.666 / 66.66666666 / 83.3333333
V.WHITE SCHEME	0.0 - 20.0 / 20.0 - 40.0 / 40.0 - 60.0 / 60.0 - 80.0 / 80.0 - 100.0 / 100.0 - 120.0 / 120.0 - 140.0

Name: V.COUNTY_POP_DENSITY

Description: rainbow color scheme 13

Style: [✓ BucketStyleRange] BucketStyleCollectio ColorSchemeStyle VariableMarkerStyle

Range: Equal [✓ Variable]

Bucket Definition:

Label	Low	High	Style
&lt; 100k		100,000	SCOTT:C.RB13_1
100k - 150k	100,000	150,000	SCOTT:C.RB13_2
150k - 250k	150,000	250,000	SCOTT:C.RB13_3
250k - 350k	250,000	350,000	SCOTT:C.RB13_4
350k - 500k	350,000	500,000	SCOTT:C.RB13_5
500k - 750k	500,000	750,000	SCOTT:C.RB13_6
750k - 1000k	750,000	...00,000	SCOTT:C.RB13_7
1000k - 1250k	...00,000	...50,000	SCOTT:C.RB13_8
1250k - 1500k	...50,000	...00,000	SCOTT:C.RB13_9
1500k - 2500k	...00,000	...00,000	SCOTT:C.RB13_10
2500k - 3500k	...00,000	...00,000	SCOTT:C.RB13_11
3500k - 5000k	...00,000	...00,000	SCOTT:C.RB13_12
5000k and up	...00,000		SCOTT:C.RB13_13

Name and **Preview** columns: List currently defined advanced styles, with a preview of each. (The styles listed depend on whether you selected the ALL_SDO_STYLES or USER_SDO_STYLES view at connection time.)

Name: Name of the style. Must be unique within a schema.

Description: Optional descriptive text about the style.

Style: Type of style:

- BucketStyleRange: individual range-based buckets (See Section A.6.1 and Section A.6.1.2.)
- BucketStyleCollection: collection-based buckets with discrete values (See Section A.6.1 and Section A.6.1.1.)
- ColorSchemeStyle: color scheme style (See Section A.6.2.)
- VariableMarkerStyle: variable marker style (See Section A.6.3.)

Range: Equal if the style contains a series of buckets that contain an equally divided range of a master range; Variable if the style contains a series of buckets that do not necessarily contain an equally divided range of a master range. (See Section A.6.1.3.)

Bucket Definition: (Options and content vary depending on *Style* and *Range* settings.)

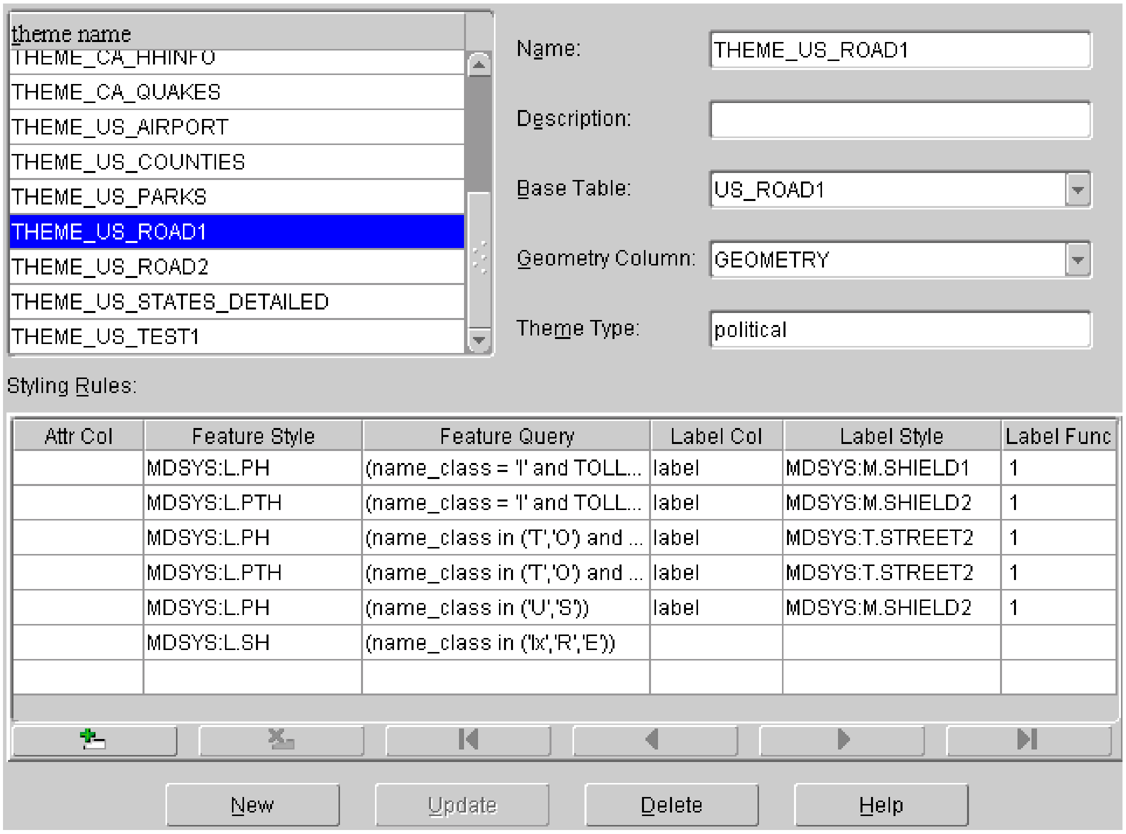
Icon Buttons: Insert an Empty Row inserts an empty row above the selected row; **Delete a Row** removes the selected row; **Move to Top** moves the selected row to the first row position; **Move Up One Row** moves the selected row above the row that is currently above it; **Move Down One Row** moves the selected row below the row that is currently below it; **Move to Bottom** moves the selected row to the last row position.

Buttons: New lets you enter information for a new style; **Insert** inserts a new style using the specified information; **Update** updates the style using the specified information; **Delete** removes the style; **Cancel** clears any information that you have entered for a new style.

7.9 Themes Page

Figure 7–8 shows the Themes page.

Figure 7–8 Themes Page



theme name
THEME_CA_HHINFO
THEME_CA_QUAKES
THEME_US_AIRPORT
THEME_US_COUNTIES
THEME_US_PARKS
THEME_US_ROAD1
THEME_US_ROAD2
THEME_US_STATES_DETAILED
THEME_US_TEST1

Name: THEME_US_ROAD1

Description:

Base Table: US_ROAD1

Geometry Column: GEOMETRY

Theme Type: political

Styling Rules:

Attr Col	Feature Style	Feature Query	Label Col	Label Style	Label Func
	MDSYS:L.PH	(name_class = 'I' and TOLL...	label	MDSYS:M.SHIELD1	1
	MDSYS:L.PTH	(name_class = 'I' and TOLL...	label	MDSYS:M.SHIELD2	1
	MDSYS:L.PH	(name_class in ('T','O') and ...	label	MDSYS:T.STREET2	1
	MDSYS:L.PTH	(name_class in ('T','O') and ...	label	MDSYS:T.STREET2	1
	MDSYS:L.PH	(name_class in ('U','S'))	label	MDSYS:M.SHIELD2	1
	MDSYS:L.SH	(name_class in ('Ix','R','E'))			

Theme Name column: Lists the names of currently defined themes.

Name: Name of the theme. Must be unique within a schema.

Description: Optional descriptive text about the theme.

Base Table: Name of the table or view that has the spatial geometry column to be associated with this theme. You can enter the name, or you can select from a list of tables. (The list contains all tables with entries in your USER_SDO_GEOM_METADATA view.)

Geometry Column: Name of the geometry column in the table or view to be associated with the theme. You can enter the name, or you can select from a list of columns. (The list contains all geometry columns in the selected table or view.)

Theme Type: Optional descriptive text to identify a type for the theme. Examples: *political*, *demographic*, *nature*.

Styling Rules: A tabular visual representation of the XML styling rules to be used with the theme. (For more information about theme definition, see Section 2.3, especially Section 2.3.1, "Styling Rules in Predefined Themes".)

Attr Col: Name of the attribute column (not of type SDO_GEOMETRY) in the table or view, or a SQL expression that references an attribute column in the table or view (for example, to specify a label that is a substring of the value in the column), to use with the bucket ranges or values in the advanced feature style (identified in the *Feature Style* column). If this column is empty or contains an asterisk (*), no attribute column is used with the feature style.

Feature Style: Name of the style to use for the styling rule.

Feature Query: A SQL condition to select rows from the table or view to use the feature style specified in the same row. You should use XML internal entities to identify special characters in the query (for example, `&lt;` instead of <). Examples:

```
name like 'I-%' and length(name) &lt; 6
name_class='U'
name_class in ('A', 'B', 'C')
```

Label Col: Name of the label column (not of type SDO_GEOMETRY) in the table or view, or a SQL expression that references one or more columns (not of type SDO_GEOMETRY) in the table or view (for example, to specify a label that is a substring of the value in the column), to use for text labels.

Label Style: Name of the text style to be used for the labels.

Label Func: A SQL expression or a value to determine whether or not the feature will be identified using the value in the label column. If the specified value or the value returned by the specified function is less than or equal to zero, the feature will not be identified. Examples:

```
0
1
8-length(label)
```

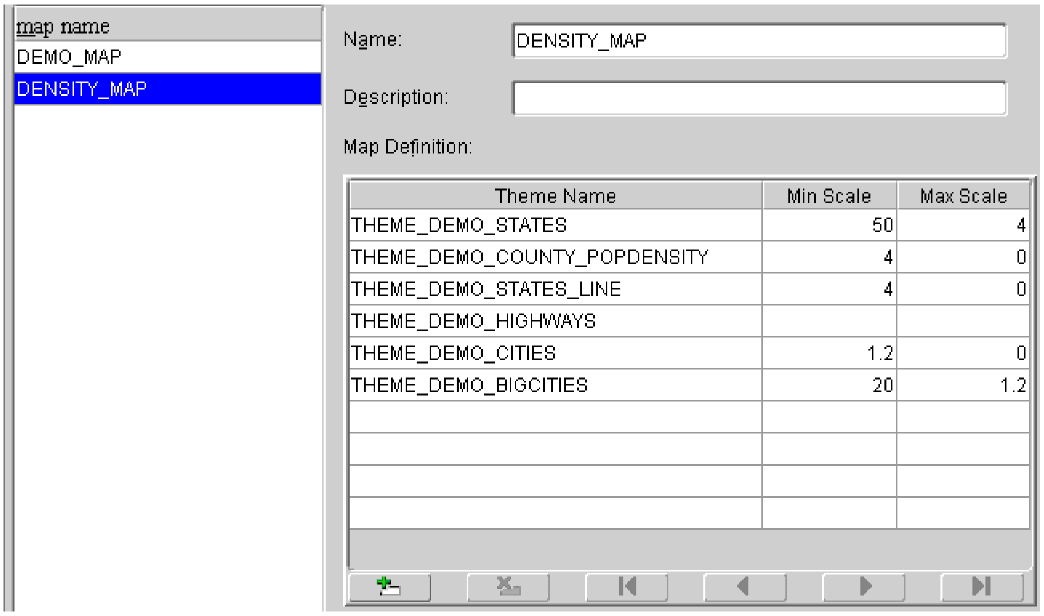
Icon Buttons: Insert an Empty Row inserts an empty row above the selected row; **Delete a Row** removes the selected row; **Move to Top** moves the selected row to the first row position; **Move Up One Row** moves the selected row above the row that is currently above it; **Move Down One Row** moves the selected row below the row that is currently below it; **Move to Bottom** moves the selected row to the last row position.

Buttons: New lets you enter information for a new theme; **Insert** inserts a new theme using the specified information; **Update** updates the theme using the specified information; **Delete** removes the theme; **Cancel** clears any information that you have entered for a new theme.

7.10 Maps Page

Figure 7–9 shows the Maps page.

Figure 7–9 Maps Page



map name
DEMO_MAP
DENSITY_MAP

Name: DENSITY_MAP

Description:

Map Definition:

Theme Name	Min Scale	Max Scale
THEME_DEMO_STATES	50	4
THEME_DEMO_COUNTY_POPDENSITY	4	0
THEME_DEMO_STATES_LINE	4	0
THEME_DEMO_HIGHWAYS		
THEME_DEMO_CITIES	1.2	0
THEME_DEMO_BIGCITIES	20	1.2

Map Name column: Lists the names of currently defined base maps.

Name: Name of the base map. Must be unique within a schema.

Description: Optional descriptive text about the base map.

Map Definition: A tabular visual representation of the XML definition of the base map. The order in which the themes are listed determines the order in which they are rendered, with the last listed theme on top. For more information about base map definition, see Section 2.4; for information about the minimum and maximum scale values, see Section 2.4.1.

Theme Name: Name of the theme to use for a layer in the base map.

Min Scale: Minimum value of the scale range for the theme.

Max Scale: Maximum value of the scale range for the theme.

Icon Buttons: Insert an Empty Row inserts an empty row above the selected row; **Delete a Row** removes the selected row; **Move to Top** moves the selected row to the first row position; **Move Up One Row** moves the selected row above the row that is currently above it; **Move Down One Row** moves the selected row below the row that is currently below it; **Move to Bottom** moves the selected row to the last row position.

Buttons: New lets you enter information for a new base map; **Insert** inserts a new base map using the specified information; **Update** updates the base map using the specified information; **Delete** removes the base map; **Cancel** clears any information that you have entered for a new base map.

A
XML Format for Styles, Themes, and Base Maps

This appendix describes the XML format for defining style, themes, and base maps using the MapViewer metadata views described in Section 2.5.

The metadata views for MapViewer styles (USER_SDO_STYLES and related views) contain a column named DEFINITION. For each style, the DEFINITION column contains an XML document that defines the style to the rendering engine.

Each style is defined using a syntax that is similar to SVG (scalable vector graphics). In the MapViewer syntax, each style's XML document must contain a single `<g>` element, which must have a `class` attribute that indicates the type or class of the style. For example, the following defines a color style with a filling color component:

```
<?xml version="1.0" standalone="yes"?>
   <svg width="1in" height="1in">
       <desc> red </desc>
           <g class="color" style="fill:#ff1100" />
   </svg>
```

Note that the MapViewer XML parser looks only for the `<g>` element in a style definition; other attributes such as the `<desc>` element are merely informational and are ignored.

The metadata views for MapViewer themes (USER_SDO_THEMES and related views) contain a column named STYLING_RULES. For each theme in these views, the STYLING_RULES column contains an XML document (a CLOB value) that defines the styling rules of the theme.

The metadata views for MapViewer base maps (USER_SDO_MAPS and related views) contain a column named DEFINITION. For each base map in these views, the DEFINITION column contains an XML document (a CLOB value) that defines the base map.

The following sections describe the XML syntax for each type of mapping metadata.

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 semicolon. (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 you 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; for example, 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`*.

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>
```

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>
   <desc></desc>
   <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.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 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 hashmark element, 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.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.

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 2.5.3).

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 that displays "Hello World!".

```
<?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>
```

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).

A.6.1 Bucket Style

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 … />
    </BucketStyle>
</AdvancedStyle>
```

In contrast with the other (primitive) styles, an advanced style always has a root element identified by the `<AdvancedStyle>` tag.

For bucket styles, a `<BucketStyle>` element is the only child of the `<AdvancedStyle>` element. Each `<BucketStyle>` element has one or more `<Buckets>` child elements, whose contents vary depending on the type of buckets.

A.6.1.1 Collection-Based Buckets with Discrete Values

If each bucket of a bucket style contains a collection of discrete values, use a `<CollectionBucket>` element to represent each bucket. Each bucket contains one or more values. The values for each bucket are listed as the content of the

<CollectionBucket> element, with multiple values delimited by commas. The following example defines three buckets.

```
<?xml version="1.0" ?>
  <AdvancedStyle>
    <BucketStyle>
      <Buckets>
        <CollectionBucket  seq="0" label="commercial"
            style="10015">commercial</CollectionBucket>
        <CollectionBucket  seq="1" label="residential"
            style="10031">residential, rural</CollectionBucket>
        <CollectionBucket  seq="2" label="industrial"
            style="10045">industrial, mining, agriculture</CollectionBucket>
    </Buckets>
  </BucketStyle>
</AdvancedStyle>
```

In the preceding example:

- The values for each bucket are one or more strings; however, the values can also be numbers.
- The name of the style associated with each bucket is given.
- The label attribute for each <CollectionBucket> element (*commercial*, *residential*, or *industrial*) is used only in a label that is compiled for the advanced style.
- The order of the <CollectionBucket> elements is significant. However, the values in the seq (sequence) attributes are informational only; MapViewer determines sequence only by the order in which elements appear in a definition.

Although not shown in this example, if you want a bucket for all other values (if any other values are possible), you can create a <CollectionBucket> element with #DEFAULT# as its attribute value. It should be placed after all other <CollectionBucket> elements, so that its style will be rendered last.

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.

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: 0-5, 6-11, 12-17, 18-23, and 24-29.

- 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 Style

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 also 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.

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: `&gt;`
For &, use: `&amp;`

A.6.3 Variable Marker Style

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.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+)>
<!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
>
```

```
<!ELEMENT rule (features, label?)>
<!ATTLIST rule column CDATA #IMPLIED>

<!ELEMENT features (#PCDATA?)>
<!ATTLIST features style CDATA #REQUIRED>

<!ELEMENT label (#PCDATA)>
<!ATTLIST label column CDATA #REQUIRED
                style  CDATA #REQUIRED>
```

The `<styling_rules>` element contains one or more `<rule>` elements and an optional `theme_type` attribute, which is used mainly used for certain kinds of predefined themes. If the value of the `theme_type` attribute is `image`, this is an image theme, and you must also specify the `image_format` and `image_column` attributes, and perhaps also the `image_resolution` and `image_unit` attributes, as explained in Section 2.3.4.1. For more information about image themes, see Section 2.3.4.

The `<styling_rules>` element can have a `key_column` attribute. This attribute is needed only if the theme is defined on a join view (a view created from multiple tables). In such a case, you must specify a column in the view that will serve as the key column to uniquely identify the geometries or images in that view. Without this key column information, MapViewer will not be able to cache geometries or images in a join view.

The `<styling_rules>` element can have a `caching` attribute, which specifies the caching scheme for each predefined theme. The `caching` attribute can have one of the following values: `NORMAL` (the default), `NONE`, or `ALL`.

- `NORMAL` causes MapViewer to try to cache the geometry data that was just viewed, to avoid repeating the costly unpickling process when it needs to reuse the geometries. Geometries are always fetched from the database, but they are not used if unpickled versions are already in the cache.
- `NONE` means that no geometries from this theme will be cached. This value is useful when you are frequently editing the data for a theme and you need to display the data as you make edits.
- `ALL` causes MapViewer to pin all geometry data of this theme entirely in the cache before any viewing request. In contrast to the default value of `NORMAL`, a value of `ALL` caches all geometries from the base table the first time the theme is viewed, and the geometries are not subsequently fetched from the database.

Each `<rule>` element must have a `<features>` element and may 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 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 filing a county geometry by looking up numeric values in the column named TOTPOP in the base table for this theme.

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.

See Section 2.3.1 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" >
```

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, MapViewer labels all features of the theme even if two or more labels will overlap in the display. (MapViewer always tries to avoid overlapping labels.) If label_always_on is FALSE (the default), when it is impossible to avoid overlapping labels, MapViewer disables the display of one or more labels so that no overlapping occurs. The label_always_on attribute can also be specified for a map feature (geoFeature element, described in Section 3.2.9), 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.

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

B
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 2.3.4.)

If you want to create a map request specifying an image theme with an image format that is not supported by MapViewer, you must first implement and register a custom image renderer for that format. For example, the ECW format in Example 3–5 in Section 3.1.5 is not supported by MapViewer; therefore, for that example to work, you must first implement and register an image renderer for ECW format images.

The interface `oracle.sdovis.CustomImageRenderer` is defined in the package `sdovis.jar`, which is located in the *`$ORACLE_HOME`*`/lbs/lib` directory in an Oracle Application Server environment. If you performed a standalone installation of OC4J, `sdovis.jar` is unpacked into `$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
 * MapViewer to support a custom image format.
 */
public interface CustomImageRenderer
{
  /**
   * The method is called by MapViewer to find out the image format supported
   * by this renderer. <br>
   * This format string must match the one specified in a custom image renderer
   * element defined in the MapViewer configuration file (mapViewerConfig.xml).
   */
  public String  getSupportedFormat() ;
  /**
   * Renders the given images. MapViewer calls this method
   * to tell the implementor the images to render, the current map
   * window in user space, and the MBR (in the same user space) for each
   * image.
   * <br>
   * The implementation should not retain any reference to the parameters
   * permanently.
   * @param g2  the graphics context to draw the images onto.
   * @param images  an array of image data stored in byte array.
   * @param mbrs an array of double[4] arrays containing one MBR for each
   *       image in the images array.
   * @param dataWindow the data space window covered by the current map.
   * @param deviceView the device size and offset.
   * @param at  the AffineTransform using which you can transform a point
   *           in the user data space to the device coordinate space. You can
   *           ignore this parameter if you opt to do the transformation
   *           yourself based on the dataWindow and deviceView information.
   * @param scaleImage a flag passed from MapViewer to indicate whether the
   *                  images should be scaled to fit the current device window.
   *                  If it is set to false, render the image as-is without
   *                  scaling it.
   */
  public void   renderImages(Graphics2D g2, byte[][] images, double[][] mbrs,
                             Rectangle2D dataWindow, Rectangle2D deviceView,
                             AffineTransform at, boolean scaleImage) ;
}
```

After you implement this interface, you must place your implementation class in a directory that is part of the MapViewer CLASSPATH definition, such as the *$MAPVIEWER*`/web/WEB-INF/lib` directory. If you use any native libraries to perform the actual rendering, you must ensure that any other required files (such as `.dll` and `.so` files) for these libraries are accessible to the Java virtual machine (JVM) that is running MapViewer.

After you place your custom implementation classes and any required libraries in the MapViewer CLASSPATH, you must register your class with MapViewer in its configuration file, `mapViewerConfig.xml` (described in Section 1.5). Examine, and edit as appropriate, the following section of the file, which tells MapViewer which class to load if it encounters a specific image format that it does not already support.

```
  <!-- ****************************************************************** -->
  <!-- ******************** Custom Image Renderers ********************** -->
```

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

In this example, for any ECW formatted image data loaded through the `<jdbc_image_query>` element of an image theme, MapViewer will load the class `com.my_corp.image.ECWRenderer` to perform the rendering.

Example B–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 purpose only, and the code shown is not necessarily optimal or even correct for all system environments. This implementation uses the ECW Java SDK, which in turn uses a native C library that comes with it. For MapViewer to be able to locate the native dynamic library, you may need to use the command-line option `-Djava.library.path` when starting the OC4J instance that contains MapViewer.

Example B–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);
    }

  }
}
```

C Using the Flash Mapping Client

This appendix explains how to use MapViewer to display maps within Macromedia Flash applications.

MapViewer provides limited Flash support through a mapping client that is a Macromedia Flash SWF file. This Flash mapping client is shipped only as a demonstration application. Its capabilities are limited and its performance has not been tuned.

The Flash mapping client can send map requests to MapViewer, load vector map data from MapViewer, and display map data. The client SWF file is named `flashmap.swf.` It is installed in the *`%MAPVIEWER_HOME%`*`/demo` directory, where *`%MAPVIEWER_HOME%`* is the top directory of the unpacked MapViewer files (such as `/oracle/lbs/MapViewer`). To use this client, you must copy the `flashmap.swf` file to the following directory:

```
%MAPVIEWER_HOME%/web
```

This SWF file can be referred to in Web pages, so that users can interact with maps in a Web browser.

C.1 How the Flash Mapping Client Works

The Flash mapping client works with MapViewer by sending an XML request and receiving vector map data (represented by an XML document) from MapViewer. Most MapViewer style types are supported by the Flash mapping client. You cannot use image markers with the Flash mapping client, except for SVG and Flash markers, as explained in Section C.4.

Maps are represented by vectors inside the Flash mapping client. All vector graphic components are drawn locally by the client. The Flash mapping client has full control over the map data, which makes rich client-side mapping capabilities possible. Mapping operations such as zooming in, zooming out, and panning can be handled locally without sending new requests to the MapViewer server, if the necessary map data is already loaded on the client system.

To provide clickable map features, you can associate any styled feature (geometry) with a hidden information string, which can specify nonspatial columns in the geometry table. The associated information can be displayed when a user clicks on or moves the mouse pointer over the styled feature. To associate styled features and attributes, use the `<hidden_info>` element in the styling rules definition of a theme, as explained in Section C.3.

To display maps without clickable features, you can use your existing base map, theme, and style definitions, although you may need to modify styles that are used in relevant themes to use styles based on SVG markers or Flash markers, as explained in Section C.4.

You can group features into themes, and you can allow users to select which themes are to be displayed.

C.2 Embedding the Flash Mapping Client in a Web Page

To embed the Flash mapping client in a Web page, insert the following lines in the source HTML file. Replace the strings in the square bracket pairs with the actual attribute values, to set the map size, data source, base map, and map center. (These attributes are described in Section 3.2.2.) Replace `[flashmap.swf URL]` with the value for *%MAPVIEWER_HOME%*`/web/flashmap.swf`, where *%MAPVIEWER_HOME%* is the name of the home directory where MapViewer is deployed.

```
<OBJECT
  id=Flashmap
  codeBase=http://download.macromedia.com/pub/shockwave/cabs/Flash/swFlash.cab#version=6,0,0,0
  height=[height of map in pixels]
  width=[width of map in pixels]
  classid=clsid:D27CDB6E-AE6D-11cf-96B8-444553540000>
  <PARAM NAME="Movie" VALUE="flashmap.swf?">
  <PARAM NAME="Src"
    VALUE="[flashmap.swf URL]?serverurl=[MapViewer URL]&datasource=[map data
source]&basemap=[base map name]&centerx=[map center X]&centery=[map center Y]&mapsize=[map
size]">
  <PARAM NAME="Quality" VALUE="High">
  <EMBED
    src="[flashmap.swf URL]?serverurl=[MapViewer URL]&datasource=[map data source]&basemap=[base
map name]&centerx=[map center X]&centery=[map center Y]&mapsize=[map size]"
    quality=high
    HEIGHT =[height of map in pixels]
    WIDTH =[width of map in pixels]
                     NAME="Flashmap"
    TYPE="application/x-shockwave-Flash"
    PLUGINSPAGE="http://www.macromedia.com/go/getFlashplayer">
  </EMBED>
</OBJECT>
```

Example C–1 shows an excerpt from an HTML file in which the MapViewer Flash mapping client is included in a table.

Example C–1 Including the Flash Mapping Client in an HTML File

```
<TABLE cellSpacing=1 cellPadding=1 width="500" border=1>
  <TR>
    <TD>
      <OBJECT
        id=Flashmap
    codeBase=http://download.macromedia.com/pub/shockwave/cabs/Flash/swFlash.cab#version=6,0,0,0
        height=400
        width=500
        classid=clsid:D27CDB6E-AE6D-11cf-96B8-444553540000>
        <PARAM NAME="Movie" VALUE="flashmap.swf?">
        <PARAM NAME="Src"
VALUE="http://www.xyzcorp.com:8888/MapViewer/flashmap.swf?datasource=mvdemo&basemap=Flash_
demo&centerx=-122.3615&centery=37.82660&mapsize=1.0">
        <PARAM NAME="Quality" VALUE="High">
        <EMBED
src="http://www.xyzcorp.com:8888/MapViewer/flashmap.swf?datasource=mvdemo&basemap=Flash_
demo&centerx=-122.3615&centery=37.82660&mapsize=1.0"
          quality=high
          WIDTH=500
          HEIGHT=400
          NAME="Flashmap"
          TYPE="application/x-shockwave-Flash"
          PLUGINSPAGE="http://www.macromedia.com/go/getFlashplayer">
        </EMBED>
      </OBJECT>
    </TD>
  </TR>
</TABLE>
```

C.3 Creating a Theme with Clickable Styled Features

You can associate each styled feature (geometry) of a theme with attributes, which can be dynamically displayed when the user's mouse cursor moves over the feature. To do this, include a `<hidden_info>` element in the styling rules of the theme. The attributes to the `<hidden_info>` element must specify columns in the same table on which the theme definition is based. You must insert the theme with clickable styled features in the USER_SDO_THEMES view; you cannot use the Map Definition Tool to create this theme.

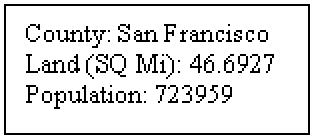
Example C–2 creates a theme that allows users to click on a county in the map to see the name, area in square miles, and population of the county.

Example C–2 Creating a Theme with Clickable Styled Features

```
<?xml version="1.0" standalone="yes"?>
<styling_rules >
  <hidden_info>
    <field column="COUNTY" name="County"/>
    <field column="LANDSQMI" name="Land (SQ Mi)"/>
    <field column="TOTPOP" name="Population"/>
  </hidden_info>
  <rule >
    <features style="SCOTT:C.COUNTIES">  </features>
  </rule>
</styling_rules>
```

In Example C–2, the column attribute of the `<hidden_info>` element associates the COUNTY, LANDSQMI, and TOTPOP columns with the theme, and the name attribute associates a label with the column value. Figure C–1 shows the display when the user clicks on San Francisco county.

Figure C–1 Display of Theme with Clickable Styled Features



County: San Francisco
Land (SQ Mi): 46.6927
Population: 723959

C.4 SVG and Flash Markers

Two types of image-related markers can be used in the Flash mapping client: SVG markers and Flash markers; otherwise, image markers are not supported with the Flash mapping client.

SVG markers are defined using a scalable vector graphics (SVG) string. An example of an SVG marker is `M.STAR`, which is one of the default styles created by the `defaultstyles.sql` file (see Section 1.4.2.3). The `M.STAR` marker has the following definition:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<desc></desc>
<g class="marker"  style="stroke:#000000;fill:#ff0000;width:15;height:15">
<polygon points="138.0,123.0, 161.0,198.0, 100.0,152.0, 38.0,198.0, 61.0,123.0,
0.0,76.0, 76.0,76.0, 100.0,0.0, 123.0,76.0, 199.0,76.0" />
</g>
</svg>
```

Flash markers are predefined in the Flash mapping client SWF file. The following Flash markers are currently supported:

- `MARKER_ECLIPSE`
- `MARKER_PIN`
- `MARKER_SOLIDPOINT`
- `MARKER_POINT`
- `MARKER_SHIELD1`
- `MARKER_SHIELD2`

To use these Flash markers, you must insert an entry for each one in the USER_SDO_STYLES view. Example C–3 inserts an entry for a style named `M.FLASH_PIN` based on the `MARKER_PIN` Flash marker.

Example C–3 Creating a Flash Marker Style

```
INSERT INTO
  user_sdo_styles(name, definition, type)
  values(
    'M.FLASH_PIN',
    '<Flash_marker name="MARKER_PIN" height="18" width="14" />',
    'FLASH_MARKER') ;
```

The name attribute value must be a Flash marker name as defined in the Flash mapping client SWF file (MARKER_PIN in Example C–3). You can use the height and width attributes to specify the size of the marker.

After a style based on a Flash marker is inserted into the USER_SDO_STYLES view, you can use it in theme definitions.

C.5 Simplifying Map Geometries

Simplifying (sometimes referred to as *generalizing*) a map geometry refers to reducing the number of vertices to produce a displayed geometry with less fine resolution than the original geometry. For example, if users do not need to see the hundreds or thousands of turns in the course of a river or a highway, you can get better performance and acceptable displays if the map geometry is simplified to show only the "major" turns.

You can use the min_dist attribute in the <styling_rules> element to simplify map geometries. The min_dist attribute specifies the minimum distance (in pixels) between two adjacent shape points in a polygon or line string geometry. Any two shape points that are within the min_dist distance are drawn as a single point by the Flash map client.

The value of the min_dist attribute must be a number greater than or equal to 1. A small value results in greater map detail, while a large value results in less map detail but faster performance.

Note: Do not use the min_dist attribute with polygon geometries that share any boundaries. For example, do not use this attribute with a map of the counties in a state, such as the one shown in Figure 3–1 in Section 3.1.7.

Example C–4 is the styling rules definition of a theme that simplifies the geometries by specifying 5 pixels as the minimum distance between adjacent shape points.

Example C–4 Simplifying Map Geometries

```
<?xml version="1.0" standalone="yes"?>
<styling_rules min_dist="5">
  <rule >
    <features style="SCOTT:L.PH">  </features>
    <label column="ROUTEN" style="SCOTT:M.FLASH_SHIELD1"> 1 </label>
  </rule>
</styling_rules>
```

D Connection Pools and Java Object Cache in MapViewer

This appendix describes how MapViewer uses Oracle JDBC connection pooling and caching of Java objects to provide efficient performance. The information in this appendix might help you to configure and use MapViewer for the best possible performance on your system.

As described in Chapter 2 and Chapter 3, MapViewer creates a cartographic map in response to a map request based on one or more of the following items specified in a map request: base map, JDBC themes, predefined themes, and geometry features. One or more sets of features from an Oracle database are associated with each of these items (except the inline geometry features), and each JDBC theme can refer to a distinct database. Fetching each set of features from a database sequentially with one JDBC connection would be a slow process, and repeatedly fetching a set of features for the same or different map requests would waste time. To achieve fast response time for each map request and higher server throughput, MapViewer uses Oracle JDBC connection pooling, Oracle Object Cache, and Java threads to perform query processing on one or more back-end databases.

When MapViewer is started in an OC4J instance, it creates a JDBC connection pool manager to maintain a set of JDBC connection pools to handle all JDBC themes in its life span. When a map engine processes a JDBC theme specified in a map request, it checks with the JDBC connection pool manager for a JDBC connection pool to the database that the theme refers to. If one already exists, the map engine gets a JDBC connection from the connection pool to process the JDBC theme. After the process is complete, the map engine returns the connection to the pool rather than closing it. If a JDBC connection pool to the database referred to in the JDBC theme does not exist, the map engine will ask the JDBC connection pool manager to create one and maintain it in its set of JDBC connection pools. There could be more than one JDBC connection pool because there could be more than one JDBC theme in a map request, and those JDBC themes could refer to two or more different databases. All JDBC connections in a JDBC connection pool share one single physical JDBC connection to a database server. It is efficient to reuse the JDBC connection without creating and closing it for each request. After a map engine finishes processing a map request, it closes all associated JDBC connections. This prevents connection leaks that could eventually lock MapViewer.

Whenever MapViewer processes an `add_data_source` request, it creates a set of map engines for the data source according to the number of mappers specified in the request, and it creates one JDBC connection pool for the map engines to use. When a map engine processes a map request, it gets a JDBC connection from this JDBC connection pool to process each predefined theme in the request.

A map engine uses a thread and a different JDBC connection to retrieve features for each theme specified in a map request, and it synchronizes them to run concurrently. On an application server system with multiple CPUs, this allows MapViewer to provide faster response time and higher system throughput.

In addition, MapViewer uses JDBC Object Cache to cache geometry objects retrieved from the connected databases. MapViewer maintains a set of geometry objects to a predefined volume in memory and on disk. This avoids the need to unpickle (unstream) geometry objects in processing the result set of a spatial query. If a geometry object is in the cache, the cached objects are used directly; otherwise, the geometry object is unpickled and saved in the cache. The behavior of this object cache is set by the attribute values used in the `spatial_data_cache` element in the `mapViewerConfig.xml` file. (For more information, see Section 1.5, especially Section 1.5.6.)

Index