Oracle® Spatial

Spatial Map Visualization Developer's Guide

23ai
F46998-04
August 2024

ORACLE

Oracle Spatial Spatial Map Visualization Developer's Guide, 23ai

F46998-04

Primary Author: Lavanya Jayapalan

Contributors: Chuck Murray, L.J. Qian, Jayant Sharma, Honglei Zhu

Contents

2 Map Visualization Concepts

3 Map Visualization Servers

4 Oracle Maps

C OGC WMS Support in the Map Visualization Component

D OGC WMTS Support in the Map Visualization Component

Index

List of Examples

List of Figures

List of Tables

Preface

Oracle Spatial Map Visualization Developer's Guide describes how to use the Oracle Spatial map visualization component (sometimes referred to as *the map visualizer* or *the visualizer*), a tool that renders maps showing different kinds of spatial data.

Audience

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

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

Documentation Accessibility

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

Access to Oracle Support

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

Related Documentation

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

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

Acknowledgments

This product includes color specifications and designs developed by Cynthia Brewer (http://colorbrewer.org/).

Use of OpenStreetMap (OSM) tiles is subject to their copyright (© OpenStreetMap contributors) and terms of use. For more information, see http://www.openstreetmap.org/copyright and http://wiki.openstreetmap.org/wiki/Tile_usage_policy.

Conventions

The following text conventions are used in this document:

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

1

Introduction to the Map Visualization Component 23.2

The Oracle Spatial map visualization component (sometimes referred to *the visualization component*) is a programmable tool for rendering maps using spatial data managed by Oracle Spatial.

The map visualization component 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.

- Overview of the Map Visualization Component
  The map visualization component enables developers to incorporate highly interactive maps and spatial analysis into business applications.
- Getting Started with the Map Visualization Component
  You can get started with the map visualization component 23.2, by performing the following tasks.
- Prerequisite Software for the Map Visualization Component
  This section explains the prerequisites required for installing the map visualization component 23.2.
- Deploying MapViewer-Based Applications with the Map Visualization Component
  If you have any MapViewer-based applications, you can deploy them with the map visualization component.
- Administering the Map Visualization Component
  This topic introduces the map visualization component Administration page and some administrative and configuration tasks that you can perform, such as adding new data sources, managing map tile layers used by Oracle Maps, and setting logging levels.
- High Availability and the Map Visualization Component (WebLogic Server Only)
  Users can benefit from the high availability features of Oracle WebLogic Server.
- Secure Map Rendering
  You can implement secure map rendering based on a web user's identity.

1.1 Overview of the Map Visualization Component

The map visualization component enables developers to incorporate highly interactive maps and spatial analysis into business applications.

Application content can be combined with maps and data from a variety of web services and data formats such as GeoJSON. It is deployed in a Java EE container or in the Oracle Java Cloud Service.

The map visualization component includes the following:

- A map rendering engine to expose cartographic render functions to web applications.

- A suite of application programming interfaces (APIs) that allow programmable access to visualization features. These APIs include XML, Java, and an HTML5-based JavaScript API.
- A graphical map builder tool that enables you to create map symbols, define spatial data rendering rules, and create and edit map visualization component objects.
- Oracle Maps, which includes map cache and FOI (feature of interest) servers that facilitate the development of interactive geospatial web applications.
- Packaged content for admin areas (countries, states, counties, and so on) as JSON files.

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

The JavaScript API enables you to create highly interactive web applications that use the Oracle Maps feature of the map visualization component.

The Map Builder tool simplifies the process of creating and managing map, theme, and symbology metadata in a spatial database. For information about this tool, see #unique_26.

Oracle Maps, built on core map visualization component features, uses a map tile server that caches map image tiles, and a feature of interest (FOI) server that streams live data out of a database to be displayed as interactive features on a map. You can use the HTML5-based JavaScript API with Oracle Maps to provide sophisticated mapping solutions. Oracle Maps also allows for advanced customization and querying capabilities.

The map visualization component supports two-dimensional vector geometries stored in Oracle Spatial, as well as GeoRaster data and data in the Oracle Spatial topology and network data models. The map visualization component is also an Open Geospatial Consortium (OGC)-compliant web map service (WMS) and web map tile service (WMTS) server.

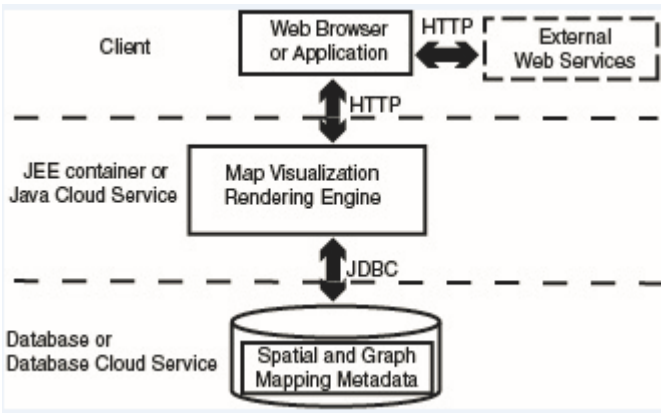
The

- Map Visualization Component Architecture

1.1.1 Map Visualization Component Architecture

The following figure illustrates the architecture of the map visualization component.

Figure 1-1 Map Visualization Component Architecture



As shown in the preceding figure:

- The map visualization component can communicate with a client web browser or application or external web services using the HTTP protocol.
- With a client web browser or application, it communicates using HTTP protocol with the map visualization rendering engine in a JEE container or the Java cloud service.
- It performs spatial data access (reading and writing Oracle Spatial data) through JDBC calls to the database or the database cloud service.
- The database or database cloud service includes Oracle Spatial, as well as mapping metadata.

1.2 Getting Started with the Map Visualization Component

You can get started with the map visualization component 23.2, by performing the following tasks.

1. Either before or after you install and deploy the map visualization component, read Map Visualization Concepts to be sure you understand important terms and concepts.
2. Ensure that you have the prerequisite software (see Prerequisite Software for the Map Visualization Component).
3. Install (if necessary) the map visualization component from the Downloads page. Deployment of the `mapviewer_23_2.ear` file enables you to get started with the map visualization component.
4. Use the map visualization component for some basic tasks. For example, create an Oracle Maps application (see #unique_29).
5. Optionally, use the Map Builder tool (described in #unique_26) to familiarize yourself with styles, themes, and maps, and the options for each, and optionally to preview spatial data.

1.3 Prerequisite Software for the Map Visualization Component

This section explains the prerequisites required for installing the map visualization component 23.2.

To use the map visualization component, you must have the following software.

- Oracle Database 12.1 or later
- A supported JEE server (see Supported JEE servers for the map visualization component)
- Java SDK (JDK) 1.8, build 1.8.0_181 or later

The map visualization component also supports the headless AWT mechanism in J2SE SDK, which enables the visualization component 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 the map visualization component:

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

1.4 Deploying MapViewer-Based Applications with the Map Visualization Component

If you have any MapViewer-based applications, you can deploy them with the map visualization component.

If you want to continue to use your existing MapViewer configuration, follow the instructions in this section.

Applications written using the MapViewer V2 API will work with the map visualization component.

(You can also use an existing MapViewer configuration in WebLogic Server on Oracle Fusion Middleware Release 12.2.1.x.0 and later.)

1.5 Administering the Map Visualization Component

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

- Logging in to the Map Visualization Component Administration Page
- Configuring the Map Visualization Component
- Creating a Map Visualization Component Data Source Using a JDBC Container
- Creating a Map Visualization Component Data Source Using Oracle Service Name
- Performing Map Visualization Component Administrative Tasks

1.5.1 Logging in to the Map Visualization Component Administration Page

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

Click the **Admin** link at the top right to display the map visualization component Administration page.

You can use this administration page to perform administrative tasks, such as clearing cached data, creating tile layers, managing tile layers, and restarting the server (use the **Monitoring** link to restart).

1.5.2 Configuring the Map Visualization Component

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

After you modify this file, you must restart the container to have the changes take effect; however, you can instead use the map visualization component Administration page to restart

only the map visualization component servlet (instead of the entire Java EE instance, which may have other applications deployed and running) if you installed the map visualization component with a standalone Glassfish instance.

The map visualization component configuration file defines the following information in XML format:

- Logging information, defined either through container-controlled logging (recommended) or in the `<logging>` element (see Specifying Logging Information)
- Map image file information, defined in the `<save_images_at>` element (see Specifying Map File Storage and Life Cycle Information)
- Web proxy information for accessing external information across a firewall, defined in the `<web_proxy>` element (see Specifying a Web Proxy)
- Global map "look and feel" configuration, defined in the `<global_map_config>` element (see Specifying Global Map Configuration Options)
- Internal spatial data cache settings, defined in the `<spatial_data_cache>` element (see Customizing the Spatial Data Cache)
- Permanent map data sources, defined in the `<map_data_source>` element (see Defining Permanent Map Data Sources)
- Security configurations, defined in the `<security_config>` element
- WMS services configurations, defined in the `<wms_config>` element
- External attribute data provider registration, defined in `<ns_data_provider>` elements
- Map tile server configurations, defined in the `<map_tile_server>` element
- UTF grid lifecycle parameters, defined in the `<utfgrid_life_cycle>` element
- External spatial data provider registration, defined in the `<s_data_provider>` element
- Map data sources, defined in the `<map_data_source>` element
- Map data server stream parameters, defined in the `<mds_config>` element
- OAM logout parameters, defined in the `<oam_logout>` element

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

Example 1-1 Sample Map Visualization Component Configuration File

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

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

<MapperConfig>

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

  <!-- Uncomment the following to modify logging. Possible values are:
       log_level = "fatal"|"error"|"warn"|"info"|"debug"|"finest"
                   default: info) ;
```

```
     log_thread_name = "true" | "false" ;
     log_time = "true" | "false" ;
     one or more log_output elements.
-->
<!--
  <logging log_level="info" log_thread_name="false"
           log_time="true">
     <log_output name="System.err" />
     <log_output name="../log/mapviewer.log" />
  </logging>
-->

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

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

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

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

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

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

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

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

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

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

```
      prohibited.

    - If you put  "*" in an <ips> element, then all IPs are allowed, except
      those specified in <ips_exclude> and <ip_range_exclude>.
      On the other hand, if you put "*" in an <ips_exclude> element, no one
      will be able to access the map visualization component (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 the map visualization component 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
       the map visualization component in map requests or for setting a logo image URL,
if
       such URLs cannot be accessed without the proxy.
   -->

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

  <!-- ********************************************************************** -->
  <!-- ********************** Security Configuration ********************** -->
  <!-- ********************************************************************** -->
  <!-- Here you can set various security related configurations of the map visualization
component.
  -->

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

  <!-- ********************************************************************** -->
  <!-- ********************** Global Map Configuration ******************** -->
  <!-- ********************************************************************** -->
  <!-- Uncomment and modify the following to specify systemwide parameters
       for generated maps. You can specify your copyright note, map title, and
```

```
       an image to be used as a custom logo shown on maps. The logo image must
       be accessible to this map visualization component and in either GIF or JPEG
format.
       Notes:
         -  To disable a global note or title, specify an empty string ("") for
            the text attribute of <note> and <title> element.
         - position specifies a relative position on the map where the
                 logo, note, or title  will be displayed. Possible values are
                 NORTH, EAST, SOUTH, WEST, NORTH_EAST, SOUTH_EAST,
                 SOUTH_WEST, NORTH_WEST, and CENTER.
         - image_path specifies a file path or a URL (starts with "http://")
                   for the image.

       <rendering> element attributes:
       - Local geodetic data adjustment: If allow_local_adjustment="true",
         The map visualization component automatically performs local data
         "flattening" with geodetic data if the data window is less than
         3 decimal degrees. Specifically, the map visualization component 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", The map visualization component will
         apply a globular projection on the fly to geometries being displayed.
         If use_globular_projection="false" (the default), The map visualization
component does no map
         projection to geodetic geometries. This option has no effect on
         non-geodetic data.
   -->

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

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

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

       To disable the cache, set max_cache_size to 0.

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

```
   -->

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

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

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

  <!-- this is the default data provider that comes with the map visualization
component;
       refer to the documentation on how to use it.

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

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

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

  </ns_data_provider>
  -->

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

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

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

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

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

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

</MapperConfig>
```

This map visualization component configuration topic includes the following subtopics.

- Specifying Logging Information
- Specifying Map File Storage and Life Cycle Information
- Specifying a Web Proxy
- Specifying Global Map Configuration Options

- Customizing the Spatial Data Cache
- Specifying the Security Configuration
- Registering a Custom Spatial Provider
- Registering Custom Nonspatial Data Providers
- Customizing SRS Mapping
- Customizing WMS GetCapabilities Responses
- Customizing WMTS GetCapabilities Responses
- Configuring the Map Tile Server for Oracle Maps
- Defining Permanent Map Data Sources
- Configuring and Securing the Map Data Server for the HTML5 API

1.5.2.1 Specifying Logging Information

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

There are two ways to configure the map visualization component's logging: container-controlled logging, and using the `<logging>` element in the configuration file.

Container-Controlled Logging

If the `<logging>` element in the `mapViewerConfig.xml` file is commented out or missing, the map visualization component uses container-controlled logging, specifically using the following loggers:

- `oracle.mapviewer.ws` for all web server (maps, tiles, features) messages
- `oracle.mapviewer.access` for all user access
- `oracle.mapviewer.sdovis` for all rendering (maps, themes, features) messages
- `oracle.mapviewer.webconsole` for all administrative console log messages

Using the <logging> Element

If the `<logging>` element in the `mapViewerConfig.xml` file is in use (that is, if the `<logging>` element is not commented out or missing), The map visualization component uses that information instead of using container-controlled logging. The `<logging>` element can have the following attributes and subelements:

- The `console_log_level` attribute controls the levels of information that are recorded in the log, which in turn affect the log output volume. Set the `console_log_level` attribute value to one of the following, listed from most restrictive logging to least restrictive logging: `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `FINEST`. The `FATAL` level outputs the least log information (only unrecoverable events are logged), and the other levels are progressively more inclusive, with the `FINEST` level causing the most information to be logged. For production work, a level of `WARN` or more restrictive (`ERROR` or `FATAL`) is recommended; however, for debugging you may want to set a less restrictive level.
- The `file_limit` attribute controls the maximum file size of a log file. The unit is Mb and the default value is 50 (that is, by the default value for the maximum log file size is 50Mb).

- The `file_count` attribute determines the number of log files created. The files are rotated (that is, when the last log file reaches its maximum size, the first log file is reused). The default value is 10.
- The `<logger>` subelement specifies the log level for a particular logger.
- The `<log_output>` subelement identifies output for the logging information. By default, log records are written to the system error console. You can change this to the system output console or to one or more files, or some combination. If you specify more than one device through multiple `<log_output>` subelements, the logging records are sent to all devices, using the same logging level and attributes.

1.5.2.2 Specifying Map File Storage and Life Cycle Information

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

A mapping client can request that the map visualization component 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 to `GIF_URL` or `PNG_URL`. In this case, the map visualization component saves the requested map image as a file on the host system where the map visualization component 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 map visualization component host. The map image URL sent to the mapping client is the map image base URL plus the map image file name. For example, if the map image base URL is `http://dev04.example.com:1521/mapviewer/images`, the map image URL for `omsmap1.gif` will be `http://dev04.example.com:1521/mapviewer/images/omsmap1.gif`.

  Default value: `url=`*`$HOST_URL`*`/mapviewer/images`
- The `path` attribute identifies the path of the directory where all map image files are saved on the map visualization component host system. This directory must be accessible by HTTP and must match the map image URL. Map image files saved in the directory specified by the `path` attribute should be accessible from the URL specified by the `url` attribute.

  However, if you are deploying the map visualization component to WebLogic Server, the default value for the `path` attribute (`../web/images`) is not correct. The path attribute value in this case should be `../../images`, because the physical "images" directory is `mapviewer_23_2.ear/web.war/images`; so using relative path, the value should be `../../images` for the `path` attribute to resolve to the physical directory.
- The `life` attribute specifies the number of minutes that a generated map image is guaranteed to stay on the file system before the image is deleted. If the `life` attribute is

specified, the `recycle_interval` attribute controls how frequently the map visualization component checks for possible files to delete.

Default: The map visualization component never deletes the generated map images.

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

  Default value: `480` (8 hours)

1.5.2.3 Specifying a Web Proxy

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

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

If the web proxy requires authentication, you can specify the `user` and `password` attributes. If the password value is preceded by the exclamation mark (`!`) character, the password value will be encrypted on the first loading of the configuration file. For example:

```
<web_proxy host="www-proxy.mycorp.com" port="80" user="uservalue" password="!pwdvalue" />
```

1.5.2.4 Specifying Global Map Configuration Options

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

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

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

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

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

Set the map title through the `<title>` element of the `<global_map_config>` element. You can also set the map title in an individual map request by specifying the `title` attribute with the `<map_request>` element, and in this case, the title in the map request is used instead of the

global title in the map visualization component 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 the map visualization component is running.
- The position attribute provides a positioning hint to the map visualization component 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 the map visualization component is running.
- The position attribute provides a positioning hint to the map visualization component 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 map visualization component 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 map visualization component 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 the map visualization component 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 the map visualization component to retrieve the logo image. For information about specifying a web proxy, see Specifying a Web Proxy.

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 Map Legend. The default position for a map legend is SOUTH_WEST.)

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

1.5.2.5 Customizing the Spatial Data Cache

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

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

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

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

  Default value: `64`

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

  Default value: `false`

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

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

For detailed information about the caching of predefined themes, see Caching of Predefined Themes.

1.5.2.6 Specifying the Security Configuration

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

However, if the map visualization component data source information is exposed, malicious attackers might be able to abuse this capability and obtain sensitive information. To prevent this from happening, you can make sure the map visualization component always connects to a database schema that has very limited access rights and hosts only non-sensitive

information, and you can also reject all `<info_request>` requests by specifying the `<security_config>` element as follows:

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

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

You can also define remote URLs that the map visualization component built-in proxy servlet is allowed to communicate with. Use commas to separate such URLs. You can end a URL with the * (asterisk) wildcard character to allow multiple URLs that start with a path. The following example specifies one remote URL:

```
<security-config>
    …
    <proxy_enabled_hosts>
      foo.com:8080/mapviewer
    </proxy_enabled_hosts>
    …
</security-config>
```

To facilitate the map visualization component's HTTPS (SSL) connection with external web sites that use self-signed certificates, you may register those certificates here. Use one entry for each server that requires an HTTPS connection. The following example shows an excerpt specifying one entry:

```
<security_config>
    …
    <certificates>
        <entry>
           <host_name>fooserver.com</host_name>
           <keystore_file>/scratch/fooserver.jks</keystore_file>
           <key>123456</key>
        </entry>
    </certificates>
    …
</security_config>
```

The subelements for each certificate's `<entry>` element are the following.

The `host_name` attribute specifies the IP address or domain name of the server that requires an HTTPS connection.

The `keystore_file` attribute specifies the file containing a single self-signed certificate. After you obtain the certificate (typically a `.pem` file) from the server site, you can create a key store by using the Java `keytool` command. For example:

```
keytool -import -file fooserver.pem -alias fooserver -keystore fooserver.jks
```

The `key` attribute specifies the password that you provided when creating the key store file. (It is used to ensure the integrity of the key store file itself.)

1.5.2.7 Registering a Custom Spatial Provider

The map visualization component can render spatial data that is in an external (non-Oracle Spatial) native format, such as shapefile, if there is a spatial provider implementation registered

for the format. For information about implementing an external spatial data provider (in connection with custom geometry themes), see Custom Geometry Themes.

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

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

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

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

1.5.2.8 Registering Custom Nonspatial Data Providers

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

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

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

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

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

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

1.5.2.9 Customizing SRS Mapping

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

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

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

1.5.2.10 Customizing WMS GetCapabilities Responses

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

> **Note:**
>
> There is a separate WMS configuration file (wmsConfig.xml) that contains more information than is in the <wms_config> element in the map visualization component configuration file. It is recommended that you define any custom WMS configuration parameters in this separate WMS configuration file; any settings there will override any conflicting settings in the <wms_config> element in the map visualization component configuration file.
>
> For more information about the wmsXonfig.xml file, see Customizing GetCapabilities Responses: Additional Options.

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

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

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

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

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

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

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

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

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

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

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

1.5.2.11 Customizing WMTS GetCapabilities Responses

The map visualization component can be used as an Open Geospatial Consortium WMTS (Web Map Tile Service) 1.0.0 compliant server, enabling tile layers defined in the USER_SDO_CACHED_MAPS metadata view to be retrieved through WMTS requests. A WMTS client can send to the map visualization component the `GetCapabilities` request. In response, the map visualization component will send back the list of tile layers that it hosts and other important information, such as the data provider's name and a list of keywords that might of interest to the requesting client. You can edit the WMTS configuration file, which is stored in the same folder as that for `mapViewerConfig.xml` with a name of `wmtsConfig.xml`, to provide such customized information.

In the `wmtsConfig.xml` file, you can use the `<wmts_config>` element to customize the descriptive information sent back to the client as part of the `GetCapabilities` response, as shown in the following example:

```
<wmts_config>
     <public_datasources>
        <public_datasource name="MVDEMO" include_all_tile_layers="true"/>
        <public_datasource name="ELOCATION">
          <tile_layers>
            <tile_layer name="WORLD_MAP"/>
          </tile_layers>
        </public_datasource>
    </public_datasources>
    <sdo_epsg_mapfile>
      ../config/epsg_srid.properties
    </sdo_epsg_mapfile>
   <ServiceAttributes>
    <ServiceIdentification>
        <Title>Web Map Tile Service by myCorp</Title>
        <Abstract> U.S. maps for state and county boundaries and big cities</Abstract>
        <Keywords>
           <Keyword>Maps,U.S. State Boundaries,Cities</Keyword>
        </Keywords>
    </ServiceIdentification>
    <ServiceProvider>
      <ProviderName>provider's name</ProviderName>
```

```
        <ProviderSite url="http://www.myCorp.com/mySite"/>
    </ServiceProvider>
  </ServiceAttributes>
</wmts_config>
```

The `<public_datasources>` element can contain `<public_datasource>` subelements, which specify which data sources' tile layers to list in the WMTS GetCapabilities response. If this `<public_datasources>` element is not defined, all data sources' tile layers will be listed; if this element is defined but contains no `<public_datasource>` subelement, then no tile layers from any data source will be listed in the response.

The `<public_datasource>` element must contain a `name` attribute, which indicates the name of the data source.

The `include_all_tile_layers` attribute is optional, and the default is `false`. When set to `true`, it indicates that all tile layers in that data source are to be listed in the response.

The `<tile_layer>` element must contain a `name` attribute, which indicates the name of the tile layer to be included in the response from the data source defined in its parent element.

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

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

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

The `<Keywords>` element specifies a collection of keywords (from its `<Keyword>` subelements) that best describe the types of layers served by this map visualization component server.

More information can be found in the comments in the `wmtsConfig.xml` file.

1.5.2.12 Configuring the Map Tile Server for Oracle Maps

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

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

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

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

1.5.2.13 Defining Permanent Map Data Sources

Every map request must have a data source attribute that specifies a map data source, which is a database user with geospatial data. You can predefine available map data sources by

using the <map_data_source> element. For instance, the following example defines a map data source by specifying the JDBC connection details:

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

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

- The **name** attribute specifies a unique data source name to the map visualization component. You must specify the data source name in all map requests that identify a data source.
- You can choose one of the following options to provide the connection details as applicable to your scenario:
  - **Specifying all the necessary JDBC connection information**
    You can specify the following connection information:
    * **jdbc_host:** Host name of the database
    * **jdbc_sid:** System identifier for the database
    * **jdbc_port:** Database listener port
    * **jdbc_user:** Database user name
    * **jdbc_password:** Password for the jdbc_user
    * **jdbc_mode:** Type of Oracle JDBC driver

    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 the map visualization component next restarts, it will automatically obfuscate and replace the clear text password.

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

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

    On database restart, the map visualization component will resume normal operation (for example responding to map requests with properly created maps) as soon as the database is back online.
  - **Specifying the Container data source name**
    The **container_ds** attribute lets you specify the Java EE container name (JNDI name) instead of specifying the jdbc_host, jdbc_sid, jdbc_port, jdbc_user, jdbc_password,

and `jdbc_mode` attributes. For example, assume you created a JDBC data source and gave it a JNDI name of `jdbc/OracleDS`. You can then define the permanent map visualization component data source as follows:

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

If you use the `container_ds` attribute, and if you want the map visualization component to resume normal operation (for example, responding to map requests with properly created maps) automatically after the database on which you defined a data source is restarted, you must instruct the container data source to always validate a connection before it can be returned to the application.

In a production deployment of the Map Visualization component, this is the recommended way of defining a data source for maximum security.

– **Specifying the Net Service name**
The **`jdbc_tns_name`** attribute identifies a net service name that is defined in the `tnsnames.ora` file.

On database restart, the map visualization component will resume normal operation (for example responding to map requests with properly created maps) as soon as the database is back online.

– **Specifying the Autonomous Database (ADB) Credentials**
This is applicable if you are using Oracle Autonomous Database as the map data source. In this case, you can create a MapViewer data source using the client credentials from the **wallet zip** file. If the MapViewer server is running behind a firewall, and it needs a proxy server to access the Autonomous Database, then you must provide the host and port details of the proxy server.

The following explores the various alternatives to define the wallet zip file's storage location.

* **Option 1:** Copy the downloaded wallet zip file onto the machine where MapViewer server is deployed and provide its absolute path as shown:

```
<map_data_source name="adb_no_proxy"
                   wallet_zip_file="/home/myfolder/mywallets/
<wallet_filename>.zip"
                   user="myCloudDBUser"
                   password="<password_for_myCloudDBUser>"
                   service="db20220510_low"
                   number_of_mappers="5"
                   allow_jdbc_theme_based_foi="false"
   />
```

* **Option 2:** Copy the downloaded wallet zip file onto the machine where MapViewer server is deployed and provide its relative path. It is relative to where the `mapViewerConfig.xml` file is stored. For example:

```
<map_data_source name="adb_require_proxy"
                   wallet_zip_file="../wallets/
<wallet_filename>.zip"
                   user="myCloudDBUser"
                   password="<password_for_myCloudDBUser>"
```

```
                    service="db20220510_low"
                    proxy_host="<proxy_host_name>"
                    proxy_port="80"
                    number_of_mappers="5"
                    allow_jdbc_theme_based_foi="false"
  />
```

* **Option 3:** Define the wallet zip file using a URL. The URL must be accessible by the MapViewer server. For example:

```
<map_data_source name="adb_url"
                    wallet_zip_file="https://
objectstorage.<region_identifier>.oraclecloud.com/p/b/
<bucket_name>/o/<wallet_filename>.zip"
                    user="myCloudDBUser"
                    password="<password_for_myCloudDBUser>"
                    service="db20220510_high"
                    number_of_mappers="5"
                    allow_jdbc_theme_based_foi="false"
  />
```

It is important to note that in all the preceding examples:

* The **`service`** attribute can be found in the `tnsnames.ora` file. This file is obtained by unzipping the wallet zip file. The `tnsnames.ora` file contains three service names, for example, `<service_name>_high`, `<service_name>_medium`, `<service_name>_low`. You can choose one of them according to your preference.
* For other attributes, such as **`user`** and **`password`**, refer to **`jdbc_user`** and **`jdbc_password`** explained earlier in this section.

Each time the MapViewer server is started, it will try to access the wallet zip file from the file system, or download it from the provided link. If the storage location of the wallet zip file is provided by a URL, then the MapViewer server will download the zip file and store it in a temporary folder. Since the MapViewer server will unzip the wallet file into the folder where this zip file is stored, ensure that the folder is writable.

- The **`number_of_mappers`** attribute identifies the maximum number of map renderers available (and thus the maximum number of map requests that the map visualization component can process in parallel for the data source) for this data source. Any unprocessed map requests are queued and eventually processed. For example, if the value is 3, the map visualization component 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 the map visualization component has finished processing one of the current requests.

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

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

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

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

1.5.2.13.1 Data Sources

A data source corresponds to a database schema or user.

Before you can draw any spatial data in a database schema, you must first define (create) a data source for the schema:

- You can define a data source by specifying its connection information and user login credentials in the map visualization component configuration file (`mapViewerConfig.xml`).
- You can define or modify a data source using the map visualization component administration (Admin) page.

Each map request must specify a data source. You can, however, specify a different data source for individual themes added to the map request. This makes it easy to aggregate data stored across different database schemas. If a theme has no specified data source, it is associated with the data source of the map request. A base map (and thus the themes included in it) is always associated with the data source. When a theme is processed, all of its underlying data, as well as the styles referenced in its definition, must be accessible from the data source or sources associated with the theme.

Each data source has associated renderers (sometimes called mappers or map makers), the number of which is determined by the `number_of_mappers` attribute in the `<map_data_source>` element. This attribute (described in Defining Permanent Map Data Sources) affects the number of database connections created for each data source when map requests are processed. The number of renderers specified in a data source also is the maximum number of concurrent requests that can be processed for that data source. Each additional renderer requires only a small amount of memory, so the main potential disadvantage of specifying a large number of renderers (such as 100) is that the underlying CPU resource might be strained

if too many map requests are allowed to come through, thus affecting the performance of the entire map visualization component server.

Each data source has its own internal metadata cache. The metadata cache holds the definitions of all accessed styles, as well as of all predefined themes that originate from the data source. This eliminates the need to query the database repeatedly for the definition of a style or predefined theme whenever it is needed.

- Catalog Data Sources

1.5.2.13.1.1 Catalog Data Sources

A catalog data source gets all of its data from local files. The local directory where the data files are stored is relative to where the `mapViewerConfig.xml` file is stored. From another perspective, a catalog data source does not need the Oracle database, because all necessary data (the spatial geometry data and its attributes, as well as the metadata for how to rend the spatial data, such as styles and themes) is stored in local files.

The following are the general steps for creating and using a catalog data source:

1. Export the Necessary Metadata from an Oracle Database
2. Export the Necessary Spatial Tables
3. Edit the Map Visualization Component Configuration File to Add the Catalog Data Source
4. Restart the Map Visualization Component Server

- Export the Necessary Metadata from an Oracle Database
- Export the Necessary Spatial Tables
- Edit the Map Visualization Component Configuration File to Add the Catalog Data Source
- Restart the Map Visualization Component Server

1.5.2.13.1.1.1 Export the Necessary Metadata from an Oracle Database

Before creating a catalog data source, you must use the Map Builder utility to export the metadata to `USER_SDO_CACHED_MAPS.xml`, `USER_SDO_THEMES.xml`, and `USER_SDO_STYLES.xml` files. The following considerations apply:

- Only external tile layers are supported in catalog data source. So, the tile layers exported into USER_SDO_CACHED_MAPS.xml must be external tile layers, such as Oracle eLocation map services.
- All styles needed by the themes in USER_SDO_THEMES.xml must be exported in to the USER_SDO_STYLES.xml file.
- You need note all required spatial data tables (themes' base tables from which to fetch spatial data) used by all of the exported themes, so that you can export each base table into a GeoJSON file (explained in Export the Necessary Spatial Tables).

1. In the Map Builder utility, select **Tools**, then **Export Metadata to XML**.
2. Select a temporary directory to store the metadata.
3. Accept the default prefix (`USER_SDO_`) for the table names.
4. Select the tile layers, themes, and styles to export. (If you are not sure which styles are needed for the desired themes, you may export all styles.)
5. Click OK to perform the export operation.

   The `USER_SDO_CACHED_MAPS.xml`, `USER_SDO_THEMES.xml`, and `USER_SDO_STYLES.xml` files are created in the specified directory.

1.5.2.13.1.1.2 Export the Necessary Spatial Tables

For a catalog data source, the spatial data sets are stored in GeoJSON files. One GeoJSON file corresponds to one spatial table is the database. To export spatial tables, you send requests to the map visualization component server.

For example, if a spatial table called OBIEE_COUNTRY is needed by a catalog data source theme (assume that the theme is also called OBIEE_COUNTRY) in the `USER_SDO_THEMES.xml` file, then this spatial table needs to be exported as a GeoJSON file. As a convention, you may name it with the same name as the name of the spatial table. In this case, it is named as `OBIEE_CONTRY.json`.

1. Check the themes definition from the `USER_SDO_THEMES.xml` file. For example, if a theme named OBIEE_COUNTRY uses the OBIEE_COUNTRY spatial table, then the spatial table needs to be exported. The table's columns and expected names must be identified, and the column names are the same as the name attributes in the theme definition. In the following example, the ISO_COUNTRY_CODE column is mapped as `Country Code`:

```
<theme>
   <name>OBIEE_COUNTRY</name>
   <description><![CDATA[OBIEE Country]]></description>
   <base_table>OBIEE_COUNTRY</base_table>
   <geometry_column>GEOMETRY</geometry_column>
   <styling_rules><![CDATA[<?xml version="1.0" standalone="yes"?><styling_rules>
 <hidden_info>
   <field column="ISO_COUNTRY_CODE" name="Country Code"/>
   <field column="NAME" name="Country Name"/>
   <field column="NAME_INIT" name="Country Name (Init)"/>
 </hidden_info>
 <rule>
   <features style="C.AIRPORTS"> </features>
   <label column="NAME_LABEL" style="T.COUNTRY_NAME_10"> 1 </label>
 </rule>
</styling_rules>]]></styling_rules>
 </theme>
  . . .
```

2. Identify the spatial table columns. In the following example, base table OBIEE_COUNTRY has the spatial geometry column named GEOMETRY:

```
SQL> describe obiee_country
 Name                          Null?    Type
 ----------------------------- -------- ----------------------------
 NAME                                   VARCHAR2(255 CHAR)
 NAME_INIT                              VARCHAR2(1020 CHAR)
 OBIEE_LOWER                            VARCHAR2(1020 CHAR)
 ISO_COUNTRY_CODE                       VARCHAR2(5)
 SQKM                                   NUMBER(11)
 NAME_LABEL                             VARCHAR2(255 CHAR)
 GEOMETRY                               MDSYS.SDO_GEOMETRY
```

3. Create request strings to the map visualization component data server. For example, if OBIEE_COUNTRY is the base table, and if the map visualization component is running at `localhost:8080`, and if the data source name is `my_ds_name` with spatial data retrieval enabled, you can send a request string like the following:

```
http://localhost:8080/ mapviewer/dataserver/my_ds_name?t=obiee_country&sql=select
iso_country_code as id, iso_country_code as "country Code", name as "Country Name",
name_init as "Country Name (Init)", name_label, geometry from
obiee_country&id_col=id&simplify=true&threshold=90&include_label_box=true
```

You can save the data set with a name `OBIEE_COUNTRY.json` in this case to the temporary folder where the exported metadata (the `USER_SDO_CACHED_MAPS.xml`, `USER_SDO_THEMES.xml`, and `USER_SDO_STYLES.xml` files) is stored.

1.5.2.13.1.1.3 Edit the Map Visualization Component Configuration File to Add the Catalog Data Source

Edit the `mapViewerConfig.xml` file and add the catalog data source. For example:

```
<map_data_source name="catalogds1"
   catalog_dir="../catalogs/datafolder1"
   private="true"
   number_of_mappers="3"
   allow_jdbc_theme_based_foi="true"
   editable="false"/>
```

For the preceding catalog data source definition, you must create the specified `catalog_dir` folder relative to where the `mapViewerConfig.xml file` is stored, and then copy all data files into the folder: that is, the three exported metadata files (`USER_SDO_CACHED_MAPS.xml`, `USER_SDO_THEMES.xml`, and `USER_SDO_STYLES.xml`) and all exported GeoJSON files, such as `OBIEE_COUNTRY.json` in this example.

1.5.2.13.1.1.4 Restart the Map Visualization Component Server

Restart the map visualization component server.

After the map visualization component server is restarted, all the exported tile layers and themes should be accessible from this catalog data source. For example, you should see a map image if you send a map request like the following:

```
http://localhost:8080/mapviewer/omserver?xml_request= <?xml version="1.0"
standalone="yes"?> <map_request  datasource = "catalogds1" width="1024" height="768"
format="PNG_STREAM"> <center size="200"> <geoFeature><geometricProperty
typeName="center"> <Point> <coordinates>0, 0</coordinates> </Point> </
geometricProperty></geoFeature> </center> <themes> <theme name="OBIEE_COUNTRY"> </theme>
</themes> </map_request>
```

1.5.2.14 Configuring and Securing the Map Data Server for the HTML5 API

Themes can be streamed by default, and the only way to protect them is by adding authentication, that is, by adding a security constraint in the map visualization component `web.xml` file and by configuring the `<mds_config>` element in the configuration file to authorize access to various themes.

The Map Data Server (MDS) server component facilitates the streaming of geospatial data in vector format to the Oracle Maps API (described in Oracle Maps JavaScript API). The MDS provides a RESTful API for browser clients to request the vector data of any predefined or dynamic (JDBC) theme from a map visualization component server instance. The only way to secure or protect the access to this service is by adding a security constraint in the map visualization component `web.xml` deployment file, as in the following example:

```
<security-constraint>
<web-resource-collection>
. . .
<url-pattern>/dataserver/*</url-pattern>
</web-resource-collection>
<auth-constraint>
<role-name>map_admin_role</role-name>
</auth-constraint>
</security-constraint>
```

The preceding example adds a security constraint to any incoming URL with the relative path `/dataserver/` in it. Because the MDS servlet responds only to URLs with `/dataserver` in its

path, this constraint effectively protects all access to the MDS. This means that any application or web client accessing the Map Data Server will require proper authentication, and only those users with the role `map_admin_role` will be granted access. (For more information on how to secure a Java EE servlet such as MDS, check the Java EE and WebLogic Server documentation.)

Access to any predefined or dynamic (JDBC) theme's vector data is blocked by default, regardless of whether you added a security constraint on the MDS URL patterns. In other words, for example, even if the `/dataserver/*` URLs are protected and an HTML5 application has passed authentication, it still cannot access a theme's data without proper authorization. When an Oracle Maps HTML5 application attempts to load or display a theme without proper authorization, the error message typically contains a statement like "This data source does not allow streaming access."

To grant access to a data source's themes, you must explicitly configure the `<mds_config>` element in the configuration file.

Example 1-2 Configuring the mds.xml File

Example 1-2 shows an `<mds_config>` element in which two map visualization component data sources, `mvdemo` and `my-data`, are configured such that certain themes of theirs can be streamed to clients.

```
<mds_config>

<data_source name="mvdemo">
<allow_predefined_themes>true</allow_predefined_themes>
<deny>my_secret_theme</deny>
<allow_dynamic_themes>true</allow_dynamic_themes>
</data_source>

<data_source name="my-data">
<allow_predefined_themes>false</allow_predefined_themes>
<allow>
<theme>public_points_theme</theme>
<theme>office_locations*</theme>
</allow>
<allow_dynamic_themes>false</allow_dynamic_themes>
</data_source>

</mds_config>
```

In Example 1-2:

- Each data source authorizes its themes in its own `<data_source>` element. With this element, the two tags `<allow_predefined_themes>` and `<allow_dynamic_themes>` provide the overall or default access control on these two types of themes. Note that for dynamic/JDBC themes, you can also disable them in the data source definition (in the main `mapViewerConfig.xml` file). If a dynamic theme is disabled in the data source definition, then that setting always has precedence (regardless of how it is set in the `<mds_config>` element.).
- The `mvdemo` data source grants all clients with access to both predefined and dynamic (JDBC) theme vector data by default; this is a reasonable choice given the nature of the data (data from publicly available samples). An exception is added, however, for the `my_secret_theme` theme, through the use of the `<deny>` tag.
- For the `my-data` data source, access to both types of themes is blocked by default. In this case exceptions (to open certain themes to streaming) are added through the `<allow>` tag. In both `<allow>` and `<deny>` tags, the theme names or patterns are case insensitive, and

the wild card character * (asterisk) can be used to match multiple themes. The example uses the `<allow>` tag to open the theme `public_points_theme` and all the themes whose name starts with `office_locations` to streaming, while all other themes are blocked. No dynamic/SQL theme is allowed on this data source.

If you modify the `<mds_config>` element, you must restart the deployed map visualization component instance for the modifications to take effect.

1.5.3 Creating a Map Visualization Component Data Source Using a JDBC Container

You can create a JDBC data source using a container defined data source in the map visualization component configuration file. However, to establish database connections to the newly created container data source, you must restart the map visualization component after performing the instructions in the following sections.

- Create a JDBC Data Source in a Map Visualization Component Container
- Create a Map Visualization Component Data Source

1.5.3.1 Create a JDBC Data Source in a Map Visualization Component Container

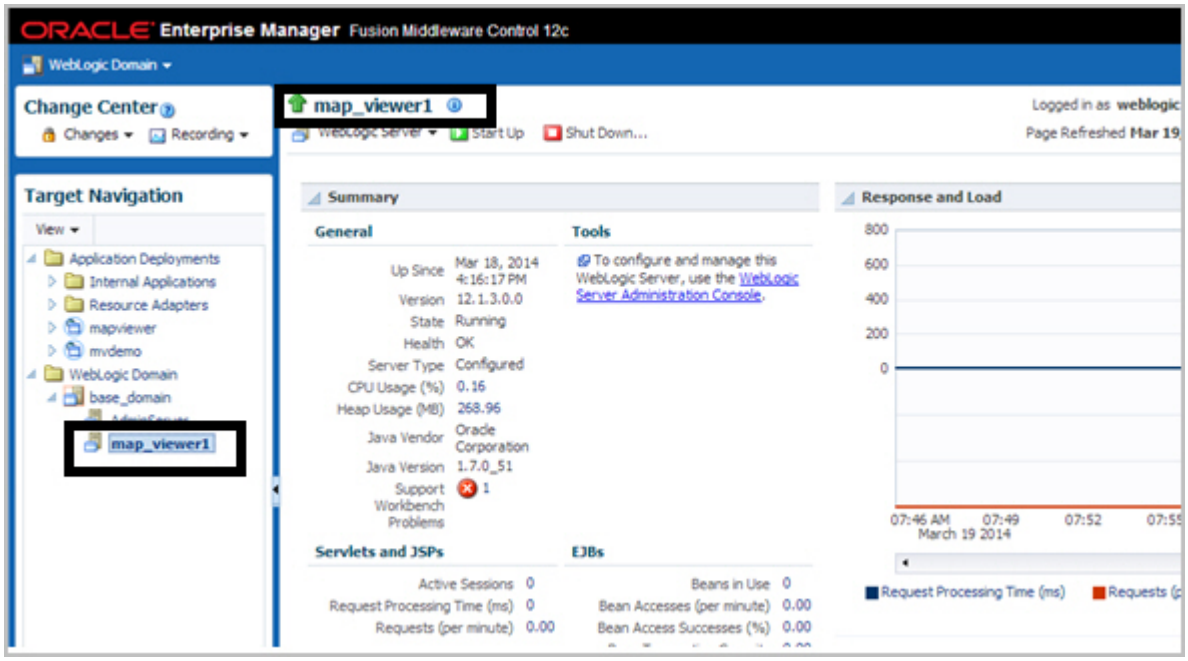
You can use Oracle Enterprise Manager 12*c* or later to create a data source that connects to the Oracle Database.

The following steps show how to create GridLink data source. (These are followed by steps showing how to create a Multi data source.)

1. Log in to Enterprise Manager and in the Target Navigation pane, click the server instance that contains the map visualization component server.

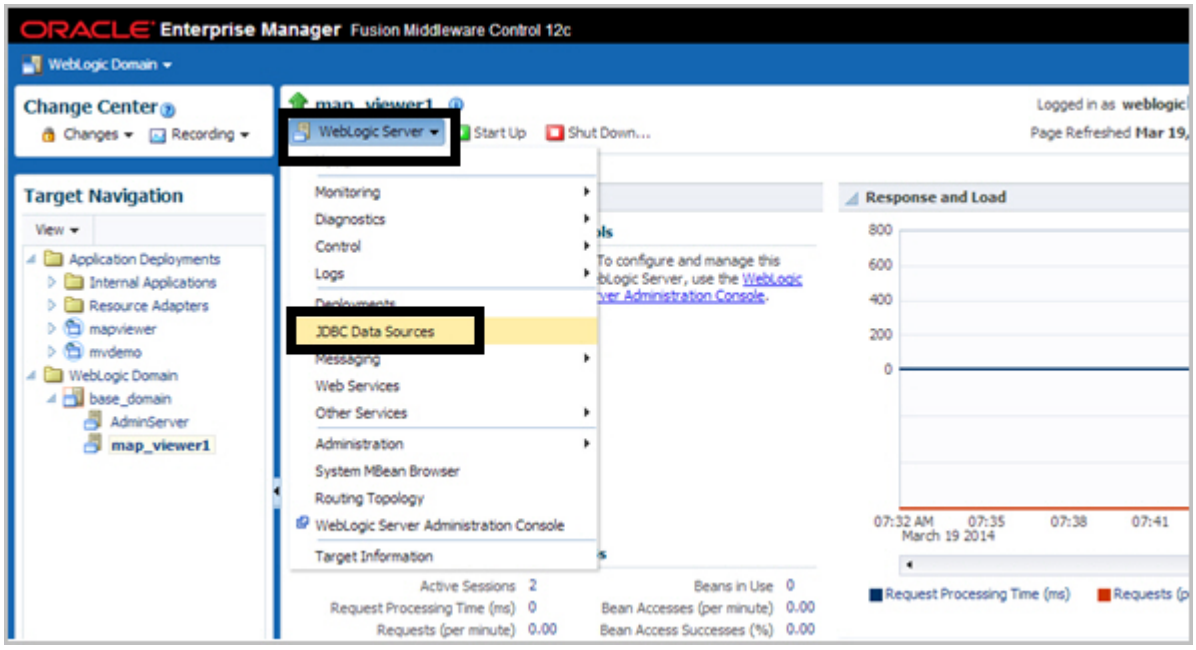
   In Figure 1-2, clicking **map_viewer1** under WebLogic Domain causes the `map_viewer1` server information to appear in the main area of the window.

Figure 1-2 Selecting the Server Instance



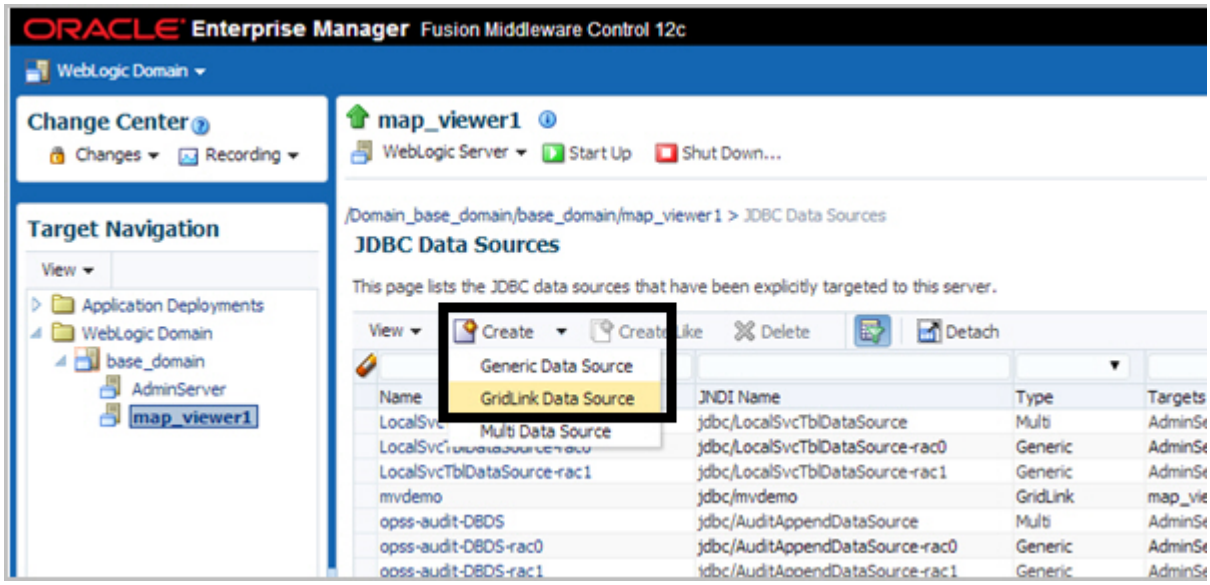
2. Click **WebLogic Server** and select **JDBC Data Sources**, as shown in Figure 1-3.

Figure 1-3 Displaying JDBC Data Sources



3. Under JDBC Data Sources, click **Create** and select **GridLink Data Source**, as shown in Figure 1-4.

Figure 1-4 Creating a GridLink Data Source



4. Enter any necessary information in the Creating New JDBC Data Source wizard. For example, to create a container data source named `jdbc/mvdemo`:

   a. **Data Source Properties**: Specify **Data Source Name** as `mvdemo`, **Driver Service Name** as `Oracle Driver (Thin XA) for GridLink Connections Versions: Any`, and **JNDI Name** as `jdbc/mvdemo`.

b. **Connection Properties**: Generate the URL for database user `mvdemo` on the appropriate host.

c. **Transaction Properties**: Accept the displayed transaction properties.

d. **ONS Properties**: Accept the displayed transaction properties, or make any changes as needed.

e. **Select Targets**: Select (check) `map_viewer1` under **Name** to deploy the JDBC data source on the desired server.

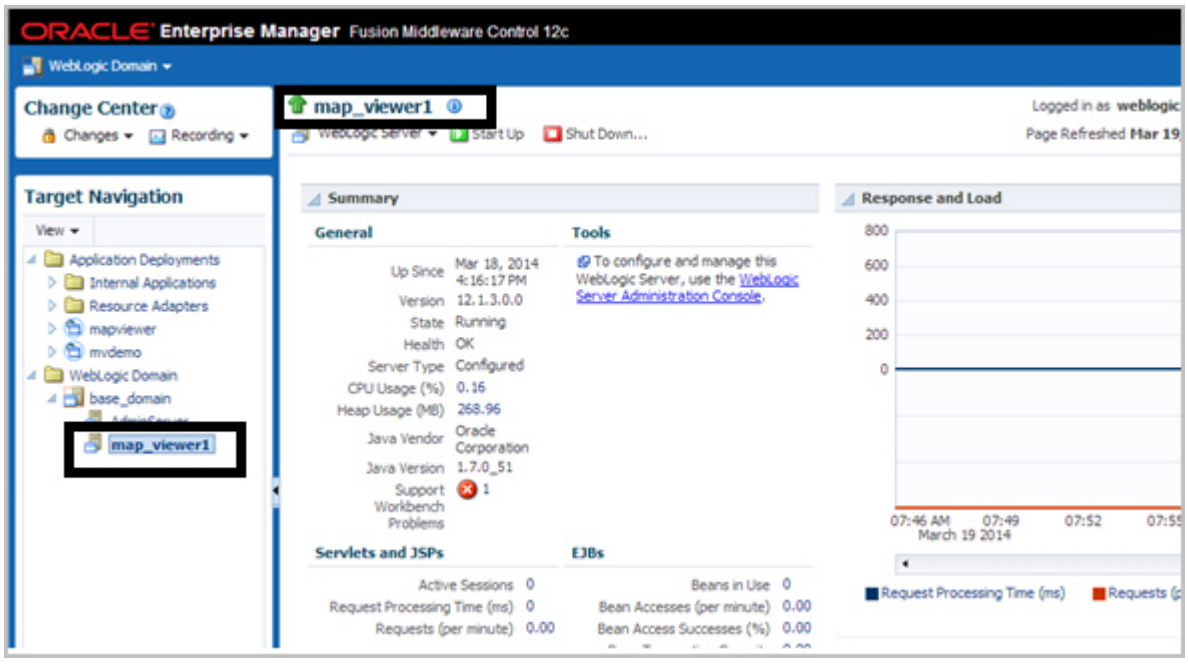
f. **Review**: Review the properties for the new data source to be created. If you need to make any changes, go back and make them and then return to this page.

To create a Multi data source instead of a GridLink data source as in the preceding instructions, adapt the steps as appropriate. For example:

1. Log in to Enterprise Manager and in the Target Navigation pane, click the server instance that contains the map visualization component server.

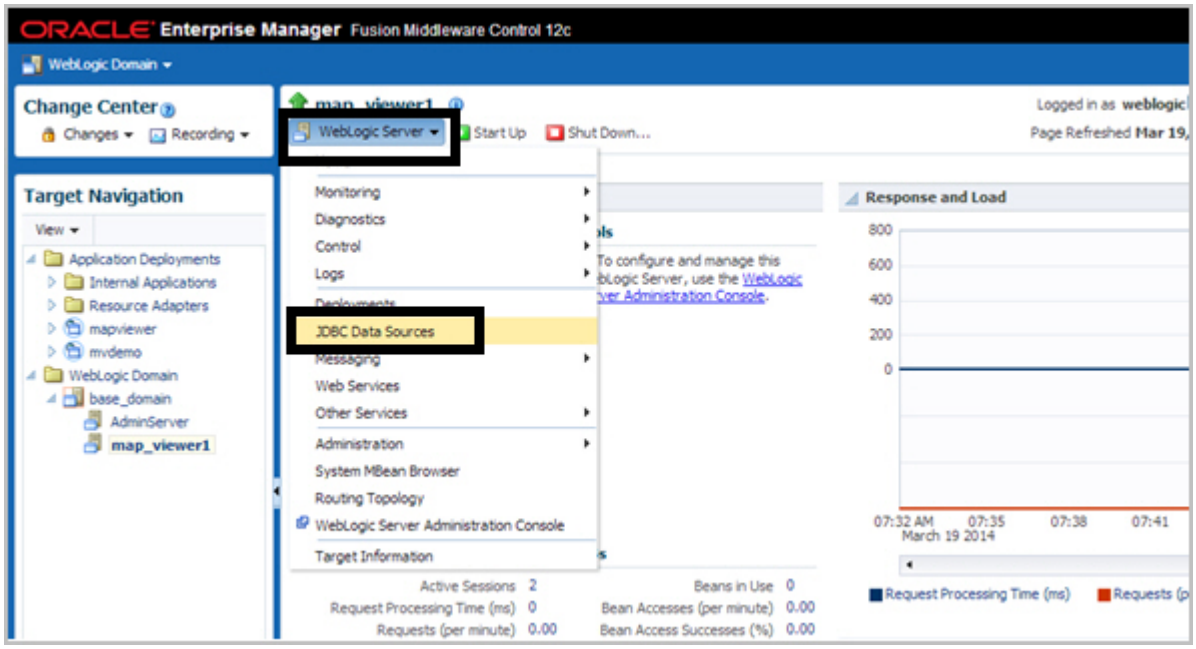
   In Figure 1-5, clicking **map_viewer1** under WebLogic Domain causes the `map_viewer1` server information to appear in the main area of the window.

Figure 1-5 Selecting the Server Instance



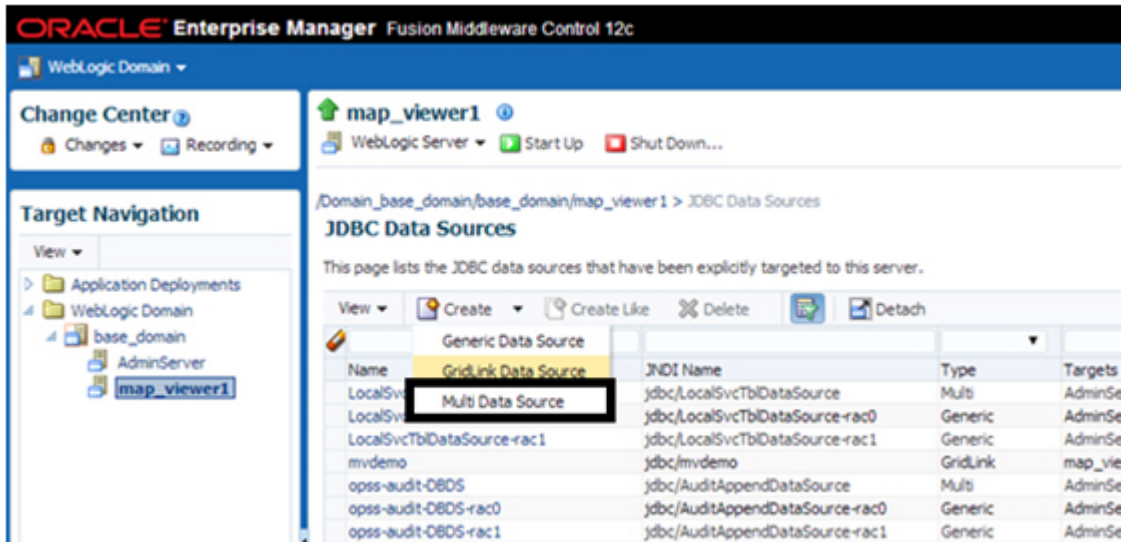
2. Click **WebLogic Server** and select **JDBC Data Sources**, as shown in Figure 1-6.

Figure 1-6 Displaying JDBC Data Sources



3. Under JDBC Data Sources, click **Create** and select **Multi Data Source**, as shown in Figure 1-7.

Figure 1-7 Creating a Multi Data Source



4. Enter any necessary information. For example:
   a. **Data Source Properties**: Specify **Data Source Name** as `mvdemo`, **JNDI Name** as `jdbc/mvdemo`, and **Algorithm Type** as `Failover`.
   b. **Select Targets**: Select (check) `map_viewer1` under **Name** .
   c. **Select Data Source Type**: Accept the default values (`non-XA Driver`).
   d. Click **Create New Data Source**.
   e. Specify properties for the first data source node, such as: **Name**: `mvdemo-rac0`, **JNDI Name**: `jdbc/mvdemo-rac0`, **Database Type**: `Oracle`.

f. For **Database Driver**, select `Oracle's Driver (Thin) for Oracle RAC Service-Instance connections: Versions: Any`.

g. Accept the default values (`Supports Global Transactions` and `One-Phase Commit`).

h. Define the connection properties for Node 1. Provide values for Service Name, Database Name, Host Name, Port, Database User Name, Password, and Protocol.

i. Verify the properties and click **Test Configuration**. If it succeeds, click **Next**.

j. Select (check) the server in which the map visualization component is deployed (`map_viewer1`), and click **Finish**.

k. On the next page, click **Create a New Data Source** to create and configure the second node.

l. Specify properties for the second data source node, such as: **Name**: `mvdemo-rac1`, **JNDI Name**: `jdbc/mvdemo-rac1`, **Database Type**: `Oracle`.

m. For **Database Driver**, select `Oracle's Driver (Thin) for Oracle RAC Service-Instance connections: Versions: Any`.

n. Accept the default values (`Supports Global Transactions` and `One-Phase Commit`).

o. Define the connection properties for Node 2. Provide values for Service Name, Database Name, Host Name, Port, Database User Name, Password, and Protocol.

p. Verify the properties and click **Test Configuration**. If it succeeds, click **Next**.

q. Select (check) the server in which the map visualization component is deployed (`map_viewer1`), and click **Finish**.

r. If you need to add more nodes, click Create a New Data Source and create each in the same way as before.

1.5.3.2 Create a Map Visualization Component Data Source

After creating a container data source in the map visualization component container (explained in Create a JDBC Data Source in a Map Visualization Component Container), create a new map visualization component data source that enables it to connect to the Oracle database by adding the `container_ds` attribute in the map visualization component data source. For example:

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

In the preceding example:

- The `name` attribute specifies the map visualization component data source name, which is required for map requests.
- The value for the `container_ds` attribute must match the JNDI Name that you specified on the Data Source Properties page of the Creating New JDBC Data Source wizard.
- The `number_of_mappers` attribute specifies the maximum number of supported concurrent map requests that can target this data source.

For more information about these attributes, see Defining Permanent Map Data Sources.

1.5.4 Creating a Map Visualization Component Data Source Using Oracle Service Name

As an alternative to creating a JDBC container data source for use with the `container_ds` attribute in the map visualization component configuration file (as explained in Creating a Map Visualization Component Data Source Using a JDBC Container), you can create a map visualization component data source directly in the map visualization component configuration file by specifying the Oracle Database connection parameters. (This type of connection works only with Oracle Database Release 11.2 and later.) For example:

```
<map_data_source name="mvdemo"
         jdbc_host="hostname.example.com"
         jdbc_sid="//srv.mycompany.com"
         jdbc_port="1521"
         jdbc_user="mvdemo"
         jdbc_password="<password_for_mvdemo>"
         jdbc_mode="thin"
         number_of_mappers="8"
         allow_jdbc_theme_based_foi="true"
         editable="false"
/>
```

In the preceding example:

- The `jdbc_host` attribute must be the Oracle IP address or host name.
- The `jdbc_sid` attribute (note the leading `//` characters in `jdbc_sid="//srv.mycompany.com"`) specifies the Oracle database service name and not the SID value.

For more information about the attributes in this example, see Defining Permanent Map Data Sources.

In order to have the newly created map visualization component data source take effect, you must restart the map visualization component. Once restarted, the map visualization component obtains the necessary database connections directly using the connection parameters.

1.5.5 Performing Map Visualization Component Administrative Tasks

Besides knowing how to configure the map visualization component, you should also know how to perform other important administrative tasks using the map visualization component administration page. To log in to this page, see the instructions in Logging in to the Map Visualization Component Administration Page.

The tasks you can do as a map visualization component administrator include the following:

- Editing the configuration file

  Click **Configuration**.
- Refreshing the list of data sources

  Click **Admin** to refresh the list automatically, or click **Refresh** to perform a manual refresh.
- Clearing cached definitions of map visualization component styles, themes, and base maps

Click **Admin**, select the data source, then click **Purge Cached Metadata**.

- Clearing cached geometry data for predefined themes

  Click **Admin**, then **Geometry Cache**, then **Purge** for a selected theme or all themes.

- Creating map tile layers for Oracle Maps

  Click **Admin**, then **Create Tile Layer**, select the tile layer type, and click **Next**.

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

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

  For other types of tile layers (Oracle Maps, Here, Google, TomTom), follow similar steps.

- Managing map tile layers for Oracle Maps

  Click **Admin**, then **Manage Tile Layers**. Then do any of the following:

  To manually refresh the tile layer list, click **Refresh**.

  To edit a map tile layer, select the row for that layer and click **Edit/View Details**.

  To view and manage map tile layer, select the row for that layer and click **View Map/ Manage Tiles**.

1.6 High Availability and the Map Visualization Component (WebLogic Server Only)

Users can benefit from the high availability features of Oracle WebLogic Server.

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

1.6.1 Deploying the Map Visualization Component on a Middle-Tier Cluster

The map visualization component can be deployed to a WebLogic Server cluster. You must take care, however, about how the generated image files on each host are named and referenced through URLs by client applications.

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

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

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

1.7 Secure Map Rendering

You can implement secure map rendering based on a web user's identity.

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

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

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

1. Create a PL/SQL package in the database schema. The package must have at least two named procedures: `set_user(username)` and `clear_user()`.
2. Create views, set access rights on database objects, and perform other tasks, based on the user identity stored in the PL/SQL package (which is set by the map visualization component through the set_user procedure for each database session).

3. Create a map visualization component data source to the schema, providing the name of the PL/SQL package as part of the data source definition. This is considered a secured data source.
4. Create map visualization component themes that are based on the views created in step 2.
5. Establish web authentication for users accessing your map visualization component application page or pages, so that when a map request reaches the map visualization component servlet, the web session object should contain an authenticated user's identity.
6. Issue map and FOI (feature of interest) requests that view the themes defined in step 4, either directly or through the use of base maps and Oracle Maps.

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

How Secure Map Rendering Works explains how secure map rendering works and provides implementation details and examples. Options for Authenticating Users describes some options for authenticating users.

1.7.1 How Secure Map Rendering Works

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

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

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

Whenever a theme is requested from a secured data source, The map visualization component invokes the `set_user` procedure in the associated PL/SQL package before it executes any data query for the theme, and it invokes the `clear_user` procedure when the querying process is complete for the theme.

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

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

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

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

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

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

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

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

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

  PROCEDURE clear_user
  AS
  BEGIN
      w_name := null;
  END;

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

Example 1-4 View for Secure Map Rendering

```
CREATE OR REPLACE VIEW customers_view
AS
  SELECT * FROM customers
  WHERE account_mgr = web_user_info.get_user;
```

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

Example 1-5 Data Source Definition for Secure Map Rendering

```
<map_data_source name="mvdemo"
                 jdbc_host="system32.example.com"
                 jdbc_sid="mv"
                 jdbc_port="15214"
                 jdbc_user="mvdemo"
                 jdbc_password="password"
                 jdbc_mode="thin"
                 number_of_mappers="3"
                 allow_jdbc_theme_based_foi="true"
                 plsql_package="web_user_info"
  />
```

1.7.2 Getting the User Name from a Cookie

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

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

Example 1-6 Data Source Definition Specifying Cookie Name

```
<map_data_source name="mvdemo"
                  jdbc_host="system32.example.com"
                  jdbc_sid="mv"
                  jdbc_port="25650"
                  jdbc_user="mvdemo"
                  jdbc_password="LfCDQ6NH59nuV7zbeY5QY06sqN7XhiUQ"
                  jdbc_mode="thin"
                  number_of_mappers="3"
                  allow_jdbc_theme_based_foi="true"
                  plsql_package="web_user_info"
                  web_user_type="MON_USER"
  />
```

The possible values for the `web_user_type` attribute are:

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

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

1.7.3 Options for Authenticating Users

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

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

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

2

Map Visualization Concepts

This chapter explains concepts that you should be familiar with before using the map visualization component.

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

- Styles define rendering properties for features that are associated with styles. For example, a text style determines how such a feature is labeled on a map, while a line style determines the rendition of a linear feature such as a road.
- A theme is a collection of features (entities with spatial and nonspatial attributes) that are associated with styles through the use of styling rules.
- A base map consists of one or more themes. (A base map should not have the same name as any theme.)
- Mapping metadata consists of a repository of styles, themes, and base maps stored in a database.
- A map is one of the components that the map visualization component 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.
- Overview of the Map Visualization Component
  When an application uses the map visualization component, 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).
- Styles
  A **style** is a visual attribute that can be used to represent a spatial feature.
- Themes
  Theme is perhaps the most important concept in the map visualization component. A **theme** is a visual representation of a particular data layer.
- Maps
  A map can consist of a combination of elements and attributes.
- How a Map Is Generated
  When a map request arrives at the map visualization component server, the server picks a free renderer associated with the data source in the request.
- Cross-Schema Map Requests
  A database user can issue a map request specifying a theme that uses data associated with another database user, to select data from tables that the other data source user is authorized to access.
- Workspace Manager Support in the Map Visualization Component
  Workspace Manager is an Oracle Database feature that lets you version-enable one or more tables in the database.
- Map Visualization Component 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.

- Oracle Maps
  Oracle Maps is the name for a suite of technologies for developing high-performance interactive web-based mapping applications. It consists of components from both the server side and the client side.

2.1 Overview of the Map Visualization Component

When an application uses the map visualization component, 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.

The map visualization component 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 map visualization component metadata views described in Map Visualization Component Metadata Views (for example, USER_SDO_STYLES, USER_SDO_THEMES, and USER_SDO_MAPS). The set of map definition objects that a given user can access is sometimes called that user's *mapping profile*. You can manage styles, themes, and base maps with the standalone Map Builder tool, described in #unique_26.

2.2 Styles

A **style** is a visual attribute that can be used to represent a spatial feature.

The basic map symbols and labels for representing point, line, and area features are defined and stored as individual styles. Each style has a unique name and defines one or more graphical elements in an XML syntax.

Each style is of one of the following types:

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

  When a marker style is specified for a line feature, the rendering engine selects a suitable point on the line and applies the marker style (for example, a shield marker for a U.S. interstate highway) to that point.
- **Line**: a line style (width, color, end style, join style) and optionally a center line, edges, and hash mark. Lines are often used for linear features such as highways, rivers, pipelines, and electrical transmission lines. You can also use cased line styles, which are useful for drawing streets and highways.
- **Area**: a color or texture, and optionally a stroke color. Areas are often used for polygonal features such as counties and census tracts.
- **Text**: a font specification (size and family) and optionally highlighting (bold, italic) and a foreground color. Text is often used for annotation and labeling (such as names of cities and rivers).

- **Advanced**: a composite used primarily for thematic mapping, which is described in Thematic Mapping. The key advanced style is `BucketStyle`, which defines the relationship between a set of simple rendering (and optionally labeling) styles and a set of buckets. For each feature to be plotted, a designated value or set of values from that feature is used to determine which bucket the feature falls into, and then the style associated with that bucket is used to plot the feature. Bucket styles are described in Bucket Styles.

  Two special types of bucket styles are also provided: color scheme (described in Color Scheme Styles) and variable (graduated) marker (described in Variable Marker Styles). Other advanced styles are dot density (described in Dot Density Marker Styles), bar chart (described in Bar Chart Marker Styles), collection (described in Collection Styles), variable pie chart (described in Variable Pie Chart Styles), and heat map (described in Heat Map Styles).

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

Table 2-1 Style Types and Applicable Geometry Types

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

All styles for a database user are stored in that user's USER_SDO_STYLES view, which is described in Map Visualization Component Metadata Views and xxx_SDO_STYLES Views.

You can also create **dynamically defined styles** (that is, temporary styles) of any style type as part of a map request. To create them, use the Oracle Maps JavaScript API, using classes and methods to create all types of styles dynamically.

What you are actually creating is the XML definition of the styles; it is the map visualization component server that actually creates such dynamically defined styles from the definitions when it processes the map request, and it discards the dynamically created styles when the request is completed.

- Scaling the Size of a Style (Scalable Styles)
- Specifying a Label Style for a Bucket
- Orienting Text Labels and Markers
- Making a Text Style Sticky
- Getting a Sample Image of Any Style
- Allowing a Text Style to Overlap or Be Overlapped

2.2.1 Scaling the Size of a Style (Scalable Styles)

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

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

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

Example 2-1 Scalable Marker Style

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

```
<style name="M.STAR_CAPITAL_CITY">
  <svg width="1in" height="1in">
    <desc/>
    <g class="marker" style="stroke:#000000;fill:#FF0000;fill-
opacity:0;width:15.0km;height:15.0km;font-family:Dialog;font-size:12;font-fill:#FF0000">
       <polyline
points="138.0,123.0,161.0,198.0,100.0,152.0,38.0,198.0,61.0,123.0,0.0,76.0,76.0,76.0,100.
0,0.0,123.0,76.0,199.0,76.0"/>
    </g>
  </svg>
</style>
```

Example 2-2 Scalable Line Style

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

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

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

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

2.2.2 Specifying a Label Style for a Bucket

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

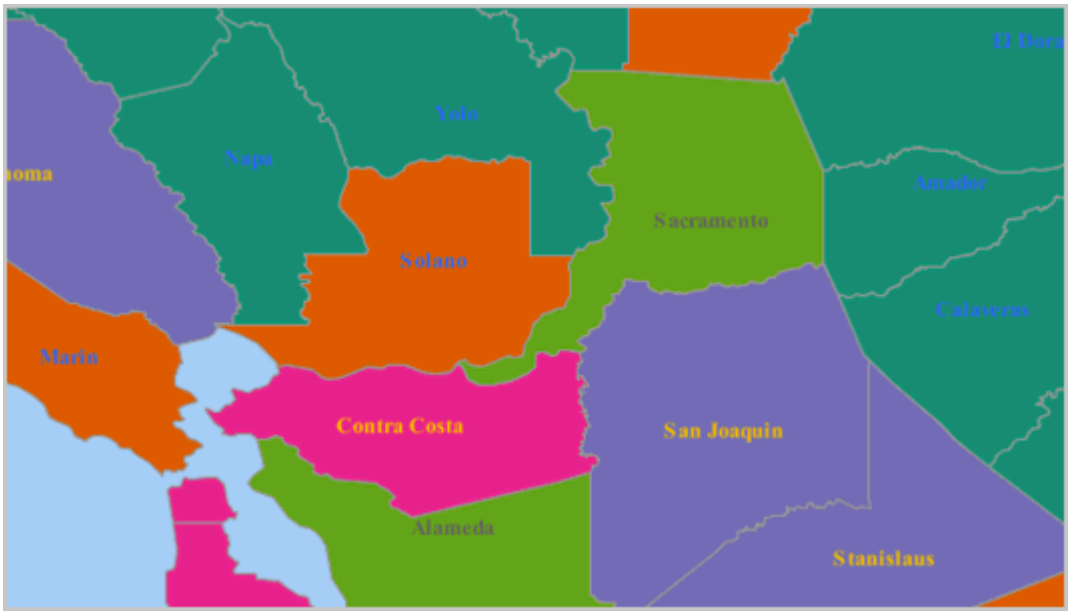
Example 2-3 Advanced Style with Text Label Style for Each Bucket

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

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

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

Figure 2-1 Varying Label Styles for Different Buckets



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

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

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

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

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

```
    </rule>
  </styling_rules>
```

2.2.3 Orienting Text Labels and Markers

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

- Controlling Text Style Orientation
- Controlling Marker Orientation

2.2.3.1 Controlling Text Style Orientation

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

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

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

Example 2-4 Labeling an Oriented Point

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

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

  <styles>
    <style name="transparent_color">
```

```
      <svg width="1in" height="1in">
        <g class="color" style="stroke:#ff0000;stroke-opacity:0">
          <rect width="50" height="50"/>
        </g>
      </svg>
    </style>
  </styles>
</map_request>
```

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

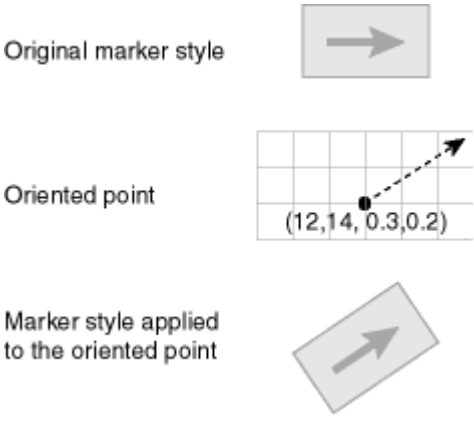
Figure 2-2 Map Display of the Label for an Oriented Point

2.2.3.2 Controlling Marker Orientation

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

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

Figure 2-3 Oriented Marker



2.2.4 Making a Text Style Sticky

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

Example 2-5 Text Style with Sticky Attribute

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

2.2.5 Getting a Sample Image of Any Style

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

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

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

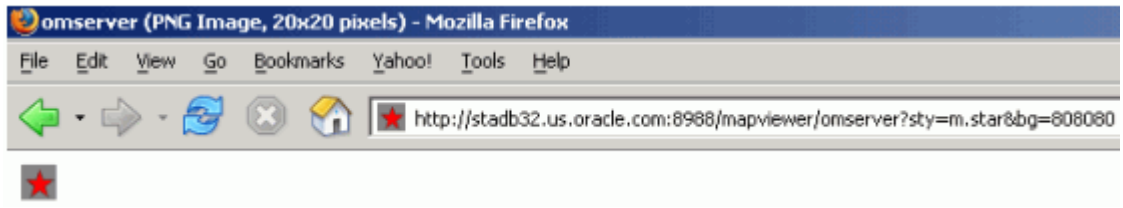
  For a transparent background, specify `bg` as an extended hexadecimal string to include the alpha values, in the format of `0x`*`aarrggbb`*. For example, 0x00ffffff will make the style image's background completely transparent, while 0x55ffffff is a white background with a transparency value of 0x55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

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

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

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

Figure 2-4 Sample Image of a Specified Marker Style

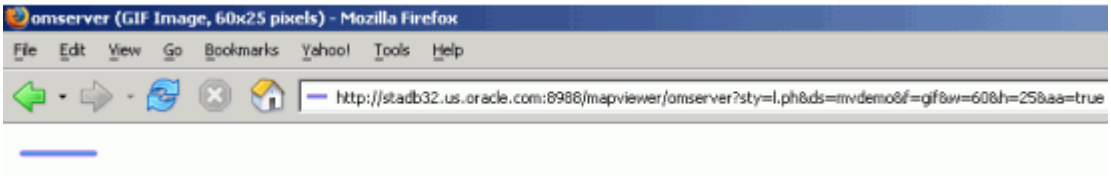


The following example generates an antialiased GIF image with the default white background, a width of 60 pixels, and a height of 25 pixels, displaying the line style named `L.PH` from the map visualization component data source named `mvdemo`:

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

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

Figure 2-5 Sample Image of a Specified Line Style



2.2.6 Allowing a Text Style to Overlap or Be Overlapped

You can specify that a text style `allow-overlap`, which means that any area feature, such as a polygon that uses it as a label style, will allow its text label be overlapped on a map. The following example shows an XML definition of a style with the `allow-overlap` attribute set to `true`.

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

For example, if a country's name is used as a label text, and if its style has the attribute value `allow-overlap="true"`, then other features' labels (such as state names) are allowed to overlap with it. The following figure shows part of a rendered map in which the country label "United States" overlaps slightly with the state labels "Colorado" and "Kansas".

Figure 2-6 Text Style with allow-overlap="true"

2.3 Themes

Theme is perhaps the most important concept in the map visualization component. A **theme** is a visual representation of a particular data layer.

Conceptually, a theme is a collection of geographic features that share similar attributes, plus the rendering and labeling rules that tell the map visualization component what styles to use to render and label the features. To be more exact, when you define a theme, you are actually providing the map visualization component with the following information: where and how to get the data, and how to render and label the data.

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

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

- Georeferenced images stored in BLOBs (image themes)
- Oracle Spatial GeoRaster data (GeoRaster themes)
- Oracle Spatial network data model (network themes)
- Oracle Spatial topology data model (topology themes)
- Cartographic annotation text (annotation themes)

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

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

- Predefined Themes
- JDBC Themes

- Image Themes
- GeoRaster Themes
- Network Themes
- Topology Themes
- WFS Themes
- WMTS Themes
- Custom Geometry Themes
- Annotation Text Themes
- LRS (Linear Referencing System) Themes
- Thematic Mapping
- Attributes Affecting Theme Appearance

2.3.1 Predefined Themes

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

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

For most types of predefined themes (but not WMS themes), you can use the Map Builder tool to create and preview themes. For information about the Map Builder tool, see #unique_26.

- Styling Rules in Predefined Spatial Geometry Themes
- How the Map Visualization Component Formulates a SQL Query for a Styling Rule
- Styling Rules with Binding Parameters
- Applying Multiple Rendering Styles in a Single Styling Rule
- Using Multiple Rendering Styles with Scale Ranges
- Caching of Predefined Themes
- Feature Labels and Internationalization
- Primary and Secondary Labels for Linear Features

2.3.1.1 Styling Rules in Predefined Spatial Geometry Themes

Each predefined theme is always associated with one or more **styling rules**, specifications in XML format that control aspects of how the theme is displayed. This section describes styling rules for predefined spatial geometry themes, such as the airport theme shown in Example 2-6.

Other types of themes, such as image, GeoRaster, network, and topology themes, have their own distinct styling rules requirements, and these are discussed in sections that explain these themes. However, the styling rules for all types of themes are grouped under the `<styling_rules>` element in an XML document, which is stored in the STYLING_RULES column for each predefined theme in the USER_SDO_THEMES view. (The `<styling_rules>` DTD is described in Themes: Styling Rules.)

> **Note:**
>
> The following naming conventions are used for prefixes in style names in the examples in this chapter: `v.` indicates variable (advanced style), `m.` indicates marker, `c.` indicates color, `l.` indicates line, and `t.` indicates text. (If the style is not under the current user's schema, you must specify the owner's schema name followed by a colon. For example: `mdsys:c.red`.)
>
> In the content (character data) of an XML document, `&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 2-6 XML Definition of Styling Rules for an Airport Theme

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

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

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

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

  ```
  <?xml version="1.0" standalone="yes"?>
  <styling_rules>
    <rule>
  ```

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

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

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

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

You can think of each styling rule as a filter into the base table or view of the theme, because it selects only a subset of the rows and applies the rendering and labeling styles of that rule. In fact, the map visualization component formulates a complete SQL query for each styling rule. This query string follows a fixed format, as described in How the Map Visualization Component Formulates a SQL Query for a Styling Rule.

2.3.1.2 How the Map Visualization Component Formulates a SQL Query for a Styling Rule

To see how the map visualization component formulates a SQL query for a styling rule, consider the first styling rule from the airport theme example (Example 2-6 in Styling Rules in Predefined Spatial Geometry Themes):

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

When the map visualization component processes this theme, it formulates a query string for this styling rule that looks like this:

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

In the preceding query string:

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

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

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

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

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

2.3.1.3 Styling Rules with Binding Parameters

As explained in How the Map Visualization Component Formulates a SQL Query for a Styling Rule, the `<features>` element of a styling rule can define a query condition to select features from the base table or view. This query condition typically contains hard-coded SQL expressions, such as `runway_num > 1` in the airport theme. However, you can instead include binding variables in the query predicate. Such a theme is often called a *templated theme*,

because it is essentially defining a template for how to display certain features, and the exact set of features is determined at runtime by providing a binding value to the query predicate.

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

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

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

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

2.3.1.4 Applying Multiple Rendering Styles in a Single Styling Rule

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

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

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

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

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

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

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

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

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

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

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

2.3.1.5 Using Multiple Rendering Styles with Scale Ranges

The `<rendering>` element (see Applying Multiple Rendering Styles in a Single Styling Rule) can have multiple `<style>` elements defined. You can also assign scale ranges for each `<style>` element. By using multiple stacked styles with scale ranges, you can define a single theme with different representations for different scales.

Example 2-8 shows the theme styling rules with stacked styles. Each style has a scale range defined.

Example 2-8 Theme Styling Rules with Stacked Styles

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <rule>
    <features> </features>
    <label column="STATE" style="T.S02_STATE_ABBREVS"> 1 </label>
    <rendering>
     <style name="C.COUNTIES" max_scale="5.0E7" scale_mode="RATIO"/>
     <style name="C.RB13_1" min_scale="5.0E7" max_scale="1.0E7" scale_mode="RATIO"/>
     <style name="L.DPH" min_scale="1.0E7" scale_mode="RATIO"/>
    </rendering>
  </rule>
</styling_rules>
```

2.3.1.6 Caching of Predefined Themes

By default, the map visualization component automatically caches the spatial data for a predefined theme when it is fetched from the database for processing by the map visualization

component rendering engine. By contrast, data for dynamic (JDBC) themes is never cached in the map visualization component. If you do not want any data for a predefined theme to be cached (such as for a theme whose underlying base table is constantly being updated), you can set the `caching` attribute to `NONE` in the `<styling_rules>` element for the theme. (The `<styling_rules>` element, including the `caching` attribute, is described in Themes: Styling Rules.)

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

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

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

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

2.3.1.7 Feature Labels and Internationalization

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

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

The steps for supplying translations and instructing the map visualization component to label a theme using a specific user language are as follows:

1. Prepare the translations.

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

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

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

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

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

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

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

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

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

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

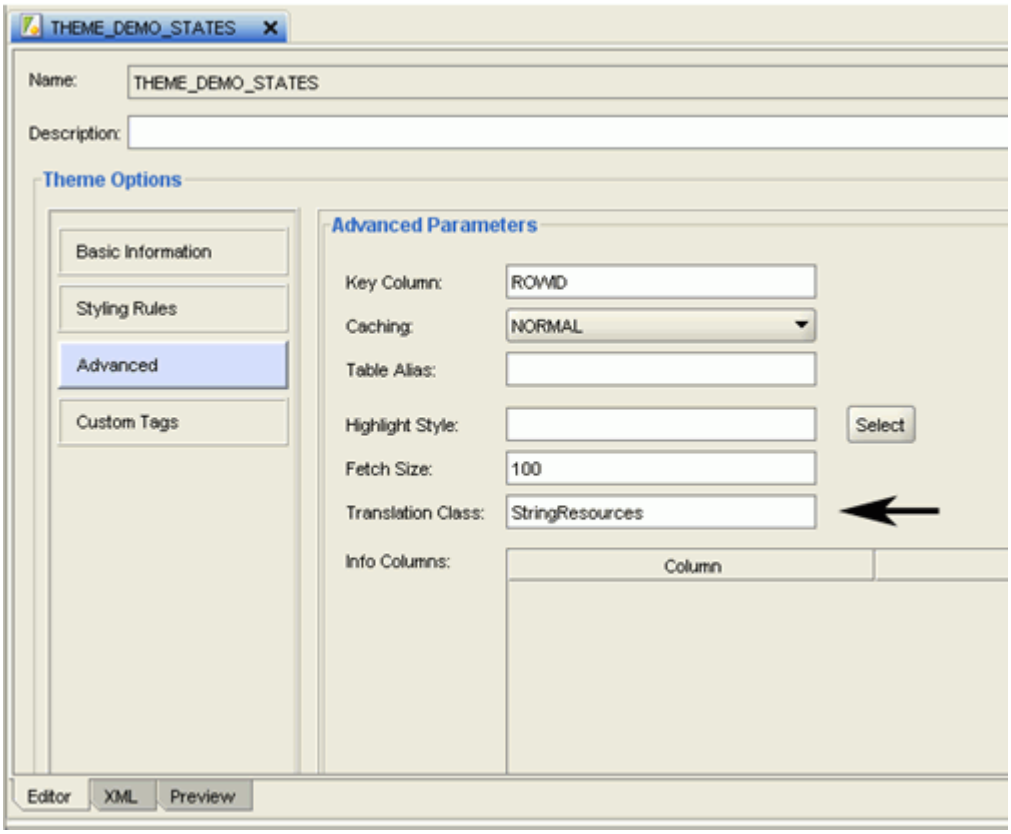
   If you place all the files of a resource bundle into a subfolder under `WEB-INF/classes`, then the name of the resource bundle (as known to the map visualization component) will need to be prefixed with this subfolder name. This is similar to how one places a Java class in a

directory structure that follows the package names. For example, if you put all the `StringResources*.properties` files in `WEB-INF/classes/i18n/`, then later when you register the resource bundle with the map visualization component, the actual name of your resource bundle should be `i18n.StringResources`.

3. Specify the name of the resource bundle in the theme definition by registering the resource bundle with the map visualization component.

   For the map visualization component to find your translated classes, you must specify the complete name of your resource bundle in the theme definition. The easiest way to do this is with the Map Builder utility, specifying the resource bundle name as the Translation Class in the Advanced Parameters pane of the theme editor. Figure 2-7 shows `StringResources` being specified for the Translation Class.

Figure 2-7 Specifying a Resource Bundle for a Theme



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

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

   Specify the preferred language for each map request the Oracle Maps JavaScript API (described in Oracle Maps JavaScript API) or the XML map request API (described in Map Visualization Servers).

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

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

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

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

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

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

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

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

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

2.3.1.8 Primary and Secondary Labels for Linear Features

The map visualization component includes support for labeling a linear feature with a primary and a secondary label. For example, a street name can be labeled with its English name as the primary label and with its local (native) language as the secondary label (or vice versa). When labeling a linear feature, the primary and secondary labels are labeled alternately along the feature when applicable.

When you specify a label column for a map visualization component theme, that column will provide both the primary and secondary text strings, delimited by "|||" (three vertical lines). For example, a text string of "Garden St|||Rua Jardim" contains the primary text string of "Garden St" (English) and secondary text string of "Rua Jardim" (Portuguese). In practice, that label

column may be a concatenation of two existing columns, and database triggers can be employed to maintain its consistency with its two base columns.

2.3.2 JDBC Themes

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

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

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

- The column of type SDO_GEOMETRY is treated as the spatial data column.
- Any column whose name or alias matches that specified in the JDBC theme's `label_column` attribute is treated as the labeling column, whose values are used as text for labels.
- Any other columns are treated as attribute data columns, which may or may not be used by the map visualization component. For example, if the rendering style is an advanced style, any attribute columns are processed by that style in the order in which they appear in the SELECT list in the query. Thus, if you are performing thematic mapping and using an advanced style, you must specify all attribute columns that are needed for the thematic mapping, in addition to the geometry column and optional labeling column. (A labeling column can also be an attribute column, in which case you do not need to specify that column in the SELECT list.)

Example 2-9 is a map request that includes a JDBC theme.

Example 2-9 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 the map visualization component executes for the JDBC theme in Example 2-9 is:

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

For this request, the map visualization component 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.

- Defining a Point JDBC Theme Based on Two Columns
- Storing Complex JDBC Themes in the Database

2.3.2.1 Defining a Point JDBC Theme Based on Two Columns

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

Example 2-10 is a JDBC theme that renders points from two columns, named LONG_LOC and LAT_LOC, of a table named POI. The x_column and y_column attributes specify the columns containing the point coordinate values. In this example, the points are rendered using the C.RED style, and the table values from the NAME column are rendered using the T.POI_NAME style.

Example 2-10 JDBC Theme Based on Columns

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

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

Example 2-11 JDBC Theme Based on Columns, with Query Window

If the table has a geometry column, you can specify SQL code to use the geometry column as a filter. Example 2-11 is similar to Example 2-10, but it adds the use of the SDO_FILTER operator to specify a query window based on the geometry in the column named GEOMETRY. In Example 2-11, the question mark (?) characters indicate that the lower-left and upper-right coordinates of the query window rectangle are taken from values supplied at runtime (not shown in this example).

```
<map_request>
  . . .
```

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

2.3.2.2 Storing Complex JDBC Themes in the Database

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

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

Example 2-12 is a sample `<styling_rules>` element of a predefined theme with a complex SQL query.

Example 2-12 Complex Query in a Predefined Theme

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

```
  </rule>
</styling_rules>
```

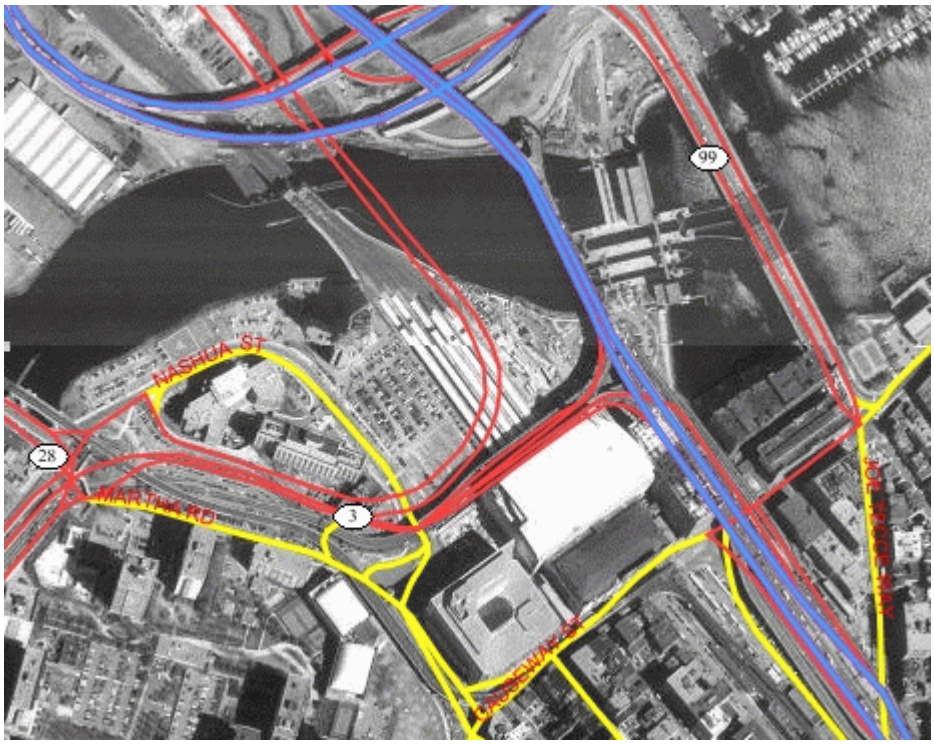
Even though Example 2-12 is defined as a predefined theme, the map visualization component still treats it as a JDBC theme at runtime when a user requests a map that includes this theme. As with a normal JDBC theme, the map visualization component by default imposes a window filtering process (if a query 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 2-12.

2.3.3 Image Themes

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

You can define an image theme dynamically or permanently (as a predefined theme) in the database. You can use image themes with vector (nonimage) themes in a map. Figure 2-8 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 2-8 Image Theme and Other Themes Showing Boston Roadways



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

- Store image data in its original format (such as JPEG) in a BLOB column in a database table, or as an Oracle Multimedia object (ORDSYS.ORDImage) that points to the original image file.
- 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 Creating Predefined Image Themes. To define a dynamic image theme in a map request, follow the guidelines for defining a JDBC theme, as explained in JDBC Themes, but note the following additional considerations with dynamic image themes:

- You must provide the original image resolution information when defining an image theme.
- The map visualization component 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, the map visualization component supports only GIF, JPEG, PNG, and TIFF image formats.

- Creating Predefined Image Themes

2.3.3.1 Creating Predefined Image Themes

To create a predefined image theme, you must store the definition of the image theme in the database by inserting a row into the USER_SDO_THEMES view (described in xxx_SDO_THEMES Views). Example 2-13 stores the definition of an image theme.

- `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`, the map visualization component can always render the image data.
- `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 2-13, the image resolution is 2 meters per pixel.

The DTD for the `<styling_rules>` element is presented in Themes: Styling Rules.

Example 2-13 Creating a Predefined Image Theme

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

```
    <features style="C.RED"> plevel=2 </features>
  </rule>
</styling_rules>' );
```

Example 2-13 creates an image theme named `IMAGE_LEVEL_2`. The base table (where all image data and associated MBRs are stored) is named IMAGES, and the minimum bounding rectangles (MBRs) for the images are stored in the column named IMAGE_MBR. In the STYLING_RULES column of the USER_SDO_THEMES view, an XML document with one `<styling_rules>` element is inserted.

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

2.3.4 GeoRaster Themes

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

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

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

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

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

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

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

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

Example 2-17 in Creating Predefined GeoRaster Themes prepares GeoRaster data for use and stores a GeoRaster theme in the database.

The map visualization component supports two types of map requests with objects from a GeoRaster table:

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

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

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

Example 2-14 defines a GeoRaster theme that contains a SQL statement that selects a single GeoRaster object. The theme assigns band 1 to the red channel, band 2 to the green channel, and band 3 to the blue channel. Because the `raster_pyramid` attribute is not specified, the map visualization component calculates the best pyramid level by using the spatial resolution values set during or after the georeferencing process. (In Example 2-14, `georid=1` in the WHERE clause refers to a column named GEORID in the GeoRaster table named PCI_IMAGE.)

Example 2-15 defines a GeoRaster theme that specifies the single GeoRaster object whose `rasterID` attribute value in the SDO_GEORASTER object is 1 (`raster_id="1"`) and associated with the raster data table named RDT_PCI. The theme specifies 2 as the pyramid level.

Subtopics:

- Creating Predefined GeoRaster Themes
- Using Bitmap Masks with GeoRaster Themes
- Reprojection of GeoRaster Themes
- Virtual Mosaic Themes

Example 2-14 GeoRaster Theme Containing a SQL Statement

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

Example 2-15 GeoRaster Theme Specifying a Raster ID and Raster Data Table

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

- Creating Predefined GeoRaster Themes
- Using Bitmap Masks with GeoRaster Themes
- Reprojection of GeoRaster Themes
- Virtual Mosaic Themes

2.3.4.1 Creating Predefined GeoRaster Themes

To create a predefined GeoRaster theme, you must store the definition of the GeoRaster theme in the database by inserting a row into the USER_SDO_THEMES view (described in xxx_SDO_THEMES Views). Example 2-16 stores the definition of a GeoRaster theme.

Example 2-16 creates a GeoRaster theme named `GEOR_BANDS_012`, in which band 0 is assigned to the red channel, band 1 to the green channel, and band 2 to the blue channel. The GeoRaster table name (GEOR_TABLE in this example) is inserted in the BASE_TABLE column of the USER_SDO_THEMES view, the GeoRaster column name (GEOR_COLUMN in this example) is inserted in the GEOMETRY_COLUMN column, and an XML document with one `<styling_rules>` element is inserted in the STYLING_RULES column.

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

The `<styling_rules>` element for a GeoRaster theme can contain the attributes described in GeoRaster Themes, including `raster_bands`, `raster_pyramid`, `raster_id`, and `raster_table`,

as shown in Example 2-16. Alternatively, the <styling_rules> element for a GeoRaster theme can be a rule definition. For example, to create a GeoRaster theme that selects a GeoRaster object from the GeoRaster table satisfying the WHERE clause condition georid=1, replace the <styling_rules> element in Example 2-16 with the following:

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

The <styling_rules> element for a GeoRaster theme can also specify one or more bitmap masks, as explained in Using Bitmap Masks with GeoRaster Themes.

The DTD for the <styling_rules> element is presented in Themes: Styling Rules.

Example 2-17 prepares GeoRaster data for use with a GeoRaster theme that is stored in the database. Comments in the code example briefly describe the main steps. For detailed information about requirements and steps for using GeoRaster data, see *Oracle Spatial GeoRaster Developer's Guide*.

Example 2-16 Creating a Predefined GeoRaster Theme

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

Example 2-17 Preparing GeoRaster Data for Use with a GeoRaster Theme

```
connect scott
Enter password: password

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

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

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

------------------------------------------------------------------
-- Create the GeoRaster DML trigger on the GeoRaster table, if
-- the Oracle Database release is before 11.1. (In Release 11.1 and later
```

```
-- this trigger is created automatically, so you do not need to create
-- it manually.)
-----------------------------------------------------------------

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

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

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

commit;

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

connect system;
Enter password: password

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

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

connect scott;
Enter password: password

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

connect system;
Enter password: password

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

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

connect scott;
Enter password: password

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

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

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

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

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

  commit;
end;/

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

declare
  gr sdo_georaster;
begin

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

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

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

  commit;
end;/

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

DECLARE
  gr sdo_georaster;
BEGIN
```

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

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

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

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

commit;

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

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

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

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

```
--------------------------------------------------------------------------
-- Create a predefined GeoRaster theme for the map visualization component.
--------------------------------------------------------------------------

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

commit;
```

2.3.4.2 Using Bitmap Masks with GeoRaster Themes

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

To specify a bitmap mask with a GeoRaster theme, use the `<bitmap_masks>` element in the `<styling_rules>` element for the predefined theme, as shown in Example 2-18.

The `<bitmap_masks>` element contains one or more `<mask>` elements, each with a mask definition for a specific GeoRaster object. In Example 2-18, a mask is defined for layers 1 and 2 of the GeoRaster object with the raster ID of 1 in the RDT_MASS_COLOR_MOSAIC table. The `<mask>` element has the following attributes:

- `raster_id` specifies the raster ID value of the GeoRaster object.
- `raster_table` specifies the raster data table (RDT).
- `layers` specifies the layer numbers in the GeoRaster object to be used for the mask.
- `zeromapping` specifies the transparency value to be applied during rendering on bitmap pixels with a value of 0 (zero). The attribute value can be from 0 (completely transparent) to 255 (completely opaque).
- `onemapping` specifies the transparency value to be applied during rendering on bitmap pixels with a value of 1. The attribute value can be from 0 (completely transparent) to 255 (completely opaque).

Example 2-18 Bitmap Mask in Predefined GeoRaster Theme

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

2.3.4.3 Reprojection of GeoRaster Themes

Effective with Oracle Spatial GeoRaster for Release 11.2.0.1, GeoRaster objects can be reprojected into a different SRID. It is recommended that you apply Oracle Database patch

10259201, to avoid black boundaries for adjacent reprojected GeoRaster objects when the objects are rendered in the map visualization component. For more information, see My Oracle Support document ID 1272931.1, *Black Lines After Reprojection Of Georaster Data Via Wms In Oracle Mapviewer*.

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

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

To specify a reprojection mode with a GeoRaster theme, use the `reproj_mode` keyword in the `<styling_rules>` element for the predefined theme, as shown in Example 2-19.

Example 2-19 Reprojection Mode in Predefined GeoRaster Theme

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

2.3.4.4 Virtual Mosaic Themes

In Oracle Spatial GeoRaster, a virtual mosaic is defined as any large collection of georeferenced GeoRaster objects, rectified or unrectified, from one or more GeoRaster tables or views that is treated as if it is a single GeoRaster object.

The virtual mosaic can be defined in different ways, like using a single or multiple GeoRaster tables (or views with a GeoRaster column), and using a full SQL query statement that results in a collection of GeoRaster objects.

The map visualization component supports the creation of the following types of predefined GeoRaster virtual mosaic themes:

- A theme based on table with a GeoRaster column and possibly containing a query condition. A list of tables with GeoRaster column can also be defined. The following XML definition defines a virtual mosaic theme based table LANDSAT5_IMAGES and GeoRaster column IMAGE:

  ```
  <?xml version="1.0" standalone="yes"?>
  <styling_rules theme_type="georaster" virtual_mosaic="true">
      <virtual_mosaic srid="32617" nodata_cell="true">
          <tables>
              <table name="LANDSAT5_IMAGES" georaster_column="IMAGE"/>
      </tables>
    </virtual_mosaic>
  </styling_rules>
  ```

- A theme based on a full SQL query. The following XML definition specifies a virtual mosaic theme based on a SQL query:

  ```
  <?xml version="1.0" standalone="yes"?>
  <styling_rules theme_type="georaster" virtual_mosaic="true">
      <virtual_mosaic srid="32617" nodata_cell="true">
          <sql>select image from landsat5_images</sql>
    </virtual_mosaic>
  </styling_rules>
  ```

The following parameters can be defined for a virtual mosaic theme.

`srid`: The spatial reference system (coordinate system) value.

nodata_cell: Specifies whether or not to consider NODATA (NODATA value or NODATA bitmap mask) when handling the overlap area. TRUE causes any cell with NODATA values to be considered as a NODATA cell, and the cell value is not involved in the overlap area calculation; FALSE causes any cell with NODATA values to be considered as normal cell, and the cell value is involved in the overlap area calculation.The default value is FALSE. If the value is TRUE and the resampling method is BILINEAR, BIQUADRATIC, CUBIC, AVERAGE4, or AVERAGE16, whenever a cell value involved in the resampling calculation is a NODATA value, the result of the resampling is also a NODATA value. The resulting NODATA value is the minimum NODATA value associated with the current raster layer, if multiple NODATA values or value ranges exist.

common_point_rule: The method for getting the cell value at the overlapping area. Can have one of the following values:

- OLDEST: The value from the GeoRaster object that has the oldest EndDataTime in the metadata is used.
- END: The value from the last encountered GeoRaster object is used.
- LATEST: The value from the GeoRaster object that has the most recent EndDataTime in the metadata is used.
- OLDEST: The value from the GeoRaster object that has the oldest EndDataTime in the metadata is used.
- CTC: The value from the GeoRaster object that is closest to the center of the output window is used.
- HIGH: The maximum cell value of all the overlapping GeoRaster objects is used.
- LOW: The minimum cell value of all the overlapping GeoRaster objects is used
- AVERAGE: The average of all cell values from the overlapping GeoRaster objects is used.
- HISHRES: The value from the GeoRaster object that has the highest spatial resolution is used.

resampling_mode: Specifies the resampling method (if resampling is involved or rectification is needed) to be used during the mosaic operation. Can have one of the following values: NN, BILINEAR, BIQUADRATIC, CUBIC, AVERAGE4, or AVERAGE16.

resampling_tolerance: Specifies the tolerance for not doing resampling when the source GeoRaster objects are not perfectly aligned. The value should be between 0 and 0.5, where the unit is pixel or cell (for example, 0.5 meaning one-half pixel or cell). If not specified, 0.5 is used, which means no resampling will occur.

color_balance: Specifies the method for color balancing. Can have one of the following values: NONE, LINEARSTRETCHING, NORMALIZATION.

min_stretch_value: Ignored if color_balance is not LINEARSTRETCHING; otherwise, specifies the lowest value in the range of the linear stretching method. Defaults to 0.

max_stretch_value: Ignored if color_balance is not LINEARSTRETCHING; otherwise, specifies the highest value in the range of the linear stretching method. Defaults to 255.

mean: Ignored if color_balance is not NORMALIZATION; otherwise, specifies the reference mean for the normalization method.

standard_deviation: Ignored if color_balance is not NORMALIZATION; otherwise, specifies the reference standard deviation for the normalization method.

2.3.5 Network Themes

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

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

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

- `<jdbc_network_query>` element: Specifies that this is a dynamically defined network theme.
- `network_name` attribute: Specifies the name of the network.
- `network_level` attribute (optional): Specifies the network hierarchy level to which this theme applies. (For a nonhierarchical network, specify 1, which is the default value.)
- `link_style` attribute (optional): Specifies the style name to be used for links.
- `direction_style` attribute (optional): Specifies the style name to be used for a link direction marker (for example, a directional arrow image).
- `bidirection_style` attribute (optional): Specifies the style name to be used for a bidirected link.
- `direction_position` attribute (optional): Specifies the position of the direction marker relative to the link start, as a number between 0 and 1. For example, 0.85 indicates 85 percent of the way between the link start and end points.
- `direction_markersize` attribute (optional): Specifies the size (number of pixels) of the direction marker.
- `direction_multimarker` attribute (optional): Specifies if the direction marker should be repeated over the link: `true` repeats the marker at a specified start position and each subsequent interval of that distance; `false` (the default) does not repeat the marker.
- `link_labelstyle` attribute (optional): Specifies the style name to be used for link labels in the column specified in the `link_labelcolumn` attribute.
- `link_labelcolumn` attribute (optional): Specifies the name of the column containing link labels to be rendered using the style specified in the `link_labelstyle` attribute.
- `node_style` attribute (optional): Specifies the style name to be used for nodes.
- `node_markersize` attribute (optional): Specifies the size (number of pixels) of the node marker.
- `node_labelstyle` attribute (optional): Specifies the style name to be used for node labels in the column specified in the `node_labelcolumn` attribute.
- `node_labelcolumn` attribute (optional): Specifies the name of the column containing node labels to be rendered using the style specified in the `node_labelstyle` attribute.
- `path_ids` attribute (optional): Specifies one or more path ID values of stored paths to be rendered. For more than one path, use commas to delimit the path ID values. For example,

`path_ids="1,3,4"` specifies that the paths with path ID values 1, 3, and 4 are to be rendered.

- `path_styles` attribute (optional): Specifies one or more style names associated with the paths specified in the `path_ids` attribute. For example, `path_styles="C.RED,C.GREEN,C.BLUE"` specifies styles to be used to render the first, second, and third paths (respectively) specified in the `path_ids` attribute.
- `path_labelstyle` attribute (optional): Specifies the style name to be used for path labels in the column specified in the `path_labelcolumn` attribute.
- `path_labelcolumn` attribute (optional): Specifies the name of the column containing path labels to be rendered using the style specified in the `path_labelstyle` attribute.

Additional network theme attributes related to network analysis are described in Using the Map Visualization Component for Network Analysis.

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

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

Example 2-20 Network Theme

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

- Creating Predefined Network Themes
- Using the Map Visualization Component for Network Analysis

2.3.5.1 Creating Predefined Network Themes

To create a predefined network theme, you must store the definition of the network theme in the database by inserting a row into the USER_SDO_THEMES view (described in xxx_SDO_THEMES Views). Example 2-21 stores the definition of a network theme.

Example 2-21 Creating a Predefined Network Theme

```
INSERT INTO user_sdo_themes  VALUES (
   'NYC_NET_1',
   'New York City network',
   'NYC_NET_LINK_TABLE',
   'GEOMETRY',
   '<?xml version="1.0" standalone="yes"?>
    <styling_rules
              theme_type="network"
```

```
            network_name="NYC_NET"
            network_level="1">
    <rule>
       <features>
         <link
            style="C.RED"
            direction_style="M.IMAGE105_BW"
            direction_position="0.85"
            direction_markersize="8">
         </link>
         <path
            ids="1,3"
            styles="C.BLUE,C.GREEN">
         </path>
         <node
            style="M.CIRCLE"
            markersize="5">
         </node>
       </features>
       <label>
         <link column="LINK_ID" style="T.STREET NAME"> 1 </link>
       </label>
    </rule>
  </styling_rules>' );
```

Example 2-21 creates a network theme named `NYC_NET_1` for level 1 of the network named `NYC_NET`. The network table name (NYC_NET_LINK_TABLE in this example) is inserted in the BASE_TABLE column of the USER_SDO_THEMES view, the link geometry column name (GEOMETRY in this example) is inserted in the GEOMETRY_COLUMN column, and an XML document with one `<styling_rules>` element is inserted in the STYLING_RULES column.

In the `<styling_rules>` element for a network theme, `theme_type` must be `network` in order for this theme to be recognized as a network theme. Elements for links, paths, and nodes can be specified in the same `<features>` element, as is done in Example 2-21:

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

Example 2-21 also contains a `<label>` element for links, specifying the link column `LINK_ID` and the label style `T.STREET NAME`.

The DTD for the `<styling_rules>` element is presented in Themes: Styling Rules.

2.3.5.2 Using the Map Visualization Component for Network Analysis

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

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

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

- `analysis_algorithm` attribute: Specifies the shortest-path analysis algorithm to use. Must be either `DIJKSTRA` or `ASEARCH`.

- `shortestpath_style` attribute: Specifies the style name to be used for the shortest path.
- `shortestpath_startnode` attribute: Specifies the start node to be used for the analysis.
- `shortestpath_endnode` attribute: Specifies the end node to be used for the analysis.
- `shortestpath_startstyle` attribute (optional): Specifies the style name to be used for the start node.
- `shortestpath_endstyle` attribute (optional): Specifies the style name to be used for the end node.

Example 2-22 defines a network theme that can be used for shortest-path analysis.

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

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

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

Example 2-22 Network Theme for Shortest-Path Analysis

```
<theme name="shortest_path_theme" user_clickable="false">
  <jdbc_network_query
    network_name="BI_TEST"
    network_level="1"
    jdbc_srid="0"
    datasource="mvdemo"
    analysis_algorithm="DIJKSTRA"
    shortestpath_style="L.PH"
    shortestpath_startnode="20"
    shortestpath_endnode="101"
    shortestpath_startstyle="M.STAR"
    shortestpath_endstyle="M.CIRCLE"
    asis="false">
  </jdbc_network_query>
</theme>
```

Example 2-23 Network Theme for Within-Cost Analysis

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

```
    asis="false">
  </jdbc_network_query>
</theme>
```

2.3.6 Topology Themes

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

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

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

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

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

Example 2-24 defines a topology theme that specifies attributes for the display of features and labels from the LAND_PARCELS table in the CITY_DATA topology. The SQL statement specifies the spatial feature column and the label column, and it includes all rows in the feature table.

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

- `edge_style` attribute: Specifies the name of the style to be used to render edges.
- `edge_label_style` attribute: Specifies the name of the text style to be used to render edge labels.
- `edge_marker_style` attribute: Specifies the name of the marker style to be used for edge markers.

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

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

Example 2-24 Topology Theme

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

Example 2-25 Topology Theme Using Debug Mode

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

- Creating Predefined Topology Themes

2.3.6.1 Creating Predefined Topology Themes

To create a predefined topology theme, you must store the definition of the topology theme in the database by inserting a row into the USER_SDO_THEMES view (described in xxx_SDO_THEMES Views). Example 2-26 stores the definition of a topology theme.

Example 2-26 creates a topology theme named `LANDPARCELS` for the topology named `CITY_DATA`. The feature table name (LAND_PARCELS in this example) is inserted in the BASE_TABLE column of the USER_SDO_THEMES view, the feature column name

(FEATURE in this example) is inserted in the GEOMETRY_COLUMN column, and an XML document with one <styling_rules> element is inserted in the STYLING_RULES column.

In the <styling_rules> element for a topology theme, theme_type must be topology in order for this theme to be recognized as a topology theme. The theme in Example 2-26 defines one styling rule that renders all land parcel features from the CITY_DATA topology using the C.RED style and using the T.TEXT STYLE label style for values in the FEATURE_NAME column of the feature table.

The DTD for the <styling_rules> element is presented in Themes: Styling Rules.

Example 2-26 Creating a Predefined Topology Theme

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

2.3.7 WFS Themes

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

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

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

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

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

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

```
  </GetCapabilities>
. . .
</Capability>
```

- `DescribeFeatureType` retrieves the feature information, including attributes and types.
- `GetFeature` retrieves the feature geometries and attributes. The output format for `GetFeature` requests is GML2.

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

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

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

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

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

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

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

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

Example 2-27 shows a request with a dynamic WFS theme. The WFS service is `geoserver`, and it is installed on the local system.

Example 2-27 WFS Request with a Dynamic WFS Theme

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

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

</map_request>
```

- Creating Predefined WFS Themes

2.3.7.1 Creating Predefined WFS Themes

To create a predefined WFS theme, you must store the definition of the WFS theme in the database by inserting a row into the USER_SDO_THEMES view (described in xxx_SDO_THEMES Views). Example 2-28 stores the definition of a WFS theme.

In Example 2-28, the WFS feature `POI` is used as the base table, and the attribute `THE_GEOM` is the spatial column. The styling rule information contains the `service_url` and `srs` information; and although not shown inExample 2-28, it can also specify a `key_column` value. The `<features>` and `<label>` elements of the styling rules are similar to the rules used in geometry themes. Hidden information (`<hidden_info>` element) can also be defined and used in Oracle Maps applications.

Example 2-29 shows a map request that uses the predefined theme created in Example 2-28.

In some cases, proxy information may affect the access to WFS servers. If this occurs, specify the appropriate proxy parameters in the map visualization component configuration file.

Example 2-28 Creating a Predefined WFS Theme

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

Example 2-29 Map Request with Predefined WFS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request
             title="Predefined WFS MAP"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="true"
             format="PNG_STREAM">

  <themes>
   <theme name="wfs_theme1" />
  </themes>

</map_request>
```

2.3.8 WMTS Themes

A **WMTS theme** supports the rendering of data delivered using the Open GIS Consortium (OGC) Web Map Tile Service (WMTS) standard, specifically the WMTS 1.0.0 implementation standard.

A WMTS theme fetches tile images from a WMTS-enabled server over the Internet and renders the images. The tile images on a WMTS-enabled server are spatially referenced with predefined content, extent, and resolution.

The WMTS operations `GetCapabilities`, `GetTile`, and `GetFeatureInfo` are used when rendering a WMTS theme. When a WMTS service is accessed by the map visualization component, it caches the capabilities information of that WMTS service. You may need to specify a proxy server in Map Builder when creating a WMTS theme and to edit the map visualization component configuration file (`MapViewerConfig.xml`) when a WMTS theme is being used.

- A `GetCapabilities` operation retrieves the server's general information, including the URL addresses to issue requests and the features available. In general, a WMTS capability request has the form:

  ```
  http://maps.opengeo.org/geowebcache/service/wmts?
  service=WMTS&version=1.0.0&request=GetCapabilities
  ```

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

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <Capabilities xmlns="http://www.opengis.net/wmts/1.0"
  xmlns:ows="http://www.opengis.net/ows/1.1"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:gml="http://www.opengis.net/gml" xsi:schemaLocation="http://www.opengis.net/
  wmts/1.0 http://schemas.opengis.net/wmts/1.0/wmtsGetCapabilities_response.xsd"
  version="1.0.0">
  <ows:ServiceIdentification>
    <ows:Title>Web Map Tile Service - GeoWebCache</ows:Title>
    <ows:ServiceType>OGC WMTS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
  </ows:ServiceIdentification>
  <ows:ServiceProvider>
    <ows:ProviderName>http://maps.opengeo.org/geowebcache/service/wmts</
  ows:ProviderName>
    <ows:ProviderSite xlink:href="http://maps.opengeo.org/geowebcache/service/wmts" />
    <ows:ServiceContact>
      <ows:IndividualName>GeoWebCache User</ows:IndividualName>
    </ows:ServiceContact>
  </ows:ServiceProvider>
  <ows:OperationsMetadata>
    <ows:Operation name="GetCapabilities">
      <ows:DCP>
        <ows:HTTP>
          <ows:Get xlink:href="http://maps.opengeo.org/geowebcache/service/wmts?">
            <ows:Constraint name="GetEncoding">
              <ows:AllowedValues>
                <ows:Value>KVP</ows:Value>
              </ows:AllowedValues>
            </ows:Constraint>
          </ows:Get>
  ```

```
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="GetTile">
      <ows:DCP>
        <ows:HTTP>
          <ows:Get xlink:href="http://maps.opengeo.org/geowebcache/service/wmts?">
            <ows:Constraint name="GetEncoding">
              <ows:AllowedValues>
                <ows:Value>KVP</ows:Value>
              </ows:AllowedValues>
            </ows:Constraint>
          </ows:Get>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="GetFeatureInfo">
      <ows:DCP>
        <ows:HTTP>
          <ows:Get xlink:href="http://maps.opengeo.org/geowebcache/service/wmts?">
            <ows:Constraint name="GetEncoding">
              <ows:AllowedValues>
                <ows:Value>KVP</ows:Value>
              </ows:AllowedValues>
            </ows:Constraint>
          </ows:Get>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
  </ows:OperationsMetadata>
  <Contents>
    <Layer>
      <ows:Title>bluemarble</ows:Title>
      <ows:WGS84BoundingBox>
        <ows:LowerCorner>-180.0 -90.0</ows:LowerCorner>
        <ows:UpperCorner>180.0 90.0</ows:UpperCorner>
      </ows:WGS84BoundingBox>
      <ows:Identifier>bluemarble</ows:Identifier>
      <Style isDefault="true">
        <ows:Identifier>_null</ows:Identifier>
      </Style>
      <Format>image/png</Format>
      <Format>image/jpeg</Format>
      <TileMatrixSetLink>
        <TileMatrixSet>EPSG:4326</TileMatrixSet>
      </TileMatrixSetLink>
      <TileMatrixSetLink>
        <TileMatrixSet>EPSG:900913</TileMatrixSet>
      </TileMatrixSetLink>
    </Layer>
    ...
    <TileMatrixSet>
      <ows:Identifier>EPSG:4326</ows:Identifier>
      <ows:SupportedCRS>urn:ogc:def:crs:EPSG::4326</ows:SupportedCRS>
      <TileMatrix>
        <ows:Identifier>EPSG:4326:0</ows:Identifier>
        <ScaleDenominator>2.795411320143589E8</ScaleDenominator>
        <TopLeftCorner>90.0 -180.0</TopLeftCorner>
        <TileWidth>256</TileWidth>
        <TileHeight>256</TileHeight>
        <MatrixWidth>2</MatrixWidth>
        <MatrixHeight>1</MatrixHeight>
```

```
    </TileMatrix>
   ...
  </TileMatrixSet>
  ...
</Contents>
<ServiceMetadataURL xlink:href="http://maps.opengeo.org/geowebcache/service/wmts?
REQUEST=getcapabilities&amp;VERSION=1.0.0"/>
</Capabilities>
```

- A `GetTile` operation retrieves a tile image from a WMTS server. A request to the server contains parameters of key-value pairs (KVP) that define the tile image.
- A `GetFeatureInfo` operation allows WMTS clients to request information at a specific position of a specific tile for a specific query layer.

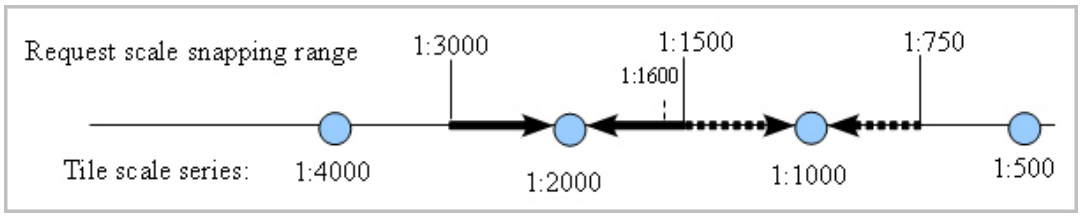
Detailed descriptions of these operations can be found at `http://www.opengeospatial.org/standards/wmts`.

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

- `current_threads` attribute (optional): an integer (int) variable; the default is 8. It is the number of concurrent threads for retrieving tile images from a WMTS server. In general, a larger number of concurrent threads allows more threads to retrieve image tiles in parallel from a map server. Other constraining factors may prevent you from using a number in the hundreds, but you may try to set it 16, 32, or a slightly larger value for faster processing.
- `format` attribute: Specifies the tile image formats stored in the WMTS server. The tif, jpeg, png, and png8 formats are supported in the map visualization component.
- `layer` attribute: Specifies the layer name for which the tile images are to be fetched.
- `matrix_set_id` attribute: Specifies the ID of the matrix set from which the tile images are to be fetched.
- `service_url` attribute: Corresponds to the capabilities address for HTTP GET requests. The `service_url` parameter for the map visualization component must be the online resource address for HTTP GET in the `<GetCapabilities>` element. In the preceding example, the value to be used is: `http://maps.opengeo.org/geowebcache/service/wmts?`
- `snap_to_tile_scale` attribute (optional): a Boolean value (`true` or `false`); the default is `false`.

  When `snap_to_tile_scale` is set to `true`, a request scale (derived from a device-window size and a request-data-window size) is snapped to a closest tile scale; the map scale will be in the tile scale. For example, if there are tiles in scale series of …, 1:4000, 1:2000, 1.1000, 1:500, …., a request scale of 1:1600 will be snapped to the 1:2000 tile scale, and the map will be using the same 1:2000 scale, as shown in Figure 2-9.

Figure 2-9 snap_to_tile_scale Attribute



If a map request has more than one WMTS theme that specifies `snap_to_tile_scale` as `true`, then only the first WMTS `snap_to_tile_scale` specification is set to `true` and all

others are reset to `false`. This is the logical behavior. For example, if two such themes both have the attribute set to `true`, but the two WMTS tile scale series are different from each other (the two map servers may be from different institutions using different scale series), the first theme will then use its closest tile scale to retrieve tile data and for the final request scale; at the same time, the second theme has to reset its `snap_to_tile_scale` to `false`, then has to find its own tile scale according to its `tile_resizing_option`, then retrieve tiles, and finally resize the tiles to match the first theme's tile scale.

- `style` attribute: Currently not used in the map visualization component; the default is the string `default`.
- `tile_resizing_option` attribute (optional): the string `unbiased`, `expand_biased`, or `contract_biased`; the default is `unbiased`. If `snap_to_tile_scale` is set to `true`, `tile_resizing_option` is ignored.

  For more information, see How the tile_resizing_option Attribute Works.
- `timeout` attribute (optional): an integer specifying a request's timeout period in milliseconds; the default is 0, (that is, no limit for a request to wait for a response).

  Specifying this attribute ensures that a map request is terminated when the map server does not respond within the specified time period, and thus frees the resources. You might try a value of 30000 (30,000 milliseconds, or 30 seconds).
- `top_left_corner_x` attribute (optional): the x coordinate value of the top left corner of the whole tile images' extent that is served by a WMTS server. If not specified, the value retrieved from the WMTS server is used. If you specify this attribute, also specify the `top_left_corner_y` attribute.
- `top_left_corner_y` attribute (optional): the y coordinate value of the top left corner of the whole tile images' extent that is served by a WMTS server. If not specified, the value retrieved from the WMTS server is used. If you specify this attribute, also specify the `top_left_corner_x` attribute.
- `version` attribute: the version of the WMTS specification implemented by the WMTS server.

- How the tile_resizing_option Attribute Works
- snap_to_tile_scale and tile_resizing_option Attribute Usage Guidelines
- Creating Predefined WMTS Themes

> **See Also:**
>
> OGC WMTS Support in the Map Visualization Component for information about the WMTS service for map visualization component, WMTS Operations, and preparing the WMTS service for the map visualization component.

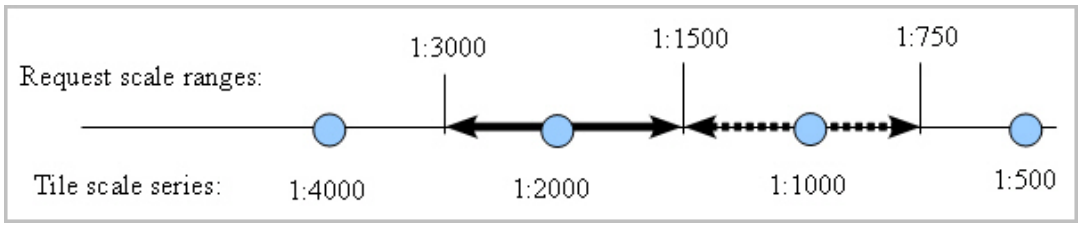
2.3.8.1 How the tile_resizing_option Attribute Works

If a WMTS theme's `snap_to_tile_scale` attribute is false (the default) or omitted, a request scale is always honored and the `tile_resizing_option` attribute value (specified or defaulted) is used when choosing a proper tile scale. However, if snap_to_tile_scale is true, the `tile_resizing_option` attribute value is ignored.

The `tile_resizing_option` attribute value can be one of the following string values: string `unbiased` (the default), `expand_biased`, or `contract_biased`.

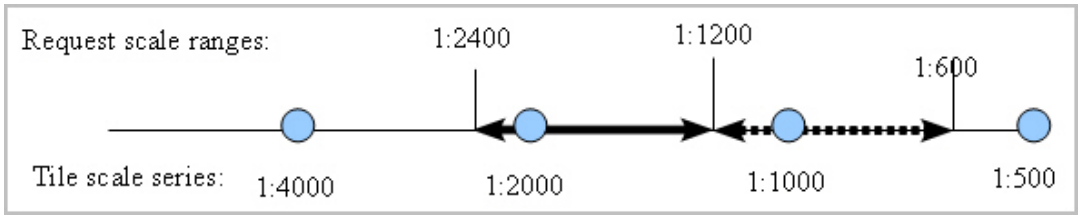
- `unbiased` (the default): The closest tile scale level is chosen, and then the tile images are expanded or contracted to generate a map in the request scale. For example, in Figure 2-10 the tile scale 1:2000 is used to generate any request scale map if a request scale falls within a scale range of 1:3000 and 1:1500.

Figure 2-10 unbiased tile_resizing_option Value



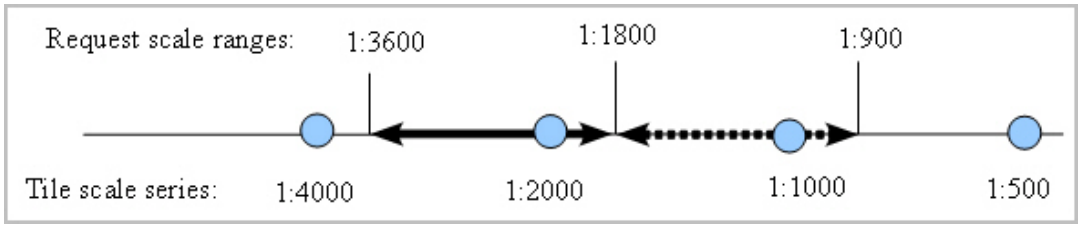
- `expand_biased`: When identifying a proper tile scale to generate a map in a request scale, a preference for expanding tile images to render a request map is used. In other words, you prefer to use a smaller tile scale to generate a map for a request scale. An 8:2 preference ratio for expanding is currently implemented. For example, in Figure 2-11 the tile scale 1:2000 is used for generating a request map if its scale is in the range of 1:2400 to 1:1200.

Figure 2-11 expand_biased tile_resizing_option Value



- `contract_biased`: When identifying a proper tile scale to generate a map in request scale, a preference for contracting tiles to render a request map is used. In other words, you prefer to use a larger tile scale to generate a map for a request scale. An 8:2 preference ratio for contracting is currently implemented. For example, in Figure 2-12 the tile scale 1:2000 is used for generating a request map if its scale is in the range of 1:3600 to 1:1800.

Figure 2-12 contract_biased tile_resizing_option Value



See also snap_to_tile_scale and tile_resizing_option Attribute Usage Guidelines.

2.3.8.2 snap_to_tile_scale and tile_resizing_option Attribute Usage Guidelines

This section presents general guidelines for using the `snap_to_tile_scale` and `tile_resizing_option` attributes to generate better quality maps. Because mapping has a wide variety of applications, you may also use your domain knowledge to set attribute values that best meet your needs.

Whenever possible, set `snap_to_tile_scale` to `true` to use a closest tile scale instead of a request scale, because original tile maps have the best map quality.

However, if you must honor a request scale, consider the following when setting the attributes:

- If you set a `tile_resizing_option` value, omit the `snap_to_tile_scale` attribute or set it to `false` (the default). If `snap_to_tile_scale` is set to `true`, a request scale will not be honored, and instead the closest tile scale will be used.
- If a map is a topographic map with annotations and thin linear features, you may want to use the `expand_biased` option.
- If a map is a thematic map, such as a land cover map, you may want to use the `contract_biased` option.
- If a map is a satellite image, you may want to use the `unbiased` or `contract_biased` option.

Regarding the tile_resizing_option possible values:

- In general, a `contract_biased` option may generate maps with more details, but these maps may need more tiles than maps from an `expand_biased` option, and retrieving more tiles takes more time.
- If a request scale is honored (when the `snap_to_tile_scale` attribute is set to `false`), when a request scale is close enough to a tile scale, then the same operation will be employed (either expanding or contracting), regardless of the specified `tile_resizing_option`. For example, if a request scale is 1:1900 in the figures in How the tile_resizing_option Attribute Works, tile maps in 1:2000 will be retrieved and expanded to render the request map in a map scale of 1:1900, regardless of whether the `unbiased`, `expand_biased`, or `contract_biased` option is specified; similarly, if a request scale is 1:2100, then tile maps in 1:2000 will also be retrieved and then contracted to render the requested map in a scale of 1:2100.

Example 2-30 shows a request with a dynamic WMTS theme, in which `snap_to_tile_scale="true"` is specified.

Example 2-30 Request with a Dynamic WMTS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request title="OpenGeo wmts theme (bluemarble)"
  datasource="mvdemo"
  width="1024"
  height="900"
  mapfilename="Bluemarble"
  format="PNG_STREAM">
  <center size="80.0">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>-112, 42.0</coordinates>
        </Point>
      </geometricProperty>
```

```
    </geoFeature>
  </center>
<themes>
  <theme name="wmtstheme: Earth" timeout="10000" snap_to_tile_scale="true">
    <wmts_gettile_request>
      <service_url> http://maps.opengeo.org/geowebcache/service/wmts </service_url>
      <version> 1.0.0 </version>
      <layer> bluemarble </layer>
      <matrix_set_id> EPSG:4326 </matrix_set_id>
      <format> image/png </format>
      <style> default </style>
      <top_left_corner_x> -180.0 </top_left_corner_x>
      <top_left_corner_y> 90.0 </top_left_corner_y>
    </wmts_gettile_request>
  </theme></themes>
</map_request>
```

2.3.8.3 Creating Predefined WMTS Themes

To create a predefined WMTS theme, you must store the definition of the WMTS theme in the database by inserting a row into the USER_SDO_THEMES view (described in xxx_SDO_THEMES Views). Example 2-31 stores the definition of a WMTS theme.

In Example 2-31, `earth_image` is the name of the WMTS theme, and `table_spaceholder` and `geom_col_spaceholder` are dummy values to fill the BASE_TABLE and GEOMETRY_COLUMN columns in the USER_SDO_THEMES view. The styling rule information contains the `service_url`, `layer`, `matrix_set_id`, `format`, and `style` information. The `top_left_corner_x` and `top_left_corner_y` attributes are not specified, so by default the values retrieved from the WMTS server are used.

Example 2-32 shows a map request that uses the predefined theme created in Example 2-31.

Example 2-31 Creating a Predefined WMTS Theme

```
INSERT INTO user_sdo_themes  VALUES (
'earth_image',
'Opengeo.org demo',
'table_spaceholder',
'geom_col_spaceholder',
'<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="wmts">
    <version> 1.0.0 </version>
    <service_url> http://maps.opengeo.org/geowebcache/service/wmts </service_url>
    <layer> bluemarble </layer>
    <matrix_set_id> EPSG:4326 </matrix_set_id>
    <format> image/png </format>
    <style> default </style>
</styling_rules>');
```

Example 2-32 Map Request with Predefined WMTS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="OpenGeo predefined wmts theme"
  datasource="mvdemo"
  width="1024"
  height="768"   mapfilename="Bluemarle: Earth"
  format="PNG_STREAM">
  <center size="10.0">
    <geoFeature>
      <geometricProperty typeName="center">
```

```
        <Point>
          <coordinates>-75.0,42.0</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
<themes>
  <theme name="earth_image" />
</themes>
</map_request>
```

2.3.9 Custom Geometry Themes

Custom geometry themes are associated with external spatial data (spatial data in a native format other than Oracle Spatial, such as shapefile). A custom geometry theme uses a spatial provider class to retrieve the native data, and the external provider must use the spatial data provider plug-in mechanism. The map visualization component provides a spatial provider interface class that the external provider must implement. The interface implementation has the following methods (some of them mainly provide information that can be used in user interfaces of applications like Map Builder):

```
public interface SDataProvider
{
  /**
   * Returns the initialization parameters for the provider.
   * @return String[] - array with initialization parameter names
   */
  public String[] getInitParameterNames();

  /**
   * Returns runtime parameter names. Runtime parameters are additional parameters
   * that the provider may use when retrieving the data objects.
   * @return String[] - array with runtime parameter names
   */
  public String[] getRuntimeParameterNames();

  /**
   * Returns a value that gives a hint for the runtime parameter value.
   * This hin can be used as a tooltip in user intefaces.
   * @param runtimeParam
   * @return   a String representing the hint value, or null if no hint is available
   */
  public String getRuntimeParameterHintValue(String runtimeParam);

  /**
   * This method is used to set the initialization parameters for the specific
   * data provider.
   * @param params - parameters to be used by the initialization method.
   * @return boolean - true if success; false otherwise
   */
  public boolean init(Properties params);

  /**
   * This method creates and returns an instance of SDataSet which contains
   * all the Spatial data produced by this provider, based on
   * the given parameters for a specific incoming map request.
   * <br>
   * The map visualization component calls this method on the custom theme producer
implementation.
   *
   * @param queryWin the search area to retrieve spatial objects. The window is
```

```
   * assumed to be already on data provider spatial reference system.
   * @param nonSpatialColumns - the list of attributes that will return with objects.
   * @param queryCondition - query condition expression (may have binding parameters).
   * @param bindingParameters - binding variables for query condition with binding
parameters.
   * @param params  - parameters that the provider may use to retrieve the data.
   * @return SDataObject - an instance of SDataSet class; null if failed.
   */
  public SDataSet buildDataSet(Rectangle2D queryWin,String []nonSpatialColumns,
                               String queryCondition,Object[] bindingParameters,
                                Properties params);

  /**
   * Returns the list of existing attributes for this data provider.
   * @param params  parameters that the provider may use to get the attribute list.
   * @return Field[] - array of attributes for this provider.
   */
  public Field[] getAttributeList(Properties params);

  /**
   * Returns the data set spatial extent MBR.
   * @param params  parameters that the provider may use to get the data extents
   * @return Rectangle2D - data spatial extent for this provider.
   */
  public Rectangle2D getDataExtents(Properties params);

 /**
   * Returns if provider can build spatial indexes.
   * If true, means that buildSpatialIndex method can be called.
   * @return
   */
  public boolean canBuildSpatialIndex();

  /**
   * Builds a spatial index on the data set.
   * @param params  parameters that the provider may use to build the spatial index.
   * @return boolean - true if saptial index creation is successful.
   */
  public boolean buildSpatialIndex(Properties params);

  /**
   * Clears provider internal caches (if provider implement caches).
   */
  public void clearCache();


  /**
   * Returns the parameter names that can be used to query for spatial tables.
   * Can be used more as information for user interfaces.
   * @return
   */
  public String[] getParametersToQuerySpatialMetadata();

  /**
   * Returns the spatial tables and spatial columns.
   * @param params must define the parameters returned from
   * getParametersToQuerySpatialMetadata.
   * @return  an array list defining the table name(index [0])
   *          and spatial column (index[1])
   */
  public String[][] getSpatialTables(Properties params);
}
```

The `init` and `buildDataSet` methods must be implemented. The other method implementations can be empty; however applications (such as the Oracle Map Builder Tool) can make use of these methods to handle the information about spatial data providers. A provider can implement its own spatial indexing mechanism; the map visualization component offers an implementation for the shapefile provider, and the `buildSpatialIndex` method creates an indexing file with the `.oix` extension in the `shapefile` directory. Creating and Registering a Custom Spatial Data Provider contains an example of how to implement and register a sample spatial provider with the map visualization component.

To render native data in the map visualization component with custom geometry themes, follow these steps:

1. Implement a spatial provider class based on the plug-in interface, and generate a jar file with the provider implementation. Copy the jar file to a directory that is part of the map visualization component CLASSPATH definition.
2. Register the provider in the map visualization component configuration. The map visualization component ships with a shapefile provider to access ESRI shapefiles, a GDAL-OGR provider to access data formats supported by OGR, and a Teradata provider to access data stored in a Teradata database. (See the GDAL-OGR and Teradata documentation for detailed information about handling spatial data in these environments.) The GDAL-OGR library `gdal.jar`, and Teradata libraries `terajdbc4.jar` and `tdgssconfig.jar`, must be on server classpath. The registration section in the map visualization component configuration file looks like this:

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

   Each provider must have `id` and `class` names defined: `id` is a unique name that identifies the provider, and `class` corresponds to the Java class implementation. The `<parameters>` element defines the initialization parameters of the provider.

   For the shapefile provider, the initialization parameter `datadir` defines where the map visualization component will look for the data files, and thus it should be a directory that is accessible to the map visualization component. The map visualization component first looks for data files based on the theme definition information; and if the data path defined in the theme definition is not accessible, the map visualization component looks for the data path defined in the configuration file.
3. Create custom geometry themes associated with the external spatial data provider.

Example 2-33 Defining a Dynamic Custom Geometry Theme

```
<theme name="custom_geom_theme_1" >
   <custom_geom_theme
     provider_id="shapefileSDP"
     srid="26986"
     render_style="C.RED"
     label_column="parcel_id"
     label_style="T.CITY NAME"
     datasource="mvdemo">
   <parameters>
    <parameter name="filename" value="/lbs/demo/shapefile/parcel.shp"/>
   </parameters>
```

```
  </custom_geom_theme>
</theme>
```

Example 2-34 Storing a Predefined Custom Geometry Theme

```
insert into user_sdo_themes values (
'SHAPE_THEME',
'Shapefile theme',
'CUSTOM_TABLE',
'GEOMETRY',
'<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="geom_custom" srid="26986" provider_id="shapefileSDP">
  <rule>
    <features style="C.RED"> </features>
    <label column="PARCEL_ID" style="T.CITY NAME"> 1 </label>
  </rule>
  <parameters>
    <parameter name="filename" value="/lbs/demo/shapefile/parcel.shp"/>
  </parameters>
</styling_rules>'
);
```

Although the external spatial data is outside the Oracle database, you still need to have a database connection to render this data. The database is used to store the metadata information related with the theme, as well as the styling information used to render and to label the data.

Example 2-33 shows the definition for a dynamic custom geometry theme. The XML element `<custom_geom_theme>` identifies a custom geometry theme. The `<parameters>` element defines the runtime parameters to be used by the provider. In this case `"filename"` is a runtime parameter, and `"/lbs/demo/shapefile/parcel.shp"` defines the file path. The map visualization component first attempts to use this file path definition; but if it is not accessible, it uses the data directory value defined in the configuration file for the shapefile spatial provider.

The runtime parameters for the available spatial providers are as follows (note that Map Builder provides the option to encrypt parameter values):

- For a shapefile provider:
  - `filename`: full path to the shapefile (.shp) on disk
- For GDAL-OGR:
  - `datasource`: a full OGR data source string. Depending on the data source format, this string can vary.(see the GDAL-OGR documentation for detailed information about connecting to different formats supported by GDAL-OGR.)

    For file formats, this parameter's value is usually the full path to the archive.

    For database connections, enter the full connection string to access the spatial table and spatial column. For example, for a Postgis table name STATES with one spatial column, the value might be:

    ```
    datasource = PG:dbname='template_postgis' host='localhost' port='5432'
    user='postgres' password='manager' tables=states
    ```
- For a Teradata provider:
  - `jdbcurl`: the JDBC connection to Teradata database. The string format is:

    ```
    jdbc:teradata://<host_address>/
    DATABASE=<db_name>,DBS_PORT=<db_port>,TMODE=ANSI,CHARSET=UTF8
    ```
  - `user`: database user.

  - `password`: database password.
  - `containerds`: Optional. Name of Teradata container data source defined on the application server (WebLogic, for example). If defined, this will be used first in the map visualization component, instead of the `jdbcurl` value.
  - `basetable`: name of the spatial table.
  - `geomcolumn`: name of the spatial column.
  - `fetchsize`: Optional. Number of rows to be prefetched.

The available attributes for a dynamic custom geometry theme are:

- `provider_id` specifies the spatial provider.
- `datasource` specifies the Oracle database connection. This connection is used to retrieve the styles to render the spatial data.
- `srid` specifies the spatial reference system (Oracle Spatial coordinate system).
- `render_style` specifies the style to be used when rendering the features.
- `label_column` specifies the name of the column containing label text to be used with the theme.
- `label_style` specifies the style to be used when labeling the features.
- `feature_attributes` specifies additional attributes that can be used with advanced styles.
- `key_column` specifies a key attribute that can be used in Oracle Maps applications.
- `query_condition` specifies the WHERE clause to filter feature selection. Shapefile providers do not support a query condition; however an OGR provider can be used to render shapefiles with a query condition. The query condition expression can define binding parameters (for example, `attr = :1`).

Example 2-34 shows how to store a predefined custom geometry theme definition. Use GEOMETRY as the geometry column name, and you can specify any name for the base table name. The `"theme_type=geom_custom"` attribute identifies the theme as a custom theme. The `<rule>` element has the same function as for an Oracle Spatial geometry theme. The `<parameters>` element defines the runtime parameters that the provider accepts. For the shapefile provider, the runtime `parameter` filename defines the path to the shapefile data.

You can override the runtime parameters section of a predefined custom geometry theme by the specifying the parameters in a map_request. For example, you can include the following in a `<map_request>` element:

```
<theme name="CUSTOM_THEME" >
  <parameters>
    <parameter name="filename" value="/lbs/demo/shapefile/counties.shp"/>
  </parameters>
</theme>
```

2.3.10 Annotation Text Themes

Oracle Spatial supports annotation text as specified in the *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture*, which defines **annotation text** as "simply placed text that can carry either geographically-related or ad-hoc data and process-related information as displayable text. This text may be used for display in editors or in simpler maps. It is usually lacking in full cartographic quality, but may act as an approximation to such text as needed by any application."

Oracle Spatial provides the ST_ANNOTATION_TEXT object type for storing annotation text, and the USER_ANNOTATION_TEXT_METADATA and ALL_ANNOTATION_TEXT_METADATA views for storing metadata related to annotation text. For more information about annotation text support, see *Oracle Spatial Developer's Guide*.

Each annotation text object may have one or more elements, and each element is defined by the following:

- Value: Text associated with element. If the value is null, the text is derived from the first non-null preceding element value. If all preceding elements have null values, the text is a text expression value derived from the metadata.
- Location: Spatial location associated with the annotation text object.
- Leader line: Linear feature associated with the annotation text object.
- Attributes: Graphic attributes used to display the text. If the value is null, graphic attributes are derived from the attributes value in the metadata.

The text expression in the metadata views can be any of the following:

- A column name.
- A function applied to a column name. For example: `substr(my_col,1,3)`
- The concatenation of two or more column names. For example: `column_1 || column_2 || column_3`
- A text value that is unrelated to a column name. In this case, it is treated as a simple text string that is used for any text element that has a null value.

Annotation text themes in the map visualization component are associated with database tables that have a column of type ST_ANNOTATION_TEXT. For each annotation text element, the map visualization component will render:

- The value (if not null) of the annotation text element as a string, using a text style that is created at real time based on the element attributes.
- The leader line (if not null) associated with the annotation text element. In this case, users can select a map visualization component style to render the leader line.

Each annotation text element has an envelope represented by a geometry, and which is used for spatial indexing. Therefore, you must do the following to use spatial indexing with annotation text tables in the map visualization component:

1. Insert a row into the USER_ANNOTATION_TEXT_METADATA view that specifies the name of the annotation text table and the PRIVATEENVELOPE attribute of the annotation text column (that is, the column of type ST_ANNOTATION_TEXT).

   The following example inserts a row for a table named ANNOTEXT_TABLE with an annotation text column named TEXTOBJ:

   ```
   INSERT INTO USER_SDO_GEOM_METADATA
     VALUES (
     'ANNOTEXT_TABLE',
     'TEXTOBJ.PRIVATEENVELOPE',
     SDO_DIM_ARRAY(
       SDO_DIM_ELEMENT('X', 0.0, 10.0, 0.0005),
       SDO_DIM_ELEMENT('Y', 0.0,10.0, 0.0005)
        ),
     null   -- SRID
   );
   ```

2. Create a spatial index on the annotation text envelope of the annotation text table.

The following example creates a spatial index named ANNO_TEXT_IDX on the annotation envelope of the table named ANNOTEXT_TABLE:

```
CREATE INDEX anno_text_idx ON annotext_table(textobj.privateenvelope)
  INDEXTYPE IS mdsys.spatial_index;
```

For themes with valid SRID information, if the metadata base map scale is defined, the element text sizes will be scaled as maps zoom in or out.

Example 2-35 defines the styling rules for a predefined annotation text theme in the map visualization component. The structure is similar to other map visualization component themes. Currently, just one styling rule is processed for each annotation theme. In this example, the theme type is `annotation`, the feature style L.`PH` is used to render leader lines, and the query condition `(id = 1 or id = 2)` is appended on the final query.

Example 2-36 shows the theme definition for a dynamic annotation text theme. The parameters defined are:

- `datasource`: the data source name
- `jdbc_srid`: the spatial reference identifier
- `annotation_table`: the annotation text table
- `annotation_column`: the annotation text column
- `leaderline_style`: the leader line style to be used

Example 2-37 is similar to Example 2-36, but it adds the behavior that if the `annotation_column` column contains a null value, then the value in the `textexpr_column` is used for the annotation instead. In Example 2-37, assume that the ANNOTATION_TABLE table contains a column named DEFAULT_ANNOTATION (which is used in Example 2-38). This additional column is specified in the `textexpr_column` attribute and in the SELECT statement.

Example 2-38 creates an annotation text table and prepares it to be used with the map visualization component.

Example 2-35 Styling Rules for a Predefined Annotation Text Theme

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="annotation">
    <rule>
        <features style="L.PH"> (id = 1 or id = 2) </features>
  </rule>
</styling_rules>
```

Example 2-36 Dynamic Annotation Text Theme Definition

```
<themes>
  <theme name="theme1" >
     <jdbc_annotation_query
       datasource="tilsmenv"
       jdbc_srid="0"
       annotation_table="ANNOTEXT_TABLE"
       annotation_column="textobj"
       leaderline_style="L.PH"
       >select textobj from annotext_table
     </jdbc_annotation_query>
   </theme>
</themes>
```

Example 2-37 Dynamic Annotation Text Theme with Default Annotation Column

```
<themes>
  <theme name="theme1" >
     <jdbc_annotation_query
       datasource="tilsmenv"
       jdbc_srid="0"
       annotation_table="ANNOTEXT_TABLE"
       annotation_column="textobj"
       textexpr_column="default_annotation"
       leaderline_style="L.PH"
       >select textobj, default_annotation from annotext_table
     </jdbc_annotation_query>
   </theme>
</themes>
```

Example 2-38 Script to Generate Annotation Text Data

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

------------------------------------------------------------------------
-- Create an annotation text table (a table that has a
-- column of ST_ANNOTATION_TEXT object type), and insert some records.
------------------------------------------------------------------------

create table annotext_table (
  id number,
  default_annotation varchar2(32),
  textobj ST_ANNOTATION_TEXT);

insert into annotext_table values (1,'Text_1',
ST_ANNOTATION_TEXT(
   ST_ANNOTATIONTEXTELEMENT_ARRAY(
          ST_ANNOT_TEXTELEMENT_ARRAY(
                ST_ANNOTATIONTEXTELEMENT('Sample Label 1',
                   SDO_GEOMETRY(2001, null, sdo_point_type(1,1,null),null,null),
                   SDO_GEOMETRY(2002,null,null,
                       SDO_ELEM_INFO_ARRAY(1,2,1),
                       SDO_ORDINATE_ARRAY(0,0, 1,1)), NULL)))));

insert into annotext_table values (2,'Text_2',
ST_ANNOTATION_TEXT(
   ST_ANNOTATIONTEXTELEMENT_ARRAY(
          ST_ANNOT_TEXTELEMENT_ARRAY(
                ST_ANNOTATIONTEXTELEMENT('Sample Label 2',
                   SDO_GEOMETRY(2001,null,sdo_point_type(10,10,null),null,null),
                   SDO_GEOMETRY(2002,null,null,
                       SDO_ELEM_INFO_ARRAY(1,2,1),
                       SDO_ORDINATE_ARRAY(5,10, 10,10)), NULL)))));

insert into annotext_table values (3, 'Text_3',
ST_ANNOTATION_TEXT(
   ST_ANNOTATIONTEXTELEMENT_ARRAY(
          ST_ANNOT_TEXTELEMENT_ARRAY(
                ST_ANNOTATIONTEXTELEMENT(null,
```

```
                    SDO_GEOMETRY(2002, null, null,
                        SDO_ELEM_INFO_ARRAY(1,2,1),
                        SDO_ORDINATE_ARRAY(2,5,4,5,6,5)),
                    SDO_GEOMETRY(2002,null,null,
                        SDO_ELEM_INFO_ARRAY(1,2,1),
                        SDO_ORDINATE_ARRAY(4,3, 4,5)),
'<?xml version="1.0" encoding="UTF-8" ?>
<textAttributes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="../../annotation_text.xsd">
 <textStyle fontFamily="Dialog" fontSize="14" fill="blue"/>
 <textlayout/>
</textAttributes>'
)))));

--------------------------------------------------------------------
-- Register the annotation text table in the user metadata view.
--------------------------------------------------------------------

insert into USER_ANNOTATION_TEXT_METADATA values(
  'ANNOTEXT_TABLE', 'TEXTOBJ', null, null, null);

--------------------------------------------------------------------
-- Update the metadata information.
--------------------------------------------------------------------

update user_annotation_text_metadata set
text_expression='default_annotation',
text_attributes =
'<?xml version="1.0" encoding="UTF-8" ?>
<textAttributes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation="../../annotation_text.xsd">
 <textStyle fontFamily="Serif" fontSize="14" fill="#ff0000"/>
 <textlayout/>
</textAttributes>';

--------------------------------------------------------------------
-- Register the annotation text geometry envelope on the user
-- metadata view of geometries.
--------------------------------------------------------------------

INSERT INTO USER_SDO_GEOM_METADATA
 VALUES (
 'ANNOTEXT_TABLE',
 'TEXTOBJ.PRIVATEENVELOPE',
 SDO_DIM_ARRAY(
   SDO_DIM_ELEMENT('X', 0.0, 10.0, 0.0005),
   SDO_DIM_ELEMENT('Y', 0.0,10.0, 0.0005)
    ),
 null   -- SRID
);

--------------------------------------------------------------------
-- Create a spatial index on the annotation text envelope.
--------------------------------------------------------------------

create index anno_text_idx on annotext_table(textobj.privateenvelope)
  indextype is mdsys.spatial_index;

-------------------------------------------------------------------------------
-- Insert a predefined theme into the map visualization component's theme view.
-------------------------------------------------------------------------------
```

```
INSERT INTO user_sdo_themes  VALUES (
  'ANNOTEXT_THEME',
  'Annotation text',
  'ANNOTEXT_TABLE',
  'TEXTOBJ',
  '<?xml version="1.0" standalone="yes"?>
   <styling_rules theme_type="annotation">
     <rule >
      <features style="L.PH"> </features>
     </rule>
   </styling_rules>' );

commit;
```

2.3.11 LRS (Linear Referencing System) Themes

An LRS theme is a special kind of map visualization component theme useful for visualizing features defined using the linear referencing system (LRS) of Oracle Spatial, which is explained in *Oracle Spatial Developer's Guide*.

Two tables are needed by an LRS theme: one (the **LRS table**) has an LRS geometry column, and the other (the **join table**) has one or two **measure columns**. Point events must have one measure column in the join table (for example, a road sign located at a measure of *m1*, and an accident that occurred at a measure of *m2*); while linear events must have two measure columns, one for a starting measure and one for an end measure (for example, from measure *m1* to measure *m2* the road pavement condition is poor, and from measure *n1* to measure *n2* the condition is fair). Each event is stored in one row in the join table.

After an LRS theme is defined, you only need to specify the theme name in a map request when you use the theme. The map visualization component automatically finds the theme's definition and constructs a query that joins these two specified tables; it then generates the geometry data by internally using an LRS function, either SDO_LRS.LOCATE_PT for point events or SDO_LRS.CLIP_GEOM_SEGMENT for linear events. The map visualization component then renders the generated point or linear segment, together with their attribute data, according to the styling rules for the theme.

To create a predefined LRS theme, you must store the definition of the LRS theme in the database by inserting a row into the USER_SDO_THEMES view. Example 2-39 stores the definition of an LRS theme named `LRS_THEME`.

In Example 2-39, the LRS table name (INTERSTATES_LRS, which has three columns, GEOM, HIGHWAY, and ROUTEN) is inserted in the BASE_TABLE column of the USER_SDO_THEMES view, the LRS geometry column (GEOM) is inserted in the GEOMETRY_COLUMN column, and XML document with one `<styling_rules>` element is inserted into the STYLING_RULES column.

In the `<styling_rules>` element for an LRS theme, `theme_type` must be `lrs` in order for this theme to be recognized as an LRS theme.

If the optional `key_column` attribute is omitted, no key column is included when constructing a query string to fetch data from a database. This is different from a predefined geometry theme, in which the ROWID column is the default key column.

The child elements `<rule>`, `<hidden_info>`, `<join_table>`, and `<join_columns>` can be included within the `<styling_rules>` element. The `<join_table>` and `<join_columns>` child elements are specific to LRS themes.

The <rule> element for an LRS theme is similar to the definition in a predefined themes, but its child element <features> must contain an asis attribute with its value set to true (asis="true").There can be 0, 1, or more <rule> elements within a <styling_rules> element.

The <hidden_info> element is optional for an LRS theme. It is defined for use in Oracle Maps applications. It specifies a list of attributes from either the LRS table or the join table, or both.

In <join_table> element, the name attribute (pavement_condition in this example) specifies the join table name. In this example, the join table contains columns such as id, seg_id, condition, from_measure, and to_measure. If two measure columns are used to define linear events in the join table, then attributes start_measure and end_measure (from_measure and to_measure in this example) must be specified, which indicate column names of measurements in the join table. If the join table contains one measure for point events, then a measure attribute is specified in this <join_table> element.

In the <join_columns> element, the lrs_table_column and join_table_column attributes must be specified, where lrs_table_column defines the column from the LRS table, and join_table_column specifies the column from the join table to join these two tables.

Example 2-40 shows a map request that uses the predefined LRS theme from Example 2-39.

> **See Also:**
>
> XML Format for Styles, Themes, Base Maps, and Map Tile Layers for reference information about elements and attributes specific to LRS themes (such as the <join_table> and <join_columns> elements).

Example 2-39 Creating an LRS Theme

```
INSERT INTO user_sdo_themes (name, description, base_table, geometry_column,
styling_rules)
VALUES (
'LRS_THEME',
'LRS theme example with 3 rules, 2 measure columns',
'INTERSTATES_LRS',
'GEOM',
'<?xml version="1.0" standalone="yes"?>
<styling_rules key_column="id" theme_type="lrs">
  <rule>
    <features  asis="true">
      condition=''good''
    </features>
    <label column="condition" style="T.STREET NAME"/>
    <rendering>
      <style name="L.PTH"/>
    </rendering>
  </rule>
  <rule>
    <features  asis="true">
      condition=''fair''
    </features>
    <label column="condition" style="T.STREET NAME"/>
    <rendering>
      <style name="L.SH"/>
    </rendering>
  </rule>
  <rule>
```

```
    <features  asis="true">
      condition=''poor''
    </features>
    <label column="condition" style="T.STREET NAME"/>
    <rendering>
      <style name="L.MAJOR STREET"/>
    </rendering>
  </rule>
  <hidden_info>
    <field column="highway"      name="highway name"/>
    <field column="routen"       name="highway number"/>
    <field column="condition"    name="pavement condition"/>
  </hidden_info>
  <join_table
    name="pavement_condition"
    start_measure="from_measure"
    end_measure="to_measure"
  />
  <join_columns
    lrs_table_column="highway"
    join_table_column="seg_id"
  />
</styling_rules>'
);
```

Example 2-40 Map Request with Predefined LRS Theme

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="LRS Theme test"
  datasource="mvdemo"
  width="640"
  height="480"
  bgcolor="#a6caf0"
  antialiase="true"
  format="PNG_STREAM">
  <center size="45">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>-95, 35</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
  <themes>
    <theme name="LRS_THEME"/>
  </themes>
</map_request>
```

2.3.12 Thematic Mapping

Thematic mapping refers to the drawing of spatial features based on their attribute values. The map visualization component uses thematic mapping to create maps in which colors or symbols are applied to features to indicate their attributes. For example, a `Counties` theme can be drawn using colors with different hues that map directly to the population density of each county, or an `Earthquakes` theme can be plotted with filled circles whose sizes map to the scale or damage of each earthquake.

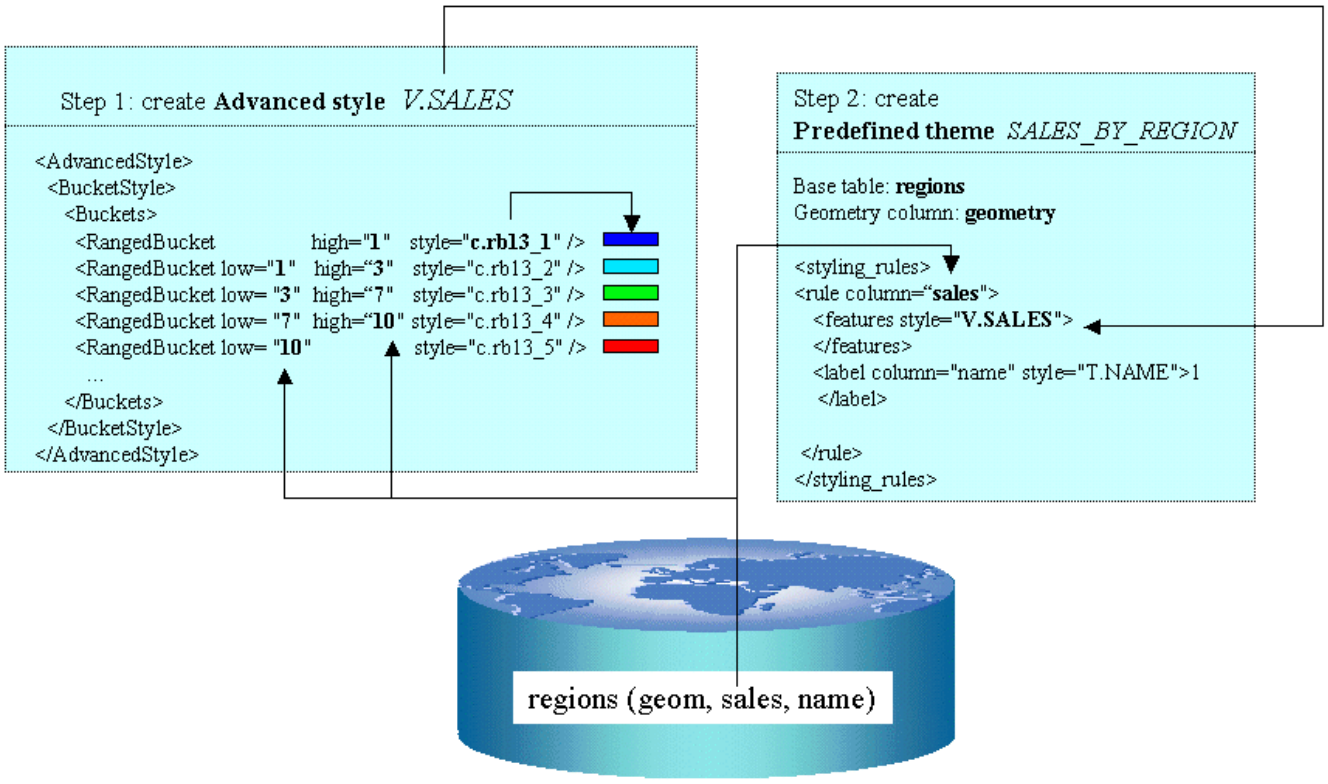
To achieve thematic mapping, you must first create an advanced style that is suitable for the type of thematic map, and then create a theme for the features specifying the advanced style

as the rendering style. In the styling rules for the theme, you must also specify attribute columns in the table or view whose values will be used to determine exactly how a feature will be rendered thematically by the advanced style.

For example, assume that you wanted to display a map in which the color used for each region reflects the level of sales for a particular product. To do this, create an advanced style that defines a series of individual range-based buckets (see Individual Range-Based Buckets), where each bucket contains a predefined range of sales values for a product, and each bucket has an associated rendering style. (Each region will be rendered using the style associated with the range in which that region's sales value falls.) Also specify the name of the column or columns that provide the attribute values to be checked against the ranges. In other words, the advanced style defines how to map regions based on their sales values, and the theme's styling rules tie together the advanced style and the attribute column containing the actual sales values.

Figure 2-13 shows the relationship between an advanced style and a theme, and how the style and the theme relate to the base table. In this figure, the advanced style named `V.SALES` defines the series of buckets. The predefined theme named `SALES_BY_REGION` specified the `V.SALES` style in its styling rules. The theme also identifies the SALES column in the REGIONS table as the column whose value is to be compared with the bucket ranges in the style. (Each bucket could be associated with a labeling style in addition to or instead of a rendering style, as explained in Specifying a Label Style for a Bucket.)

Figure 2-13 Thematic Mapping: Advanced Style and Theme Relationship



In addition to the individual range-based buckets shown in Figure 2-13, the map visualization component supports other bucket styles, as explained in Bucket Styles. You can also use more than one attribute column for thematic mapping, such as when drawing pie charts.

The rest of this section presents additional examples of thematic mapping.

Example 2-41 is the XML definition for an `Earthquakes` theme.

> **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 the map visualization component, which determines sequence only by the order in which elements appear in a definition.

Example 2-42 used the `<VariableMarkerStyle>` tag. The following examples use the `<ColorSchemeStyle>` tag in creating thematic maps of census blocks in California. Example 2-43 illustrates the use of a graduated color scale for a thematic mapping of population density. Example 2-44 is a thematic mapping of average household income using a graduated color scale. Example 2-45 is also a thematic mapping of average household income, but it uses a specific color style for each income range rather a graduated scale.

- `m.esso gasstation`, `m.texaco gasstation`, and the other style names have a space between the words in their names.
- The names are not case-sensitive. Therefore, be sure not to use case as a way of differentiating names. For example, `m.esso gasstation` and `M.ESSO GASSTATION` are considered the same name.
- A default collection bucket can be specified by using `#DEFAULT#` as its value. This bucket is used for any column values (gas station names) that are not specified in the other buckets.

A theme (`theme_gasstation`) is then defined that specifies the column (MERK) in the table that contains company names. The styling rules of the theme are shown in Example 2-47.

Table 2-2 Table Used with Gasoline Stations Theme

Column	Data Type
FID	NOT NULL NUMBER
ID	NUMBER
NAAM	VARCHAR2(31)
STRAAT_	VARCHAR2(30)
NR	NUMBER
TV	VARCHAR2(1)
AAND	VARCHAR2(2)
PCODE	VARCHAR2(6)
PLAATS	VARCHAR2(10)
GEOM	SDO_GEOMETRY
MERK	VARCHAR2(40)

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

The styling rules for the `theme_gasstation` theme specify that the marker (style `v.gasstations`) at a location specified by the content of the GEOM column is determined by

the value of the MERK column for that row. The style `v.gasstations` (see Example 2-46) 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`.

Example 2-41 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 2-41 has only one rule. The `<rule>` element includes an attribute named `column` that does not appear in the `Airport` theme in Example 2-6. The `column` attribute specifies one or more columns (comma-delimited) that provide the attribute values needed for thematic mapping. The style specified for the `<features>` element is named `v.earthquakes`, and it is an advanced style.

Another part of the definition of the `Earthquakes` theme specifies the table that contains the data to be rendered. This table must contain a column named RICHTER_SCALE in addition to a column (of type SDO_GEOMETRY) for the spatial data. (The table and the column of type SDO_GEOMETRY must be identified in the BASE_TABLE and GEOMETRY_COLUMN columns, respectively, of the USER_SDO_THEMES view, which is described in xxx_SDO_THEMES Views.) The RICHTER_SCALE column must be of type NUMBER. To understand why, look at the advanced style definition in Example 2-42.

Example 2-42 Advanced Style Definition for an Earthquakes Theme

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

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

Example 2-43 Mapping Population Density Using a Graduated Color Scheme

```
# ca pop density usbg_hhinfo
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="political">
<rule column="densitycy">
   <features style="v.CA Pop density">
```

```
   </features>
 </rule>
</styling_rules>
```

The table named USBG_HHINFO includes a column named DENSITYCY (used in Example 2-43). 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 2-43, Example 2-44 illustrates the use of a base color and a graduated color scheme, this time to show household income.

Example 2-44 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 2-44 and Example 2-45). 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 2-45 uses specific color styles for each average household income range.

Example 2-45 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 2-46.

Example 2-46 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 2-46:

Example 2-47 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).

- Thematic Mapping Using External Attribute Data

2.3.12.1 Thematic Mapping Using External Attribute Data

Previous discussion of thematic mapping has assumed that both the attribute data (such as population of sales totals) and the geospatial data (geometry objects representing boundaries, locations, and so on) are in the same database. However, the attribute data can come from a source outside the current database; for example, the attribute data might reflect aggregated results of a business intelligence (BI) query performed on a different database, or the attribute data might come from a comma-delimited list of sales values exported from a spreadsheet. Such attribute data, from outside the database that contains the geospatial data, is called **external attribute data**.

To use external attribute data with the map visualization component, you must use the **nonspatial data provider** plug-in mechanism, in which a custom data provider is associated with a map visualization component theme (predefined or dynamic) in the same map request. When the map visualization component processes the theme, it calls the nonspatial data provider to join nonspatial attribute data with the spatial data that has been fetched for the theme.

To use a nonspatial data provider, follow these steps:

1. Implement your Java nonspatial data provider by implementing the map visualization component defined interface `oracle.mapviewer.share.ext.NSDataProvider`.
2. Register the nonspatial data provider implementation with the map visualization component (in its configuration file). There you can also specify a set of global parameters that your implementation may depend on. Each custom data provider implementation class must have a unique ID that you assign.

3. Place a library containing the nonspatial data provider implementation classes in the library path of the map visualization component, such as its `web/WEB-INF/lib` directory.
4. Include the nonspatial data provider implementation in a map request by invoking the following method on the map visualization component Java client API class MapViewer:

```
addNSDataProvider(java.lang.String providerId,
                  java.lang.String forTheme,
                  java.lang.String spatialKeyColumn,
                  java.lang.String customRenderingStyle,
                  java.util.Properties params,
                  long timeout)
```

   For information about the `addNSDataProvider` parameters, see the Javadoc reference information for the map visualization component, available at a URL in the form `http://host:port/mapviewer/mapclient`, where *host* and *port* indicate where Oracle Fusion Middleware listens for incoming requests. For example: `http://www.mycorp.com:8888/mapviewer/mapclient`

Example 2-48 shows a simple nonspatial data provider implementation. This implementation is also supplied with the map visualization component as a default nonspatial data provider.

Example 2-48 Nonspatial (External) Data Provider Implementation

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Properties;
import java.util.Vector;

import oracle.mapviewer.share.ext.NSDataSet;
import oracle.mapviewer.share.ext.NSDataProvider;
import oracle.mapviewer.share.ext.NSRow;
import oracle.lbs.util.Logger;

import oracle.mapviewer.share.Field;

/**
 * A simple implementation of the NSDataProvider interface. When invoked, it supplies
tabular attribute data to the map visualization component out
 * of a file or URL. The data in the file must be orgazined as following: <br>
 * <UL>
 *   <LI> The first line contain a single character which is the delimiter
 *        between columns in the subsequent lines.
 *   <LI> Each line after the first in the file represent one data row
 *   <LI> Each field in the row must be separated by the delimiter char only
 *   <LI> The first field in each line must be a string (key) that serves as the
 *         key; the rest of the fields must be numeric values
 * </UL>
 *
 * When incorporating this data provider in a map request, one of the following
 * two parameters must be specified:
 * <UL>
 *   <LI> file  if the custom data is stored in a local file; this parameter
 *        specifies the full path to that file
 *   <LI> url  if the custom data can be accessed from a web; this parameter
 *        specifeis the full URL to the data file.
 * </UL>
 *
 *
 */
public class NSDataProviderDefault implements NSDataProvider
{
```

```
private static Logger log = Logger.getLogger("oracle.sdovis.nsdpDefault");

public boolean init(Properties params)
{
  return true;
}

public NSDataSet buildDataSet(Properties params)
{
  String file = params.getProperty("file");
  if(file!=null)
    return readFromFile(file);

  String url = params.getProperty("url");
  if(url!=null)
    return readFromUrl(url);

  log.error("Must supply either file or url for default NS data provider.");
  return null;
}

public void destroy()
{
}

protected NSDataSet readFromFile(String file)
{
  BufferedReader in = null;
  try{
    in   = new BufferedReader(new FileReader(file));
    String line = in.readLine();
    String delimiter = line.substring(0,1);

    Vector rows = new Vector();

    while ( (line=in.readLine()) != null)
    {
      NSRow row = buildRow(line, delimiter);
      if(row!=null)
        rows.add(row);
    }

    NSDataSet res = new NSDataSet(rows);
    return res;
  }catch(Exception ex)
  {
    log.error(ex);
    return null;
  } finally
  {
    try{
      if(in!=null)
        in.close();
    }catch(Exception e){}
  }
}

protected NSDataSet readFromUrl(String url)
{
  log.error("url not supported yet.");
  return null;
}
```

```
  protected NSRow buildRow(String line, String delimiter)
  {
    if(line==null || line.length()<1)
      return null;

    String[] fields = line.split(delimiter);
    if(fields==null || fields.length==0)
      return null;

    Field[] row = new Field[fields.length];

    Field a = new Field(fields[0]);
    a.setKey(true);

    row[0] = a;

    for (int i = 1; i < fields.length; i++)
    {
      try{
        double d = Double.parseDouble(fields[i]);
        a = new Field(d);
        row[i] = a;
      }catch(Exception e)
      {
        log.warn("invalid row field (key="+fields[0]+")");
        return null;
      }
    }

    return new NSRow(row);

  }
}
```

2.3.13 Attributes Affecting Theme Appearance

Some attributes of the `<theme>` element affect only the appearance of the map display, rather than determining the data to be associated with the theme. These appearance-related attributes control whether and how the theme is processed and rendered when a map is generated. Examples include the following attributes:

- `min_scale` and `max_scale` determine whether or not a theme is displayed at various map scales (levels of resolution). For example, if you are displaying a map of streets, there are certain map scales at which the streets would become too dense for a usable display, such as when viewing an entire state or province. In this case, you should create a theme for streets, and specify minimum and maximum scale values to ensure that individual streets affected by the theme are displayed when the scale is appropriate and otherwise are not displayed.
- `labels_always_on` determines whether or not labels for the theme will be displayed if they would overlap another label. By choosing appropriate `labels_always_on` values and choosing an appropriate order of themes to be processed within a map request, you can control how cluttered the labels might become and which labels have priority in getting displayed.
- `fast_unpickle` determines the unpickling (unstreaming) method to be used, which can involve a trade-off between performance and precision in the display.

- `fixed_svglabel`, `visible_in_svg`, `selectable_in_svg`, `onclick`, `onmousemove`, `onmouseover`, and `onmouseout` affect the appearance of SVG maps.

To specify any appearance-related attributes, use the `<theme>` element (described in [31.2.20]) with the XML API.

2.4 Maps

A map can consist of a combination of elements and attributes.

These elements and attributres can include the following:

- Background image
- Title
- Legend
- Query window
- Footnote (such as for a copyright notice)
- Base map
- Predefined themes (in addition to any in the base map)
- JDBC themes (with dynamic queries)
- Dynamically defined (temporary) styles

These elements and attributes, when specified in a map request, define the content and appearance of the generated map. Map Visualization Servers 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 Map Visualization Component Metadata Views and xxx_SDO_MAPS Views. The DEFINITION column in the USER_SDO_MAPS view contains an XML definition of a base map.

Example 2-49 shows a base map definition.

Example 2-49 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 2-49, 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 Map Size and Scale.)

The display order of themes in a base map is the same as their order in the base map definition. In Example 2-49, 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`.

- Map Size and Scale
- Map Legend

2.4.1 Map Size and Scale

Map size is the height of the map in units of the map data space. For example, if the map data is in WGS 84 geographic coordinates, the map center is (-120.5, 36.5), and the size is 2, then the height of the map is 2 decimal degrees, the lower Y (latitude) value is 35.5 degrees, and the upper Y value is 37.5 decimal degrees.

Map scale is expressed as units in the user's data space that are represented by 1 inch on the screen or device. Map scale for the map visualization component 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 the map visualization component) 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 the map visualization component) 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 the map visualization component 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 the map visualization component 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 the map visualization component 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 map visualization component 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 2-49 in Maps includes `min_scale` and `max_scale` values.

You also assign scale values for theme labels, to enable the showing or hiding of labels with values different from the base theme scales, by using the theme label scale parameters `label_min_scale` and `label_max_scale`. These parameters are similar to the `min_scale` and `max_scale` parameters, but the labels are shown if the map scale is in the visible range defined by `label_min_scale` and `label_max_scale`. (The label scale values are ignored if the theme is not in the visible scale range defined by `min_scale` and `max_scale`.) The following is a theme definition with label scale values; the labels will be shown when the map scale is between 5 and 2, but the theme features will be shown when the map scale is between 10 and 0:

```
<theme name="theme_us_states" min_scale="10" max_scale="0"
     label_min_scale="5" label_max_scale="2"/>
```

To determine the current map scale for a map returned by the map visualization component, 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. The map visualization component assumes a typical screen resolution of 96 dpi. Because 96 pixels equals 1 inch, the height of the returned map is 3.646 inches (350/96 = 3.646). In this example, the size of the map is 10, and therefore the map scale is approximately 2.743 (10/3.646 = 2.743).

Alternatively, you can request a map using a map scale value without specifying a unit, such as 50000 for a scale of 1:50000, by specifying the `scale_mode` theme attribute value as `ratio`. (If the `scale_mode` theme attribute value is `screen_inch`, the scale refers to a unit.) To use a scale defined without a unit, request the map specifying the center and ratio scale.

To find the equivalent map visualization component screen inch scale for a ratio scale, follow these steps:

1. Find the numerical fraction of a meter associated with one screen pixel. For example, if the screen resolution is 96 dpi (dots per inch), the number of meters on the screen for each screen pixel is 0.000265 (that is, 0.0254/96).
2. Find the map scale for one screen pixel (the `mapdotScale` value), as follows:
   - For projected data (meters), multiply the result of step 1 by the ratio scale. For example, if the ratio scale is 50000 (50 thousand) and the screen resolution is 96 dpi, the result is 13.25 meters for each pixel (50000 * 0.000265).
   - For geodetic data (degrees), multiply the result of step 1 by the number of meters (on the surface of the Earth) for each degree. (This number will depend on the coordinate system associated with the data.) For example, if one degree = 111195 meters and if the screen resolution is 96 dpi, the result is 29.466675 meters for each pixel (111195 * 0.000265).
   - For data using any other unit, use the approach for projected data using meters.

3. Because the map visualization component scale is per screen inch instead of per screen pixel, multiply the result of step 2 by the dpi value. For example, if the result of step 2 is 13.25 meters at 96 dpi, the number of meters for each screen inch is 1272 (13.25 * 96).

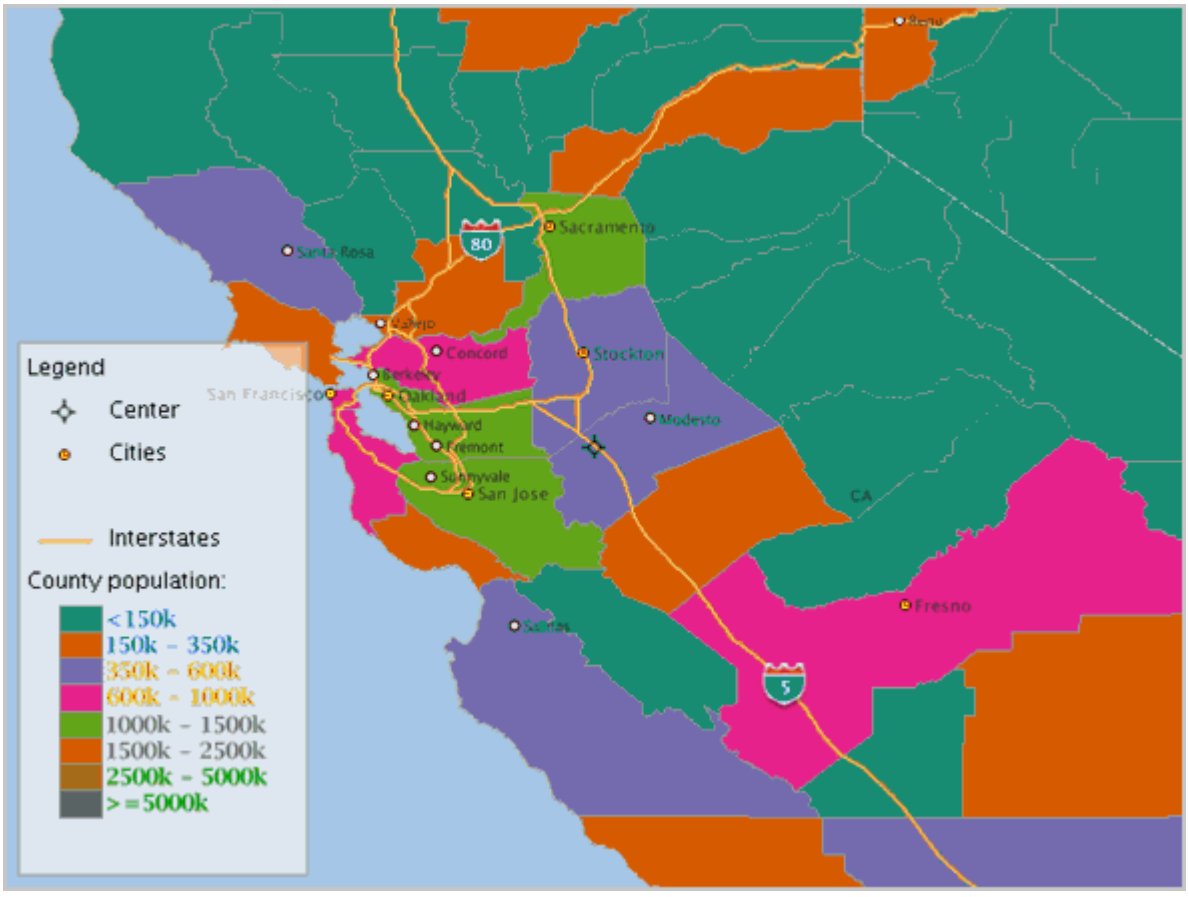
2.4.2 Map Legend

A **map legend** is an inset illustration drawn on top of the map and describing what various colors, symbols, lines, patterns, and so on represent. You have flexibility in specifying the content and appearance of the legend. You can:

- Customize the background, border style, and font
- Have one or more columns in the legend
- Add space to separate legend entries
- Indent legend entries
- Use any map visualization component style, including advanced styles

Example 2-50 is an excerpt from a request that includes a legend.

Figure 2-14 Map with Legend



Notes on Example 2-50 and Figure 2-14:

- This example shows a legend with a single column, although you can create multiple columns in a legend.

- Each entry in the column definition can identify label text and whether the text is the legend title (`is_title="true"`), a style name and associated text, or a separator (`is_separator="true"`) for vertical blank space to be added (after the *cities* entry in this example).

As an alternative to specifying the legend content in one or more <column> elements, you can request an **automatic legend** based on the map request. With an automatic legend, you specify the legend header, and the map visualization component generates the legend based on the themes that have any interaction with the map area. Themes from the map request and from the base map are considered. (Some legend items might not be visible, though, such as if a theme interacts with the query window but no features of the theme are visible on the map.)

Example 2-51 is a map request that requests an automatic legend (because the `<legend>` element does not include any `<column>` elements).

Example 2-52 requests an automatic legend in which the `<legend>` elements specifies the themes to be used to generate the legend items. In this example, even if the map result shows more themes, the legend items are based on the `THEME_COUNTIES_3397829` and `THEME_US_AIRPORT` themes specified in the `<legend>` element.

You cannot combine an automatic legend with the use of `<column>` elements. If the `<legend>` element contains any `<column>` elements, a column/entry legend is created.

The map visualization component uses the following considerations when it builds automatic legend items:

- Each legend column has a maximum of five entries (an advanced style is considered one entry).
- The legend text for simple rendering styles comes from the theme description if defined, otherwise from the theme name.
- If a rendering style is used in more than one theme, the style is repeated in the legend but with text related to the theme to which it applies.
- Labeling styles are not repeated in the legend. The style text for labeling styles comes from the style description.
- Advanced styles are not repeated in the legend.

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 Specifying Global Map Configuration Options.) The default position for a legend is `SOUTH_WEST`.

Example 2-50 Legend Included in a Map Request

```
<?xml version="1.0" standalone="yes"?>
<map_request
             basemap="density_map"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="false"
             format="PNG_STREAM">
  <center size="4.2">
     <geoFeature render_style="m.image134_bw">
         <geometricProperty typeName="center">
             <Point srsName="SDO:8307">
                 <coordinates>-121.2615, 37.5266</coordinates>
             </Point>
         </geometricProperty>
```

```
    </geoFeature>
  </center>

  <legend bgstyle="fill:#ffffff;fill-opacity:100;stroke:#a3a3a3" profile="MEDIUM"
position="SOUTH_WEST">
          <column>
            <entry text="Legend" is_title="true" />
            <entry style="M.IMAGE134_BW" text="Center" />
            <entry style="M.ALL_CITY_L2" text="Cities" />
            <entry is_separator="true" />
            <entry style="L.S04_ROAD_INTERSTATE" text="Interstates" />
            <entry text="County population:" />
            <entry style="V.COUNTY_POP_DENSITY_8" tab="1" />
          </column>
  </legend>

  <!--
  <themes>

  </themes>
  -->

</map_request>
```

Figure 2-14 shows a map with the legend specified in Example 2-50.

Example 2-51 Map Request with Automatic Legend

```
<?xml version="1.0" standalone="yes"?>
<map_request
             title="Automatic legend"
             datasource = "mvdemo"
             width="640"
             height="480"
             bgcolor="#a6cae0"
             antialiase="false"
             format="PNG_STREAM">
  <center size="4.5">
     <geoFeature >
         <geometricProperty typeName="center">
             <Point>
                 <coordinates>-122.2615, 37.5266</coordinates>
             </Point>
         </geometricProperty>
     </geoFeature>
  </center>

  <themes>
    <theme name="THEME_COUNTIES_3397829" />
    <theme name="THEME_US_ROAD1" />
    <theme name="THEME_US_AIRPORT" />
  </themes>

 <legend bgstyle="fill:#ffffff;fill-opacity:128;stroke:#ff0000;stroke-opacity:128"
profile="medium" font="Courier">
  </legend>

</map_request>
```

Example 2-52 Automatic Legend with Themes Specified

```
<map_request
             title="Legend with themes defined"
```

```
            datasource = "mvdemo"
            width="640"
            height="480"
            bgcolor="#a6cae0"
            antialiase="false"
            format="PNG_STREAM">
  <center size="4.5">
     <geoFeature >
         <geometricProperty typeName="center">
             <Point>
                 <coordinates>-122.2615, 37.5266</coordinates>
             </Point>
         </geometricProperty>
     </geoFeature>
  </center>

  <themes>
    <theme name="THEME_COUNTIES_3397829" />
    <theme name="THEME_US_ROAD1" />
    <theme name="THEME_US_AIRPORT" />
  </themes>

 <legend bgstyle="fill:#ffffff;fill-opacity:128;stroke:#ff0000;stroke-opacity:128"
profile="medium" font="Courier">
  <themes>
    <theme name="THEME_COUNTIES_3397829" />
    <theme name="THEME_US_AIRPORT" />
  </themes>
 </legend>

</map_request>
```

2.5 How a Map Is Generated

When a map request arrives at the map visualization component server, the server picks a free renderer associated with the data source in the request.

This topic describes the process that the map visualization component server follows to generate a map. In brief, the map visualization component performs the following steps:

1. Parse and process the incoming XML map request.
2. Prepare the data for each theme (executed in parallel).
3. Render and label each theme.
4. Generate final images or files.

Each map generated by the map visualization component results from its receiving a valid XML map request. The XML map request is parsed and its content is validated. The map visualization component then creates any dynamic styles specified in the XML request. It builds a theme list from all themes included in the base map (if a base map is specified), as well as any specified predefined or JDBC themes. All individual features in the request are grouped into a single temporary theme. In other words, after parsing the incoming request, all data that must be shown on the map is presented in a list of themes to the map visualization component rendering engine.

The ordering of the themes in the list is important, because it determines the order in which the themes are rendered. All themes included in the base map (when present) are added to the list first, followed by all specified themes (predefined or JDBC). The theme that contains all the

individual features is added as the last theme on the list. Any other requested features of a map (such as legend, map title, or footnote), are created and saved for rendering later.

For each theme in the request, the map visualization component then creates a separate execution thread to prepare its data, so that preparation of the themes takes place in parallel. For a predefined theme, this means formulating a query based on the theme's definition and any other information, such as the current map request window. This query is sent to the database for execution, and the result set is returned. The map visualization component creates individual renderable objects based on the result set.

- For predefined themes that are fully cached, no query is sent to the database, because all renderable objects are readily available.
- For JDBC themes, the query supplied by the user is either executed as is (when the `asis` attribute value is `TRUE` in the JDBC theme definition) or with a spatial filter subquery automatically applied to it. The spatial filter part is used to limit the results of the user's query to those within the current requested window.
- For themes that already have renderable features (such as the one containing all individual features in a request), there is no need to create renderable objects.

After all themes for the map request have been prepared and all necessary data has been collected, the map visualization component starts to render the map. It creates an empty new in-memory image to hold the result map, and paints the empty image with the necessary backgrounds (color or image). It then renders all of the themes in the theme list.

> **Note:**
>
> All image or GeoRaster themes are always rendered first, regardless of their position in the theme list. All other themes, however, are rendered in the order in which they appear in the theme list.

For each theme, features are rendered in an order determined internally by the map visualization component. The rendering of each feature involves invoking the drawing methods of its rendering style. After all themes have been rendered, the labeling process starts. For each theme whose features must be labeled with text, the map visualization component invokes algorithms to label each feature, with the specific algorithm depending on the type of feature (such as polygon or line).

After all themes have been rendered and (when needed) labeled, the map visualization component plots any additional map features (such as a legend) on the internal map image. The map visualization component then converts that image into the desired format (such as PNG or GIF) specified in the original map request; however, for SVG maps, instead of using an internal image, the map visualization component initially creates an empty SVG map object, then creates an SVG document as a result of the rendering process, and inserts it into the map object.

2.6 Cross-Schema Map Requests

A database user can issue a map request specifying a theme that uses data associated with another database user, to select data from tables that the other data source user is authorized to access.

For example, assume that user SCOTT wants to issue a map request using data associated with user MVDEMO. In general, user SCOTT must be granted SELECT access on relevant

tables owned by user MVDEMO, and the `<theme>` element should generally specify any tables in *schema-name.table-name* format. In this example scenario:

- For a geometry table, grant the SELECT privilege on the geometry table of MVDEMO to SCOTT (see Example 2-53).
- For a GeoRaster table, grant the SELECT privilege on the GeoRaster table and raster data table or tables of MVDEMO to SCOTT (see Example 2-54).
- For a topology data model table, grant the SELECT privilege on the topology table, topology column index table, and related topology information tables (*topology-name*_EDGE$, *topology-name*_NODE$, *topology-name*_FACE$, *topology-name*_RELATION$) of MVDEMO to SCOTT (see Example 2-55).
- For network data model tables, grant the SELECT privilege on the network link, node, path, and path-link tables of MVDEMO to SCOTT (see Example 2-56).

Example 2-53 shows a dynamic theme that accesses the MVDEMO.STATES geometry table from a data source defined on the SCOTT user.

Example 2-54 shows a dynamic theme that accesses the MVDEMO.GEORASTER_TABLE GeoRaster table and its RDT from a data source defined on the SCOTT user. Specify the base (GeoRaster) table in *schema-name.table-name* format.

Example 2-55 shows a dynamic theme that accesses the MVDEMO.LAND_PARCELS topology table and information tables for the CITY_DATA topology from a data source defined on the SCOTT user. Specify the feature table and the topology in *schema-name.object-name* format, if they are owned by a different schema than the one associated with the data source.

In Example 2-55, you must grant SELECT on the topology column index table name (*<topology-column-index-table-name>*) because the spatial index table associated with the feature table topology column is used by the map visualization component in topology queries. You can determine the topology column index table name as follows. Assume the following information:

- Topology feature table owner: MVDEMO
- Topology feature table name: LAND_PARCELS
- Topology feature table topology column name: FEATURE

The following query returns the index table name (in this example, MDTP_14E60$):

```
SQL> select sdo_index_table from all_sdo_index_info
     where table_owner = 'MVDEMO'
     and table_name = 'LAND_PARCELS'
     and column_name = 'FEATURE'

SDO_INDEX_TABLE
--------------------------------
MDTP_14E60$
```

Then, modify the last GRANT statement in Example 2-55 to specify the *<topology-column-index-table-name>*. In this case:

```
SQL> grant select on MDTP_14E60$ to SCOTT;
```

Example 2-56 shows a dynamic theme that accesses the MVDEMO.BI_TEST network and its link, node, path, and path-link tables. Specify the network name in *schema-name.network-name* format.

Example 2-53 Cross-Schema Access: Geometry Table

```
SQL> grant select on STATES to SCOTT;
. . .
<themes>
  <theme name="theme1">
     <jdbc_query
       datasource="scottds"
       spatial_column="geom"
       render_style="MVDEMO:C.COUNTIES"
       jdbc_srid="8265"
       >SELECT geom from MVDEMO.STATES</jdbc_query>
   </theme>
</themes>
```

Example 2-54 Cross-Schema Access: GeoRaster Table

```
SQL> grant select on GEORASTER_TABLE to SCOTT;
SQL> grant select on RDT_GEOR1 to SCOTT;
. . .
<themes>
  <theme name="georaster_theme">
    <jdbc_georaster_query
       georaster_table="MVDEMO.georaster_table"
       georaster_column="georaster"
       raster_table="rdt_geor1"
       raster_id="1"
       jdbc_srid="8307"
       datasource="scottds"
       asis="false">
     </jdbc_georaster_query>
  </theme>
</themes>
```

Example 2-55 Cross-Schema Access: Topology Feature Table

```
SQL> grant select on CITY_DATA_FACE$ to SCOTT;
SQL> grant select on CITY_DATA_EDGE$ to SCOTT;
SQL> grant select on CITY_DATA_NODE$ to SCOTT;
SQL> grant select on CITY_DATA_RELATION$ to SCOTT;
SQL> grant select on LAND_PARCELS to SCOTT;
SQL> grant select on <topology-column-index-table-name> to SCOTT;
. . .
<themes>
  <theme name="topo_theme" >
     <jdbc_topology_query
       topology_name="MVDEMO.CITY_DATA"
       feature_table="MVDEMO.LAND_PARCELS"
       spatial_column="FEATURE"
       render_style="MVDEMO:C.COUNTIES"
       jdbc_srid="0"
       datasource="scottds"
       asis="false">select feature from MVDEMO.land_parcels
     </jdbc_topology_query>
   </theme>
</themes>
```

Example 2-56 Cross-Schema Access: Network Tables

```
SQL> grant select on BI_TEST_LINK$ to SCOTT;
SQL> grant select on BI_TEST_NODE$ to SCOTT;
SQL> grant select on BI_TEST_PATH$ to SCOTT;
SQL> grant select on BI_TEST_PLINK$ to SCOTT;
```

```
. . .
<themes>
  <theme name="net_theme" >
     <jdbc_network_query
       network_name="MVDEMO.BI_TEST"
       network_level="1"
       jdbc_srid="0"
       datasource="scottds"
       link_style="MVDEMO:C.RED"
       node_style="MVDEMO:M.CIRCLE"
       node_markersize="5"
       asis="false">
     </jdbc_network_query>
   </theme>
</themes>
```

2.7 Workspace Manager Support in the Map Visualization Component

Workspace Manager is an Oracle Database feature that lets you version-enable one or more tables in the database.

After a table is version-enabled, users in a workspace automatically see the correct version of database rows in which they are interested. For detailed information about Workspace Manager, see *Oracle Database Workspace Manager Developer's Guide*.

You can request a map from a specific workspace, at a specific savepoint in a workspace, or at a point close to a specific date in a workspace. The following attributes of the `<theme>` element are related to support for Workspace Manager:

- `workspace_name` attribute: specifies the name of the workspace from which to get the map data.
- `workspace_savepoint` attribute: specifies the name of the savepoint to go to in the specified workspace.
- `workspace_date` attribute: specifies the date to go to (that is, a point at or near the specified date) in the specified workspace.
- `workspace_date_format` attribute: specifies the date format. The default is `mmddyyyyhh24miss`. This attribute applies only if you specified the `workspace_date` attribute.
- `workspace_date_nlsparam` attribute: specifies globalization support options. The options and default are the same as for the `nlsparam` argument to the `TO_CHAR` function for date conversion, which is described in *Oracle Database SQL Language Reference*.
- `workspace_date_tswtz` attribute: specifies a Boolean value. `TRUE` means that the input date is in timestamp with time zone format; `FALSE` (the default) means that the input date is a date string.

The `workspace_name` attribute is required for the use of Workspace Manager support in the map visualization component.

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

Example 2-57 shows the definition of a dynamic theme that uses attributes (shown in bold) related to Workspace Manager support. In this example, the map visualization component will render the data related to workspace `wsp_1` at the savepoint `sp1`.

The following considerations apply to the map visualization component caching of predefined themes (explained in Caching of Predefined Themes) and the use of Workspace Manager-related map visualization component attributes:

- The Workspace Manager-related attributes are ignored for predefined themes if the `caching` attribute is set to `ALL` in the `<styling_rules>` element for the theme.
- No caching data is considered if you specify the `workspace_name` attribute.

Example 2-57 Workspace Manager-Related Attributes in a Map Request

```
<?xml version="1.0" standalone="yes"?>
<map_request
 . . .
  <themes>
    <theme name="wmtheme" user_clickable="false"
       workspace_name="wsp_1" workspace_savepoint="sp1" >
       <jdbc_query
         spatial_column="GEOM"
         render_style="stylename"
         jdbc_srid="8307"
         datasource="mvdemo"
         asis="false"> select GEOM,ATTR from GEOM_TABLE
       </jdbc_query>
     </theme>
  </themes>
 . . .
</map_request>
```

2.8 Map Visualization Component 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 map visualization component user has the following views available in the schema associated with that user:

- USER_SDO_STYLES and ALL_SDO_STYLES contain information about styles. These views are described in xxx_SDO_STYLES Views.
- USER_SDO_THEMES and ALL_SDO_THEMES contain information about themes. These views are described in xxx_SDO_THEMES Views.
- USER_SDO_MAPS and ALL_SDO_MAPS contain information about base maps. These views are described in xxx_SDO_MAPS Views.
- USER_SDO_CACHED_MAPS and ALL_SDO_CACHED_MAPS contain information about configuration settings for map tile layers. These views are described in xxx_SDO_CACHED_MAPS Views.

> **Note:**
>
> You can use the Map Builder tool (described in #unique_26) to manage most mapping metadata. However, for some features you must use SQL statements to update the map visualization component metadata views.

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

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

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

The following rules apply for accessing the mapping metadata:

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

The preceding map visualization component metadata views are defined in the following file:

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

The map visualization component also uses some other metadata views, which may be defined in other files. You should never modify the contents of these views, which include the following:

- MDSYS.USER_SDO_TILE_ADMIN_TASKS includes information about long tasks related to map tile management. If you stop a long map tile layer task such as prefetching and then restart the task, the map visualization component uses the information in the USER_SDO_TILE_ADMIN_TASKS view to resume the task rather than start over at the beginning.
- xxx_SDO_STYLES Views
- xxx_SDO_THEMES Views
- xxx_SDO_MAPS Views
- xxx_SDO_CACHED_MAPS Views

2.8.1 xxx_SDO_STYLES Views

The USER_SDO_STYLES and ALL_SDO_STYLES views have the columns listed in Table 2-3.

Table 2-3 xxx_SDO_STYLES Views

Column Name	Data Type	Description
OWNER	VARCHAR2	Schema that owns the style (ALL_SDO_STYLES only)
NAME	VARCHAR2	Unique name to be associated with the style
TYPE	VARCHAR2	One of the following values: COLOR, MARKER, LINE, AREA, TEXT, or ADVANCED
DESCRIPTION	VARCHAR2	Optional descriptive text about the style
DEFINITION	CLOB	XML definition of the style
IMAGE	BLOB	Image content (for example, `airport.gif`) for marker or area styles that use image-based symbols (for markers) or fillers (for areas)
GEOMETRY	SDO_GEOMETRY	(Reserved for future use)

Depending on the Oracle Database release, the ALL_SDO_STYLES view may contain sample styles owned by the MDSYS schema. If these styles are defined on your system, you can specify them in theme definitions and map requests, and you can examine the XML definitions for ideas to use in defining your own styles.

To specify a style (or other type of map visualization component object) that is owned by a schema other than the one for the current user, you must specify the schema name, and you must use a colon (:), not a period, between the schema name and the object name. The following excerpt from a `<jdbc_query>` element refers to the style named `C.RED` owned by the MDSYS schema:

```
<jdbc_query . . . render_style="MDSYS:C.RED">
. . .
 </jdbc_query>
```

Example 2-58 finds the names of all currently defined styles owned by the MDSYS schema, and it displays the type, description, and XML definition of one of the styles. (The example output is reformatted for readability.)

Example 2-58 Finding Styles Owned by the MDSYS Schema

```
SELECT owner, name FROM all_sdo_styles
  WHERE owner = 'MDSYS';

OWNER                          NAME
------------------------------ --------------------------------
MDSYS                          C.BLACK
MDSYS                          C.BLACK GRAY
MDSYS                          C.BLUE
MDSYS                          C.COUNTIES
MDSYS                          C.FACILITY
. . .
MDSYS                          L.MAJOR STREET
MDSYS                          L.MAJOR TOLL ROAD
MDSYS                          L.MQ_ROAD2
MDSYS                          L.PH
MDSYS                          L.POOR_ROADS
MDSYS                          L.PTH
MDSYS                          L.RAILROAD
MDSYS                          L.RAMP
MDSYS                          L.SH
```

```
MDSYS                          L.STATE BOUNDARY
. . .
MDSYS                          M.REDSQ
MDSYS                          M.SMALL TRIANGLE
MDSYS                          M.STAR
MDSYS                          M.TOWN HALL
MDSYS                          M.TRIANGLE
MDSYS                          T.AIRPORT NAME
MDSYS                          T.CITY NAME
MDSYS                          T.MAP TITLE
MDSYS                          T.PARK NAME
MDSYS                          T.RED STREET
MDSYS                          T.ROAD NAME
MDSYS                          T.SHIELD1
MDSYS                          T.SHIELD2
MDSYS                          T.STATE NAME
MDSYS                          T.STREET NAME
. . .

-- Display the type, description, and XML definition of one style.
SET LONG 4000;
SELECT owner, name, type, description, definition
  FROM all_sdo_styles WHERE name = 'L.PH';

OWNER   NAME     TYPE       DESCRIPTION
------  -----    ------     -----------------
MDSYS   L.PH     LINE       Primary highways

DEFINITION
--------------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<desc></desc>
<g class="line" style="fill:#33a9ff;stroke-width:4">
<line class="parallel" style="fill:#aa55cc;stroke-width:1.0"/>
</g>
</svg>
```

2.8.2 xxx_SDO_THEMES Views

The USER_SDO_THEMES and ALL_SDO_THEMES views have the columns listed in Table 2-4.

Table 2-4 xxx_SDO_THEMES Views

Column Name	Data Type	Description
OWNER	VARCHAR2	Schema that owns the theme (ALL_SDO_THEMES only)
NAME	VARCHAR2	Unique name to be associated with the theme
DESCRIPTION	VARCHAR2	Optional descriptive text about the theme
BASE_TABLE	VARCHAR2	Table or view containing the spatial geometry column
GEOMETRY_COLUMN	VARCHAR2	Name of the spatial geometry column (of type SDO_GEOMETRY)
STYLING_RULES	CLOB	XML definition of the styling rules to be associated with the theme

2.8.3 xxx_SDO_MAPS Views

The USER_SDO_MAPS and ALL_SDO_MAPS views have the columns listed in Table 2-5.

Table 2-5 xxx_SDO_MAPS Views

Column Name	Data Type	Description
OWNER	VARCHAR2	Schema that owns the base map (ALL_SDO_MAPS only)
NAME	VARCHAR2	Unique name to be associated with the base map
DESCRIPTION	VARCHAR2	Optional descriptive text about the base map
DEFINITION	CLOB	XML definition of the list of themes and their scale value range information to be associated with the base map

2.8.4 xxx_SDO_CACHED_MAPS Views

The USER_SDO_MAPS and ALL_SDO_MAPS views have the columns listed in Table 2-6.

Table 2-6 xxx_SDO_CACHED_MAPS Views

Column Name	Data Type	Description
NAME	VARCHAR2	Unique name of the cached map source
DESCRIPTION	VARCHAR2	Optional descriptive text about the cached map source
TILES_TABLE	VARCHAR2	(Not currently used)
IS_ONLINE	VARCHAR2	`YES` if the map tile layer is online, or `NO` if the map tile layer is offline. When a tile is missing from the cache and the map tile layer is online, the map tile server will fetch the tile and return the fetched tile to the client. When a tile is missing and the map tile layer is offline, the map tile server will not fetch the tile but will return a blank image to the client.
IS_INTERNAL	VARCHAR2	`YES` if the map source is an internal map source, or `NO` if the map source is an external map source
DEFINITION	CLOB	XML definition of the map tile layer, as described later in this section.
BASE_MAP	VARCHAR2	Name of the cached map visualization component base map, if the map source is an internal map source
MAP_ADAPTER	BLOB	The jar file that contains the adapter Java classes of the external map services provider, as described later in this section.

For detailed information about using the USER_SDO_CACHED_MAPS view, see Map Tile Server Configuration.

2.9 Oracle Maps

Oracle Maps is the name for a suite of technologies for developing high-performance interactive web-based mapping applications. It consists of components from both the server side and the client side.

- Overview of Oracle Maps
- Architecture for Oracle Maps Applications

2.9.1 Overview of Oracle Maps

Oracle Maps consists of the following main components:

- A map tile server that caches and serves pregenerated map image tiles upon a map image tile request
- A map server that generates maps form spatial data to the mapping client upon a client map image request
- A map data server that fetches spatial data from a spatial data provider to the mapping client
- An Ajax-based JavaScript mapping client. (Ajax is an acronym for asynchronous JavaScript and XML.) This client provides functions for browsing and interacting with maps, as well as a flexible application programming interface (API).

The map tile server (map image caching engine, described in Map Tile Server) automatically fetches and caches map image tiles rendered by the Oracle Spatial map visualization component or other web-enabled map providers. It also serves cached map image tiles to the clients, which are web applications developed using the Oracle Maps client API. The clients can then automatically stitch multiple map image tiles into a seamless large map. Because the map image tiles are normally pregenerated and cached, the application users will experience fast map viewing performance.

The JavaScript mapping client is a browser side map display engine that fetches map content from the servers, and presents it to client applications. It:

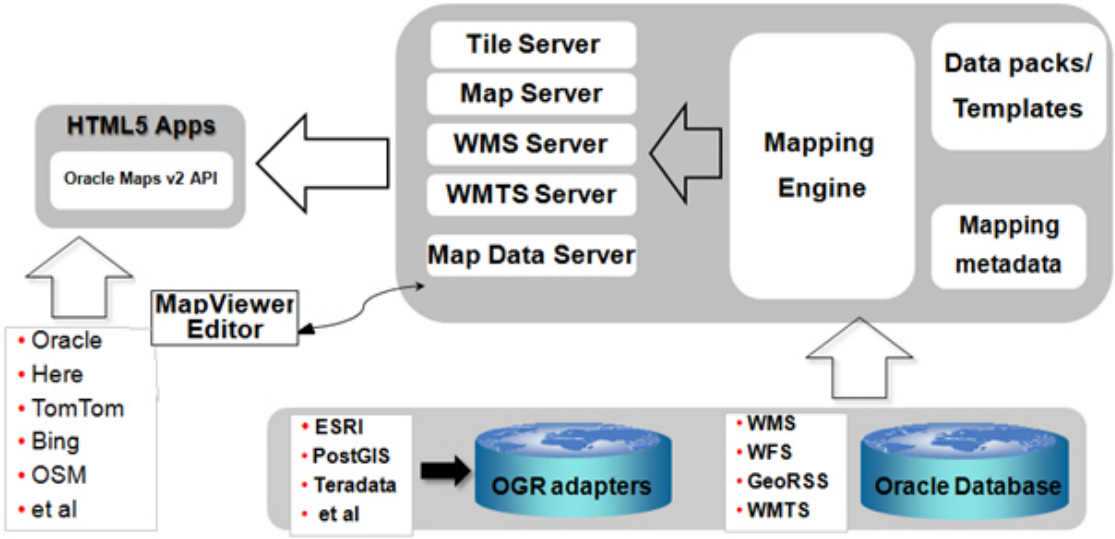
- Fetches map images tiles from a map visualization component server, from a third party map service provider, or from its local disk-storage
- Retrieves spatial data from a map visualization component server, from a third party spatial data provider, or from its local disk-storage and then renders the spatial data into map images by itself on the client side
- Provides customizable map-related user interaction controls, such as map dragging and clicking, for the application
- Provides other built-in features and utilities to customize the map contents and map layout design

The JavaScript mapping client can be easily integrated with any web application or portal.

2.9.2 Architecture for Oracle Maps Applications

Figure 2-15 shows the architecture of web mapping applications that are developed using Oracle Maps.

Figure 2-15 Architecture for Oracle Maps Applications



Referring to Figure 2-15, applications interact with the Oracle Maps architecture as follows:

- The application is developed using JavaScript, and it runs inside the JavaScript engine of the web browser.
- The application invokes the JavaScript map client to fetch the map image tiles from the map tile server, and then it displays the map in the web browser.
- The application invokes the JavaScript map client to fetch the map image in other form of map services (such as WMS and WMTS) from the server and then displays the map image.
- The application invokes the JavaScript map client to fetch spatial data from the map data server and then renders the map image to display.
- The JavaScript map client controls map-related user interaction for the application.
- When the map visualization component server receives a map image request, the request can be one of its several supported types: a map image tile request to its map tile server, a map image request to its map server, a WMS map request to its WMS server, or a WMTS map image tile request to its WMTS server. Each request will be handled by the server accordingly.

  Example 1: When the map tile server receives a map image tile request, it first checks to see if the requested tile is already cached. If the tile is already in the cache, the cached tile is then returned to the client. If the tile is not in the cache, then the map tile server fetches the tile into the cache and returns it to the client. Tiles can be fetched either directly from the map visualization component map rendering engine or from an external web map service provider.

  Example 2: When the server receives a map image request (instead of a map tile request) to its map server, the map server renders the image and returns it to the client. Note that in contrast to the map tile server, the map server generates a map according to a specified data bounds and renders the data onto an image with specified dimension, but it does not cache the map image.
- When the map data server receives a map data request, it retrieves the spatial data and returns the spatial data, not map images, to the client.

3
Map Visualization Servers

The map visualization component, as a set of Java Enterprise Edition packages, contains a collection of servers to provide mapping services.

These services include a map server, a map data server, a map tile server, a vector tile server, a WMS server (see OGC WMS Support in the Map Visualization Component), and a WMTS server (see OGC WMTS Support in the Map Visualization Component).When you develop Oracle Maps applications, the Oracle Maps API and the map visualization component server will identify the needed service and then issue map service request to the proper server. You normally do not need to send service requests explicitly in your map applications; however, if you are familiar with the map visualization component servers described in this chapter, it can help if you need to debug or optimize the application.

- Map Visualization Component Map Data Server
  The map visualization component map data server provides services for streaming live data from a database server to a client.
- Map Tile Server
  The map tile server is a map image caching engine that fetches, caches, and serves pregenerated, fixed-size map image tiles.
- Vector Tile Server
  The vector tile server is a vector tile caching engine that fetches, caches, and serves pregenerated vector tiles.

3.1 Map Visualization Component Map Data Server

The map visualization component map data server provides services for streaming live data from a database server to a client.

The data can be consumed by the Oracle Maps JavaScript API client, or be edited by the API editing utilities. It is also used as the middle-tier component for handling data synchronization tasks.

At the most basic level, a client sends a request to the map data server, specifying the name of a theme and an optional bounding box. The server then returns the live data (including both geometries and attributes) in compressed GeoJSON format.

- Domains and Map Data Server URL Patterns
- Map Data Server Request Parameters
- Interpreting Data Returned from the Map Data Server
- Map Data Server Error Handling

3.1.1 Domains and Map Data Server URL Patterns

The map data server uses domains, where each **domain** corresponds to a map visualization component data source. For example, to request data from the `mvdemo` data source, the URL must have the following pattern:

```
http://example.com:8080/mapviewer/dataserver/mvdemo?
```

In this URL request, `/dataserver` refers to the map data server. `/mvdemo` is the path information, which indicates which domain or data source this request is directed against. There is no need to specify the data source explicitly in the rest of the URL request. This URL pattern provides flexibility in terms of protection. A user or a web administrator can easily set up different levels of protection for different domains using such a URL pattern.

To get a quick help regarding the full list of supported HTTP request parameters, you can issue a request that includes the help parameter. For example:

```
http://example:8080/mapviewer/dataserver/mvdemo?help=true
```

This example returns a list of supported parameters. (Note that the `/mvdemo` path is still required even for this help request.)

3.1.2 Map Data Server Request Parameters

In a map data server request to get data, the URL must include appropriate query parameters. The data could be from a map visualization component predefined geometry theme or JDBC theme.

- Getting Data from a Predefined Geometry Theme
- Getting Data from a JDBC Theme
- Getting Annotation Text from a JDBC Theme
- Getting Topology Data

3.1.2.1 Getting Data from a Predefined Geometry Theme

Before getting data from a predefined geometry theme, be sure that the predefined geometry theme is properly defined. If it is not properly defined, modify its definition or create a new predefined geometry theme with the proper definition.

Besides the geometry column of a spatial table, if you also need some attribute columns for an application, then your geometry theme's STYLING_RULES column must have the `<hidden_info>` element for defining needed attributes. For example, if your application needs a geometry theme (such as CUSTOMERS) to represent a spatial table, and if you also need each customer's name, city, and sales amount, then the STYLING_RULES should look similar to the following:

```
SQL> select STYLING_RULES from user_sdo_themes where name='CUSTOMERS';

STYLING_RULES
--------------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="name" name="Name"/>
    <field column="city" name="City"/>
    <field column="sales" name="Sales"/>
  </hidden_info>
  <rule>
    <features style="M.STOPLIGHT_RED"> </features>
  </rule>
</styling_rules>
```

The Map Builder utility is the recommended tool for modifying or creating geometry themes.

For information using styling rules in a predefined geometry theme, see Styling Rules in Predefined Spatial Geometry Themes.

The following parameters are available for a map data server request to get data from a map visualization component predefined geometry theme. The `t` (theme name) parameter is required; the others are optional.

- `t`: Name of the theme.
- `bbox`: Bounding box. Must be a comma-delimited list of `minx,miny,maxx,maxy`.
- `to_srid`: SRID (spatial reference system) in which to return the data.
- `bbox_srid`: SRID (spatial reference system) of the bounding box, if different from the native SRID of the data.
- `seq`: Sequence ID, to be used when getting data in multipart format.
- `dadp`: Digits after decimal point: the maximum number of digits after the decimal point for the coordinates in the returned data.
- `include_style_info`: Determines whether styling information (rendering/labeling style name and related columns) should be included with each feature. (The default is `true`.)
- `include_label_box`: Determines whether a label box should be included with each polygon feature. (The default is `true`.) A label box is a near maximum rectangle inside the polygon. A label placed inside this rectangle is guaranteed to be completely inside the polygon feature. This parameter is ignored for non-polygon features.
- `simplify`: Determines whether geometries will be simplified. (The default is `false`.)
- `threshold`: If `simplify` is `true`, then `threshold` controls the approximate percentage of vertices from the input geometry to be eliminated during the simplification. (Must be a number between 1 and 99.) For example, if the value is 10, then relatively few (approximately 10%) of the vertices will be eliminated and most (approximately 90%) will be kept; but if the value is 90, then approximately 90% of the vertices will be eliminated (and only approximately 10% will be kept).

The following are some examples.

To get all the data from the CITIES table in the 3857 SRID:

```
http://example:8080/mapviewer/dataserver/mvdemo?t=theme_demo_cities&to_srid=3857
```

To get all the data from the CITIES table that interacts with a specified bbox in 3857 SRID:

```
http://example:8080/mapviewer/dataserver/mvdemo?
t=theme_demo_cities&bbox=-122.0,25,-100,45&to_srid=3857
```

To get all the data from the CITIES table that interacts with a specified bbox in 3857 SRID, with 3 digits after the decimal points, and without styling information:

```
http://example:8080/mapviewer/dataserver/mvdemo?
t=theme_demo_cities&bbox=-122.0,25,-100,45&to_srid=3857&dadp=3&include_style_info
=no
```

To get all data from the COUNTIES table and include label boxes:

```
http://example:8080/mapviewer/dataserver/mvdemo?
t=theme_demo_counties&include_label_box=yes
```

3.1.2.2 Getting Data from a JDBC Theme

The following parameters are available for a map data server request to get data from a map visualization component theme based on a dynamic JDBC theme. The `t` (theme name) and `sql` (SQL query) parameters are required; the others are optional.

- `t`: Name of the theme.
- `sql`: The complete SQL query, properly URL encoded.
- `asis`: Determines whether the query should be executed "as is". The default is `false`, which causes the map visualization component to embed the SQL query as a subquery of its spatial filter query. If the value is `true`, the map visualization component does not attempt to modify the supplied query string.
- `bbox`: Bounding box. Must be a comma-delimited list of `minx,miny,maxx,maxy`.
- `to_srid`: SRID (spatial reference system) in which to return the data.
- `bbox_srid`: SRID (spatial reference system) of the bounding box, if different from the native SRID of the data.
- `seq`: Sequence ID, to be used when getting data in multipart format.
- `dadp`: Digits after decimal point: the maximum number of digits after the decimal point for the coordinates in the returned data.
- `include_style_info`: Determines whether styling information (rendering/labeling style name and related columns) should be included with each feature. (The default is `true`.)
- `include_label_box`: Determines whether a label box should be included with each polygon feature. (The default is `true`.) A label box is a near maximum rectangle inside the polygon. A label placed inside this rectangle is guaranteed to be completely inside the polygon feature. This parameter is ignored for non-polygon features.

The following are some examples.

To get all the data of the CITIES table in SRID 3857:

```
http://example:8080/mapviewer/dataserver/mvdemo?
t=theme1&sql=select+*+from+cities&to_srid=3857
```

To get all the data of the table CITIES within a given bbox in SRID 3857:

```
http://example:8080/mapviewer/dataserver/mvdemo?
t=theme1&sql=select+*+from+cities&to_srid=3857&bbox=-122.0,25,-100,45
```

To run the query as is, without a bbox and return data in SRID 3857:

```
http://example:8080/mapviewer/dataserver/mvdemo?
t=theme1&sql=select+*+from+cities&to_srid=3857&bbox=-122.0,25,-100,45&asis=t
```

3.1.2.3 Getting Annotation Text from a JDBC Theme

A request to retrieve annotation text elements from a JDBC theme is similar to that in Getting Data from a JDBC Theme

The following parameters are available for a map data server request to get data from a map visualization component theme based on a dynamic JDBC theme. The `t` (theme name), `sql` (SQL query, `geom_type`, and `base_table` parameters are required; the others are optional.

- `t`: Name of the theme.
- `sql`: The complete SQL query, properly URL encoded.
- `geom_type`: Must be specified as `annotation` to indicate that the spatial column is on annotation type.
- `base_table`: The database table (for the server to read annotation text metadata information for that table).

- `asis`: Determines whether the query should be executed "as is". The default is `false`, which causes the map visualization component to embed the SQL query as a subquery of its spatial filter query. If the value is `true`, the map visualization component does not attempt to modify the supplied query string.
- `bbox`: Bounding box. Must be a comma-delimited list of `minx,miny,maxx,maxy`.
- `to_srid`: SRID (spatial reference system) in which to return the data.
- `bbox_srid`: SRID (spatial reference system) of the bounding box, if different from the native SRID of the data.
- `seq`: Sequence ID, to be used when getting data in multipart format.
- `dadp`: Digits after decimal point: the maximum number of digits after the decimal point for the coordinates in the returned data.
- `include_style_info`: Determines whether styling information (rendering/labeling style name and related columns) should be included with each feature. (The default is `true`.)
- `include_label_box`: Determines whether a label box should be included with each polygon feature. (The default is `true`.) A label box is a near maximum rectangle inside the polygon. A label placed inside this rectangle is guaranteed to be completely inside the polygon feature. This parameter is ignored for non-polygon features.

The following example retrieves annotation text information:

```
http://example:8080/mapviewer/dataserver/tilsmenv?
t=theme1&sql=select+*+from+annotext_table&geom_col=textobj&geom_type=annotation&b
ase_table=annotext_table&bbox=0,0,10,10
```

A typical response includes the annotation text table metadata information plus the annotation text feature. Each annotation text feature can have one or more text elements. Each text element can be defined by a text value, a location, a leader line, and graphic attributes. Refer to OGC specification of annotation texts for additional information.

The response looks like the following:

```
{"type":"AnnotationText",
"collectionName":"theme1",
"srs":0,
"geodetic":false,
"bbox":[0, 0, 10, 10],
"attr_names":["ID"],
"attr_types":["double"],
"default_text_attributes":
{"fontWeight":"Normal","fontStyle":"Normal","textDecoration":"None","letterSpacing":"Norm
al","wordSpacing":"Normal","fill":"black","fill-
opacity":1.0,"stroke":"black","strokeWidth":1.0,"stroke-
opacity":1.0,"horizontalAlignment":"start","verticalAlignment":"top","multilineJustificat
ion":"left","multilineSpacing":0.0},
"metadata":{"textExpression":"name","textAttributes":
{"fontFamily":"Serif","fontSize":14.0,"fill":"#ff0000"}},
"features":[
{"type":"AnnoText", "elements":[{"location":{"type":"Point", "coordinates":[1,
1]},"textValue":"Sample Label 1","leaderLine":{"type":"LineString", "coordinates":
[0,0,1,1]}}],"envelope":{"type":"Rectangle", "coordinates":[0,0,1,1]},"properties":
{"ID":"1.0"}},
{"type":"AnnoText", "elements":[{"location":{"type":"LineString", "coordinates":
[2,5,4,5,6,5]},"leaderLine":{"type":"LineString", "coordinates":
[4,3,4,5]},"textAttributes":
{"fontFamily":"Dialog","fontSize":14.0,"fill":"blue"}}],"envelope":{"type":"Rectangle",
"coordinates":[2,3,6,5]},"properties":{"ID":"3.0"}},
```

```
{"type":"AnnoText", "elements":[{"location":{"type":"Point", "coordinates":[10,
10]},"textValue":"Sample Label 2","leaderLine":{"type":"LineString", "coordinates":
[5,10,10,10]}}],"envelope":{"type":"LineString", "coordinates":
[5,10,10,10]},"properties":{"ID":"2.0"}}
]}
```


3.1.2.4 Getting Topology Data

A topology set is defined by a set of topology primitives (faces, edges, and nodes(. Each topology feature can be associated with one or more topology primitives.

A request to retrieve the topology primitives must contain the `topology` parameter, as in the following example:

```
http://example:8080/mapviewer/dataserver/tilsmenv?
topology=city_data&bbox=10,10,35,35
```

The response includes all primitives that interact with input MBR:


```
{"type":"TopologyPrimitives",
"topology":"city_data",
"srs":0,
"bbox":[0, 0, 62, 42],
"face_attr_names":
["FACE_ID","BOUNDARY_EDGE_ID","ISLAND_EDGE_ID_LIST","ISLAND_NODE_ID_LIST"],
"face_attr_types":["integer","integer","array:integer","array:integer"],
"edge_attr_names":
["EDGE_ID","START_NODE_ID","END_NODE_ID","NEXT_LEFT_EDGE_ID","PREV_LEFT_EDGE_ID","NEXT_RI
GHT_EDGE_ID","PREV_RIGHT_EDGE_ID","LEFT_FACE_ID","RIGHT_FACE_ID"],
"edge_attr_types":
["integer","integer","integer","integer","integer","integer","integer","integer","integer
"],
"node_attr_names":["NODE_ID","EDGE_ID","FACE_ID"],
"node_attr_types":["integer","integer","integer"],
"primitives":[
{"type":"Face", "mbr_geometry":{"type":"Rectangle", "coordinates":[3,30,15,38]},
"properties":{"FACE_ID":"1", "BOUNDARY_EDGE_ID":"1", "ISLAND_EDGE_ID_LIST":[25]}},
{"type":"Face", "mbr_geometry":{"type":"Rectangle", "coordinates":[9,14,21,22]},
"properties":{"FACE_ID":"3", "BOUNDARY_EDGE_ID":"19"}},
{"type":"Face", "mbr_geometry":{"type":"Rectangle", "coordinates":[9,6,21,14]},
"properties":{"FACE_ID":"6", "BOUNDARY_EDGE_ID":"20"}},
{"type":"Face", "mbr_geometry":{"type":"Rectangle", "coordinates":[17,30,31,40]},
"properties":{"FACE_ID":"2", "BOUNDARY_EDGE_ID":"2", "ISLAND_NODE_ID_LIST":[4]}},
{"type":"Face", "mbr_geometry":{"type":"Rectangle", "coordinates":[21,6,35,14]},
"properties":{"FACE_ID":"7", "BOUNDARY_EDGE_ID":"10"}},
{"type":"Face", "mbr_geometry":{"type":"Rectangle", "coordinates":[21,14,35,22]},
"properties":{"FACE_ID":"4", "BOUNDARY_EDGE_ID":"17"}},
{"type":"Face", "mbr_geometry":{"type":"Rectangle", "coordinates":[35,14,47,22]},
"properties":{"FACE_ID":"5", "BOUNDARY_EDGE_ID":"15"}},
{"type":"Face", "mbr_geometry":{"type":"Rectangle", "coordinates":[35,6,47,14]},
"properties":{"FACE_ID":"8", "BOUNDARY_EDGE_ID":"16"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":
[8,30,16,30,16,38,3,38,3,30,8,30]}, "properties":{"EDGE_ID":"1", "START_NODE_ID":"1",
"END_NODE_ID":"1", "NEXT_LEFT_EDGE_ID":"1", "PREV_LEFT_EDGE_ID":"1",
"NEXT_RIGHT_EDGE_ID":"-1", "PREV_RIGHT_EDGE_ID":"-1", "LEFT_FACE_ID":"1",
"RIGHT_FACE_ID":"-1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":
[4,31,7,31,7,34,4,34,4,31]}, "properties":{"EDGE_ID":"26", "START_NODE_ID":"20",
"END_NODE_ID":"20", "NEXT_LEFT_EDGE_ID":"26", "PREV_LEFT_EDGE_ID":"26",
"NEXT_RIGHT_EDGE_ID":"-26", "PREV_RIGHT_EDGE_ID":"-26", "LEFT_FACE_ID":"9",
"RIGHT_FACE_ID":"1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[9,22,21,22]},
```

```
"properties":{"EDGE_ID":"6", "START_NODE_ID":"16", "END_NODE_ID":"17",
"NEXT_LEFT_EDGE_ID":"7", "PREV_LEFT_EDGE_ID":"21", "NEXT_RIGHT_EDGE_ID":"-21",
"PREV_RIGHT_EDGE_ID":"19", "LEFT_FACE_ID":"-1", "RIGHT_FACE_ID":"3"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[9,14,9,22]},
"properties":{"EDGE_ID":"21", "START_NODE_ID":"15", "END_NODE_ID":"16",
"NEXT_LEFT_EDGE_ID":"6", "PREV_LEFT_EDGE_ID":"22", "NEXT_RIGHT_EDGE_ID":"9",
"PREV_RIGHT_EDGE_ID":"-6", "LEFT_FACE_ID":"-1", "RIGHT_FACE_ID":"3"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[9,14,21,14]},
"properties":{"EDGE_ID":"9", "START_NODE_ID":"15", "END_NODE_ID":"14",
"NEXT_LEFT_EDGE_ID":"19", "PREV_LEFT_EDGE_ID":"-21", "NEXT_RIGHT_EDGE_ID":"-22",
"PREV_RIGHT_EDGE_ID":"20", "LEFT_FACE_ID":"3", "RIGHT_FACE_ID":"6"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[9,6,21,6]}, "properties":
{"EDGE_ID":"12", "START_NODE_ID":"8", "END_NODE_ID":"9", "NEXT_LEFT_EDGE_ID":"20",
"PREV_LEFT_EDGE_ID":"-22", "NEXT_RIGHT_EDGE_ID":"22", "PREV_RIGHT_EDGE_ID":"-13",
"LEFT_FACE_ID":"6", "RIGHT_FACE_ID":"-1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[9,35,13,35]},
"properties":{"EDGE_ID":"25", "START_NODE_ID":"21", "END_NODE_ID":"22",
"NEXT_LEFT_EDGE_ID":"-25", "PREV_LEFT_EDGE_ID":"-25", "NEXT_RIGHT_EDGE_ID":"25",
"PREV_RIGHT_EDGE_ID":"25", "LEFT_FACE_ID":"1", "RIGHT_FACE_ID":"1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[9,6,9,14]}, "properties":
{"EDGE_ID":"22", "START_NODE_ID":"8", "END_NODE_ID":"15", "NEXT_LEFT_EDGE_ID":"21",
"PREV_LEFT_EDGE_ID":"-12", "NEXT_RIGHT_EDGE_ID":"12", "PREV_RIGHT_EDGE_ID":"-9",
"LEFT_FACE_ID":"-1", "RIGHT_FACE_ID":"6"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":
[25,30,31,30,31,40,17,40,17,30,25,30]}, "properties":{"EDGE_ID":"2",
"START_NODE_ID":"2", "END_NODE_ID":"2", "NEXT_LEFT_EDGE_ID":"3",
"PREV_LEFT_EDGE_ID":"-3", "NEXT_RIGHT_EDGE_ID":"-2", "PREV_RIGHT_EDGE_ID":"-2",
"LEFT_FACE_ID":"2", "RIGHT_FACE_ID":"-1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[21,6,35,6]},
"properties":{"EDGE_ID":"13", "START_NODE_ID":"9", "END_NODE_ID":"10",
"NEXT_LEFT_EDGE_ID":"18", "PREV_LEFT_EDGE_ID":"-20", "NEXT_RIGHT_EDGE_ID":"-12",
"PREV_RIGHT_EDGE_ID":"-14", "LEFT_FACE_ID":"7", "RIGHT_FACE_ID":"-1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[21,22,35,22]},
"properties":{"EDGE_ID":"7", "START_NODE_ID":"17", "END_NODE_ID":"18",
"NEXT_LEFT_EDGE_ID":"8", "PREV_LEFT_EDGE_ID":"6", "NEXT_RIGHT_EDGE_ID":"-19",
"PREV_RIGHT_EDGE_ID":"17", "LEFT_FACE_ID":"-1", "RIGHT_FACE_ID":"4"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[21,6,21,14]},
"properties":{"EDGE_ID":"20", "START_NODE_ID":"9", "END_NODE_ID":"14",
"NEXT_LEFT_EDGE_ID":"-9", "PREV_LEFT_EDGE_ID":"12", "NEXT_RIGHT_EDGE_ID":"13",
"PREV_RIGHT_EDGE_ID":"10", "LEFT_FACE_ID":"6", "RIGHT_FACE_ID":"7"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[35,14,21,14]},
"properties":{"EDGE_ID":"10", "START_NODE_ID":"13", "END_NODE_ID":"14",
"NEXT_LEFT_EDGE_ID":"-20", "PREV_LEFT_EDGE_ID":"18", "NEXT_RIGHT_EDGE_ID":"17",
"PREV_RIGHT_EDGE_ID":"-19", "LEFT_FACE_ID":"7", "RIGHT_FACE_ID":"4"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[21,14,21,22]},
"properties":{"EDGE_ID":"19", "START_NODE_ID":"14", "END_NODE_ID":"17",
"NEXT_LEFT_EDGE_ID":"-6", "PREV_LEFT_EDGE_ID":"9", "NEXT_RIGHT_EDGE_ID":"-10",
"PREV_RIGHT_EDGE_ID":"-7", "LEFT_FACE_ID":"3", "RIGHT_FACE_ID":"4"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[25,30,25,35]},
"properties":{"EDGE_ID":"3", "START_NODE_ID":"2", "END_NODE_ID":"3",
"NEXT_LEFT_EDGE_ID":"-3", "PREV_LEFT_EDGE_ID":"2", "NEXT_RIGHT_EDGE_ID":"2",
"PREV_RIGHT_EDGE_ID":"3", "LEFT_FACE_ID":"2", "RIGHT_FACE_ID":"2"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[35,6,47,6]},
"properties":{"EDGE_ID":"14", "START_NODE_ID":"10", "END_NODE_ID":"11",
"NEXT_LEFT_EDGE_ID":"16", "PREV_LEFT_EDGE_ID":"-18", "NEXT_RIGHT_EDGE_ID":"-13",
"PREV_RIGHT_EDGE_ID":"-16", "LEFT_FACE_ID":"8", "RIGHT_FACE_ID":"-1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[35,14,47,14]},
"properties":{"EDGE_ID":"11", "START_NODE_ID":"13", "END_NODE_ID":"12",
"NEXT_LEFT_EDGE_ID":"15", "PREV_LEFT_EDGE_ID":"-17", "NEXT_RIGHT_EDGE_ID":"-18",
"PREV_RIGHT_EDGE_ID":"16", "LEFT_FACE_ID":"5", "RIGHT_FACE_ID":"8"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[35,6,35,14]},
"properties":{"EDGE_ID":"18", "START_NODE_ID":"10", "END_NODE_ID":"13",
```

"NEXT_LEFT_EDGE_ID":"10", "PREV_LEFT_EDGE_ID":"13", "NEXT_RIGHT_EDGE_ID":"14",
"PREV_RIGHT_EDGE_ID":"-11", "LEFT_FACE_ID":"7", "RIGHT_FACE_ID":"8"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[35,22,47,22]},
"properties":{"EDGE_ID":"8", "START_NODE_ID":"18", "END_NODE_ID":"19",
"NEXT_LEFT_EDGE_ID":"-15", "PREV_LEFT_EDGE_ID":"7", "NEXT_RIGHT_EDGE_ID":"-17",
"PREV_RIGHT_EDGE_ID":"15", "LEFT_FACE_ID":"-1", "RIGHT_FACE_ID":"5"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[35,14,35,22]},
"properties":{"EDGE_ID":"17", "START_NODE_ID":"13", "END_NODE_ID":"18",
"NEXT_LEFT_EDGE_ID":"-7", "PREV_LEFT_EDGE_ID":"-10", "NEXT_RIGHT_EDGE_ID":"11",
"PREV_RIGHT_EDGE_ID":"-8", "LEFT_FACE_ID":"4", "RIGHT_FACE_ID":"5"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":
[36,38,38,35,41,34,42,33,45,32,47,28,50,28,52,32,57,33]}, "properties":{"EDGE_ID":"4",
"START_NODE_ID":"5", "END_NODE_ID":"6", "NEXT_LEFT_EDGE_ID":"-5",
"PREV_LEFT_EDGE_ID":"-4", "NEXT_RIGHT_EDGE_ID":"4", "PREV_RIGHT_EDGE_ID":"5",
"LEFT_FACE_ID":"-1", "RIGHT_FACE_ID":"-1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":
[41,40,45,40,47,42,62,41,61,38,59,39,57,36,57,33]}, "properties":{"EDGE_ID":"5",
"START_NODE_ID":"7", "END_NODE_ID":"6", "NEXT_LEFT_EDGE_ID":"-4",
"PREV_LEFT_EDGE_ID":"-5", "NEXT_RIGHT_EDGE_ID":"5", "PREV_RIGHT_EDGE_ID":"4",
"LEFT_FACE_ID":"-1", "RIGHT_FACE_ID":"-1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[47,14,47,22]},
"properties":{"EDGE_ID":"15", "START_NODE_ID":"12", "END_NODE_ID":"19",
"NEXT_LEFT_EDGE_ID":"-8", "PREV_LEFT_EDGE_ID":"11", "NEXT_RIGHT_EDGE_ID":"-16",
"PREV_RIGHT_EDGE_ID":"8", "LEFT_FACE_ID":"5", "RIGHT_FACE_ID":"-1"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[47,6,47,14]},
"properties":{"EDGE_ID":"16", "START_NODE_ID":"11", "END_NODE_ID":"12",
"NEXT_LEFT_EDGE_ID":"-11", "PREV_LEFT_EDGE_ID":"14", "NEXT_RIGHT_EDGE_ID":"-14",
"PREV_RIGHT_EDGE_ID":"-15", "LEFT_FACE_ID":"8", "RIGHT_FACE_ID":"-1"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[4, 31]}, "properties":
{"NODE_ID":"20", "EDGE_ID":"26", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[8, 30]}, "properties":
{"NODE_ID":"1", "EDGE_ID":"1", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[9, 6]}, "properties":
{"NODE_ID":"8", "EDGE_ID":"12", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[9, 35]}, "properties":
{"NODE_ID":"21", "EDGE_ID":"25", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[9, 14]}, "properties":
{"NODE_ID":"15", "EDGE_ID":"21", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[9, 22]}, "properties":
{"NODE_ID":"16", "EDGE_ID":"6", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[13, 35]}, "properties":
{"NODE_ID":"22", "EDGE_ID":"-25", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[20, 37]}, "properties":
{"NODE_ID":"4", "EDGE_ID":"0", "FACE_ID":"2"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[21, 14]}, "properties":
{"NODE_ID":"14", "EDGE_ID":"19", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[21, 22]}, "properties":
{"NODE_ID":"17", "EDGE_ID":"7", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[21, 6]}, "properties":
{"NODE_ID":"9", "EDGE_ID":"20", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[25, 30]}, "properties":
{"NODE_ID":"2", "EDGE_ID":"2", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[25, 35]}, "properties":
{"NODE_ID":"3", "EDGE_ID":"-3", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[35, 14]}, "properties":
{"NODE_ID":"13", "EDGE_ID":"17", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[35, 6]}, "properties":
{"NODE_ID":"10", "EDGE_ID":"18", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[35, 22]}, "properties":
{"NODE_ID":"18", "EDGE_ID":"8", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[36, 38]}, "properties":
{"NODE_ID":"5", "EDGE_ID":"4", "FACE_ID":"0"}},

```
{"type":"Node", "geometry":{"type":"Point", "coordinates":[41, 40]}, "properties":
{"NODE_ID":"7", "EDGE_ID":"5", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[47, 14]}, "properties":
{"NODE_ID":"12", "EDGE_ID":"15", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[47, 6]}, "properties":
{"NODE_ID":"11", "EDGE_ID":"-14", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[47, 22]}, "properties":
{"NODE_ID":"19", "EDGE_ID":"-15", "FACE_ID":"0"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[57, 33]}, "properties":
{"NODE_ID":"6", "EDGE_ID":"-4", "FACE_ID":"0"}}
]}
```


To retrieve topology features, the following example specifies the `topology`, `base_table`, and `geom_col` parameters (where `grom_col` refers to the topology column):

```
http://example:8080/mapviewer/dataserver/tilsmenv?
topology=city_data&base_table=land_parcels&geom_col=feature
```

The response to this request is similar to the following:


```
{"type":"TopologyFeatures",
"topology":"CITY_DATA",
"topology_id":5,
"topology_owner":"TILSZUSER",
"tolerance":5.0E-5,
"srs":0,
"table_schema":"TILSZUSER",
"table_name":"LAND_PARCELS",
"topo_column":"FEATURE",
"layer_id":1,
"layer_type":"POLYGON",
"layer_level":0,
"child_layer":0,
"node_sequence":"CITY_DATA_NODE_S",
"edge_sequence":"CITY_DATA_EDGE_S",
"face_sequence":"CITY_DATA_FACE_S",
"feature_sequence":"CITY_DATA_TG_S",
"digits_right_decimal":16,
"attr_names":["FEATURE_NAME"],
"attr_types":["string"],
"features":[
{"type":"topology", "tg_id":4, "primitives":[{"topo_id":3,"topo_type":3},
{"topo_id":6,"topo_type":3}], "properties":{"FEATURE_NAME":"P1"}},
{"type":"topology", "tg_id":5, "primitives":[{"topo_id":4,"topo_type":3},
{"topo_id":7,"topo_type":3}], "properties":{"FEATURE_NAME":"P2"}},
{"type":"topology", "tg_id":6, "primitives":[{"topo_id":5,"topo_type":3},
{"topo_id":8,"topo_type":3}], "properties":{"FEATURE_NAME":"P3"}},
{"type":"topology", "tg_id":7, "primitives":[{"topo_id":2,"topo_type":3}], "properties":
{"FEATURE_NAME":"P4"}},
{"type":"topology", "tg_id":8, "primitives":[{"topo_id":1,"topo_type":3}], "properties":
{"FEATURE_NAME":"P5"}}
]}
```


To specific primitive faces, edges, and nodes, the following example define the primitive identifiers:

```
http://example:8080/mapviewer/dataserver/tilsmenv?
topology=city_data&face_ids=-1&edge_ids=3,4&node_ids=5
```

The response to this request is similar to the following:


```
{"type":"TopologyPrimitives",
"topology":"city_data",
```

```
"srs":0,
"bbox":[0, 0, 57, 38],
"face_attr_names":
["FACE_ID","BOUNDARY_EDGE_ID","ISLAND_EDGE_ID_LIST","ISLAND_NODE_ID_LIST"],
"face_attr_types":["integer","integer","array:integer","array:integer"],
"edge_attr_names":
["EDGE_ID","START_NODE_ID","END_NODE_ID","NEXT_LEFT_EDGE_ID","PREV_LEFT_EDGE_ID","NEXT_RI
GHT_EDGE_ID","PREV_RIGHT_EDGE_ID","LEFT_FACE_ID","RIGHT_FACE_ID"],
"edge_attr_types":
["integer","integer","integer","integer","integer","integer","integer","integer","integer
"],
"node_attr_names":["NODE_ID","EDGE_ID","FACE_ID"],
"node_attr_types":["integer","integer","integer"],
"primitives":[
{"type":"Face", "properties":{"FACE_ID":"-1", "BOUNDARY_EDGE_ID":"0",
"ISLAND_EDGE_ID_LIST":[-1,-2,4,6]}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":[25,30,25,35]},
"properties":{"EDGE_ID":"3", "START_NODE_ID":"2", "END_NODE_ID":"3",
"NEXT_LEFT_EDGE_ID":"-3", "PREV_LEFT_EDGE_ID":"2", "NEXT_RIGHT_EDGE_ID":"2",
"PREV_RIGHT_EDGE_ID":"3", "LEFT_FACE_ID":"2", "RIGHT_FACE_ID":"2"}},
{"type":"Edge", "geometry":{"type":"LineString", "coordinates":
[36,38,38,35,41,34,42,33,45,32,47,28,50,28,52,32,57,33]}, "properties":{"EDGE_ID":"4",
"START_NODE_ID":"5", "END_NODE_ID":"6", "NEXT_LEFT_EDGE_ID":"-5",
"PREV_LEFT_EDGE_ID":"-4", "NEXT_RIGHT_EDGE_ID":"4", "PREV_RIGHT_EDGE_ID":"5",
"LEFT_FACE_ID":"-1", "RIGHT_FACE_ID":"-1"}},
{"type":"Node", "geometry":{"type":"Point", "coordinates":[36, 38]}, "properties":
{"NODE_ID":"5", "EDGE_ID":"4", "FACE_ID":"0"}}
]}
```


3.1.3 Interpreting Data Returned from the Map Data Server

The map data server returns data in a compressed GeoJSON format. Some minor changes and additions to the standard GeoJSON are made to improve performance and the usefulness of the information.

The following is a sample response:


```
{"type":"FeatureCollection",
"collectionName":"theme1",
"srs":3857,
"geodetic":false,
"bbox":[-17566686.86258, 2414218.89842, -7905675.57465, 8629389.76988],
"attr_names":["CITY","STATE_ABRV","POP90","RANK90"],
"attr_types":["string","string","double","double"],
"features":[
{"type":"Feature","_id":"AAASQ3AAEAAAAMbAAA","geometry": {"type":"Point", "coordinates":
[-119.99823, 38.9052]},"properties":{"CITY":"SOUTH LAKE TAHOE", "SALES":"125.8",
"NAME":"FACTORY STORES AT THE Y", "_label_":"FACTORY STORES AT THE Y"},"styles":
{"rendering":{"style":"M.SMALL CIRCLE"},"labeling":{"style":"T.RED STREET", "columns":
["_label_"]}}},
{"type":"Feature","_id":"AAASQ3AAEAAAAMbAAB","geometry": {"type":"Point", "coordinates":
[-121.95073, 37.53356]},"properties":{"CITY":"FREMONT", "SALES":"186.8", "NAME":"OHLONE
VILLAGE", "_label_":"OHLONE VILLAGE"},"styles":{"rendering":{"style":"M.SMALL
CIRCLE"},"labeling":{"style":"T.RED STREET", "columns":["_label_"]}}},
{"type":"Feature","_id":"AAASQ3AAEAAAAMbAAC","geometry": {"type":"Point", "coordinates":
[-118.48844, 34.02353]},"properties":{"CITY":"SANTA MONICA", "SALES":"9.1",
"NAME":"SANTA MONICA PLACE", "_label_":"SANTA MONICA PLACE"},"styles":{"rendering":
{"style":"M.SMALL CIRCLE"},"labeling":{"style":"T.RED STREET", "columns":["_label_"]}}},
{"type":"Feature","_id":"AAASQ3AAEAAAAMbAAD","geometry": {"type":"Point", "coordinates":
[-118.55093, 34.42104]},"properties":{"CITY":"SANTA CLARITA", "SALES":"52.6",
"NAME":"VALENCIA TOWN CENTER", "_label_":"VALENCIA TOWN CENTER"},"styles":{"rendering":
{"style":"M.SMALL CIRCLE"},"labeling":{"style":"T.RED STREET", "columns":["_label_"]}}},
```

{"type":"Feature","_id":"AAASQ3AAEAAAAMbAAE","geometry": {"type":"Point", "coordinates":
[-122.56007, 38.08187]},"properties":{"CITY":"NOVATO", "SALES":"119.1", "NAME":"VINTAGE
OAKS AT NOVATO", "_label_":"VINTAGE OAKS AT NOVATO"},"styles":{"rendering":
{"style":"M.SMALL CIRCLE"},"labeling":{"style":"T.RED STREET", "columns":["_label_"]}}}]}

The response contains a minimal header plus an array of features. The header includes the spatial reference system (`srs`) ID and the minimum bounding box of the result data. The array of features includes attribute names and their types. Possible type names include: `"byte","short","int","long","float","double","char","string","boolean", "date"`

For each feature, the following fields apply:

- `type`: Always `Feature`.
- `_id`: Optional ID or key attribute.
- `geometry`: The actual geometry encoded in the modified GeoJSON format.
- `properties`: An object containing all the properties (name-value pairs) for the feature.
- `styles`: An optional styling information object. Contains two embedded objects, `rendering` and `labeling`, which share the same structure: basically an object containing a `style` field and an optional `columns` array. Currently only predefined themes support including styling information in the response; a dynamic theme's response contains no styling information.
- `label_box`: A 4-element array specifying the `minX`, `minY`, `maxX`, and `maxY` of the label box. Only a polygon can have a label box.

Note that the labeling text is always included as a pseudo-property with the name `_label_` in the property list.

3.1.4 Map Data Server Error Handling

If the map data server cannot process a data request, it will send a JSON response containing an error object. This JSON error object may look like the following:

```
{"error":
  {"code": "ora-500",
   "message": "Table requested does not exist",
   "details": "maybe a stack trace here..."
  }
}
```

In the preceding JSON object, `code` is the error code known only to the map data serve, `message` contains a short message that can be displayed to the end user in a warning dialog, and `details` is an optional field that may contain more details (such as the stack trace if included).

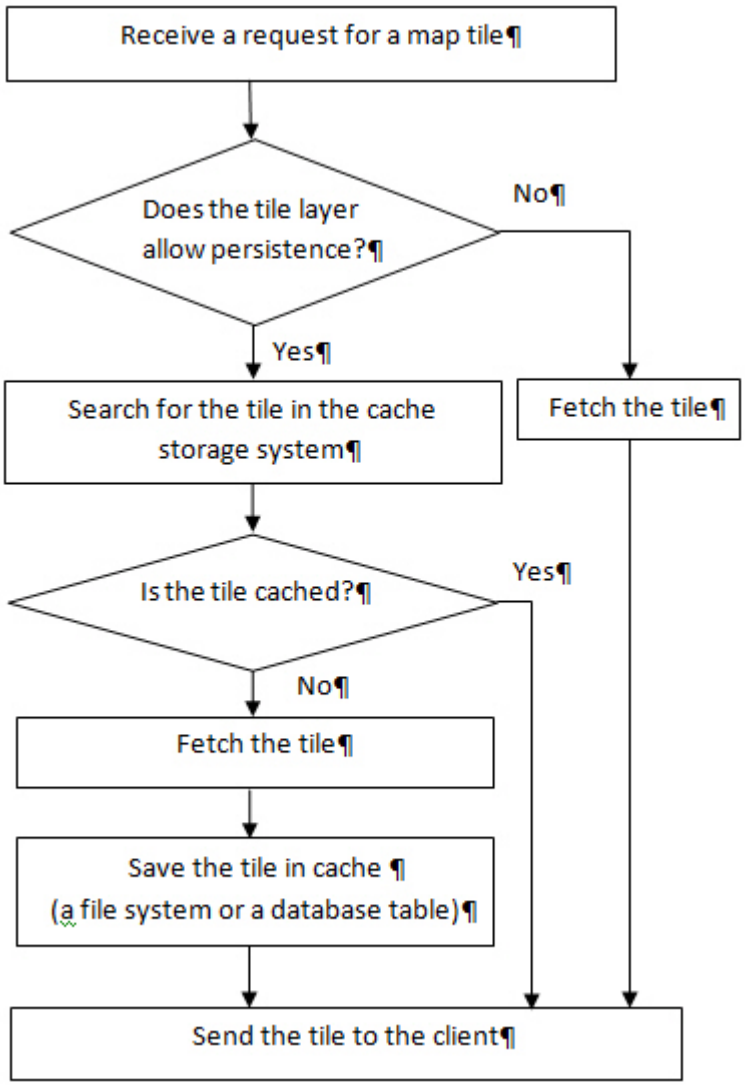
3.2 Map Tile Server

The map tile server is a map image caching engine that fetches, caches, and serves pregenerated, fixed-size map image tiles.

It is implemented as a Java servlet that is part of the map visualization component server. The map tile server accepts requests that ask for map image tiles specified by tile zoom level and tile location (mesh code), and it sends the requested tiles back to clients.

Figure 3-1 shows the basic workflow of the map tile server.

Figure 3-1 Workflow of the Map Tile Server



As shown in Figure 3-1, when the map tile server receives a request for a map tile, it searches for the tile in the cache storage system. If the tile is cached, the map tile server sends the tile to the client. If the tile is not cached, the map tile server fetches the tile, saves it in the cache, and sends it to the client.

You can use the map visualization component administration tool to manage the map tile server.

- Map Tile Server Concepts
- Map Tile Server Configuration
- Map Cache Auto-Update
- UTFGrid for Map Tiles: Including Text Information About Features
- External Map Source Adapter

3.2.1 Map Tile Server Concepts

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

- Map Tile Layers and Map Tile Sources
- Storage of Map Image Tiles
- Coordinate System for Map Tiles
- Tile Mesh Codes
- Map Tile Requests
- Tiling Rules
- Tile Background Color and Out-of-Bounds Color

3.2.1.1 Map Tile Layers and Map Tile Sources

All map tile layers are managed by the map tile server. The **map tile server** fetches and stores the map image tiles that belong to the map tile layer and returns map image tiles to the client. The map tile server can manage multiple map tile layers.

Each map tile layer can have multiple predefined zoom levels. Each zoom level is assigned a zoom level number ranging from 0 to n-1, where n is the total number of zoom levels. Zoom level 0 is the most zoomed out level and zoom level n-1 is the most zoomed in level.

The map is evenly divided into same-sized small map image tiles on each zoom level. Clients specify a map tile by its zoom level and tile mesh code.

A map tile layer can come from two different types of sources:

- Internal map visualization component base maps rendered by the map visualization component map rendering engine. A map visualization component base map consists of a set of predefined themes and must be predefined in the database view USER_SDO_MAPS.
- Maps rendered by an external web map services providers. An external web map services provider is a server that renders and serves maps upon client requests over the web. If you properly configure an adapter that can fetch maps from the external map services provider, the map tile server can fetch and cache map tiles generated by the external map services provider. (A map visualization component instance other than the map visualization component inside which the map tile server is running is also considered an external map services provider.)

3.2.1.2 Storage of Map Image Tiles

Oracle Maps has three options for handling the storage of map image tiles:

- Store the tiles using the local file system .

  If you use the local file system for caching, you can customize the path that is used for this storage as part of the map tile server configuration settings.
- Store the tiles in a database table .

  If you use a database table for caching, you must create the database table, and set the TILES_TABLE column to that table for the tile layer in the USER_SDO_CACHED_MAPS view.

- Stream the tiles directly without storing them .

  If you do not want to cache any image tiles, you must indicate that in the tile layer's definition.

- Store the tiles using the local file system
- Store the tiles in a database table
- Stream the tiles directly without storing them

3.2.1.2.1 Store the tiles using the local file system

In a file system, each tile layer has its own storage root directory, which is specified by the `<cache_storage>` element's `root_path` attribute in the tile layer definition. If that attribute is not specified, then the default storage location specified in the `mapViewerConfig.xml` file `<tile_storage>` element is used as the root path. For example, if the root path is defined as `/scratch/tilecache/`, and a data source named MVDEMO has a tile layer named DEMO_MAP with 19 zoom levels, after the server is instantiated the folder `/scratch/tilecache/MVDEMO.DEMO_MAP` is created, and it contains 19 subfolders (`/0`, `/1`, …, `/18`), each for storing the image tiles in that zoom level.

Under each zoom level, there are two options to organize its subfolders for map tiles. One is the default option, which uses a mesh code tree structure; the other, called xyz storage scheme, uses the tile's row and column values as subfolder and tile name to store the map tile. Both storage options start from the tile's mesh code value for each zoom level (see Tile Mesh Codes for details about the tile mesh code). Each tile in a zoom level can be represented using its mesh code value pair (`mx, my`), where the `mx` and `my` are integer values in the horizontal and vertical directions respectively. The tile at the lower left corner has a value of (0, 0). A tile can also be located using its tile row and tile column value pair (`tile_column, tile_row`). A tile at the upper left corner has a value of (0, 0).

3.2.1.2.2 Store the tiles in a database table

Image tiles can be stored in a database table, as follows.

1. Create a table for storing the image tiles. For example:

```
CREATE TABLE tile_dbt (
  tile_layer varchar2(64),
  zoom_level number,
  x number,
  y number,
  modified TIMESTAMP,
  data BLOB);
COMMIT;
```

2. Update the TILES_TABLE column in USER_SDO_CACHED_MAPS view for the tile layer. For example, if you have a tile layer DEMP_MAP and you want to use the table created in step 1 to store its map tiles, then update the tile layer's TILES_TABLE column as follows:

```
UPDATE user_sdo_cached_maps SET tiles_table='tiles_dbt' WHERE name='DEMP_MAP';
```

3. Restart the map visualization component server to make the changes take effect.

Using this example, the map image tiles for tile layer DEMP_MAP will be stored in the database table TILE_DBT.

3.2.1.2.3 Stream the tiles directly without storing them

If the tile contents may change constantly (such as a real-time cloud cover satellite image map) or if you do not want to store the image tiles, you can stream the tiles directly without storing them.

To choose this option, set the `persistent_tiles` attribute to `false` in the `<map_tile_layer>` element in the tile layer's definition. (The default value for the `persistent_tiles` attribute is `true`.). Example 3-1 inserts a tile layer named DEMO_MAP, with the `persistent_tiles` attribute sets to `false` so that no map image tiles will be cached for this tile layer.

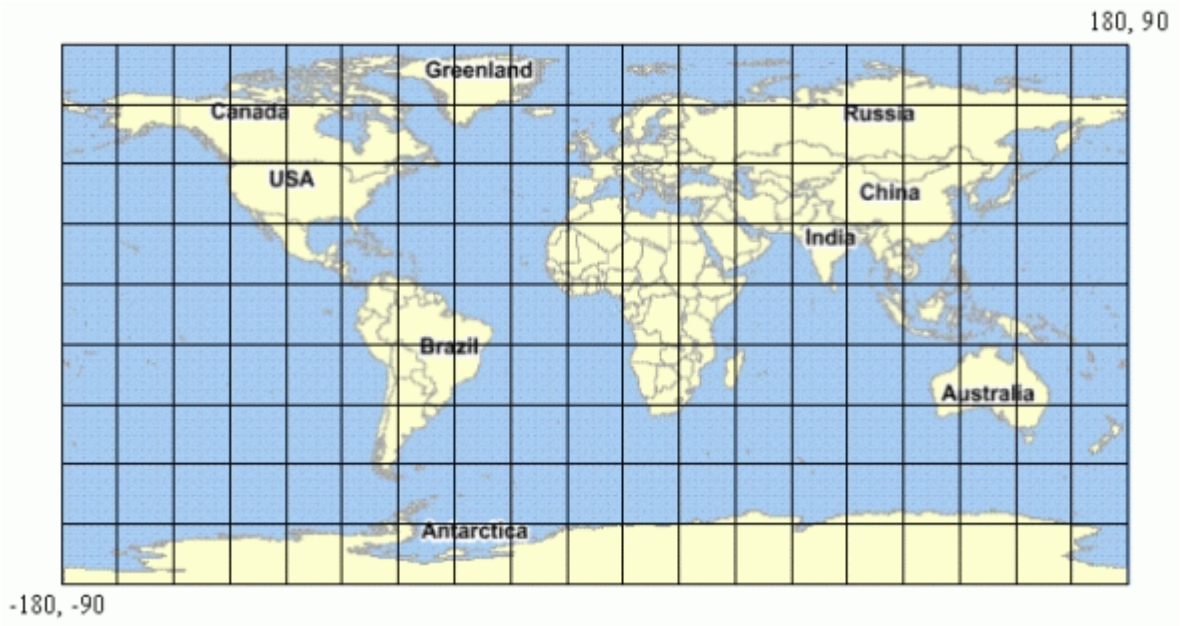
Example 3-1 Streaming Tiles Without Storing Them

```
INSERT INTO user_sdo_cached_maps values(
'DEMO_MAP',
'an example tile layer that does not cache image tiles',
'',
'YES',
'YES',
'<map_tile_layer name="DEMO_MAP_TREEMESH" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3" persistent_tiles="false">
   <internal_map_source data_source="mvdemo" base_map="DEMO_MAP" bgcolor="#dddddd"
out_of_bounds_color="#eeddff"/>
   <tile_storage root_path="/temp" short_path="false" />
   <coordinate_system srid="8307" minX="-180.0" maxX="180.0" minY="-90.0" maxY="90.0"/>
   <tile_image width="256" height="256"/>
   <tile_dpi  value="90.7142857"/>
   <tile_meters_per_unit  value="111319.49079327358"/>
   <zoom_levels levels="19" min_scale="2132.729583849784" max_scale="559082264.0287178"/>
</map_tile_layer>',
'DEMO_MAP',
'');
COMMIT;
```

3.2.1.3 Coordinate System for Map Tiles

Map images are cached and managed by the map tile server as small same-size rectangular image tiles. Currently we support tiling on any two-dimensional Cartesian coordinate system. A geodetic coordinate system can also be supported when it is mapped as if it is a Cartesian coordinate system, where longitude and latitude are treated simply as two perpendicular axes, as shown in Figure 3-2.

Figure 3-2 Tiling with a Longitude/Latitude Coordinate System



On each zoom level, the map tiles are created by equally dividing the whole map coordinate system along the two dimensions (X and Y, which inFigure 3-2 represent latitude and longitude). The map tile server needs this dimensional information of the map coordinate system in order to create map image tiles, and therefore you must include this information in the map tile layer configuration settings.

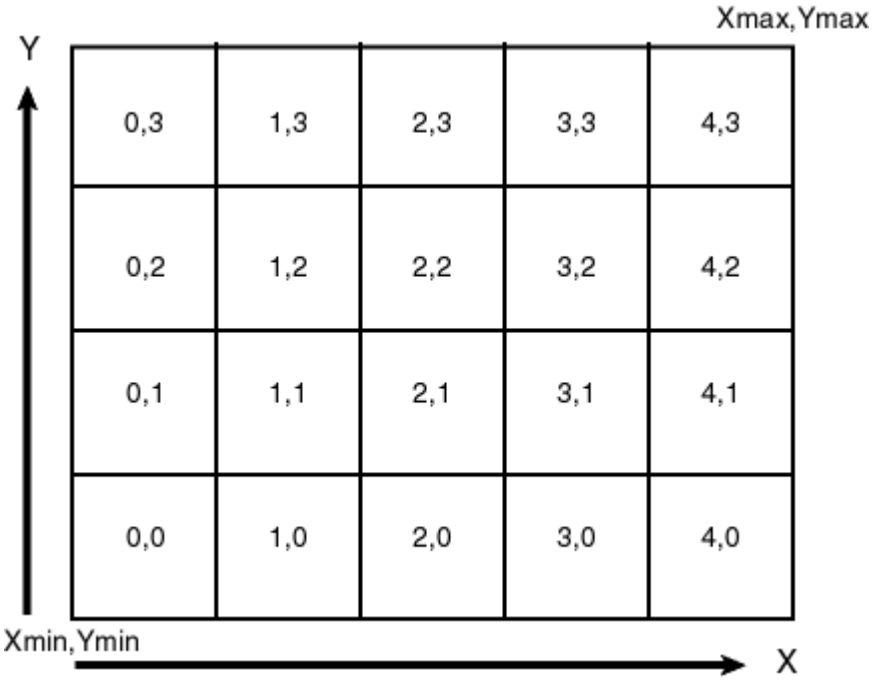
The whole map coordinate system can be represented by a rectangle, and its boundary is specified by (Xmin, Ymin) and (Xmax, Ymax), where Xmin is the minimum X value allowed in the coordinate system, Ymin is the minimum Y value allowed, Xmax is the maximum X value allowed and Ymax is the maximum Y value allowed. In Figure 3-2, Xmin is –180, Ymin is –90, Xmax is 180, and Ymax is 90.

You must also specify the spatial referencing ID (SRID) of the coordinate system to enable the map tile server to calculate map scales.

3.2.1.4 Tile Mesh Codes

Each map tile is specified by a mesh code, which is defined as a pair of integers (Mx, My), where Mx specifies the X dimension index of the tile and My specifies the Y dimension index of the tile. If the tile is the *i*th tile on X dimension starting from Xmin, then Mx should be i-1. If the tile is the *j*th tile on Y dimension starting from Ymin, then My should be j-1. Figure 3-3 shows the mesh codes of the tiles on a map.

Figure 3-3 Tile Mesh Codes



The JavaScript map client automatically calculates which tiles it needs for displaying the map in the web browser, and it sends requests with the mesh codes to the server. Mesh codes are transparent to the application, and application developers do not need to deal with mesh codes directly.

3.2.1.5 Map Tile Requests

The map tile server handles map tile requests. A map tile request can be in key/value pair format or REST format:

- Map tile request in key/value pair format

  For example, if data source DS_NAME has a tile layer TL_NAME, then to get a map tile image in PNG format from zoom level 2 and mesh code (3.2), the URL may be formatted as:

  ```
  http://localhost:8080/mapviewer/mcserver?
  request=gettile&format=PNG&zoomlevel=2&mapcache=DS_NAME.TL_NAME&mx=3&my=2
  ```

- Map tile request in REST format

  A general format of a request in REST format is:

  ```
  http://localhost:8080/mapviewer/mcserver/DS_NAME/TL_NAME/{zoom}/{row}/{column}.png
  ```

  For example, to send the same map tile request shown in the preceding key/value pair example but in a REST format, the URL may be:

  ```
  http://localhost:8080/mapviewer/mcserver/DS_NAME/TL_NAME/2/1/3.png
  ```

The mesh code in the key/value pair format has an origin of the lower-left corner, but the REST format requests a tile's row and column, and thus the origin is at the upper-left corner. Because the two formats have different origins, the `my` value is different from the `row` value, but the `mx` value is the same as the `column` value.

Normally, map tile requests are encapsulated in the map visualization component JavaScript API libraries, so the API handles map tile requests. However, if you are using a third party's JavaScript API to communicate with the map visualization component server, you may need to specify the map tile request format in your application. For example, if you use the Leaflet JavaScript API to get map tiles from a map visualization component server, you may need to set the URL template for its `L.tileLayer` as:

```
http://localhost:8080/mapviewer/mcserver/DS_NAME/TL_NAME/{z}/{y}/{x}.png.
```

3.2.1.6 Tiling Rules

You must create tiling rules that determine how the map is divided and how tiles are created. The map tile server uses these tiling rules to divide the map into small map image tiles that are stored in the tile storage system. These rules are also used by the JavaScript map client.

Because all tiles on a given zoom level are the same size, the map tile server needs to know the following information to perform the tile division:

- The map tile image size (width and height), specified in screen pixels. This is the physical size of the tile images.
- The tile size specified according to the map coordinate system. For example, if the map uses a geodetic coordinate system, the tile width and height should be defined in degrees. The size can be specified either explicitly by tile width and height or implicitly by map scale. (Map scale, combined with tile image size, can be used to derive the tile width and height according to the map coordinate system.)

The preceding information constitutes the tiling rule for a given zoom level. Each zoom level must have its own tiling rule. You must define the tiling rules when you specify the configuration settings for the map tile server, as described in Map Tile Server Configuration.

3.2.1.7 Tile Background Color and Out-of-Bounds Color

Two attributes in a tile layer metadata definition which affect a tile's color: `bgcolor` (background color) and `out_of_bounds_color` (out-of-bounds color). The `bgcolor` attribute value is used for filling areas within the valid data area of a tile layer (the valid data area is defined by `minX, minY, maxY, maxY`), while the `out_of_bounds_color` attribute value is used for filling areas that are outside the valid data area. Both attributes have the same default values (`Color(192, 192, 192)`).

If a tile-fetching process failed due to an exception on the attempt to generate a tile, then the tile filled with out-of-bounds color is used as its substitute, regardless of whether it is within the valid data area. However, such a substitute tile due to tile-fetching exception is not permanently stored on disk; rather, it is streamed to the client on a temporary basis. The map visualization component will retry the tile generation on subsequent requests, if the temporary tile data in the client browser's cache is purged or if a different client initiates the request.

If `bgcolor` is set to `none`, then the tile becomes transparent; that is, the background color of the HTML page replaces the attribute values for both `bgcolor` and `out_of_bounds_color`.

3.2.2 Map Tile Server Configuration

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

- Global Map Tile Server Configuration
- Map Tile Layer Configuration

- Map Tile Storage Schemes: Internal Mesh Code or XYZ
- Creating a Map Tile Layer Using an External Web Map Source

3.2.2.1 Global Map Tile Server Configuration

Global map tile server settings, such as logging options and the default cache storage directory, are stored in the map visualization component configuration file `mapViewerConfig.xml`, which is under the directory *`$MAPVIEWER_HOME`*`/web/WEB-INF/conf`.

The map tile server configuration settings are defined in element `<map_tile_server>` inside the top-level `<mapperConfig>` element, as shown in the following example:

```
<map_tile_server>
   <tile_storage default_root_path="/scratch/tilecache/"/>
</map_tile_server>
```

The `<tile_storage>` element specifies the map tiles storage settings. The `default_root_path` attribute specifies the default file system directory under which the cached tile images are to be stored. If the default root directory is not set or not valid, the default root directory is *`$MAPVIEWER_HOME`*`/web/tilecache`. A subdirectory under this directory will be created and used for a map tile layer if the map tile layer configuration does not specify the map tiles storage directory for itself. The name of the subdirectory will be the same as the name of the map tile layer.

3.2.2.2 Map Tile Layer Configuration

The configuration settings for a map tile layer are stored in the USER_SDO_CACHED_MAPS metadata view, which is described in xxx_SDO_CACHED_MAPS Views. You should normally not manipulate this view directly, but should instead use the map visualization component administration tool, which uses this view to configure map tile layers.

Each database user (schema) has its own USER_SDO_CACHED_MAPS view. Each entry in this view stores the configuration settings for one map tile layer. If the map tile layer is based on an internal map visualization component base map or themes, the base map or themes associated with the map tile layer must be defined in the same database schema where the map tile layer configuration settings are stored.

The map tile server obtains the map source configuration by querying the USER_SDO_CACHED_MAPS view using the database connections specified by the map visualization component data sources. This happens when the map tile server is started or a new data source is added to the map visualization component as the result of a map visualization component administration request.

For the DEFINITION column in the USER_SDO_CACHED_MAPS view, the map source definition has the following general format:

```
<map_tile_layer
   name = "map tile layer name"
   image_format ="tile-image-format">
  <internal_map_source
    data_source="name-of-data-source"
    base_map="name-of-MapViewer-base-map"
    bgcolor="base-map-background-color"
    antialias="whether-to-turn-on-antialiasing"
    />
  </internal_map_source>
  <external_map_source
     url="external-map-service-url"
```

```
      adapter_class="name-of-adapter-class"
      proxy_host=" proxy-server-host "
      proxy_port="proxy-server-port"
      timeout="request-timeout"
      request_method="http-request-method: 'GET'|'POST'">
     <properties>
       <property name="property-name" value="property-value"/>
       …
     </properties>
   </external_map_source>
   <tile_storage
      root_path="disk-path-of-cache-root-directory"
   </tile_storage>
   <coordinate_system
      srid="coordinate-system-srid"
      minX="minimum-allowed-X-value"
      maxX="maximum-allowed-X-value"
      minY="minimum-allowed-Y-value"
      maxY="maximum-allowed-Y-value">
   </coordinate_system>
   <tile_image
      width="tile-image-width-in-screen-pixels"
      height="tile-image-height-in-screen-pixels" >
   </tile_image>
   <tile_bound>
      <coordinates> … </coordinates>
   </tile_bound>
   <zoom_levels
      levels="number-of-zoom-levels"
      min_scale="map-scale-at-highest-zoom-level"
      max_scale="map-scale-at-lowest-zoom-level"
      min_tile_width="tile-width-specified-in-map-data-units-at-
                      highest-zoom-level"
      max_tile_width="tile-width-specified-in-map-data-units-at-
                      lowest-zoom-level">
     <zoom_level
        description="zoom-level-description"
        level_name="zoom-level-name"
        scale="map-scale-of-zoom-level"
        tile_width ="tile-width-specified-in-map-data-units"
        tile_height ="tile-height-specified-in-map-data-units">
       <tile_bound>
         <coordinates> … </coordinates>
       </tile_bound>
     </zoom_level>
     …
   </zoom_levels>
 </map_tile_layer>
```

The DTD of the map tile layer definition XML is listed in Map Tile Layers.

Example 3-2 shows the XML definition of an internal map tile layer that is based on a base map; Example 3-3 shows the XML definition of an internal map tile layer that is based on themes (not on a base map); and Example 3-4 shows the XML definition of an external map tile layer. Explanations of the `<map_tile_layer>` element and its subelements follow these examples.

Example 3-2 XML Definition of an Internal Map Tile Layer Based on a Base Map

```
<?xml version = '1.0'?>
<!-- XML definition of an internal map tile layer.
-->
```

```
   <map_tile_layer image_format="PNG">
      <internal_map_source base_map="demo_map"/>
      <tile_storage root_path="/scratch/mapcache/"/>
      <coordinate_system
         srid="8307"
         minX="-180" maxX="180"
         minY="-90" maxY="90"/>
      <tile_image width="250" height="250"/>
      <zoom_levels>
         <zoom_level description="continent level" scale="10000000"/>
         <zoom_level description="country level" scale="3000000"/>
         <zoom_level description="state level" scale="1000000"/>
         <zoom_level description="county level" scale="300000"/>
         <zoom_level description="city level" scale="100000"/>
         <zoom_level description="street level" scale="30000"/>
         <zoom_level description="local street level" scale="10000"/>
      </zoom_levels>
   </map_tile_layer>
```

Example 3-3 XML Definition of an Internal Map Tile Layer Based on Themes

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<map_tile_layer name="TL1" image_format="PNG" http_header_expires="168.0" utfgrid="true"
utfgrid_resolution="4" concurrent_fetching_threads="3">
     <internal_map_source data_source="MVDEMO" bgcolor="none"
out_of_bounds_color="#ffffff" base_map="" db_tile_table=""/>
     <tile_storage root_path="/temp" xyz_storage_scheme="true"/>
     <coordinate_system srid="3857" minX="-2.0037508E7" minY="-2.0037508E7"
maxX="2.0037508E7" maxY="2.0037508E7"/>
     <tile_image width="256" height="256"/>
     <tile_dpi value="90.714"/>
     <tile_meters_per_unit value="1.0"/>
     <zoom_levels levels="19" min_scale="2132.72958384" max_scale="5.59082264028E8"
min_tile_width="152.874538064" max_tile_width="4.00751429065E7">
          <zoom_level level="0" name="level0" tile_width="4.00751429065E7"
tile_height="4.00751429065E7"/>
          <zoom_level level="1" name="level1" tile_width="2.003757145325E7"
tile_height="2.003757145325E7"/>
          <zoom_level level="2" name="level2" tile_width="1.0018785726625258E7"
tile_height="1.001878572662E7"/>
          <zoom_level level="3" name="level3" tile_width="5009392.86331"
tile_height="5009392.86331"/>
          <zoom_level level="4" name="level4" tile_width="2504696.431656"
tile_height="2504696.431656"/>
          <zoom_level level="5" name="level5" tile_width="1252348.215828"
tile_height="1252348.215828"/>
          <zoom_level level="6" name="level6" tile_width="626174.1079140786"
tile_height="626174.107914"/>
          <zoom_level level="7" name="level7" tile_width="313087.0539570393"
tile_height="313087.053957"/>
          <zoom_level level="8" name="level8" tile_width="156543.5269785"
tile_height="156543.5269785"/>
          <zoom_level level="9" name="level9" tile_width="78271.763489"
tile_height="78271.763489"/>
          <zoom_level level="10" name="level10" tile_width="39135.8817446"
tile_height="39135.8817446"/>
          <zoom_level level="11" name="level11" tile_width="19567.9408723"
tile_height="19567.9408723"/>
          <zoom_level level="12" name="level12" tile_width="9783.9704361"
tile_height="9783.9704361"/>
          <zoom_level level="13" name="level13" tile_width="4891.9852180"
tile_height="4891.9852180"/>
```

```
          <zoom_level level="14" name="level14" tile_width="2445.9926090"
tile_height="2445.99260903"/>
          <zoom_level level="15" name="level15" tile_width="1222.99630451"
tile_height="1222.99630451"/>
          <zoom_level level="16" name="level16" tile_width="611.49815225"
tile_height="611.49815225"/>
          <zoom_level level="17" name="level17" tile_width="305.74907612"
tile_height="305.74907612"/>
          <zoom_level level="18" name="level18" tile_width="152.87453806"
tile_height="152.87453806"/>
   </zoom_levels>
     <auto_update finest_level_to_refresh="13" dirty_mbr_batch="100"
dirty_mbr_cap="1000">
          <dirty_mbr_table name="TL1MBR"/>
          <logtable name="TL1LOG"/>
   </auto_update>
     <themes>
          <theme name="UTFGRID_THEME_DEMO_STATES" from_level="0" to_level="18"/>
          <theme name="UTFGRID_THEME_DEMO_COUNTIES" from_level="0" to_level="18"/>
          <theme name="UTFGRID_THEME_DEMO_HIGHWAYS" from_level="0" to_level="18"/>
          <theme name="UTFGRID_THEME_DEMO_CITIES" from_level="0" to_level="18"/>
   </themes>
</map_tile_layer>
```

Example 3-4 XML Definition of an External Map Tile Layer

```
<?xml version = '1.0'?>
<!-- XML definition of an external map tile layer.-->
<map_tile_layer name="TILELAYER1" image_format="PNG">
   <external_map_source
     url="http://mycorp.com:7001/mapviewer/wms/"
     request_method="GET"
     adapter_class="oracle.lbs.mapcache.adapter.WMSAdapter"
     adapter_class_path="">
      <properties>
         <property name="datasource" value="mvdemo"/>
         <property name="version" value="1.1.1"/>
         <property name="srs" value="EPSG:4326"/>
         <property name="layers" value="THEME_DEMO_COUNTIES,THEME_DEMO_HIGHWAYS"/>
         <property name="format" value="image/png"/>
         <property name="transparent" value="true"/>
      </properties>
   </external_map_source>
   <tile_storage root_path="/scratch/tmp/"/>
   <coordinate_system srid="8307" minX="-180.0" minY="-90.0" maxX="180.0" maxY="90.0"/>
   <tile_image width="256" height="256"/>
   <!-
         The following <zoom_levels> element does not have any
         <zoom_level> element inside it. But since it has its levels,
         min_scale and max_scale attributes set, map tile server will
         automatically generate the <zoom_level> elements for the 10
         zoom levels.
   -->
   <zoom_levels levels="10" min_scale="1000.0" max_scale="2.5E8">
   </zoom_levels>
</map_tile_layer>
```

The top-level element is `<map_tile_layer>`. The `image_format` attribute specifies the tile image format; the currently supported values for this attribute are `PNG`, `GIF`, and `JPG`. PNG and GIF images are generally better for vector base maps, while JPG images are generally better for raster maps, such as satellite imagery, because of a better compression ratio. Currently, only tile images in PNG format can have transparent background.

- The `http_header_expires` attribute specifies the number of hours after which a cached tile layer can be considered stale.
- The `utfgrid` attribute, when set to `true`, indicates that a companion UTFGrid dataset for an image file will be generated. (The default value is `false`.)
- The `utfgrid_resolution` attribute specifies how fine the grid data is compared to the image tile. For example, a value of 4 (the default) indicates that one grid cell represents 4 by 4 pixels in its companion image tile.
- The `concurrent_fetching_threads` attribute defines the maximum number of concurrent tile fetching threads that may be created for fetching image tiles for this tile layer. (The default value is 3.)
- The `fetch_larger_tile` attribute , when `true` (the default), tells the tile server that if the tile is not available in the cache, the tile's adjacent tiles will also be generated. This parameter does bot affect the `getTile` performance when the requested tiles are already available in the cache. The default setting (`true`) is intended to improve server response time.
- The `persistent_tiles` attribute, when `true` (the default), specifies that image tiles will be saved is the server's disk cache. If this attribute is set to false, then each getTile request invokes an image tile rendering process in the map visualization component server, and the newly acquired image tile is not cached for future use. Cases where you may want to specify `false` include (A) a GeoRaster theme is used by a tile layer and you do not want duplicate raster images in the map visualization component's disk cache, or (B) you want the tile server always to provide up-to-date image tiles.

The `<internal_map_source>` element is required only if the map tiles are rendered by the local map visualization component instance.

- The `base_map` attribute is required and specifies the predefined map visualization component base map that is cached by the map tile server; its value should match an entry in the BASE_MAP column in the USER_SDO_CACHED_MAPS view.
- The `bgcolor` attribute is optional and specifies the background color of the map. If the value of this attribute is set to `NONE`, the background will be transparent. (Currently, only tile images in PNG format can have a transparent background.)
- The `out_of_bounds_color` attribute is optional and specifies the color for areas that are outside the data boundaries. The data boundaries are specified by the attributes of the `<coordinate_system>` element.
- The `db_tile_table` attribute is optional, specifies the database table in which to cache the tile layer's image tiles. If the attribute is not specified, the tiles are cached in the map visualization component's disk cache.

The `<external_map_source>` element is required only if the map tiles are rendered by an external map services provider. This element has the following attributes:

- The `url` attribute is required and specifies the map service URL from which the map tiles can be fetched (for example, `http://myhost/mapviewer/omserver`).
- The `adapter_class` attribute is required and specifies the full name of the map adapter class, including the package names (for example, `mcsadapter.MVAdapter`).
- The `proxy_host` and `proxy_port` attributes are needed only if the external map provider server must be accessed through a proxy server; these attributes specify the host name and port number, respectively, of the proxy server. If `proxy_host` is specified as `NONE`, all map tile requests will be sent directly to the remote server without going through any proxy server. If `proxy_host` is omitted or specifies an empty string, the global map visualization

component proxy setting defined in the `mapViewerConfig.xml file` will be used when map tile requests are sent.

- The `timeout` attribute is optional and specifies the number of milliseconds for which the map tile server must wait for an external map tile image before giving up the attempt. The default timeout value is 15000.
- The `request_method` attribute is optional and the HTTP request method for sending map tile requests; its value can be `POST` (the default) or `GET`.

For more information about external map tile layers and an example, see Creating a Map Tile Layer Using an External Web Map Source.

The <properties> element in the `<external_map_source>` element can include multiple `<property>` elements, each of which specifies a user-defined parameter for use by the map adapter when it fetches map tiles. The same map source adapter can use different set of parameters to fetch different map tile layers. For example, the sample map visualization component adapter `mcsadapter.MVAdapter` shipped with the map visualization component accepts parameters defined as follows:

```
<properties>
   <property name="data_source" value="elocation"/>
   <property name="base_map" value="us_base_map"/>
</properties>
```

However, by changing the `value` attribute values, you can use this adapter to fetch a different base map from the same data source or a different data source.

The `<tile_storage>` element specifies storage settings for the map tile layer.

- The optional `root_path` attribute specifies the file system directory to be used as the root directory of the tile storage. If this attribute is omitted or invalid, the default root directory defined in the `mapViewerConfig.xml` file is used.
- The optional `xyz_storage_scheme` element controls how the directory structure of map tiles in the disk cache is organized. The default value (`false`) causes the map visualization component internal mesh code storage scheme to be used. The value `true` causes the XYZ storage scheme to be used. For more information, see Map Tile Storage Schemes: Internal Mesh Code or XYZ

The `<coordinate_system>` element specifies the map coordinate system, and it has several required attributes. The `srid` attribute specifies the spatial reference ID of the coordinate system. The `minX` attribute specifies the lower bound of the X dimension; the `minY` attribute specifies the lower bound of the Y dimension; the `maxX` attribute specifies the upper bound of the X dimension; and the `maxY` attribute specifies the upper bound of the Y dimension. For the standard longitude/latitude (WGS 84) coordinate system, the `srid` value is 8307; and the `minX`, `minY`, `maxX`, and `maxY` values are -180, -90, 180, and 90, respectively.

For an internal map tile layer, the map coordinate system can be different from the data coordinate system. If the two are different, the map tile server transforms the map data into the coordinate system defined in the `<coordinate_system>` element and renders map tile images using this coordinate system.

The `<tile_image>` element specifies the tile image size settings, and it has the following required attributes: `width` specifies the width of the tile images in screen pixels, and `height` specifies the height of the tile images in screen pixels.

The optional `<tile_bound>` element specifies the bounding box of the cached map tiles. The map tile server only fetches tiles inside this box, and returns a blank tile if the requested tile is

outside this box. The bounding box is specified by a rectangle in the map data coordinate system. The rectangle is specified by a <coordinates> element in the following format:

```
<coordinates>minX, minY, maxX, maxY</coordinates>
```

The default cache bounding box is the same bounding box specified in the <coordinate_system> element.

The optional <tile_dpi> element specifies the map display screen resolution as a "dots per inch" value. If this element is not specified, the value specified in the mapViewerConfig.xml file will be assigned to the tile layer. If the map visualization component must comply with OGC standards in exposing the tile layer in its WMTS service, then you must specify this element with the following value: 90.714

The optional <tile_meters_per_unit> element specifies the meters per unit. The unit is defined indirectly by the srid attribute in the <coordinate_system> element. For example, if the srid is 3857, its unit is meters, and thus the value for this attribute must be 1.0; or if the srid is 8307, its unit is decimal degrees, and thus the value may be set to 111319.49. If this element is not specified, a map visualization component internal process calculates the value according to the data bounds, and the result is very close to 111319.49. If this tile layer is to be exposed by the map visualization component to provide WMTS services, and if srid is 8307, then you must set this element's value to 111319.49 to comply with OGC standards.

The <zoom_levels> element specifies the predefined zoom levels. Only image tiles at predefined zoom levels will be cached and served by the map tile server. The <zoom_levels> element can have multiple <zoom_level> elements, each of which specifies one predefined zoom level. If there are no <zoom_level> elements, the map tile server automatically generates the <zoom_level> elements by using the following attributes inside the <zoom_levels> element. (These attributes can be omitted and will be ignored if any <zoom_level> elements exist.)

- levels specifies the total number of zoom levels.
- min_scale specifies the scale of map images at the highest (zoomed in the most) zoom level.
- max_scale specifies the scale of map images at the lowest (zoomed out the most) zoom level.
- min_tile_width specifies the width of map tiles at the highest zoom level. The width is specified in map data units.
- max_tile_width specifies the width of the map tiles at the lowest zoom level. The width is specified in map data units.

For the map tile server to be able to generate the definitions of individual zoom levels automatically, you must specify either of the following combinations of the preceding attributes:

- levels, min_scale, and max_scale
- levels, min_tile_width, and max_tile_width

When the zoom levels are defined this way, the map tile server automatically derives the definition of all the individual zoom levels and updates the XML definition with the <zoom_level> elements generated for the zoom levels. You can then make adjustments to each zoom level if you want.

Each zoom level is assigned a zoom level number by the map tile server based on the order in which the zoom levels are defined. The first zoom level defined in the <zoom_levels> element is zoom level 0, the second zoom level is zoom level 1, and so on. These zoom level numbers are used in the tile requests to refer to the predefined zoom levels.

The <zoom_level> element specifies a predefined zoom level, and it has several attributes. The description attribute is optional and specifies the text description of the zoom level. The level_name attribute is optional and specifies the name of the zoom level. The scale attribute specifies the map scale of the zoom level; it is required if the attributes tile_width and tile_height are not defined. The tile_width and tile_height attributes specify the tile width and height, respectively, in map data units. The fetch_larger_tiles attribute is optional and specifies whether to fetch larger map images instead of the small map image tiles; a value of TRUE (the default) means that larger map images that may consist multiple map tiles will be fetched and broken into small map image tiles, which might save network round trips between the map tile server and the map services provider.

In the <zoom_level> element, you must specify either the scale attribute or both the tile_width and tile_height elements.

The <tile_bound> element within the <zoom_level> element optionally specifies the bounding box of the cached map tiles for the zoom level. The map tile server only fetches tiles inside this box, and returns a blank tile if the requested tile is outside this box. The bounding box is specified by a rectangle specified in map data coordinate system. The rectangle is specified by a <coordinates> element (explained earlier in this topic) If you specify the <tile_bound> element within the <zoom_level> element, it overrides the overall cache bounding box settings specified by the <tile_bound> element that is above it in the XML hierarchy.

The <auto_update> element defines how the tile layer's disk cache is to be automatically updated when spatial data in the base table is modified, For details, see Add the <auto_update> element to tile layer definition.

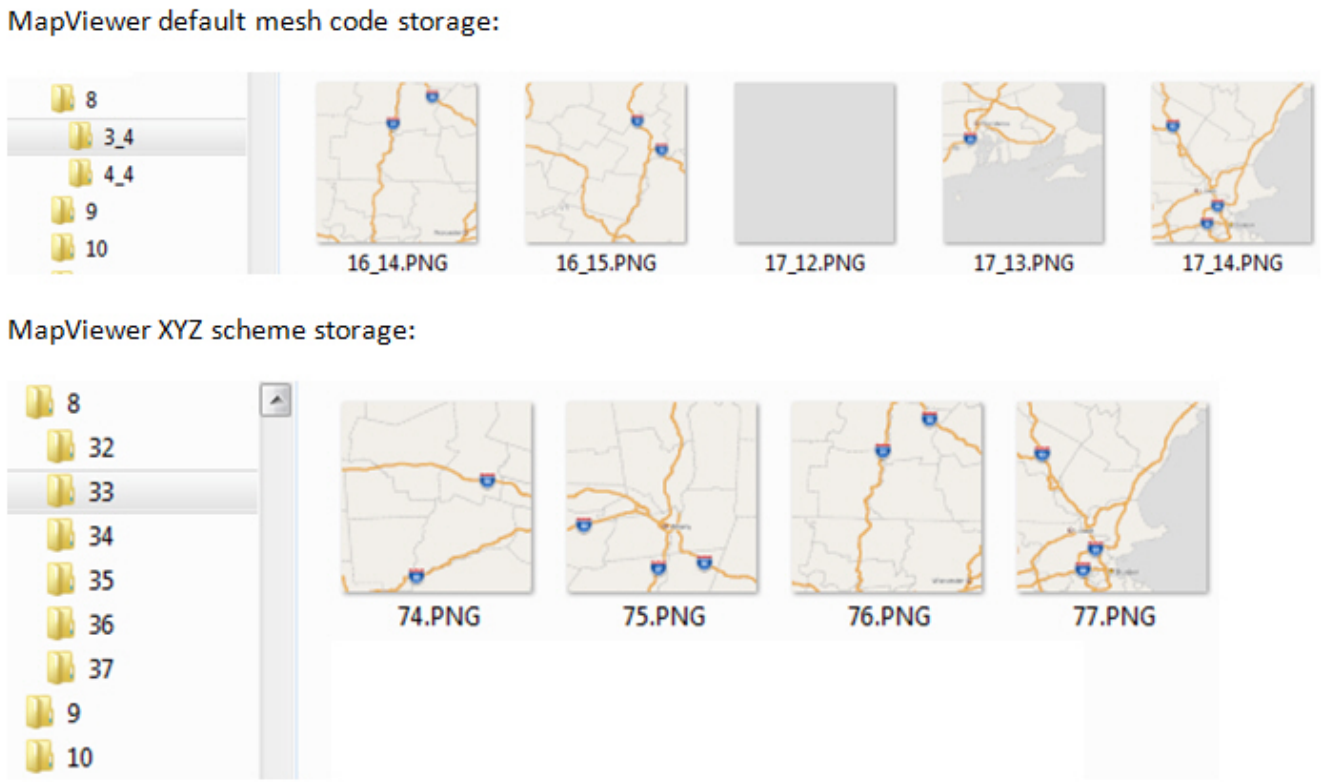
The <themes> element defines a layer based on themes (as opposed to a layer based on a base map). The themes to be used to render image tiles are listed in its subelements. In each subelement <theme>, the name attribute is required to specify a predefined theme in the metadata. The other two optional attributes, from_level and to_level, define the visibility of the that theme. The default values for those two attributes are from_level of 0 (which contains the least map detail) and to_level of the last level (which contains the most map detail).

3.2.2.3 Map Tile Storage Schemes: Internal Mesh Code or XYZ

The xyz_storage_scheme attribute of the <file_storage> element (described in Map Tile Layer Configuration) controls how the directory structure of map tiles in the disk cache is organized. The default scheme uses the map visualization component's internal mesh codes, but you can instead choose the XYZ storage scheme.

Figure Figure 3-4 shows both storage schemes.

Figure 3-4 Internal Mesh Code and XYZ Map Tile Storage Schemes



The details of the map visualization component internal mesh code are nor important for users. You need only know that there are two different storage scheme options, and that the XYZ storage scheme is analogous to the scheme used by several map data providers. Therefore, if you want to export tiles from some zoom levels for an offline project (local tile layer), if you find the XYZ storage scheme more intuitive, you can specify it.

In the map visualization component XYZ storage scheme, the naming of subdirectories is in the form `/z/y/x`, where `z` is the zoom level, `y` is the tile's row number, and `x` in the tile's column number. For example, in Figure 3-4, the map visualization component XYX storage scheme shows the images (`74.png`, `75.png`, `76,png`, `77.png`) in the tile columns for tile row 33 for zoom level 8.

3.2.2.4 Creating a Map Tile Layer Using an External Web Map Source

You can use an external web map source to create a map tile layer, as follows:

1. Log in to the map visualization component administrative console.
2. Click **Create Tile Layer**.
3. Select **External** as this tile layer's map image source, and click **Continue**.
4. On the Create a map tile layer for external map source page, enter the appropriate information:

   Name: Tile layer name. Example: `TILELAYER1`

   Data Source: Name of the data source for the tile layer. Example MVDEMO

Max browser tile cache age (hours): Maximum number of hours in the cache before a tile is refreshed. Example: 168

Map Service URL: Example: `http://mycorp.com:7001/mapviewer/wms`

Request Method: `HTTP GET`

Adapter class: `oracle.lbs.mapcache.adapter.WMSAdapter`

Jar file location: Example: `adapterlibs/`

Background: `transparent`

Adapter properties: (Check it and click **Add** multiple times as appropriate, entering the values for the following each time.)

- **datasource**: Example: `mvdemo`
- **version**: `1.1.1`
- **srs**: `EPSG:4326`
- **layers**: Example: `THEME_DEMO_COUNTIES,THEME_DEMO_HIGHWAYS`
- **format**: `image/png`
- **transparent**: `true`

Tile storage: Example: `C:\temp`

SRID: `8307`

Min X: `-180.0`

Max X: `180.0`

Min Y: `-90.0`

Max Y: `90.0`

Tile width (pixels): `256`

Tile height (pixels): `256`

File format: `PNG`

 # Zoom Levels: `10`

Minimum map scale: `1000`

Maximum map scale: `250000000`

5. Click **Submit** to create the tile layer.

   The message *Information: New map tile layer created successfully* is displayed.

To verify the created layer, you can click **Manage Tile Layers** on the left, select the tile layer, and click **View map / Manage tiles** to preview the map.

3.2.3 Map Cache Auto-Update

The map cache auto-update feature periodically updates cached map tiles when they become "dirty". A cached map tile becomes **dirty** when data in base tables is changed by an update, insert, or delete operation, because such changes may affect the shape, annotation, or choice of rendering style for the already cached map tiles.Updating a dirty tile invokes one of the following operations: a) Refresh: delete the cached dirty tiles and then re-fetch them; or b) Clear: only delete the cached dirty tiles. To enable automatic updating for a tile layer, you perform the following major steps:

1. Add the <dirty_tile_auto_update> element to the mapViewerConfig.xml configuration file.
2. Add the <auto_update> element to tile layer definition.
3. Create the dirty MBR table, base tables' log table, and triggers.

To test the automatic tile updating, follow the instructions in Start the map visualization component server and test the map cache auto-update feature.

- Add the <dirty_tile_auto_update> element to the mapViewerConfig.xml configuration file
- Add the <auto_update> element to tile layer definition
- Create the dirty MBR table, base tables' log table, and triggers
- Start the map visualization component server and test the map cache auto-update feature

3.2.3.1 Add the <dirty_tile_auto_update> element to the mapViewerConfig.xml configuration file

Add the `<dirty_tile_auto_update>` element as a child element of the `<map_tile_server>` element. For example:

```
<map_tile_server>
   <tile_storage default_root_path="/scratch/tilecache/"/>
   <dirty_tile_auto_update
     auto_update="true"
     auto_update_interval="1"
     recycle_interval="168"/>
</map_tile_server>
```

The `auto_update` attribute enables server's automatic update when set to `true` (the default is `false`). If this attribute is set to `true`, all qualified tile layers on the server will be automatically updated. To qualify, a tile layer must have a proper definition (see Add the <auto_update> element to tile layer definition).

The `auto_update_interval` attribute sets the recurring interval for checking the base tables' log table and the dirty MBR table. Its value is in minutes, and the default value is 1. The value should not be more than a few minutes, and it should not be very large (even if you update the base tables far less frequently, such as daily or weekly). The base tables' log table and the dirty MBR table should be created before starting the map visualization component server. Sample scripts for creating these two tables and related sequences and triggers are included in Create the dirty MBR table, base tables' log table, and triggers.

The `recycle_interval` attribute specifies how long to keep a processed row in the log table and dirty MBR table. Its value is in hours, and the default value is 168. Processed rows older than that will be deleted.

When the map visualization component starts, it loads the `<dirty_tile_auto_update>` element from `mapViewerConfig.xml` configuration file, and the element is applied to all tile layers of this server. If that element is not found, then the server disables any tile layer's auto-update capability.

3.2.3.2 Add the <auto_update> element to tile layer definition

Add an `<auto_update>` element after the `<zoom_levels>` element in a tile layer definition. You can add this element manually or using the map visualization component web console. For example:

```
<auto_update
     finest_level_to_refresh="15"
     dirty_mbr_batch="100"
     dirty_mbr_cap="1000">
     <dirty_mbr_table name="mbr_mcau"/>
     <logtable name="log_mcau_tl"/>
  </auto_update>
```

The `finest_level_to_refresh` attribute specifies the finest level to refresh. Levels starting from level 0, level 1, level 2, …, until this specified level will be refreshed, and dirty tiles in the remaining zoom levels will be cleared. If the data modifications in the base tables are often geographically small features (as most data modifications should be), such as changing the name of a restaurant or inserting a newly developed street, the value can be the finest zoom level found in the tile layer definition. For example, if there are 19 zoom levels from 0 to 18, then this attribute can be set to 18.

The `dirty_mbr_batch` attribute specifies the maximum number of rows from the dirty MBR table for an update. The default value is 100. This value prevents the server from getting a very large number of tiles to update at one time. Instead, if a large number of dirty tiles need a refresh or clear operation, these tiles will be processed in many updates, and one interval processes just one batch. To determine an optimal value for this attribute, consider the following factors:

- The `auto_update_interval` value (see Add the <dirty_tile_auto_update> element to the mapViewerConfig.xml configuration file)and how many tiles the server is able finish in the interval
- The amount of memory that the server can use
- The number of tile layers that are enabled for map cache auto update and the frequency and magnitude of the changes in their base tables

Because there is no formula for a precise calculation of an optimal `dirty_mbr_batch` value, the best practice is to set up an environment to test different settings. When selecting a value, consider the worst case scenario. Two extreme scenarios to avoid are: a) the value is so small that the server is idling after finishing a refresh operation while the number of dirty MBR rows in the dirty MBR table keeps growing; or b) the value is so large that the server runs out of memory, throws an out-of-memory exception, and shuts down all services.

The `dirty_mbr_cap` attribute specifies the maximum number of dirty tiles for a log table to generate in one interval. This constraint may affect the finest zoom level for refresh operation, and the remaining zoom levels are then set for a clear operation. The accumulation counter for dirty tiles starts from zoom level 0, to level 1, level 2, and so on, until the cap is reached or the finest level to refresh is reached.

When a cap is reached at level *n* before reaching the specified finest level to refresh, the already counted tiles in level 0, level 1, level 2, …, and level *n*-1 are for a refresh operation (delete and then re-fetch), and the current zoom level (level *n* in this example) and all other finer levels are for a clear operation (delete from the map tile cache and no re-fetch). For example, if the cap is given a value of 1000, the dirty tile counter reaches the cap at zoom level 4, then all the counted dirty tiles from level 0 to level 3 are for refresh. After that, each dirty tile in level 3 will be used to define a rectangle (the rectangular area the tile covers on the ground), and this rectangle is taken as an MBR to clear all zoom levels starting from level 4, level 5, and all other finer levels in this tile layer.

The `<dirty_mbr_table>` element specifies the name of the dirty MBR table, where the dirty MBRs are to be stored, retrieved, and updated. You need to manually create this table before starting the map visualization component server (see the example in Create the dirty MBR table, base tables' log table, and triggers).

The `<logtable>` element specifies the name of the base table's log table. If a tile layer depends on more than one base table (as is often the case), then every change in each base table should be inserted into this log table by its trigger; if the schema is accessed by more than one data source defined in `mapViewerConfig.xml`, then one change in a base table should insert one row for each data source.

It is recommended that each tile layer have its own log table and its own dirty MBR table. Both tables should be manually created before starting the server (see the example in Create the dirty MBR table, base tables' log table, and triggers).

3.2.3.3 Create the dirty MBR table, base tables' log table, and triggers

This section contains examples that, when taken together, show the actions to create the dirty MBR table, base tables' log table, and triggers. The example segments assume that a base map named DEMO_MAP is already defined and that there is one data source named MVDEMO accessing the schema.

The example code segments include explanatory comments, and they perform the following actions:

1. Create a tile layer that includes an `<auto_update>` element.

```
insert into user_sdo_cached_maps values(
'MCAU_TL',
'a test case for map cache auto update',
'',
'YES',
'YES',
'<map_tile_layer name="MCAU_TL" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3" fetch_larger_tiles="false">
   <internal_map_source data_source="mvdemo" base_map="DEMO_MAP"/>
   <coordinate_system srid="8307" minX="-180.0" maxX="180.0" minY="-90.0"
maxY="90.0"/>
   <tile_image width="256" height="256"/>
   <tile_dpi value="90.7142857"/>
   <tile_meters_per_unit value="111319.49079327358"/>
   <zoom_levels levels="19" min_scale="2132.729583849784"
max_scale="559082264.0287178">
   </zoom_levels>
   <auto_update
      finest_level_to_refresh="15"
      dirty_mbr_batch="100"
      dirty_mbr_cap="1000">
      <dirty_mbr_table name="mbr_MCAU_TL"/>
      <logtable name="log_MCAU_TL"/>
   </auto_update>
</map_tile_layer>',
'DEMO_MAP',
'');
commit;
```

2. Create a dirty MBR table and its trigger.

   The dirty MBR table stores the dirty MBRs for refresh and clear operations. This table is populated using the geometries from the log table. There is also a sequence and a trigger created for generating unique IDs for this table's ID column.

```
-- create the dirty MBR table
CREATE TABLE mbr_MCAU_TL (     -- dirty MBR table name
  id             number,       -- id, used for tracking the status
  datasource     varchar2(32), -- data source name
  tile_layer     varchar2(32), -- tile layer name
```

```
  logtable       varchar2(32), -- basetable's log-table
  refresh_status varchar2(1),  -- n: not refreshed, y: refreshed, p:pending, f:
failed
  clear_status   varchar2(1),  -- n: not cleared,   y: cleared,   p:pending, f:
failed
  mbr_to_clear   varchar2(1),  -- y/n: use tile's mbr for clearing finer levels
  zoom_level     number,       -- zoom level of this tile
  mx             number,       -- mesh x ordinate
  my             number,       -- mesh Y ordinate
  minx           number,       -- tile's minimum x coordinate
  miny           number,       -- tile's minimum y coordinate
  maxx           number,       -- tile's maximum x coordinate
  maxy           number,       -- tile's maximum y coordinate
  insert_time    Date,         -- when the tile MBR was inserted into this table
  update_time    Date          -- most recent update (refresh/clear) time
  );

-- create a sequence for mbr_MCAU_TL
CREATE SEQUENCE mbr_MCAU_TL_seq
  START WITH 1
  INCREMENT BY 1
  CYCLE
  MAXVALUE 9999999999;

-- create a trigger to get a unique id
create or replace trigger mbr_MCAU_TL_br
 before insert on mbr_MCAU_TL   -- before inserting the row
 referencing new as new old as old
 for each row  -- for each row
begin
  select mbr_MCAU_TL_seq.nextval INTO :new.id FROM dual;
end;
/
```

3. Create a log table.

   The base tables' log table is for recording the rows changed in the base tables of the tile layer. Because each tile layer depends on the data in its base tables when generating a map tile, any change made to the base table (such as modifications of geometries or changes to attributes values) may affect their representation in their corresponding map tile. The change log table records such changes through a trigger created on its base table.

   The following statements create a log table, a sequence, and a trigger on the table using the sequence to generate unique ID values.

```
-- create the log table
create table log_MCAU_TL(
  id number,
  geom0 sdo_geometry,  -- the affected geometry or its attributes, original geometry
  geomN sdo_geometry,  -- the affected geometry or its attributes, new geometry
  modified Date,       -- when the modified occurred
  status varchar2(1),  -- y: processed, n: not processed
  datasource varchar2(32), -- data source name, more than one ds may access the same
log
  tile_layer varchar2(32), -- tile layer name
  basetable  varchar2(32) -- base table name, more than one base table may insert
into this log
);

-- create a sequence for log_MCAU_TL
CREATE SEQUENCE log_MCAU_TL_seq
  START WITH 1
```

```
  INCREMENT BY 1
  CYCLE
  MAXVALUE 9999999999;

-- create a trigger for log_MCAU_TL to create a unique id
create or replace trigger log_MCAU_TL_br
 before insert on log_MCAU_TL   -- before inserting
 referencing new as new old as old
 for each row                          -- for each row
begin
  select log_MCAU_TL_seq.nextval INTO :new.id FROM dual;
end;
/
```

4. Create a trigger on each base table to insert changes into the log table.

   In a base table's trigger, any geometries inserted into the log table are transformed into the same spatial reference system (coordinate system) as the tile layer. If there are multiple data sources defined in the `mapViewerConfig.xml` configuration file accessing the same schema, then one INSERT statement should be used for each of these data sources in each trigger definition.

   The following statements make these assumptions:

   - One data source named MVDEMO is accessing the schema that contains the MCAU_TL tile layer. (Thus, there is only one INSERT statement in each trigger definition.)
   - The tile layer's spatial reference system (SRID) is 8307 (WGS 84 longitude/latitude).
   - There are four base tables to monitor for this tile layer: states, counties, interstates, and cities.

```
--states trigger
create or replace trigger states_MCAU_TL_ar
 after insert or update or delete on states   -- any change
 referencing new as new old as old
 for each row
 when (old.geom IS NOT NULL OR new.geom IS NOT NULL)
declare
  oldGeom SDO_GEOMETRY;
  newGeom SDO_GEOMETRY;
  tileSRID  number;
begin
   tileSRID := 8307;
   oldGeom  := :old.geom;
   if (:old.geom IS NOT NULL) then
     if (:old.geom.SDO_SRID != tileSRID) then
       select sdo_cs.transform(:old.geom, tileSRID)  into oldGeom from dual;
     end if;
   end if;
   newGeom:=:new.geom;
   if (:new.geom IS NOT NULL) then
     if (:new.geom.SDO_SRID!= tileSRID) then
       select sdo_cs.transform(:new.geom, tileSRID)  into newGeom from dual;
     end if;
   end if;

   insert into log_MCAU_TL (id, geomO, geomN, modified, status, datasource,
tile_layer, basetable)
       values(null, oldGeom, newGeom, sysdate, 'n', 'MVDEMO', 'MCAU_TL', 'states');
end;
/
```

```
--counties trigger
create or replace trigger counties_MCAU_TL_ar
 after insert or update or delete on counties
 referencing new as new old as old
 for each row
 when (old.geom IS NOT NULL OR new.geom IS NOT NULL)
declare
  oldGeom SDO_GEOMETRY;
  newGeom SDO_GEOMETRY;
  tileSRID  number;
begin
   tileSRID := 8307;
   oldGeom  := :old.geom;
   if (:old.geom IS NOT NULL) then
     if (:old.geom.SDO_SRID!=tileSRID) then
       select sdo_cs.transform(:old.geom, tileSRID)  into oldGeom from dual;
     end if;
   end if;
   newGeom:=:new.geom;
   if (:new.geom IS NOT NULL) then
     if (:new.geom.SDO_SRID!= tileSRID) then
       select sdo_cs.transform(:new.geom, tileSRID)  into newGeom from dual;
     end if;
   end if;
   insert into log_MCAU_TL (id, geomO, geomN, modified, status, datasource,
tile_layer, basetable)
        values(null, oldGeom, newGeom, sysdate, 'n', 'MVDEMO', 'MCAU_TL',
'counties');
end;
/

--interstates trigger
create or replace trigger interstates_MCAU_TL_ar
 after insert or update or delete on interstates
 referencing new as new old as old
 for each row
 when (old.geom IS NOT NULL OR new.geom IS NOT NULL)
declare
  oldGeom SDO_GEOMETRY;
  newGeom SDO_GEOMETRY;
  tileSRID  number;
begin
   tileSRID := 8307;
   oldGeom  := :old.geom;
   if (:old.geom IS NOT NULL) then
     if (:old.geom.SDO_SRID!=tileSRID) then
       select sdo_cs.transform(:old.geom, tileSRID)  into oldGeom from dual;
     end if;
   end if;
   newGeom:=:new.geom;
   if (:new.geom IS NOT NULL) then
     if (:new.geom.SDO_SRID!= tileSRID) then
       select sdo_cs.transform(:new.geom, tileSRID)  into newGeom from dual;
     end if;
   end if;

   insert into log_MCAU_TL (id, geomO, geomN, modified, status, datasource,
tile_layer, basetable)
        values(null, oldGeom, newGeom, sysdate, 'n', 'MVDEMO', 'MCAU_TL',
'interstates');

end;
```

```
/

--cities trigger
create or replace trigger cities_MCAU_TL_ar
 after insert or update or delete on cities
 referencing new as new old as old
 for each row
 when (old.location IS NOT NULL OR new.location IS NOT NULL)
declare
  oldGeom SDO_GEOMETRY;
  newGeom SDO_GEOMETRY;
  tileSRID  number;
begin
   tileSRID := 8307;
   oldGeom  := :old.location;
   if (:old.location IS NOT NULL) then
     if (:old.location.SDO_SRID!=tileSRID) then
       select sdo_cs.transform(:old.location, tileSRID)  into oldGeom from dual;
     end if;
   end if;
   newGeom:=:new.location;
   if (:new.location IS NOT NULL) then
     if (:new.location.SDO_SRID!= tileSRID) then
       select sdo_cs.transform(:new.location, tileSRID)  into newGeom from dual;
     end if;
   end if;

  insert into log_MCAU_TL (id, geomO, geomN, modified, status, datasource,
tile_layer, basetable)
      values(null, oldGeom, newGeom, sysdate, 'n', 'MVDEMO', 'MCAU_TL', 'cities');
end;
/
commit;
```

3.2.3.4 Start the map visualization component server and test the map cache auto-update feature

To test the automatic tile updating, start the map visualization component server and then change one or more rows in the base table.

1. Modify a row in the base table. For example:

   ```
   update cities set city='Worcester' where city='Worcester' and state_abrv='MA';
   ```

2. Check the log table. For example:

   ```
   select * from log_mcau_tl;
   ```

   The result should include a row that was just inserted by the base table's trigger.

3. Wait for about one interval (one minute in this example), then check the dirty MBR table. For example:

   ```
   select count(*) from mbr_mcau_tl;
   ```

   The result should include some dirty MBR rows inserted by the server.

You can also look for changes in the `refresh_status` column in the dirty MBR table. When rows are initially inserted, the status is set to `n` for *not processed*; then it changes to `p` for *pending* when they are being processed; and after the update is done, it changes to `y` for *processed*. Meanwhile, on the server you can see that the server has been updating the tiles (the server's logger needs to be set to `finest` level to see the finest logging information).

3.2.4 UTFGrid for Map Tiles: Including Text Information About Features

When a tile layer has UTFGrid enabled, a data set named UTFGrid becomes a "companion" of an image tile, containing text information about features in the image tile. This text information can be displayed by the browser when responding to a mouse event, such as mouse click on a map feature.

A UTFGrid data set is stored in JSON format, and has two components: (a) a grid data set that mirrors its companion image tile, and (b) attributes of all grid cells. The value at each grid cell serves as the key to link the image tile cell and its attributes. Each image pixel on a map is associated with a UTFGrid grid cell, and the cell's value indicates where its attributes (as text strings) can be retrieved. For storage efficiency, the value of each grid cell is encoded in a revised UTF-8 encoding scheme.

- Enabling the UTFGrid Option for a Tile Layer
- Encoding a Key and Decoding a Grid Cell's Value
- Building a UTFGrid Test Case

3.2.4.1 Enabling the UTFGrid Option for a Tile Layer

To enable the UTFGrid option for a map tile layer, then in the USER_ADO_CACHED_MAPS row for the tile layer, specify `utfgrid="true"` in the `<map_tile_layer>` element, and optionally specify the UTFGrid resolution with the `utfgrid_resolution` attribute (the default is 4). For example:

```
insert into user_sdo_cached_maps values(
'UTFGRID_TL',
'utfgrid enalbled test case ',
'',
'YES',
'YES',
'<map_tile_layer name="UTFGRID_TL" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3" fetch_larger_tiles="false"
    persistent_tiles="true" utfgrid="true" utfgrid_resolution="4">
   <internal_map_source data_source="mvdemo" base_map="UTFGRID_BASEMAP"
bgcolor="#dddddd" out_of_bounds_color="#eeddff"/>
   <tile_storage root_path="/temp" xyz_storage_scheme="true"/>
   <coordinate_system srid="8307" minX="-180.0" maxX="180.0" minY="-90.0" maxY="90.0"/>
   <tile_image width="256" height="256"/>
   <tile_dpi  value="90.7142857"/>
   <tile_meters_per_unit  value="111319.49079327358"/>
   <zoom_levels levels="19" min_scale="2132.729583849784" max_scale="559082264.0287178">
   </zoom_levels>
</map_tile_layer>',
'UTFGRID_BASEMAP',
'');
commit;
```

The `utfgrid_resolution` attribute determines how fine the grid cells are in an UTFGrid data set, and the default value of 4 indicates that one grid cell represents 4 by 4 image pixels in its companion image tile. For example, if 256x256 is an image's dimension, then the grid's dimension will be 64x64, and a grid cell at row=10, column=20 represents the pixels in the tile image from row 40 to row 43 and from column 80 to column 83 (note that one grid cell represents 16 image tile pixels). When multiple features in the map image tile fall into one grid cell at the indicated resolution, then the feature with the majority or plurality of the pixels is assigned to the grid cell. The value stored in the grid cell is encoded using UTF-8 encoding.

If a tile layer has UTFGrid enabled, when the map server generates a map image tile, it will also generate an UTFGrid data set stored in JSON format. When a client requests a map image tile, both the image tile and its UTFGrid JSON file will be returned in the response.

In the map visualization component server, the UTFGrid JSON files are stored in the same location with their image tiles. For example, if an image tile is cached at `/mapcache/MVDEMO.UTFGRID_TL/0/1/2.PNG`, you should also see its companion UTFGrid file `/mapcache/MVDEMO.UTFGRID_TL/0/1/2.JSON`.

This UTFGrid option is currently not supported if the image tiles are stored in a database table (see Store the tiles in a database table) when disk cache is disabled for a tile layer (see Stream the tiles directly without storing them).

3.2.4.2 Encoding a Key and Decoding a Grid Cell's Value

A tile layer can contain multiple data layers or themes. Each theme will have its own grid data set, and an UTFGrid JSON object may contain multiple grid data sets. In each theme's grid, features shown in its image tile are also "drawn" using styles to render each feature. As for its associated image tile, a feature is drawn using its key, and the key is an ordinal number assigned to each feature. This number is used to assign a UTF-8 encoded value for the cell's value.

When a mouse event is triggered on a feature in an image tile, the image pixel's corresponding grid cell can be located, and the cell's value (an UTF-8 encoded value) can then be retrieved. This UTF-8 encoded value can then be decoded to derive its key, an ordinal number. Using this key, the attribute stored can be obtained. (See the examples near the end of Building a UTFGrid Test Case.)

3.2.4.3 Building a UTFGrid Test Case

This test case focuses on the server side. It has two major steps: creating a UTFGrid-enabled tile layer, and sending some hard-coded requests to the server. (In a real application, you need to have a client side map application to make use of the tile layer.)

1. Create a UTFGrid-enabled tile layer.

   If all the spatial base tables and styles exist in the metadata, the following example creates four themes, a basemap, and an UTFGrid-enabled tile layer. Because the feature attributes stored in an UTFGrid JSON object come from the `<hidden_info>` element of a theme, the `<hidden_info>` element is defined in three of the four themes. (If none of the four themes has the `<hidden_info>` element defined, then enabling this tile layer's UTFGrid option would not make sense, because the UTFGrid object would be empty and creating it would still consume server resources.)

```
-- insert  theme 1
insert into USER_SDO_THEMES (name, description, base_table, geometry_column,
styling_rules)
values (
 'UTFGRID_THEME_DEMO_STATES',
 'for utfgrid testing',
 'STATES',
 'GEOM',
 '<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="STATE" name="State"/>
    <field column="STATE_ABRV" name="Abrv."/>
    <field column="TOTPOP" name="Population"/>
  </hidden_info>
```

```
  <rule>
    <features style="C.S02_COUNTRY_AREA"> </features>
    <label column="STATE_ABRV" style="T.S02_STATE_ABBREVS"> 1 </label>
  </rule>
</styling_rules>');


-- insert theme 2
insert into USER_SDO_THEMES (name, description, base_table, geometry_column,
styling_rules)
values (
 'UTFGRID_THEME_DEMO_COUNTIES',
 'for utfgrid testing',
 'COUNTIES',
 'GEOM',
 '<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <!--<hidden_info>
    <field column="COUNTY" name="County"/>
    <field column="FIPSSTCO" name="Fips"/>
    <field column="TOTPOP" name="Population"/>
    <field column="STATE_ABRV" name="State"/>
  </hidden_info>-->
  <rule>
    <features style="L.S06_BORDER_STATE"> </features>
  </rule>
</styling_rules>');

-- insert theme 3
insert into USER_SDO_THEMES (name, description, base_table, geometry_column,
styling_rules)
values (
 'UTFGRID_THEME_DEMO_HIGHWAYS',
 'for utfgrid testing',
 'INTERSTATES',
 'GEOM',
 '<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="HIGHWAY" name="Highway"/>
    <field column="ROUTEN" name="No."/>
  </hidden_info>
  <rule>
    <features style="L.S04_ROAD_INTERSTATE"> </features>
    <label column="routen" style="M.HWY_USA_INTERSTATE_NARROW"> (3-length(routen)) </
label>
  </rule>
</styling_rules>');

-- insert theme 4
insert into USER_SDO_THEMES (name, description, base_table, geometry_column,
styling_rules)
values (
 'UTFGRID_THEME_DEMO_CITIES',
 'for utfgrid testing',
 'CITIES',
 'LOCATION',
 '<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="CITY" name="City"/>
    <field column="POP90" name="Population"/>
```

```
  </hidden_info>
  <rule>
    <features style="M.ALL_CITY_L2"> (pop90 between 200000 AND 1000000 ) </features>
    <label column="city" style="T.S07_CITIES_L2"> 1 </label>
  </rule>
  <rule>
    <features style="M.ALL_CITY_L3"> (pop90 between 0 and  200000) </features>
    <label column="city" style="T.S07_CITIES_L3"> 1 </label>
  </rule>
</styling_rules>');

-- insert a basemap
INSERT INTO USER_SDO_MAPS (name, description, definition)
values (
 'UTFGRID_BASEMAP',
 'for utfgrid testing',
 '<?xml version="1.0" standalone="yes"?>
  <map_definition>
    <theme name="UTFGRID_THEME_DEMO_STATES" min_scale="1.5E8" max_scale="0.0"
scale_mode="RATIO"/>
    <theme name="UTFGRID_THEME_DEMO_COUNTIES" min_scale="8500000.0" max_scale="0.0"
scale_mode="RATIO"/>
    <theme name="UTFGRID_THEME_DEMO_HIGHWAYS" min_scale="4.5E7" max_scale="0.0"
scale_mode="RATIO"/>
    <theme name="UTFGRID_THEME_DEMO_CITIES" min_scale="7500000.0" max_scale="0.0"
scale_mode="RATIO"/>
  </map_definition>' );

-- insert the UTFGrid enabled tile layer
insert into user_sdo_cached_maps values(
 'UTFGRID_TL',
 'a utfgrid ',
 '',
 'YES',
 'YES',
 '<map_tile_layer name="UTFGRID_TL" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3" fetch_larger_tiles="false"
    persistent_tiles="true" utfgrid="true" utfgrid_resolution="8">
   <internal_map_source data_source="mvdemo" base_map="UTFGRID_BASEMAP"
bgcolor="#dddddd" out_of_bounds_color="#eeddff"/>
   <tile_storage root_path="/temp" xyz_storage_scheme="true"/>
   <coordinate_system srid="8307" minX="-180.0" maxX="180.0" minY="-90.0"
maxY="90.0"/>
   <tile_image width="256" height="256"/>
   <tile_dpi  value="90.7142857"/>
   <tile_meters_per_unit  value="111319.49079327358"/>
   <zoom_levels levels="19" min_scale="2132.729583849784"
max_scale="559082264.0287178">
   </zoom_levels>
</map_tile_layer>',
 'UTFGRID_BASEMAP',
 '');
commit;
```

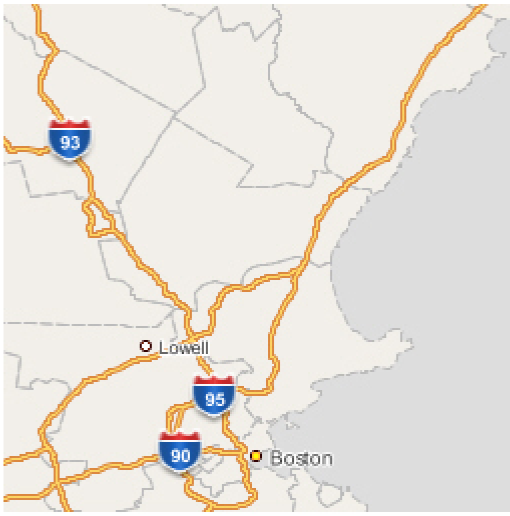
2. Request the map image and its UTFGrid object.

   In an actual application, requesting a map image tile and its companion UTFGrid JSON file is handled by the map visualization component HTML5 API; however, for illustration purposes, two substeps with hard-coded requests are shown here.

   a. Request an image tile from the server. For example:

```
http://localhost:8080/mapviewer/mcserver?
request=gettile&format=png&zoomlevel=8&mapcache=MVDEMO.UTFGRID_TL&mx=77&my
=94
```

The server's response:



b. Request the image tile's companion UTFGrid from the server. For example:

```
http://localhost:8080/mapviewer/mcserver?
request=getutfgrid&format=json&zoomlevel=8&mapcache=MVDEMO.UTFGRID_TL&mx=7
7&my=94
```

The server's response:


```
{"UTFGRID_THEME_DEMO_STATES":
{"keys":["1","2","3"],
"data":{
"1":{"State":"New Hampshire","Abrv.":"NH","Population":"1109252"},
"2":{"State":"Maine","Abrv.":"ME","Population":"1227928"},
"3":{"State":"Massachusetts","Abrv.":"MA","Population":"6016424"}},
"grid":[
"!!!!!!!!!!!!!!!!!###############",
"!!!!!!!!!!!!!!!!!###############",
"!!!!!!!!!!!!!!!!!############## ",
"!!!!!!!!!!!!!!!!!###############",
"!!!!!!!!!!!!!!!!!############## ",
"!!!!!!!!!!!!!!!!!#############  ",
"!!!!!!!!!!!!!!!!!!#########     ",
"!!!!!!!!!!!!!!!!!!!########     ",
"!!!!!!!!!!!!!!!!!!!!#######     ",
"!!!!!!!!!!!!!!!!!!!!######      ",
"!!!!!!!!!!!!!!!!!!!!######      ",
"!!!!!!!!!!!!!!!!!!!!!!###       ",
"!!!!!!!!!!!!!!!!!!!!!!!#        ",
"!!!!!!!!!!!!!!!!!!!!!!!         ",
"!!!!!!!!!!!!!!!!!!!!!!          ",
"!!!!!!!!!!!!!!!!!!!!!!          ",
"!!!!!!!!!!!!!!!!!$$!!           ",
"!!!!!!!!!!!!!!!$$$$$$           ",
"!!!!!!!!!!!!$$$$$$$$$           ",
"!!!!!!!!!!!$$$$$$$$$$$          ",
"$$$$$!!!!$$$$$$$$$$$$$$ $$      ",
"$$$$$$$$$$$$$$$$$$$$$$$$$$      ",
"$$$$$$$$$$$$$$$$$$$$$$$$$$      ",
```

```
"$$$$$$$$$$$$$$$$$$$$$$$$       ",
"$$$$$$$$$$$$$$$$$$$$           ",
"$$$$$$$$$$$$$$$$$$$            ",
"$$$$$$$$$$$$$$$$$$             ",
"$$$$$$$$$$$$$$$$$              ",
"$$$$$$$$$$$$$$$$$              ",
"$$$$$$$$$$$$$$$$$$$            ",
"$$$$$$$$$$$$$$$$$$$$           ",
"$$$$$$$$$$$$$$$$$$$$$$         "]},
"UTFGRID_THEME_DEMO_HIGHWAYS":{
"keys":["1","2","3","4","5","6","7","8","9"],
"data":{
"1":{"Highway":"I 93","No.":"93"},
"2":{"Highway":"I 95","No.":"95"},
"3":{"Highway":"I 89","No.":"89"},
"4":{"Highway":"I 293","No.":"293"},
"5":{"Highway":"I 495","No.":"495"},
"6":{"Highway":"I 190","No.":"190"},
"7":{"Highway":"88","No.":"99"},
"8":{"Highway":"I 290","No.":"290"},
"9":{"Highway":"I 90","No.":"90"}},
"grid":[
"  !                          ## ",
"  !                        ###  ",
"  !!                       ##   ",
"  !!                      ##    ",
"  !                      ##     ",
"  !                      ##     ",
"  !!                    ##      ",
"   !!                   ##      ",
"$  !!                   #       ",
"$$$$!                  ##       ",
"    !!                ###       ",
"     !              ###         ",
"     %!             ##          ",
"     %!!           ##           ",
"     %%!           #            ",
"       !!         ##            ",
"        !!        ##            ",
"         !!   &&&&#             ",
"          !! &&& ##             ",
"          !! &   #              ",
"           !&&  ##              ",
"         &&&!   ##              ",
"       &&&& !   ##              ",
"    &&&&    !!  ##              ",
"'  &&&     ##!###               ",
"' &&&     #((!!                 ",
"  &&      #( !!                 ",
"  &&      (  !!!                ",
")))&  *********!                ",
")  &***** ##   !                ",
"*****      ##  !                ",
"***&&      ##!!!                "]},
"UTFGRID_THEME_DEMO_CITIES":{
"keys":["1","2"],
"data":{
"1":{"City":"Lowell","Population":"103439"},
"2":{"City":"Boston","Population":"574283"}},
"grid":[
"                                ",
"                                ",
```

```
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"        !!                      ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"                                ",
"               ##               ",
"                                ",
"                                ",
"                                ",
"                                "]},
"resolution":"8","rows":"32","columns":"32"}
```

The preceding response shows a UTFGrid object that contains properties for the themes `UTFGRID_THEME_DEMO_STATES`, `UTFGRID_THEME_DEMO_HIGHWAYS`, and `UTFGRID_THEME_DEMO_CITIES`. (The theme `UTFGRID_THEME_DEMO_COUNTIES` did not define the `<hidden_info>` element, and so nothing about that theme is listed in the response.)

3.2.5 External Map Source Adapter

An external map source adapter is the interface between a map tile server and an external map services provider. When a map image tile needs to be fetched from the external map services provider, the map tile server calls the adapter with information about the zoom level, size, and location of the tile. The adapter then constructs a provider-specific request, sends the request to the external map services provider, and return the resulting image tile to the map tile server.

The external map source adapter is a Java class that must extends the abstract Java class `oracle.mapviewer.share.mapcache.MapSourceAdapter`, which is defined as follows:

```
public abstract class MapSourceAdapter
{
   public abstract String getMapTileRequest(TileDefinition tile);
   public byte[] getTileImageBytes(TileDefinition tile) ;
   public Properties getProperties() ;
}
```

An adapter that extends this class must implement the following method:

- `public String getMapTileRequest(TileDefinition tile)`

This method should implement the logic to construct the HTTP request string that can be sent to the map services provider to fetch the map image tile. For example, if the URL of a map tile is `http://myhost/mymapserver?par1=v1&par2=v2&par3=v3`, the HTTP request string returned by this method should be `par1_v1&par2=v2&par3=v3`.

When the map tile server cannot find a specific map tile, it calls the `getTileImageBytes` method to fetch the binary data of the tile image, and that method calls the `getMapTileRequest` method to construct the map tile request before fetching the tile. The `getMapTileRequest` method takes one parameter: a `TileDefinition` object that specifies the zoom level, bounding box, image size and image format of the requested tile. This method returns the HTTP request string.

The map source adapter also inherits all methods implemented in class `MapSourceAdapter`. Among them, the following methods are more important than the others:

- `public byte[] getTileImageBytes(TileDefinition tile)`

  This method fetches the actual binary map tile image data from the external map service provider. This method is already implemented. It calls the abstract method `getMapTileRequest` to construct the map tile request and sends the request to the external map services provider. If the map tiles cannot be fetched by sending HTTP requests, you can override this method to implement the appropriate logic to fetch an image tile from the map source. This method takes one parameter: a `TileDefinition` object that specifies the zoom level, bounding box, image size, and image format of the requested tile. This method returns the binary tile image data encoded in the image format specified in the map tile layer configuration settings.

- `public Properties getProperties()`

  This method returns the provider-specific parameters defined in the map tile layer configuration settings explained in Map Tile Layer Configuration.

The `MapSourceAdapter` and `TileDefinition` classes are packaged inside `mvclient.jar`, which can be found under the directory *`$MAPVIEWER_HOME`*`/web/WEB/lib`.

Example 3-5 shows an external map source adapter.

Example 3-5 External Map Source Adapter

```
/**
 * This is a sample map source adapter that can be used to fetch map
 * tiles from a map visualization component instance.
 */
package mcsadapter ;

import java.awt.Dimension;
import java.net.URL;
import java.util.Properties;
import oracle.lbs.mapclient.MapViewer;
import oracle.lbs.mapcommon.MapResponse;
import oracle.mapviewer.share.mapcache.*;

/**
 * The map source adapter must extend class
 * oracle.lbs.mapcache.cache.MapSourceAdapter.
 */

public class MVAdapter extends MapSourceAdapter
{
  /**
   * Gets the map tile request string that is to be sent to the map
   * service provider URL.
```

```
   * @param tile tile definition
   * @return request string
   */
  public String getMapTileRequest(TileDefinition tile)
  {
    // Get map source specified parameters
    Properties props = this.getProperties() ;
    String dataSource = props.getProperty("data_source") ;
    String baseMap = props.getProperty("base_map") ;
    // Use oracle.lbs.mapclient.MapViewer to construct the request string
    MapViewer mv = new MapViewer(this.getMapServiceURL()) ;
    mv.setDataSourceName(dataSource);
    mv.setBaseMapName(baseMap);
    mv.setDeviceSize(new Dimension(tile.getImageWidth(),
                                   tile.getImageHeight()));
    mv.setCenterAndSize(tile.getBoundingBox().getCenterX(),
                        tile.getBoundingBox().getCenterY(),
                        tile.getBoundingBox().getHeight());
    int format = MapResponse.FORMAT_PNG_STREAM ;
    String req = null ;
    switch(tile.getImageFormat())
    {
      case TileDefinition.FORMAT_GIF:
        mv.setImageFormat(MapResponse.FORMAT_GIF_URL);
        req = mv.getMapRequest().toXMLString().replaceFirst(
                        "format=\"GIF_URL\"", "format=\"GIF_STREAM\"") ;
        break ;
      case TileDefinition.FORMAT_PNG:
        mv.setImageFormat(MapResponse.FORMAT_PNG_URL);
        req = mv.getMapRequest().toXMLString().replaceFirst(
                        "format=\"PNG_URL\"", "format=\"PNG_STREAM\"") ;
        break ;
      case TileDefinition.FORMAT_JPEG:
        mv.setImageFormat(MapResponse.FORMAT_JPEG_URL);
        req = mv.getMapRequest().toXMLString().replaceFirst(
                        "format=\"JPEG_URL\"", "format=\"JPEG_STREAM\"");
        break ;
    }

    byte[] reqStr = null ;
    try
    {
      reqStr = req.getBytes("UTF8") ;
    }
    catch(Exception e)
    {}
    // Return the request string.
    return "xml_request="+ new String(reqStr);
  }
}
```

Example 3-6 shows the implementation of the `MapSourceAdapter.getTileImageBytes` method.

Example 3-6 MapSourceAdapter.getTileImageBytes Implementation

```
/**
 * Fetches the map image tile from the external map service provider by
 * sending the HTTP map tile request to the map service provider, and
 * return the binary tile image data. You can rewrite this method so that
 * the adapter can fetch the tile from an external map service provider
 * that does not accept HTTP requests at all.
 * @param tile the tile definition
 * @return the binary tile image data.
```

```
 * @throws Exception
 */
public byte[] getTileImageBytes(TileDefinition tile)
  throws Exception
{
  // construct request string
  String request = getMapTileRequest(tile) ;

  if(request == null)
  {
    throw new Exception("Null map tile request string in map source adapter!") ;
  }

  // set proxy settings
    Proxy proxy = null ;

  /* If the proxyHost is "NONE", the request is sent directly to the
   * external server. If the proxyHost is a valid host, that host will
   * be used as the proxy server. If the proxyHost is empty of omitted,
   * the global proxy setting in mapViewerConfig.xml will be in effect.
   */
  boolean noProxy = "NONE".equalsIgnoreCase(getProxyHost()) ;
  if(getProxyHost()!=null && !noProxy)
  {
    SocketAddress addr = new InetSocketAddress(proxyHost, proxyPort);
    proxy = new Proxy(Proxy.Type.HTTP, addr);
  }

  // send the request and get the tile image binary
  PrintWriter wr = null ;
  BufferedInputStream bis = null;
  try
  {
    String urlStr = mapServiceURL ;
    if("GET".equalsIgnoreCase(httpMethod))
      urlStr = mapServiceURL + "?" + request ;
    log.finest("http "+httpMethod+": "+urlStr);

    URL url = new URL(urlStr);
    // Open a URL connection based on current proxy setting
    URLConnection conn =
      proxy!=null? url.openConnection(proxy):
                   (noProxy? url.openConnection(Proxy.NO_PROXY):
                             url.openConnection()) ;
    conn.setConnectTimeout(timeOut);
    if("GET".equalsIgnoreCase(getHTTPMethod()))
      conn.connect();
    else
    {
      conn.setDoOutput(true);
      wr = new PrintWriter(conn.getOutputStream());
      wr.print(request);
      wr.flush();
      wr.close();
      wr = null ;
    }
    bis = new BufferedInputStream(conn.getInputStream());
    byte[] result = toBytes(bis) ;
    bis.close();
    bis = null ;
    return result;
  }
```

```
  catch(Exception ioe)
  {
    throw new Exception("Failed to fetch external map tile.", ioe);
  }
  finally
  {
    try
    {
      if(bis != null)
      {
        bis.close();
        bis = null;
      }
      if(wr != null)
      {
        wr.close();
        wr = null;
      }
    }
    catch(IOException ioee)
    {
      throw ioee;
    }
  }
}
```

3.3 Vector Tile Server

The vector tile server is a vector tile caching engine that fetches, caches, and serves pregenerated vector tiles.

A vector tile is encoded in ProtoBuf format (see https://en.wikipedia.org/wiki/Protocol_Buffers). This format allows serializing the structured data and it is neutral to programming languages and operating systems.

Currently, the vector tiles support only the World Mercator projection. Its Oracle SRID is 3857. Its tiling coordinates, including the number of zoom levels, and the map's data bounds, are identical to the Google Maps tiling scheme.

One vector tile is only for one layer, and it cannot have more than one layer. In Map Visualization Component terminology, a layer is called as a theme, and thus one vector tile is only for one theme; a vector tile cannot have two or more themes.

Example 3-7 Use mapbox-gl JavaScript Library to Display a Vector Tile

This example uses the `mapbox-gl` JavaScript library and displays a vector tile layer provided by the Map Visualization Component's vector tile server.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8' />
    <title>Simple Demo Using Oracle Visualization Vector Tile Service</title>
    <meta name='viewport' content='initial-scale=1,maximum-scale=1,user-
scalable=no' />
    <script src="https://api.mapbox.com/mapbox-gl-js/v1.6.1/mapbox-gl.js"></
script>
    <link href="https://api.mapbox.com/mapbox-gl-js/v1.6.1/mapbox-gl.css"
rel="stylesheet" />
```

```
    <style>
        body { margin:0; padding:0; }
        #map { position:absolute; top:0; bottom:0; width:100%; }
    </style>
  </head>
  <body>
    <style>
      .mapboxgl-popup {
          max-width: 400px;
          font: 12px/20px 'Helvetica Neue', Arial, Helvetica, sans-serif;
        }
    </style>
    <div id='map'></div>

    <script>
      var mapInstance = new mapboxgl.Map({
                                container: 'map',
                                // this defines where to load the background
map tiles.
                                style: 'map-styles/world_map/
oracle_world_map.json',
                                center: [-122, 37],
                                zoom: 4.5
                               });
      var addMapViewerPointLayer = function() {
        var mvURL ="http://"+document.location.host+"/mapviewer";
        var mvDataSrc = "mvdemo";
        var mvTheme = "CUSTOMERS";
        var vectorTileUrl = mvURL+"/vt/"+mvDataSrc+"/{z}/{x}/{y}.mvt?
t="+mvTheme ;
        var layerDef1 = {
                        "id" : "my-layer-1",
                        "type" : "circle",
                        "source": {
                            "type" : "vector",
                            "tiles" : [vectorTileUrl], // a vector tile source
                            "minzoom" : 0,
                            "maxzoom" : 19
                        },
                        "source-layer" : mvTheme,
                        "paint" : {
                             "circle-radius": 10,
                             "circle-color": 'rgb(111, 111, 111)',
                             "circle-opacity": .8
                          }
                      };
                      mapInstance.addLayer(layerDef1);
                  };
        mapInstance.on('load', addMapViewerPointLayer);
    </script>
  </body>
</html>
```

In this example, the content of the style property defined in the file `oracle_world_map.json` is as follows:

```
{
    "version": 8,
    "id"     : "elocation-mercator-worldmap-mb",
    "name"   : "World Map (Oracle)",
    "metadata" : {
        "sgtech:version": "19.1.0",
        "sgtech:sources:type": "raster"
    },

    "sources": {
        "elocation-worldmap-mb": {
            "type": "raster",
            "attribution": "<a target=''_blank'' href=''https://
elocation.oracle.com/elocation/legal.html''> &copy; 2019 Oracle Corporation
</a>   <a target=''_blank'' href=''http://elocation.oracle.com/elocation/
legal.html''> Map data &copy; 2019 HERE </a>",
            "tiles": [
                "https://elocation.oracle.com/mapviewer/mcserver/
ELOCATION_MERCATOR/world_map_mb/{z}/{y}/{x}.png"
            ],
            "tileSize": 256
        }
    },

    "layers": [
        {
            "id": "Oracle World Map",
            "type": "raster",
            "source": "elocation-worldmap-mb",
            "minzoom": 0,
            "maxzoom": 19
        }
    ],
    "sprite": "http://localhost:8080/mapviewer/private/mapbox-demos/map-
styles/world_map/sprites/sgtech-maki",
    "glyphs": "http://localhost:8080/mapviewer/private/mapbox-demos/map-
styles/world_map/fonts/glyphs/{fontstack}/{range}.pbf"
}
```

As you see, a vector tile request sent to the `mapviewer` server is formulated using this template:

```
"http://"+document.location.host+"/mapviewer"+"/vt/"+mvDataSrc+"/{z}/{x}/
{y}.mvt?t="+mvTheme ;
```

When the example is running, if you check the browser's network traffic, you will see vector tile requests similar to the following:

```
http://localhost:8080/mapviewer/vt/mvdemo/4/2/6.mvt?t=CUSTOMERS
```

For any data source accessible by the server, its predefined geometry themes are accessible by the vector tile server.

4

Oracle Maps

Oracle Maps is the name for a suite of technologies for developing high-performance interactive web-based mapping applications.

Oracle Maps is included with the map visualization component.

The map visualization component allows users who have a valid license and comply with map tile providers' Terms of Use to display map tiles from those licensed services.

- Oracle Maps JavaScript API
  The Oracle Maps JavaScript client is a browser-based map visualization engine that works on top of the map tile server and that uses HTML5 technology.
- Developing Oracle Maps Applications
  This topic describes how to develop Oracle Maps applications.
- Using Google Maps
  Applications can display Google Maps tiles as a built-in map tile layer, by creating and adding to the map window an instance of `MVGoogleTileLayer` .
- Transforming Data to a Spherical Mercator Coordinate System
  Popular online map services such as Google Maps use a spherical Mercator projection for their maps.
- Using Dynamic Tile Layers
  A **dynamic tile layer** is a tile layer defined on the client side.

4.1 Oracle Maps JavaScript API

The Oracle Maps JavaScript client is a browser-based map visualization engine that works on top of the map tile server and that uses HTML5 technology.

It implements the following functions:

- Fetching map tiles from the map tile server and displaying them as a map tile layer in the web browser.
- Controlling user interaction, such as dragging for map navigation, clicking features, drawing rectangles, and redlining.

  Drawing a rectangle refers to the application user creating a rectangle by clicking and holding the mouse button at one corner of the rectangle, dragging the mouse to the diagonally opposite corner, and releasing the mouse button.

  Redlining refers to the application user creating a polygon or polyline by clicking the mouse button and then moving the mouse and clicking multiple times, with each click extending the redline by a straight line. (Redline drawings are often rendered in red, although you can specify a line style that uses any color.)

To access these functions, use the JavaScript API, which consists of several JavaScript classes. For detailed information about all classes in the Oracle Maps JavaScript API, see the Javadoc-style reference documentation, which is available at:

`http://<host>:<port>/mapviewer/jslib/<version>/apidoc/index.html`

4.1.1 About the Oracle Maps JavaScript API

The Oracle Maps JavaScript API takes advantage of the capabilities of modern browsers and HTML5. Some of its features include:

- Built-in support of various third party map tile services, such as maps.oracle.com, Nokia Maps, OpenStreet Maps, and other mapping service providers
- Rich client side rendering of geospatial data with on-the-fly application of rendering styles and effects such as gradients, animation, and drop-shadows
- Autoclustering of large number of points and client side heat map generation
- Client side feature filtering based on attribute values as well as spatial predicates (query windows)
- A rich set of built-in map controls and tools, including a customizable navigation bar and information windows, configurable layer control, and red-lining and distance measurement tools

Existing server-side predefined styles and themes will work with the API. For example, the following code snippet creates an interactive vector layer based on a predefined theme `mvdemo.customers`, which has an associated predefined style:

```
var baseURL  = "http://"+document.location.host+"/mapviewer";
var layer = new OM.layer.VectorLayer("layer1",
 {
     def:{
                type:OM.layer.VectorLayer.TYPE_PREDEFINED,
                dataSource:"mvdemo",
                theme:"customers",
                url: baseURL
            }
  });
```

The API has the following top-level classes and subpackages, all of which are in the namespace `OM`:

- The `Map` class is the main class of the API.
- The `Feature` class represents individual geo features.
- The `MapContext` class a top-level class encapsulating some essential contextual information, such as the current map center point and zoom level. It is typically passed into event listeners.
- The `control` package contains all the map controls, such as navigation bar and overview map.
- The `event` package contains all the map and layer event classes.
- The `filter` package contains all the client-side filters (spatial or relational) for selecting, or subsetting, the displayed vector layer features.
- The `geometry` package contains various geometry classes.
- The `layer` package contains various tile and vector layer classes. The tile layer classes include access to a few online map services such as Oracle, Nokia, and OpenStreetMap. The vector layers are interactive feature layers.
- The `infowindow` package contains the customizable information windows and their styles.

- The `style` package contains styles applicable to vector data on the client side. It also includes visual effects such as animation, gradients, and drop shadows.
- The `tool` package contains various map tools such as for distance measuring, red-lining, and geometry drawing.
- The `universe` package contains built-in, or predefined, *map universes*. A map universe defines the bounding box and set of zoom level definitions for the map content.
- The `util` package contains various utility classes.
- The `visualfilter` package provides an interface for the various visual effects, such as gradients and drop shadows.

`OM.Map` is the main entry class for all map operations inside the web browser. This and other classes provide interfaces for adding application-specific logic, operations, and interactivity in web mapping applications. The application logic and operations can include the following:

- Create a map client instance and associate it with the map container DIV object created in the web page.
- Configure map parameters such as map center and map zoom level.
- Optionally, create and manipulate map tile layers. A map tile layer is not required. An application can have only interactive vector layers using a custom Universe that programmatically defines the zoom levels and scales.
- Create and manipulate vector layers.
- Display an information window on the map.
- Create fixed map decorations, such as a map title, a copyright note, and map controls.
- Access built-in utilities such as a navigation panel, rectangle or circle tool, scale bar, and overview map panel.
- Use event listeners to customize event handling and thus map interactions.

Other notes and considerations on using the Oracle Maps API:

- The "FOI server" in a previous version of the API is now a data server that streams the vector geometries and attributes for features to the client for local rendering. Therefore, the "FOI layers" in a previous version are now called **vector layers**.
- A background map tile layer is not required in order to display interactive vector layers. For example, an application can display a thematic map of states (such as color-filled by population quintile) with no background tile layer.
- The API depends on and includes `JQuery` and `JQueryUI`. So, `oraclemapsv2.js` includes `jquery-1.7.2.min.js` and `jquery-ui-1.8.16.min.js`. If your application also uses JQuery and JQueryUI and includes them already, then use the file `oraclemapsv2_core.js` in the `<script>` tag instead to load the Oracle Maps library. That is, use the following:

  ```
  <script src="/mapviewer/jslib/v2/oraclemapsv2_core.js"></script>
  ```

  instead of:

  ```
  <script src="/mapviewer/jslib/v2/oraclemapsv2.js"></script>
  ```

For information about developing applications using the API, see Developing Oracle Maps Applications and the Oracle-supplied tutorials and demos.

4.2 Developing Oracle Maps Applications

This topic describes how to develop Oracle Maps applications.

If the underlying base map and layers are managed in an Oracle database, each map tile layer displayed in the client application must have a corresponding database metadata entry in the USER_SDO_CACHED_MAPS metadata view (described in xxx_SDO_CACHED_MAPS Views) . Similarly, if an interactive layer is based on database content, it must have a metadata entry in the USER_SDO_THEMES view (described in Map Visualization Component Metadata Views, especially xxx_SDO_THEMES Views). These tile and interactive layers, and the styles and styling rules for them, can be defined using the Map Builder tool (described in #unique_26).

To develop Oracle Maps applications using the API, follow these basic steps:

1. Import the `oraclemapsv2.js` library.

   The API is provided in a single JavaScript library packaged as part of the map visualization component EAR archive.

2. After the map visualization component is deployed and started, load the library through a `<script>` tag, for example:

   ```
   <script  type="text/javascript" url="http://localhost:8080/mapviewer/jslib/v2/
   oraclemapsv2.js"/>
   ```

   > **Note:**
   >
   > If you need a specific jQuery version other than the one packaged in `oraclemapsv2.js` (shown in the preceding example), then you can add your preferred jQuery first, followed by a list of libraries. For example:
   >
   > ```
   > <script src="your_preferred_jQuery_library_listed_here_first.js "></script>
   > <script src="v2/jquery/jquery.hammer-full.min.js"></script>
   > <script src="v2/jquery/jquery.i18n.properties-min-1.0.9.js"></script>
   > <script src="v2/jquery/jquery.mousewheel.min.js"></script>
   > <script src="v2/rtree/rtree-min.js"></script>
   > <script src="v2/fortknox-libs/fortknox-canvas.js"></script>
   > <script src="v2/oraclemapsv2_core.js"></script>
   > ```

3. Create a `<DIV>` tag in the HTML page, which will contain the interactive map.

4. Create a client-side map instance that will handle all map display functions.

   The class is named `OM.Map` and is the main entry point of the API.

5. Set up a map universe (unless you also do the optional next step).

   A map universe basically defines the overall map extent, the number of zoom levels, and optionally the resolution (in map units per pixel) at each zoom level. Note that a predefined tile layer is not necessary in order to display interactive vector layers or themes. For example, an interactive thematic map of sales by region does not need to have a background map, or tile layer.

6. (Optional) Add a tile layer that serves as the background map.

   The tile layer can be from the database, such as `mvdemo.demo_map`, or from a supported service, such as Nokia Maps. Adding a tile layer also implicitly defines a map universe, and therefore the preceding step (setting up a map universe) is not necessary in this case.

7. Add one or more interactive vector layers.

   `OM.VectorLayer` uses HTML5 (Canvas or SVG) technology to render all the data in the browser. So, unless specified otherwise, all vector layer content is loaded once and there are no subsequent database queries, or data fetching, on map zoom or pan operations.

8. Add one or more map controls, tools, and other application-specific UI controls so that users can set the displayed layers, styling, and visual effects.

If you need to prevent themes from being streamed by default, you must protect them is by adding authentication, that is, by adding a security constraint in the map visualization component `web.xml` file and by configuring the `mds.xml` file to authorize access to various themes. For information, see Configuring and Securing the Map Data Server for the HTML5 API.

For detailed instructions and related information about using the API, see the Oracle-supplied tutorials and demos.

- Creating the Client Application with the API

4.2.1 Creating the Client Application with the API

Oracle Maps applications run inside web browsers and require only HTML5 (Canvas) support and JavaScript enabled. No additional plugins are required.

The source for an Oracle Maps application is typically packaged in an HTML page, which consists of the following parts:

- A `<script>` element that loads the Oracle Maps client library into the browser's JavaScript engine. For example:

  ```
  <script src="/mapviewer/jslib/v2/oraclemapsv2.js"></script>
  ```

- An HTML `<div>` element that will contain the map. For example:

  ```
  <div id="map" style="width: 600px; height: 500px"></div>
  ```

- JavaScript code that creates the map client instance and sets the initial map content (tile and vector layer), the initial center and zoom, and map controls. This code should be packaged inside a function which is executed when the HTML page is loaded or ready. The function is specified in the `onload` attribute of the `<body>` element of the HTML page. For example:

  ```
  function on_load_mapview()
  {
    var baseURL  = "http://"+document.location.host+"/mapviewer";
    // Create an OM.Map instance to display the map
    var mapview = new OM.Map(document.getElementById("map"),
                             {
                               mapviewerURL:baseURL
                             });
    // Add a map tile layer as background.
    var tileLayer = new OM.layer.TileLayer(
          "baseMap",
          {
              dataSource:"mvdemo",
              tileLayer:"demo_map",
              tileServerURL:baseURL+"/mcserver"
          });
    mapview.addLayer(tileLayer);
    // Set the initial map center and zoom level
    var mapCenterLon = -122.45;
    var mapCenterLat = 37.7706;
    var mapZoom = 4;
    var mpoint = new OM.geometry.Point(mapCenterLon,mapCenterLat,8307);
    mapview.setMapCenter(mpoint);
    mapview.setMapZoomLevel(mapZoom);
    // Add a theme-based FOI layer to display customers on the map
  ```

```
  customersLayer = new OM.layer.VectorLayer("customers",
        {
            def:
                {
                type:OM.layer.VectorLayer.TYPE_PREDEFINED,
                dataSource:"mvdemo", theme:"customers",
                url: baseURL,
                loadOnDemand: false
                }
        });
  mapview.addLayer(customersLayer);
  // Add a navigation panel on the right side of the map
  var navigationPanelBar = new OM.control.NavigationPanelBar();
  navigationPanelBar.setStyle(
{backgroundColor:"#FFFFFF",buttonColor:"#008000",size:12});
  mapview.addMapDecoration(navigationPanelBar);
  // Add a scale bar
  var mapScaleBar = new OM.control.ScaleBar();
  mapview.addMapDecoration(mapScaleBar);
  // Display the map.
  // The initialization and display is done just once.
  mapview.init();
}
```

- Additional HTML elements and JavaScript code that implement other application-specific user interface and control logic. For example, the HTML `<input>` element and JavaScript function `setLayerVisible` together implement a layer visibility control. The `setLayerVisible` function is coded as follows:

  ```
  function setLayerVisible(checkBox)
  {
    // Show the customers vector layer if the check box is checked and
    // hide it otherwise.
    if(checkBox.checked)
      customersLayer.setVisible(true) ;
    else
      customersLayer.setVisible(false);
  }
  ```

  The function is specified in the `onclick` attribute of the `<input>` element defining the checkbox. In the following example, the function is executed whenever the user clicks on the Show Customers check box:

  ```
  <INPUT TYPE="checkbox" onclick="setLayerVisible(this)" checked/>Show Customers
  ```

4.3 Using Google Maps

Applications can display Google Maps tiles as a built-in map tile layer, by creating and adding to the map window an instance of `MVGoogleTileLayer` .

Internally, the Oracle Maps client uses the official Google Maps API to display the map that is directly served by the Google Maps server.

To use the Google Maps tiles, your usage of the tiles must meet the terms of service specified by Google (see `https://developers.google.com/readme/terms`).

If you need to overlay your own spatial data on top of the Google Maps tile layer, see also Transforming Data to a Spherical Mercator Coordinate System.)

The following sections describe the two options for using built-in map tile layers.

- Defining Google Maps Tile Layers on the Client Side
- Defining the Built-In Map Tile Layers on the Server Side

4.3.1 Defining Google Maps Tile Layers on the Client Side

To define a built-in map tile layer on the client side, you need to create a `MVGoogleTileLayer` object, and add it to the MVMapView object. (As of Oracle Fusion Middleware Release 11.1.1.6, `MVGoogleTileLayer` uses the Google Maps Version 3 API by default.)

For example, to use Google tiles, add the Google tile layer to your map:

```
mapview = new MVMapView(document.getElementById("map"), baseURL);
tileLayer = new MVGoogleTileLayer() ;
mapview.addMapTileLayer(tileLayer);
```

In your application, you can invoke the method `MVGoogleTileLayer.setMapType` to set the map type to be one of the types supported by the map providers, such as road, satellite, or hybrid.

For usage examples and more information, see the JavaScript API documentation for `MVGoogleTileLayer`, and the tutorial demo *Built-in Google Maps Tile Layer*.

4.3.2 Defining the Built-In Map Tile Layers on the Server Side

You can define a built-in map tile layer (also called an external tile layer) on the server side in the metadata, and use it as a regular map visualization component tile layer on the client side. To define a built-in map tile layer on the server side, follow these steps:

1. Log into the map visualization component administration page (explained in Logging in to the Map Visualization Component Administration Page).
2. Select the Admin tab and click**Create tile layer**.
3. For **Tile Layer Type**, choose Oracle Maps, Google Maps, HERE Maps, OpenStreet Maps, Tomtom Maps, or LocationBox Maps, and click **Next**.
4. Provide a name for this external tile layer (such as GOOGLE_MAP), and select the data source where the tile layer definition is to be stored.
5. Set the license key that you have obtained from the map provider and other relevant properties (details are available for specific external tile layers), and click **Next**.
6. Review the XML Description of the external tile layer. You may click **Previous** to go to the previous page to make changes.
7. Click **Submit** to create the tile layer.

After you have created the built-in map tile layer on the server side, you can use it like any other tile layer served by MapViewer. You do not need to add any `<script>` tag to load the external JavaScript library.

The following code snippetshows a Google Maps tile layer defined on the server side:

```
var tileLayer = new OM.layer.TileLayer("google1", {
     dataSource:"mvdemo",
     tileLayer:"GOOGLE_MAP"
     });

 map.addLayer(tileLayer) ;
```

> **Note:**
>
> Each built-in tile layer has its own specific properties. You may need to check with the service provider for requirements.
>
> If Google Maps will be used, you may need to provide the following properties:
>
> - `key`: the key you obtained from Google Maps.
> - `client`: the client ID, an optional property you may obtain from Google Maps. The key property is still needed when the client ID is provided.
> - `lib_url`: the Google Maps API URL, for example: `http://maps.google.com/maps/api/js?sensor=false`. If not provided, a default value will be used.
> - `version`: the current API version is 3.
> - `mapTyelList`, a list of map types may be provided: MVGoogleTileLayer.TYPE_ROAD; MVGoogleTileLayer.TYPE_SHADED; MVGoogleTileLayer.TYPE_SATELLITE; MVGoogleTileLayer.TYPE_HYBRID.
> - `mapTypeVisible`, a Boolean `true` or `false` value to indicate whether to have the map type buttons be displayed for selecting map types.

4.4 Transforming Data to a Spherical Mercator Coordinate System

Popular online map services such as Google Maps use a spherical Mercator projection for their maps.

If you are using an Oracle Database release earlier than 11.1.0.7, and if you need to overlay your own spatial data on top of such a tile layer, such as a Google Maps tile layer, you must set up the database to properly handle coordinate system transformation between the coordinate system of that tile layer and your own data coordinate system, if the two coordinate systems are not the same.

> **Note:**
>
> To perform the actions in this section, your database must be Release 10.2.0.1 or later.

Google Maps uses a Spherical Mercator coordinate system (EPSG: 3785), which is also widely used among commercial API providers such as Yahoo! Maps. This coordinate system (SRID 3785) was not provided with Oracle Spatial before Release 11.1.0.7. In order to enable the map visualization componentthe map visualization component andOracle Spatial to transform your own data to this coordinate system, you must first add this coordinate system definition into your Oracle database if it is not already defined.

To check if this coordinate system is defined, you can enter the following statement:

```
SELECT srid FROM mdsys.cs_srs WHERE srid=3785;
```

If the preceding statement returns a row, you do not need to perform the actions in this section. If the preceding statement does not return a row, you must perform the actions in this section in order to be able to overlay your own spatial data on top of the tile layer.

Follow these steps:

1. Connect to the database as a privileged user, such as one with the DBA role.
2. Run the `csdefinition.sql` script, as follows. (Replace *`$MAPVIEWER_HOME`* with the root directory of the WebLogic Server instance where your MapViewer is deployed, and enter the command on a single line.)
   - Linux: *`$MAPVIEWER_HOME`*`/j2ee/home/applications/mapviewer/web/WEB-INF/admin/csdefinition.sql`
   - Windows: *`$MAPVIEWER_HOME`*`\j2ee\home\applications\mapviewer\web\WEB-INF\admin\csdefinition.sql`
3. If necessary, create a transformation rule to cause Oracle Spatial to skip datum conversion when transforming data from a specified coordinate system to the Spherical Mercator system. To find out if you need to create such a transformation rule, see Creating a Transformation Rule to Skip Datum Conversion.
4. Either pre-transform your spatial data for better performance, or let the map visualization component transform the data at runtime ("on the fly"). Note that if your database release is earlier than 10.2.0.4, pre-transforming is the only option.
   - To pre-transform all your data into the Spherical Mercator coordinate system, use the SDO_CS.TRANSFORM_LAYER procedure on all the data, and use the transformed data for mapping. (See the SDO_CS.TRANSFORM_LAYER reference section in *Oracle Spatial Developer's Guide*.)
   - To let the map visualization component transform the data at runtime, do not transform the data before using it for mapping.

- Creating a Transformation Rule to Skip Datum Conversion

4.4.1 Creating a Transformation Rule to Skip Datum Conversion

Spatial data is often in a coordinate system based on an ellipsoid datum, such as WGS84 or BNG. In such cases, Oracle Spatial by default applies datum conversion when transforming the data into the Spherical Mercator system. This will introduce a small amount of mismatch or error between your data and the Google Maps other map service tiles. If you want to address this issue, you can create transformation rules that tell Oracle Spatial to skip datum conversion when transforming data from a specified coordinate system to the Spherical Mercator system.

Example 4-1 shows SQL statements that are included in the `csdefinition.sql` script and that create such transformations rules. However, if the coordinate system of your spatial data is not covered by the rules shown in Example 4-1, you can create your own rule if the coordinate system of your data is not covered by these rules. (For more information about creating coordinate system transformation rules, see *Oracle Spatial Developer's Guide*.)

Example 4-1 Transformation Rules Defined in the csdefinition.sql Script

```
-- Create the tfm_plans, that is, the transformation rules.
-- Note: This will result in an incorrect conversion since it ignores a datum
-- datum between the ellipsoid and the sphere. However, the data will match
-- up better on Google Maps.

-- For wgs84 (8307)
call sdo_cs.create_pref_concatenated_op( 83073785, 'CONCATENATED OPERATION 8307 3785',
TFM_PLAN(SDO_TFM_CHAIN(8307, 1000000000, 4055, 19847, 3785)), NULL);
```

```
-- For 4326, EPSG equivalent of 8307
call sdo_cs.create_pref_concatenated_op( 43263785, 'CONCATENATED_OPERATION_4326_3785',
TFM_PLAN(SDO_TFM_CHAIN(4326, 1000000000, 4055, 19847, 3785)), NULL);

-- For OS BNG, Oracle SRID 81989
call sdo_cs.create_pref_concatenated_op( 819893785, 'CONCATENATED OPERATION 81989 3785',
TFM_PLAN(SDO_TFM_CHAIN(81989, -19916, 2000021, 1000000000, 4055, 19847, 3785)), NULL);

-- For 27700, EPSG equivalent of 81989
call sdo_cs.create_pref_concatenated_op( 277003785, 'CONCATENATED_OPERATION_27700_3785',
TFM_PLAN(SDO_TFM_CHAIN(27700, -19916, 4277, 1000000000, 4055, 19847, 3785)), NULL);
commit;
```

4.5 Using Dynamic Tile Layers

A **dynamic tile layer** is a tile layer defined on the client side.

A dynamic tile layer differs from a regular tile layer in the following ways:

- It is defined by the client application, so it does not need to be predefined in the USER_SDO_CACHED_MAPS metadata.
- The server does not cache the map tiles.
- It can use third party maps services, when a map tile image URL generation function is provided.

Thus, a dynamic tile layer can be defined dynamically by a client application, and the map images may come from a map visualization component server or from a third party map service.

4.5.1 Creating a Universe and Configuration Instance for a Dynamic Tile Layer

Before creating a dynamic tile layer instance, you must create a universe and a tile layer configuration for a dynamic tile layer.

For example, assume that a variable named baseURL is properly defined (such as http://myapp.mycorp.com/mapviewer) and that the specified data source exists. You must create a tile layer configuration instance and a universe instance. You can use an Oracle Maps JavaScript API built-in universe or create your own. For simplicity, a built-in universe is used here.

A tile layer configuration instance only needs to define the dimension of an image tile, and 256 pixels for a tile image's width and height are commonly used. The following code snippet creates this instance:

```
  var  myuniv= new OM.universe.LatLonUniverse();  // a built-in universe
  var  myconfig=new OM.layer.TileLayerConfig(  // a tile layer configure
instance
          {
            tileImageWidth: 256,
            tileImageHeight: 256
          });
```

4.5.2 Creating a Dynamic Tile Layer Using a Base Map

Creating a dynamic tile layer using a base map is the simplest approach, because the map contents are already defined by a server-side map visualization component base map. Follow these steps.

1. Create a `ServerMapRequest` instance and set its properties. For example:

```
        var req = new OM.server.ServerMapRequest(baseURL);
        req.setProperties({
            dataSource:"MVDEMO",
            basemap: 'demo_map',  // this basemap property defines the map
contents
            transparent:true,
            antialiase:"false"
        });
```

2. Create an object containing properties of this dynamic tile layer. For example:

```
        var dtl_props = {
            universe: myuniv,
            tileLayerConfig: myconfig,
            tileServerURL: baseURL + "/omserver",  // map images come from
this omserver
            enableUTFGrid: true,  // the UTFGrid is enabled
            enableUTFGridInfoWindow: true,  // the info window for
displaying UTFGrid is enabled
            utfGridResolution: 4   // one grid cell represents 16 (4 by 4)
image pixels
        };
```

In this example, the string value in variable `tileServerURL` is pointing to a map visualization component server. It could be the same server as where the web application is deployed, or another map visualization component server instance. For example, it can be in either of the following forms:

- From the same Oracle Maps server instance:. For example:

```
var baseURL=document.location.protocol+"//"+document.location.host+"/
mapviewer";
```

- From another Oracle Maps server instance. For example:

```
var baseURL="http://myapp.mycorp.com/mapviewer ";
```

3. Create a dynamic tile layer. For example:

```
  var dynamic_tilelayer_1 = new OM.layer.DynamicTileLayer("dtl1",
dtl_props, req);
```

The dynamic tile layer requests map images from the map visualization component's map server (omserver), which renders a fresh map image and does not cache the images on disk.

From the preceding examples, notice that three statements are given to create a dynamic tile layer: (1) create a map request instance (`ServerMapRequest`); (2) create an object containing the dynamic tile layer related properties; and (3) instantiate a dynamic tile layer.

The dynamic tile layer requests map images from the map visualization component's map server (`omserver`), which renders a fresh map image and does not cache the images on disk.

4.5.3 Creating a Dynamic Tile Layer Using Predefined Themes

Another approach to creating a dynamic tile layer is to use predefined themes. Follow these steps.

1. Create a `ServerMapRequest` instance and set its properties. For example:

```
        var req = new OM.server.ServerMapRequest(baseURL);
        req.setProperties({
            dataSource:"MVDEMO",
            transparent:true,
            antialiase:"false",
        });
        // the three themes below are defined in the metadata already.
        var t1 = new OM.server.ServerPredefinedTheme("THEME_DEMO_STATES");
        var t2 = new
OM.server.ServerPredefinedTheme("THEME_DEMO_HIGHWAYS");
        var t3 = new OM.server.ServerPredefinedTheme("THEME_DEMO_CITIES");
        req.addThemes([t1, t2, t3]);
```

2. Create an object containing properties of this dynamic tile layer. For example:

```
        var dtl_props = {
            universe: myuniv,
            tileLayerConfig: myconfig,
            tileServerURL: baseURL + "/omserver",
            enableUTFGrid: true,
            enableUTFGridInfoWindow: true,
            utfGridResolution: 4
        };
```

3. Create a dynamic tile layer. For example:

```
  var dynamic_tilelayer_2 = new OM.layer.DynamicTileLayer("dtl2",
dtl_props, req);
```

The dynamic tile layer requests map images from the map visualization component's map server (omserver), which renders a fresh map image and does not cache the images on disk.

Instead of specifying a server base map, in this approach the `ServerMapRequest` instance adds some map visualization component server predefined themes. From a practical perspective, this allows the client-side application to create tile layers using the server-side defined themes in the view USER_SDO_THEMES.

4.5.4 Creating a Dynamic Tile Layer Using Spatial Tables Directly

You can directly use spatial tables for a dynamic tile layer by creating `ServerJBCTheme` instances and adding them into a `ServerMapRequest` instance. Follow these steps.

1. Create a style. For example:

```
var myc1 = new OM.style.Color({
    styleName: "mycolor1",
    stroke: "#333333",
    strokeOpacity: 1.0,
    fill: "#F2EFE9",
    fillOpacity: 1.0
});
```

2. Create a JDBC theme and set its properties. Spatial tables are used in a SQL query statement. The statement, as a string type variable, is set as an attribute in the `OM.server.ServerJDBCTheme` instance. For example:

```
var jdbcTStates= new
OM.server.ServerJDBCTheme('theme_jdbc_states');
jdbcTStates.setDataSourceName('mvdemo');
jdbcTStates.setSRID('8307');
jdbcTStates.setGeometryColumnName('geom');
var sql='select totpop, poppsqmi, state, state_abrv, geom from
states'; // the query
jdbcTStates.setQuery(sql);  // set the SQL string
jdbcTStates.addInfoColumn({column: 'state_abrv', name:'State'});
jdbcTStates.addInfoColumn({column: 'totpop', name:'Population'});
jdbcTStates.addInfoColumn({column: 'poppsqmi', name:'Pop.
Density'});
jdbcTStates.setRenderingStyleName('mycolor1');
```

3. Create a `ServerMapRequest` instance and set its properties. For example:

```
var req = new OM.server.ServerMapRequest(baseURL);
req.setProperties({
    dataSource:"MVDEMO",
    transparent:true,
    antialiase:"false",
});
req.addTheme(jdbcTStates);
req.addStyle(myc1);
```

4. Create an object containing properties of this dynamic tile layer. For example:

```
var dtl_props = {
    universe: myuniv,
    tileLayerConfig: myconfig,
    tileServerURL: baseURL + "/omserver",
    enableUTFGrid: true,
    enableUTFGridInfoWindow: true,
    utfGridResolution: 4
};
```

5. Create a dynamic tile layer. For example:

```
  var dynamic_tilelayer_3 = new OM.layer.DynamicTileLayer("dtl3",
dtl_props, req);
```

In this example, a dynamically defined style and a `ServerJDBCTheme` object are created first. The `ServerJDBCTheme` can make use the spatial tables directly using a SQL statement. The tables are then used by the `addTheme` and `addStyle` methods of a `ServerMapRequest` object.

4.5.5 Creating a Dynamic Tile Layer Using Third Party Map Services

Another approach to create a dynamic tile layer is to use third party map services. The map service URL of a third party is provided by a client provided URL generation function. Follow these steps.

1. Using the third party map service provider's specifications, create a URL construction function for building map image requests by a dynamic tile layer. For example:

```
var urlb = function (w, h, minX, minY, maxX, maxY, options){
      var str="http://my.mycorp.com:8080/geoserver/ows?";
      var optParams="";
      str = str+"request=getmap"+
      "&bbox="+minX+","+minY+","+maxX+","+maxY+
      "&width="+w+
      "&height="+h;

    if (!OM.isNull(options)) {
      for (var key in options) {
        optParams=optParams +"&"+key +"="+options[key];
      }
    }
    return str+optParams;
  }
```

2. Create an object containing properties of this dynamic tile layer. For example:

```
var dtl_props = {
    universe: myuniv,
    tileLayerConfig: myconfig,
    urlBuilder: urlb,
    urlBuilderOptions: {
              "layers":  "topp:states,sf:sfdem,sf:roads,sf:streams",
              "CRS":     "CRS:84",
              "service": "WMS",
```

```
                "version": "1.3.0",
                "format":  "image/png"
        }
};
```

3. Create a dynamic tile layer. For example:

```
  var dynamic_tilelayer_4 = new OM.layer.DynamicTileLayer("dl4",
dtl_props);
```

In this example, a dynamic tile layer instance (`dynamic_tilelayer_4`) contains a `urlBuilder` property to provide a URL generation function (`iurlb`). The URL generation function will be used by the dynamic tile layer for fetching image tiles from the specified map service provider. This example uses a third party map provider, a `GeoServer` instance, to provide map images.

Additional map service provider required parameters can be provided in the `urlBuilderOptions` property. This example specifies several additional parameters (`layers`, `CRS`, `service`, `version`, and `format`) needed by the `GeoServer` instance.

4.5.6 Dynamic Tile Layer Use Case

This topic describes a dynamic tile layer use case.

The code examples come from a dynamic tile layer application. The application uses the map visualization component server to provide map image tiles. It uses spatial tables directly via an `OM.server.ServerJDBCTheme` instance.

1. Obtain the SQL query's "where" clause from an application's user interface using the `getWhereClauseFromUI()` function, assuming that this function is properly defined already in that application and returns expected results. For example:

```
    var  wherec = getWhereClauseFromUI();
```

2. Create a JDBC theme and set its properties. For example:

```
    var objJDBCTheme= new OM.server.ServerJDBCTheme();
    objJDBCTheme.setDataSourceName('storm');
    objJDBCTheme.setSRID('3857');
    objJDBCTheme.setXColumnName('long_loc');
    objJDBCTheme.setYColumnName('lat_loc');

    objJDBCTheme.addInfoColumn({column: 'YEAR', name:'Year'});
    objJDBCTheme.addInfoColumn({column: 'MONTH', name:'Month'});
    objJDBCTheme.addInfoColumn({column: 'DAY', name:'Day'});
    objJDBCTheme.addInfoColumn({column: 'STATE', name:'State'});
    objJDBCTheme.addInfoColumn({column: 'LOSS', name:'Loss'});
    objJDBCTheme.addInfoColumn({column: 'CROPLOSS', name:'Crop_loss'});
    objJDBCTheme.addInfoColumn({column: 'FATALITIES', name:'Fatalities'});
    objJDBCTheme.addInfoColumn({column: 'INJURIES', name:'Injuries'});

    objJDBCTheme.setName('theme_tornado');
    objJDBCTheme.setQuery("SELECT s.geom.sdo_point.x
long_loc,s.geom.sdo_point.y
lat_loc,year,month,day,state,fatalities,injuries,loss,croploss from
tornado_3857 s"+
```

```
(wherec === "" ?"": wherec));
    objJDBCTheme.setRenderingStyleName('V.TORNADO');
```

3. Create a `ServerMapRequest` instance and set its properties. For example:

```
var req = new OM.server.ServerMapRequest(baseURL);
req.setProperties({
    dataSource:'storm',
    transparent:true,  // map image is set to be transparent
    antialiase:"false"
});
req.addTheme(objJDBCTheme);
```

4. Create an object containing properties of this dynamic tile layer. For example:

```
var dtl_props = {
    universe: myuniv,
    tileLayerConfig: myconfig,
    tileServerURL: baseURL + "/omserver",
    enableUTFGrid: true,
    enableUTFGridInfoWindow: true,
    utfGridResolution: 4
};
```

5. Create a dynamic tile layer. For example:

```
  var tornado_damage = new OM.layer.DynamicTileLayer("tornado_damage",
dtl_props, req);
```

The code examples in the preceding steps illustrate the following:

- The map application fetches map images from a map visualization component server. Therefore, it may use all available features from the map visualization component server.
- The map image is set to be transparent (that is, the value for property `transparent` is `true`). This setting makes all of the non-overlapping features visible when multiple dynamic tile layers are stacked for display. For example, when you overlay a `tornado_damage` dynamic tile layer on top of a `hail_damage` dynamic tile layer, you may see both layers on the map when such events (as their names suggest) were happening at non-overlapping locations. This setting is more useful for multiple layers containing linear and point features. Note that since the API allows you to turn on/off a layer's visibility, you can always make a layer of interest stand out and be visible in your application.
- The dynamic tile layer also enables the map visualization component server's `UTFGrid` support (specifically, the value for property `enableUTFGrid` is set to `true`, with a resolution value set to 4 for the `utfGridResolution` property). This setting instructs map visualization component's API to request an additional `UTFGrid` file as a companion of its map tile image. It is a JSON file, and it contains a two-dimensional grid table in UTF encoding, that matches the tile image. It also contains some text attributes for each grid cell.

  When a mouse event (for example, a mouse click) is triggered at an image pixel, its corresponding grid cell can then be located, its UTF code be obtained, and subsequently, the text attributes associated with that grid cell's UTF code can be retrieved and displayed in a popup info window.

The text attributes contained in that JSON file are specified using the `addInfoColumn()` method. Those text attributes will be displayed in an info window for a pixel being clicked on the map.

Note that if server-side predefined geometry themes are used by a dynamic tile layer, the text attributes for an info window must have been specified in theme's `<hidden_info>` element. This also applies for the base map-based dynamic tile layers, where the predefined themes are found in the map visualization component's base map definition

A resolution value (for example, `"utfGridResolution":4`) indicates the width and height of the grid cell compares with its companion map tile image pixel. In this example, one grid cell in the `UTFGrid` data set represents 16 pixels (4x4=16).

- This dynamic tile layer makes use of spatial table directly via a `ServerJDBCTheme` to define its map contents. In its embedded SQL statement, the client web application has the flexibility to construct the whole statement, including the tables to use, the columns to select, and the "where" clause to refine the query. In the code snippets, it assumes that a client function, `getWhereClauseFromUI()`, only constructs the 'where' clause in the JDBC theme definition.

The dynamic tile layer supported in the Oracle Maps JavaScript API allows the map visualization component server metadata to be used by the tile layer, and it offers flexibility in how a dynamic tile layer is defined and where to fetch its map images by the client application.

5
Oracle Map Builder Tool

This chapter briefly describes the map visualization component Map Builder tool, also referred to as Oracle Map Builder. It does not provide detailed information about the tool's interface; for that you should use see online help available when you use Oracle Map Builder.

Oracle Map Builder is a standalone application that lets you create and manage the mapping metadata (about styles, themes, and base maps) that is stored in the database. For example, use this tool to create a style or to modify the definition of a style. Besides handling the metadata, the tool provides interfaces to preview the metadata (for example, to see how a line style will appear on a map) and also spatial information.

Whenever possible, you should use Oracle Map Builder instead of directly modifying map visualization component metadata views to create, modify, and delete information about styles, themes, and maps. For any modifications made outside Oracle Map Builder, such as with SQL statements, you should refresh the database connection in Oracle Map Builder to get the current items.

To use Oracle Map Builder effectively, you must understand the map visualization component concepts explained in Map Visualization Concepts and the information about map requests in Map Visualization Servers.

- Running Oracle Map Builder
  Oracle Map Builder is shipped as a JAR file (`mapbuilder.jar`).
- Oracle Map Builder User Interface
  Oracle Map Builder generally uses the left side for navigation to find and select objects, and the right side to display information about selected objects.

5.1 Running Oracle Map Builder

Oracle Map Builder is shipped as a JAR file (`mapbuilder.jar`).

You can run it as a standalone Java application in a Java Development Kit (J2SE SDK) 1.5 or later environment, as follows:

```
% java -jar mapbuilder.jar [Options]
```

Options:

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

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

`-connect` causes Map Builder at startup to register connections for all data sources specified in the `oasmapbuilder.xml` preferences file or the file specified with the `-config` option, and it automatically connects to the first available data source. This option increases the application startup time. If this option is not defined, startup is faster, but you must then use the File menu

or an icon to connect to any data sources that you want to use (see Oracle Map Builder User Interface).

`-help` displays information about the available options.

- Java Libraries for Theme Creation with GDAL and Teradata

5.1.1 Java Libraries for Theme Creation with GDAL and Teradata

To create themes in Map Builder with GDAL-OGR (version 1.8 or later) or Teradata (version 13 or later), Map Builder must be started with GDAL-OGR (`gdal.jar`) and/or Teradata (`terajdbc4.jar` and `tdgssconfig.jar`) jar files on the class path. These Java libraries can be found in the GDAL installation on your system and on the Teradata website. (For GDAL, see the GDAL website for more information.)

The following example commands start Map Builder with additional libraries in the class path.

Windows: `java -Xmx512M -cp .\mapbuilder.jar;.\gdal.jar;.\terajdbc4.jar;.\tdgssconfig.jar oracle.mapviewer.builder.MapBuilder`

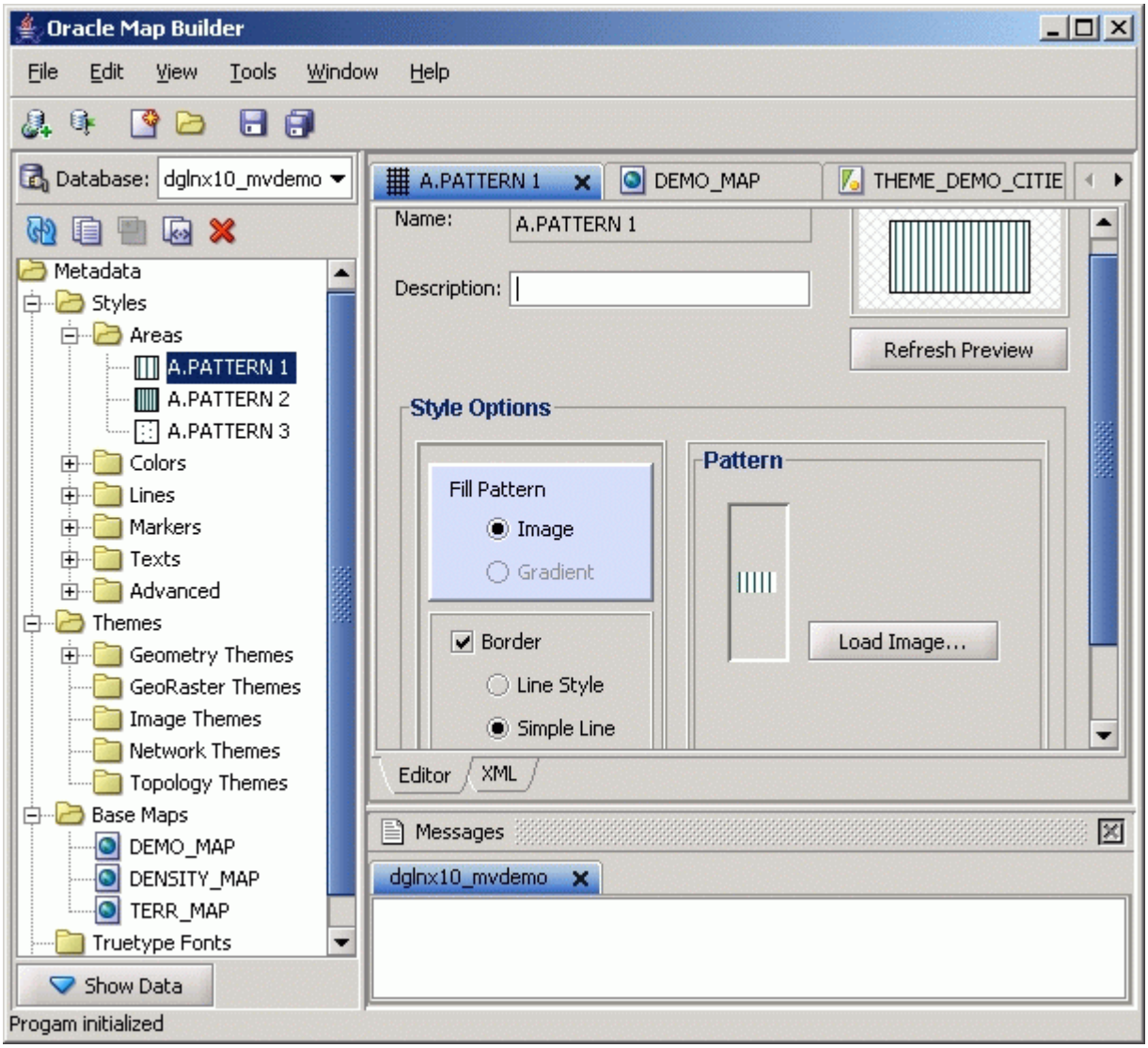
Linux: `java -Xmx512M -cp ./mapbuilder.jar:./gdal.jar:./terajdbc4.jar:./tdgssconfig.jar oracle.mapviewer.builder.MapBuilder`

5.2 Oracle Map Builder User Interface

Oracle Map Builder generally uses the left side for navigation to find and select objects, and the right side to display information about selected objects.

Figure 5-1 shows the main window of Oracle Map Builder, with the metadata navigation tree on the left and a detail pane for a selected area style on the right.

Figure 5-1 Oracle Map Builder Main Window



The menus at the top contain standard entries, plus entries for features specific to Oracle Map Builder.

You can use shortcut keys to access menus and menu items: for example Alt+F for the File menu and Alt+E for the Edit menu; or Alt+H, then Alt+A for Help, then About.

Icons under the menus perform the following actions:

- **Add new connection** creates a new database connection for Oracle Map Builder to use.
- **Load/Add/Remove connection** loads or adds database connection for Oracle Map Builder to use, or removes a database connection from the available connections that Oracle Map Builder can use.
- **Create new metadata** creates a new base map, theme, or style.
- **Open** opens a base map, theme, or style.
- **Save** saves any changes to the currently selected object.
- **Save All** saves any changes to all open objects.

The left side of the Oracle Map Builder window has the Metadata navigator, including a database connection selector, icons for performing actions, and a hierarchical tree display for the map visualization component metadata objects (categorized by object type) accessible to the currently selected database connection. To select an object, expand the appropriate tree node or nodes, then double-click the object.

The right side of the Oracle Map Builder window has tabs and panes for detail views of objects that you select or open

To switch among objects, click the desired tabs; to close a tab, click the X in the tab. If you make changes to an object and click the X, you are asked if you want to save the changes.

The Messages area is used for feedback information as appropriate (for example, results of an action, or error or warning messages).

Detailed help is available within the Oracle Map Builder interface. See the online help for more information about Oracle Map Builder, including information about specific panes and dialog boxes.

A

XML Format for Styles, Themes, Base Maps, and Map Tile Layers

This appendix describes the XML format for defining style, themes, and base maps using the map visualization component metadata views.

These metadata views are described in Map Visualization Component Metadata Views.

The metadata views for map visualization component 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 map visualization component syntax, each style's XML document must contain a single `<g>` element, which must have a `class` attribute that indicates the type or class of the style. For example, the following defines a color style with a filling color component:

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

The map visualization component 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.

> **Note:**
>
> You can make the size of a style scalable by specifying a unit other than the default pixel (`px`) -- for example, `width:15.0km` or `stroke-width:10.0m`. For information about using scalable styles, see Scaling the Size of a Style (Scalable Styles).

The metadata views for map visualization component 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.

- Color Styles
  A color style has a fill color, a stroke color, or both.
- Marker Styles
  A marker style represents a marker to be placed on point features or on label points of area and linear features.

- Line Styles
  A line style is applicable only to a linear feature, such as a road, railway track, or political boundary.
- Area Styles
  An area style defines a pattern to be used to fill an area feature.
- Text Styles
  A text style defines the font and color to be used in labeling spatial features.
- Advanced Styles
  Advanced styles are structured styles made from simple styles.
- Themes: Styling Rules
  A theme definition contains one `<styling_rules>` element, which may have several other elements depending on the theme type.
- Base Maps
  A base map definition consists of one or more themes.
- Map Tile Layers
  An Oracle Maps map tile layer which assembles and displays pregenerated map image tiles from the map tile server,

A.1 Color Styles

A color style has a fill color, a stroke color, or both.

When applied to a shape or geometry, the fill color (if present) is used to fill the interior of the shape, and the stroke color (if present) is used to draw the boundaries of the shape. Either color can also have an alpha value, which controls the transparency of that color.

For color styles, the `class` attribute of the `<g>` element must be set to `"color"`. The `<g>` element must have a `style` attribute, which specifies the color components and their optional alpha value. For example:

- `<g class="color" style="fill:#ff0000">` specifies a color style with only a fill color (whose RGB value is #ff0000).
- `<g class="color" style="fill:#ff0000;stroke:blue">` specifies a color style with a fill color and a stroke color (blue).

You can specify a color value using either a hexadecimal string (such as #00ff00) or a color name from the following list: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow.

To specify transparency for a color style, you can specify `fill-opacity` and `stroke-opacity` values from 0 (completely transparent) to 255 (opaque). The following example specifies a fill component with half transparency:

```
<g class="color" style="fill:#ff00ff;fill-opacity:128">
```

The following example specifies both stroke and fill opacity:

```
<g class="color" style= "stroke:red;stroke-opacity:70;
                         fill:#ff00aa;fill-opacity:129">
```

The syntax for the `style` attribute is a string composed of one or more `name:value` pairs delimited by semicolons. (This basic syntax is used in other types of styles as well.)

For stroke colors, you can define a stroke width. The default stroke width when drawing a shape boundary is 1 pixel. To change that, add a `stroke-width:value` pair to the `style` attribute string. The following example specifies a stroke width of 3 pixels:

```
<g class="color" style="stroke:red;stroke-width:3">
```

A.2 Marker Styles

A marker style represents a marker to be placed on point features or on label points of area and linear features.

A marker can be either a vector marker or raster image marker. A marker can also have optional notational text. For a vector marker, the coordinates of the vector elements must be defined in its XML document. For a marker based on a raster image, the XML document for the style indicates that the style is based on an external image.

The marker XML document specifies the preferred display size: the preferred width and height are defined by the `width:value;height:value` pairs in the `style` attribute of the `<g>` element. The `class` attribute must be set to `"marker"`. Some markers must be overlaid with some notational text, such as a U.S. interstate highway shield marker, which, when rendered, must also have a route number plotted on top of it. The style for such notational text is a style attribute with one or more of the following name-value pairs: `font-family:value`, `font-style:value`, `font-size:value`, and `font-weight:value`.

The following example defines an image-based marker that specifies font attributes (shown in bold) for any label text that may be drawn on top of the marker:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<desc></desc>
<g class="marker"
  style="width:20;height:18;font-family:sans-serif;font-size:9pt;fill:#ffffff">
    <image x="0" y="0" width="9999" height="9999" type="gif"
           href="dummy.gif"/>
</g>
</svg>
```

In the preceding example, when the marker is applied to a point feature with a labeling text, the label text is drawn centered on top of the marker, using the specified font family and size, and with the fill color (white in this case) as the text foreground. The label text (`495`) in Figure A-1 in Using Marker Styles on Lines has the text attributes specified in this example.

- Vector Marker Styles
- Image Marker Styles
- TrueType Font-Based Marker Styles
- Using Marker Styles on Lines

A.2.1 Vector Marker Styles

A vector marker can be a simple polygon, an optimized rectangle (defined using two points), a single polyline, or a circle, but not any combination of them. For each type of vector marker, its `<g>` element must contain a corresponding subelement that specifies the geometric information (coordinates for the polygon, optimized rectangle, or polyline, or radius for the circle):

- A polygon definition uses a `<polygon>` element with a `points` attribute that specifies a list of comma-delimited coordinates. For example:

```
<g class="marker">
   <polygon points="100,20,40,50,60,80,100,20"/>
</g>
```

- An optimized rectangle definition uses a `<rect>` element with a `points` attribute that specifies a list of comma-delimited coordinates. For example:

```
<g class="marker">
  <rect points="0,0, 120,120"/>
</g>
```

- A polyline definition uses a `<polyline>` element with a `points` attribute that specifies a list of comma-delimited coordinates. For example:

```
<g class="marker">
   <polyline points="100,20,40,50,60,80"/>
</g>
```

- A circle definition uses a `<circle>` element with an `r` attribute that specifies the radius of the circle. For example:

```
<g class="marker">
   <circle r="50"/>
</g>
```

You can specify a stroke or fill color, or both, for any vector-based marker. The syntax is the same as for the style attribute for a color style. The following example defines a triangle marker that has a black border and that is filled with a half-transparent yellow:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<g class="marker"  style="stroke:#000000;fill:#ffff00;fill-opacity:128">
     <polygon points="201.0,200.0, 0.0,200.0, 101.0,0.0"/>
</g>
</svg>
```

If a marker is scalable, you can set the marker's maximum size in pixels on a map by using the `max_size_in_px` attribute. Setting this attribute to an appropriate value will prevent the marker from getting so large as to block other map features when the user zooms in to view fine map details. The following example sets the marker's maximum size to 64 pixels:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
    <g class="marker" max_size_in_px="64"
style="stroke:#0000BB;fill:#0033FF;width:3.0mile;height:3.0mile;font-family:Dialog;font-
size:12;font-fill:#FF0000">
        <polygon points="0.0,0.0,0.0,100.0,100.0,100.0,100.0,0.0,0.0,0.0"/>
    </g>
</svg>
```

A.2.2 Image Marker Styles

For an image marker, its XML document contains an `<image>` element that identifies the marker as based on an image. The image must be in GIF format, and is stored in the IMAGE column in the styles metadata views.

The following example is an XML document for an image marker:

```
<?xml version="1.0" standalone="yes"?>
<svg>
   <g class="marker"
             style="width:20;height:18;font-family:sansserif;font-size:9pt">
     <image x="0" y="0" width="9999" height="9999" type="gif" href="dummy.gif"/>
```

```
   </g>
</svg>
```

Note that in the preceding example, it would be acceptable to leave the `<image>` element empty (that is, `<image/>`) to create a valid definition with the image to be specified later.

A.2.3 TrueType Font-Based Marker Styles

For a TrueType font-based marker, its marker symbol is stored in a TrueType font file, which has the .ttf file extension and which typically contains many individual symbols or glyphs. Many GIS software packages come with TrueType font files that contain symbols useful for mapping.

Before the map visualization component can use a symbol in a TrueType font file, you must do the following:

1. Import the TrueType font file into the database, preferably by using the Map Builder tool (described in #unique_26), which causes the symbols in the font file to be inserted into a single row in the system view USER_SDO_STYLES. In this new row, the TYPE column contains the string TTF, and the IMAGE column contains the contents of the TrueType font file. After the import operation, you can use the Map Builder tool to view all the glyphs or symbols contained inside the TrueType font file. Also, because the font file is now physically stored inside a database, it can be shared by all the map visualization component users.
2. Create a map visualization component marker style based on a glyph or symbol inside an imported TrueType font, preferably using the Map Builder tool.

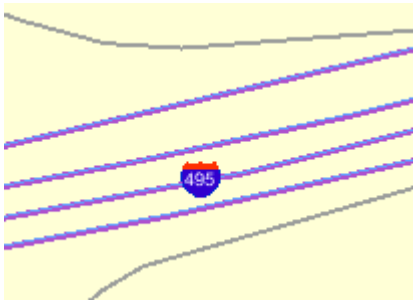
The following example shows the use of a TrueType font-based marker (with TrueType-specific material in bold):

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<g class="marker"  style="fill:#ff0000;width:25;height:25">
  <ttfSymbol fontName="ERS_INCIDENTS" charCode="118" />
</g>
</svg>
```

A.2.4 Using Marker Styles on Lines

Marker styles are usually applied to point features, in which case the marker style is rendered on the point location that represents the feature. However, with line (line string) features such as highways, the marker must be placed at some point along the line to denote some information about the feature, such as its route number. For example, on maps in the United States, a shield symbol is often placed on top of a highway, with a route number inside the symbol, as shown with Route 495 in Figure A-1.

Figure A-1 Shield Symbol Marker for a Highway



To achieve the result shown in Figure A-1, you must do the following:

1. Choose a marker style, and add a text style definition (font family, font size, fill color, and so on), as shown in the example in Marker Styles.
2. Specify the marker style as the labeling style in the styling rules for the theme. The following example shows the XML document with the styling rules for a theme to show highways. A marker style (shown in bold in the example) is specified. The label text (`495` in Figure A-1) is a value from the label column, which is named LABEL in this example.

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="political">
<rule>
   <features style="L.PH"> (name_class = 'I' and TOLL=0) </features>
   <label column="label" style="M.SHIELD1">1</label>
</rule>
<styling_rules>
```

The map visualization component automatically determines the optimal position on the line for placement of the marker style (the shield in this example).

A.3 Line Styles

A line style is applicable only to a linear feature, such as a road, railway track, or political boundary.

In other words, line styles can be applied only to Oracle Spatial geometries with an SDO_GTYPE value ending in 2 (line) or 6 (multiline). (For information about the SDO_GEOMETRY object type and SDO_GTYPE values, see *Oracle Spatial Developer's Guide*.)

When the map visualization component draws a linear feature, a line style tells the rendering engine the color, dash pattern, and stroke width to use. A line style can have a base line element which, if defined, coincides with the original linear geometry. It can also define two edges parallel to the base line. Parallel line elements can have their own color, dash pattern, and stroke width. If parallel lines are used, they must be located to each side of the base line, with equal offsets to it.

To draw railroad-like lines, you need to define a third type of line element in a line style called *hashmark*. For a `<line>` element of class `hashmark`, the first value in the dash array indicates the gap between two hash marks, and the second value indicates the length of the hash mark to either side of the line. The following example defines a hash mark line with a gap of 8.5 screen units and a length of 3 screen units at each side of the base line:

```
<line class="hashmark" style="fill:#003333"  dash="8.5,3.0"/>
```

The following example defines a complete line style.

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
   <g class="line" style="fill:#ffff00;stroke-width:5">
      <line class="parallel" style="fill:#ff0000;stroke-width:1.0"/>
      <line class="base" style="fill:black;stroke-width:1.0" dash="10.0,4.0"/>
   </g>
</svg>
```

In the preceding example, `class="line"` identifies the style as a line style. The overall fill color (#ffff00) is used to fill any space between the parallel lines and the base line. The overall line width (5 pixels) limits the maximum width that the style can occupy (including that of the parallel lines).

The line style in the preceding example has both base line and parallel line elements. The parallel line element (`class="parallel"`) is defined by the first `<line>` element, which defines its color and width. (Because the definition does not provide a dash pattern, the parallel lines or edges will be solid.) The base line element (`class="base"`) is defined by the second `<line>` element, which defines its color, width, and dash pattern.

A marker (such as a direction marker) can be defined for a line style. The `marker-name` parameter specifies the name of a marker style, the `marker-position` parameter specifies the proportion (from 0 to 1) of the distance along the line from the start point at which to place the marker, and the `marker-size` parameter specifies the number of display units for the marker size. The marker orientation follows the orientation of the line segment on which the marker is placed.

The following example defines a line style with direction marker:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
  <g class="line" style="fill:#33a9ff;stroke-width:4;
      marker-name:M.IMAGE105_BW;marker-position:0.15;marker-size=8">
    <line class="parallel" style="fill:red;stroke-width:1.0"/>
  </g>
</svg>
```

To get multiple markers, add the `multiple-marker` attribute to the style definition. In this case the marker-position will define the position for the first marker and the space in between markers. The following example defines a line style with a direction marker that starts at position 0.15 and that is repeated continually with a space of 0.15 between each occurrence.

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
 <g class="line" style="fill:#33a9ff;stroke-width:4;
     marker-name:M.IMAGE105_BW; marker-position:0.15;
     marker-size=8; multiple-marker=true">
   <line class="parallel" style="fill:red;stroke-width:1.0"/>
 </g>
</svg>
```

A.4 Area Styles

An area style defines a pattern to be used to fill an area feature.

In the current release, area styles must be image-based. That is, when you apply an area style to a geometry, the image defining the style is plotted repeatedly until the geometry is completely filled.

The definition of an area style is similar to that of an image marker style, which is described in Image Marker Styles.

The following example defines an area style:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
   <g class="area"  style="stroke:#000000">
      <image/>
   </g>
</svg>
```

In the preceding example, `class="area"` identifies the style as an area style. The stroke color (`style="stroke:#000000"`) is the color used to draw the geometry boundary. If no stroke color

is defined, the geometry has no visible boundary, although its interior is filled with the pattern image.

You can also specify any line style to be used as the boundary for an area style. The following area style definition uses the `line-style` keyword (shown in bold in the example) to specify a line style to be used for the borders of features:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
  <g class="area" style="line-style:L.DPH">
    <image x="0" y="0" width="9999" height="9999" type="gif" href="dummy.gif"/>
  </g>
</svg>
```

As with the image marker style, the image for an area style must be stored in a separate column (identified in the IMAGE column in the USER_SDO_STYLES and ALL_SDO_STYLES metadata views, which are described in xxx_SDO_STYLES Views).

A.5 Text Styles

A text style defines the font and color to be used in labeling spatial features.

The `class` attribute must have the value `"text"`. For the font, you can specify its style (plain, italic, and so on), font family, size, and weight. To specify the foreground color, you use the `fill` attribute.

The following example defines a text style:

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
   <g class="text" style="font-style:plain; font-family:Dialog; font-size:14pt;
             font-weight:bold; fill:#0000ff">
    Hello World!
   </g>
</svg>
```

In the preceding example, the text "Hello World!" is displayed only when the style itself is being previewed in a style creation tool, such as the Map Builder tool. When the style is applied to a map, it is always supplied with an actual text label that the map visualization component obtains from a theme.

A text style can provide a floating white background around the rendered text, to make the labels easier to read on a map that has many features. Figure A-2 shows the label Vallejo with a white background wrapping tightly around the letters.

Figure A-2 Text Style with White Background

To achieve the result shown in Figure A-2, you must specify the `float-width` attribute in the `<g>` element of the text style definition. The following example uses the `float-width` attribute (shown in bold in the example) to specify a white background that extends 3.5 pixels from the boundary of each letter. (The Hello World! text is ignored when the style is applied to the display of labels.)

```
<?xml version="1.0" standalone="yes"?>
<svg width="1in" height="1in">
<desc></desc>
<g class="text"  float-width="3.5"
   style="font-style:plain; font-family:Dialog; font-size:12pt; font-weight:bold;
     fill:#000000">
 Hello World!
</g>
</svg>
```

A.6 Advanced Styles

Advanced styles are structured styles made from simple styles.

Advanced styles are used primarily for thematic mapping. The core advanced style is the bucket style (`BucketStyle`), and every advanced style is a form of bucket style. A bucket style is a one-to-one mapping between a set of primitive styles and a set of buckets. Each bucket contains one or more attribute values of features to be plotted. For each feature, one of its attributes is used to determine which bucket it falls into or is contained within, and then the style assigned to that bucket is applied to the feature.

Two special types of bucket styles are also provided: color scheme (described in Color Scheme Styles) and variable (graduated) marker (described in Variable Marker Styles).

Other advanced styles are dot density (described in Dot Density Marker Styles), bar chart (described in Bar Chart Marker Styles), collection (described in Collection Styles), and variable pie chart (described in Variable Pie Chart Styles).

- Bucket Styles
- Color Scheme Styles
- Variable Marker Styles
- Dot Density Marker Styles
- Bar Chart Marker Styles
- Collection Styles
- Variable Pie Chart Styles
- Heat Map Styles

A.6.1 Bucket Styles

A bucket style defines a set of buckets, and assigns one primitive style to each bucket. The content of a bucket can be either of the following:

- A collection of discrete values (for example, a bucket for all counties with a hurricane risk code of 1 or 2, a bucket for all counties with a hurricane risk code of 3, and so on).
- A continuous range of values (for example, a bucket for all counties with average family income less than $30,000, a bucket for all counties with average family income from $30,000 through $39,999, and so on). In this case, the ranges of a series of buckets

can be individually defined (each defined by an upper-bound value and lower-bound value) or equally divided into an overall range.

The following code excerpt shows the basic format of a bucket style:

```
<?xml version="1.0" ?>
<AdvancedStyle>
  <BucketStyle>
    <Buckets>
    . . .
    </Buckets>
  </BucketStyle>
</AdvancedStyle>
```

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

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

- Collection-Based Buckets with Discrete Values
- Individual Range-Based Buckets
- Equal-Ranged 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; the map visualization component determines sequence only by the order in which elements appear in a definition.

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

To apply label styles to collection-based buckets with discrete values, see Specifying a Label Style for a Bucket.

A.6.1.2 Individual Range-Based Buckets

If each bucket of a bucket style contains a value range that is defined by two values, use a <RangedBucket> element to represent each bucket. Each bucket contains a range of values. The following example defines four buckets.

```
<?xml version="1.0" ?>
 <AdvancedStyle>
   <BucketStyle>
     <Buckets>
          <RangedBucket high="10" style="10015"/>
          <RangedBucket low="10" high="40" style="10024"/>
          <RangedBucket low="40" high="50" style="10025"/>
          <RangedBucket low="50" style="10029"/>
     </Buckets>
   </BucketStyle>
</AdvancedStyle>
```

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

For example, the second bucket in this example (low="10" high="40") will contain any values that are exactly 10, as well as values up to but not including 40 (such as 39 and 39.99). Any values that are exactly 40 will be included in the third bucket.

As with the <CollectionBucket> element, the style associated with each <RangedBucket> element is specified as an attribute.

To apply label styles to individual range-based buckets, see Specifying a Label Style for a Bucket.

A.6.1.3 Equal-Ranged Buckets

If a bucket style contains a series of buckets that contain an equally divided range of a overall range, you can omit the use of <RangedBucket> elements, and instead specify in the <Buckets> element the overall upper-bound value and lower-bound value for the overall range, the number of buckets in which to divide the range, and a list of style names (with one for each bucket). The following example defines five buckets (nbuckets=5) of equal range between 0 and 29:

```
<?xml version="1.0" ?>
<AdvancedStyle>
   <BucketStyle>
      <Buckets low="0" high="29" nbuckets="5"
          styles="10015,10017,10019,10021,10023"/>
```

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

In the preceding example:

- If all values are integers, the five buckets hold values in the following ranges: 0 to 5, 6 to 11, 12 to 17, 18 to 23, and 24 to 29.
- The first bucket is associated with the style named `10015`, the second bucket is associated with the style named `10017`, and so on.

The number of style names specified must be the same as the value of the `nbuckets` attribute. The buckets are arranged in ascending order, and the styles are assigned in their specified order to each bucket.

A.6.2 Color Scheme Styles

A color scheme style automatically generates individual color styles of varying brightness for each bucket based on a base color. The brightness is equally spaced between full brightness and total darkness. Usually, the first bucket is assigned the brightest shade of the base color and the last bucket is assigned the darkest shade.

You can include a stroke color to be used by the color style for each bucket. The stroke color is not part of the brightness calculation. So, for example, if a set of polygonal features is rendered using a color scheme style, the interior of each polygon is filled with the color (shade of the base color) for each corresponding bucket, but the boundaries of all polygons are drawn using the same stroke color.

You can include an opacity value (0 to 255, for transparent to opaque) for the base color (using the `basecolor_opacity` attribute) and for the stroke color (using the `strokecolor_opacity` attribute).

The following example defines a color scheme style with a black stroke color and four buckets associated with varying shades of the base color of blue.

```
<?xml version="1.0" ?>
<AdvancedStyle>
 <ColorSchemeStyle basecolor="blue" strokecolor="black">
   <Buckets>
        <RangedBucket label="&lt;10"  high="10"/>
        <RangedBucket label="10 - 20" low="10" high="20"/>
        <RangedBucket label="20 - 30" low="20" high="30"/>
        <RangedBucket label="&gt;=30" low="30"/>
   </Buckets>
  </ColorSchemeStyle>
</AdvancedStyle>
```

> **Note:**
>
> For the following special characters, use escape sequences instead.
>
> For <, use: `&lt`
>
> For >, use: `>`
>
> For &, use: `&`

A.6.3 Variable Marker Styles

A variable marker style generates a series of marker styles of varying sizes for each bucket. You specify the number of buckets, the start (smallest) size for the marker, and the size increment between two consecutive markers.

Variable marker styles are conceptually similar to color scheme styles in that both base buckets on variations from a common object: with a color scheme style the brightness of the base color varies, and with a variable marker style the size of the marker varies.

The following example creates a variable marker style with four buckets, each associated with different sizes (in increments of 4) of a marker (`m.circle`). The marker for the first bucket has a radius of 10 display units, the marker for the second bucket has a radius of 14 display units, and so on. This example assumes that the marker named `m.circle` has already been defined.

```
 <?xml version="1.0" ?>
<AdvancedStyle>
  <VariableMarkerStyle basemarker="m.circle" startsize="10" increment="4">
     <Buckets>
         <RangedBucket label="&lt;10"  high="10"/>
         <RangedBucket label="10 - 20" low="10" high="20"/>
         <RangedBucket label="20 - 30" low="20" high="30"/>
         <RangedBucket label="&gt;=30"   low="30"/>
     </Buckets>
  </VariableMarkerStyle>
</AdvancedStyle>
```

A.6.4 Dot Density Marker Styles

A dot density advanced marker style, when applied to an area feature such as states or counties, randomly draws a set of dots inside the area. The number of dots drawn inside each area is determined by the count value associated with the area. When you define a dot density style, you must specify a marker style that will be used for each of the dots.

The following example shows the XML definition of a simple dot density style:

```
<?xml version="1.0" ?>
<AdvancedStyle>
        <DotDensityStyle MarkerStyle="M.STAR" DotWidth="8" DotHeight="8">
        </DotDensityStyle>
</AdvancedStyle>
```

In the preceding example, the marker style M.STAR is used for each dot, and the size of each dot is 8 pixels wide and high.

When you use a dot density style, you should "scale" the count value to a proper range. For example, if you want to apply a dot density style based on the population count for each county, you would not want to use the population count directly (one dot for each person), because this will result in an unacceptable number of drawn dots (for example, if a county has 15,000 people). Instead, supply a scaled down value or expression, such as `population/1000`, when you define the styling rules for the theme. (The map visualization component does not perform any scaling-down internally, so you must do it at the SQL query level.)

A.6.5 Bar Chart Marker Styles

A bar chart advanced marker style is similar to a pie chart style, except that it draws a bar graph for each feature to which it is applied. The following example shows the XML definition of a bar chart style:

```
<?xml version="1.0" ?>
<AdvancedStyle>
   <BarChartStyle width="30" height="25" show_x_axis="true">
       <Bar name="1990" color="#FF0000" />
       <Bar name="1995" color="#FFC800" />
       <Bar name="1998" color="#0000FF" />
       <Bar name="2000" color="#00FF00" />
       <Bar name="2002" color="#00FFFF" />
   </BarChartStyle>
</AdvancedStyle>
```

In the preceding example, width and height specify the overall size of the bar chart, including all individuals bars within it.

When a bar chart is drawn on a feature based on a set of values associated with that feature, the height of each bar can be determined by either of two approaches: locally scaled or globally scaled. A locally scaled bar chart determines the height of each bar only from the associated values for that feature; and thus, for example, you cannot compare the second bar of one chart to the second bar on another chart on the same theme. A globally scaled bar chart uses the same bar scale for all charts on the map; and thus, for example, you can compare the second bar of one chart to the second bar on another chart on the same theme.

So, if you want to compare bars not only within the same chart, but also among all the charts showing on the map, you must use globally scaled bar chart style by specifying `share_scale="true"` in the definition of the bar chart style, as shown in the following example:

```
<?xml version="1.0" ?>
<AdvancedStyle>
   <BarChartStyle width="40" height="30" share_scale="true"
      min_value="0.0" max_value="100">
       <Bar name="1990" color="#FF0000" />
       <Bar name="1995" color="#FFC800" />
       <Bar name="1998" color="#0000FF" />
       <Bar name="2000" color="#00FF00" />
       <Bar name="2002" color="#00FFFF" />
   </BarChartStyle>
</AdvancedStyle>
```

When the bar chart style in the preceding example is applied to a theme, the map visualization component considers the global range of values of all features in that theme, and then determines the height of each bar based on where a specific value falls in the global range from the minimum value to the maximum value.

A.6.6 Collection Styles

A collection advanced style is simply a collection of other types of styles that are applied together to a feature. This can result in faster rendering of a collection theme compared to using multiple themes based on different styles.

For example, a bar chart style, when applied to a county, draws only the bar chart somewhere inside the county, but the county itself (its boundary and interior area) is not drawn. However, you probably want to see the underlying boundaries of the counties, to see which bar chart

belongs to which county. To do this without a collection style, you would have to define a second theme in which each county is being associated with a color or area style. This approach would result in two rendering passes (because two themes are involved) for essentially the same group of features.

However, by using a collection style in this example, you can define a single style that refers to both the bar chart and the color or area style, and then apply the collection style to the theme for the counties. This theme, when rendered by the map visualization component, will show both the bar charts and the boundaries on the map.

Another typical use of a collection style is for rendering collection type topology features, each of which can contain multiple types of geometries, such as polygons (areas), points, and lines. In such cases, a collection style can include styles that are most appropriate for each type of geometry in a collection topology feature.

The following example shows the XML definition of a collection style:

```
<?xml version="1.0" standalone="yes"?>
 <AdvancedStyle>
   <CollectionStyle>
     <style name="C.COUNTIES" shape="polygon" />
     <style name="L.PH" shape="line" />
     <style name="M.CIRCLE" shape="point" />
   </CollectionStyle>
  </AdvancedStyle>
```

A.6.7 Variable Pie Chart Styles

A variable pie chart generates a series of pie circles of varying sizes for each bucket. You specify the pie slice information, the start (smallest) radius size for a pie circle, and the radius size increment between two consecutive circles.

Variable pie chart styles are conceptually similar to variable marker styles. With a variable marker style the base marker size varies, whereas with the variable pie chart style the circle radius varies.

The following example creates a definition for a variable pie chart style with four buckets, each associated with different sizes (in increments of 4) of a circle with start radius of 5. The circle radius for the first bucket has a radius of 5 display units, the circle for the second bucket has a radius of 9 display units, and so on.

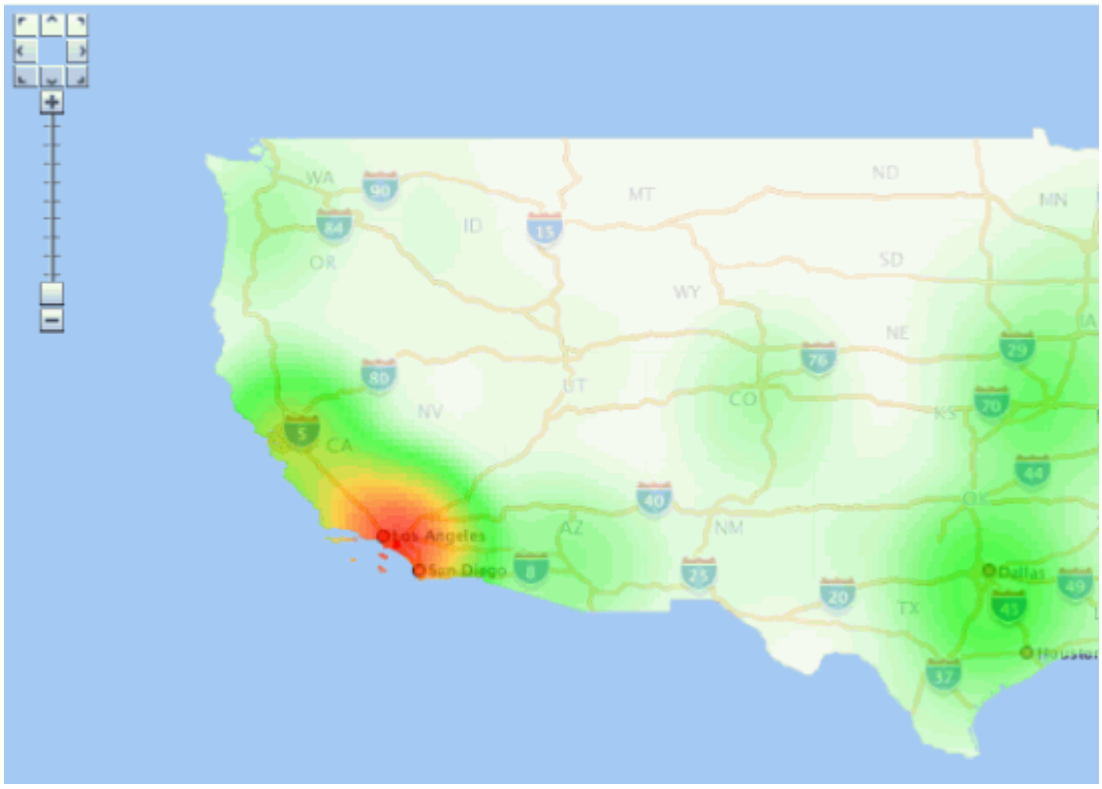
```
<?xml version="1.0" ?>
<AdvancedStyle>
  <VariablePieChartStyle startradius="5" increment="4">
    <PieSlice name="WHITE" color="#FFFFFF"/>
    <PieSlice name="BLACK" color="#000000"/>
    <PieSlice name="HISPANIC" color="#FF0000"/>
   <Buckets>
    <RangedBucket seq="0" label="0 - 6194757.2" low="0" high="6194757.2" />
    <RangedBucket seq="1" label="6194757.2 - 1.23895144E7" low="6194757.2"
high="1.23895144E7"/>
    <RangedBucket seq="2" label="1.23895144E7 - 1.85842716E7" low="1.23895144E7"
high="1.85842716E7"/>
    <RangedBucket seq="3" label="1.85842716E7 - 2.47790288E7" low="1.85842716E7"
high="2.47790288E7"/>
    <RangedBucket seq="4" label="2.47790288E7 - 3.0973786E7" low="2.47790288E7"
high="3.0973786E7"/>
   </Buckets>
  </VariablePieChartStyle>
</AdvancedStyle>
```

A.6.8 Heat Map Styles

A heat map style can be used to generate a two-dimensional (2D) color map of any point-type data set. The colors represent the distribution density or pattern of the points or events across the region. Internally, the map visualization component creates a 2D matrix and assigns a value to each grid cell based on the result of a distance-weighted algorithm run against the point data set.

You can create a heat map style using the Map Builder tool, and assign it as the rendering style for a point-type geometry theme. You can then add this theme to a base map, or add it as a theme-based FOI layer to an interactive Oracle Maps application. Figure A-3 shows a map displayed using a theme based on a heat map style. This map shows the concentration of pizza restaurants: red areas have the highest concentration of pizza restaurants, with concentrations progressively lower for orange, yellow, dark green, lighter green, pale green, and white areas.

Figure A-3 Heat Map Showing Pizza Restaurant Concentration



The following example creates a definition for a heat map style.

```
<?xml version="1.0" ?>
<AdvancedStyle>
     <HeatMapStyle>
          <color_stops num_steps="200" alpha="128">
            FFFFFF,00FF00, FFC800,FF0000
          </color_stops>
          <spot_light_radius>75.0mile</spot_light_radius>
          <grid_sample_factor>2.5</grid_sample_factor>
          <container_theme>THEME_DEMO_STATES</container_theme>
```

```
    </HeatMapStyle>
</AdvancedStyle>
```

The preceding example defines these essential aspects of the heat map:

- **Color stops**. Color stops are used to generate a color gradient. In this example, the color gradient will go from white (maps to grid cells with a zero value) to green, to orange, and finally to full red (maps to grid cells with highest values). The gradient will have 200 colors that span these 4 color stops. All the colors will have an alpha value of 128 (half transparent, where 0 would be fully transparent and 255 would be opaque).
- **Spot light radius**. The spot light radius defines the radius around each grid cell where events or points within this radius will be contributing to the final aggregated value of that cell. The contribution of each point decreases as its distance from the cell center increases, and becomes zero beyond this radius.

  You can specify the radius in pixels or in a real ground unit such as `mile`. When you specify the radius in pixels (the default if you do not specify a unit), the mapping from the color gradient to the grid cells will vary as the user zooms in and out on the map. This occurs because the number of points fall within the radius is constantly changing as the user zooms in and out. To achieve a fixed heat map regardless of map scale, you must specify the spotlight radius in a ground unit such as `meter`, `km`, or `mile`. The preceding example uses `mile`.
- **Grid sample factor**. The grid sample factor is used to sample the current map window size when creating the internal matrix or grid for heat map calculation. For example, a sample factor of 4 means that the internal heat map grid will be one-fourth (0.25) the actual map window size. So, if the map is 1000x1000 pixels, the internal heat map grid is 250x250. Thus, the lower the grid sample factor value, the larger the internal heat map grid will be; and the higher the value, the smaller the internal heat map grid will be.

  The grid sample factor value probably has more effect on heat map rendering performance than any other attribute, because a large internal heat map grid (resulting from a low grid sample factor value) will significantly increase the overall computation time. A good general guideline is to specify a grid sample factor value high enough so that the internal heat map grid will be 512x512 pixels or smaller.
- **Container theme name**. The container theme name specifies the name of a theme (predefined geometry theme in the same database schema) that defines the boundary of the map for the heat map theme. For example, if you are generating a heat map for a point data set that scatters all over the entire United States of America, choose a theme that represents the US national boundary or all the states as its container theme.

  The specified container theme does not affect how the heat map itself is calculated (which is solely based on the point distribution and the spotlight radius). Instead, the container theme it masks out all colored cells that are outside the boundary of the study region. This helps to ensure a "clean" look for the heat map.

After you create a heat map style, you can create a theme for point data and assign the new heat map style as the rendering style for the theme.

Unlike other types of advanced styles, heat map styles do not require any attribute or value columns.

Labels are not supported for themes rendered using heat map styles.

A.7 Themes: Styling Rules

A theme definition contains one `<styling_rules>` element, which may have several other elements depending on the theme type.

This <styling_rules> element is specified in the STYLING_RULES column of the USER_SDO_THEMES metadata view, using the following DTD:

```
<!ELEMENT styling_rules (rule+, hidden_info?, join_table?, join_columns?, operations?,
bitmap_masks?, parameters?)>
<!ATTLIST styling_rules theme_type         CDATA #IMPLIED
                        key_column         CDATA #IMPLIED
                        caching            CDATA #IMPLIED "NORMAL"
                        image_format       CDATA #IMPLIED
                        image_column       CDATA #IMPLIED
                        image_resolution   CDATA #IMPLIED
                        image_unit         CDATA #IMPLIED
                        raster_id          CDATA #IMPLIED
                        raster_table       CDATA #IMPLIED
                        raster_pyramid     CDATA #IMPLIED
                        raster_bands       CDATA #IMPLIED
                        polygon_mask       CDATA #IMPLIED
                        transparent_nodata CDATA #IMPLIED
                        network_name       CDATA #IMPLIED
                        network_level      CDATA #IMPLIED
                        topology_name      CDATA #IMPLIED
                        service_url        CDATA #IMPLIED
                        srs                CDATA #IMPLIED
                        feature_ids        CDATA #IMPLIED
                        provider_id        CDATA #IMPLIED
                        srid               CDATA #IMPLIED>

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

<!ELEMENT features (#PCDATA?, link?, node?, path?)>
<!ATTLIST features style CDATA #REQUIRED>

<!ELEMENT label (#PCDATA?, link?, node?, path?)>
<!ATTLIST label column CDATA #REQUIRED
                style  CDATA #REQUIRED>

<!ELEMENT link (#PCDATA)>
<!ATTLIST link style                CDATA #REQUIRED
               direction_style      CDATA #IMPLIED
               direction_position   CDATA #IMPLIED
               direction_markersize CDATA #IMPLIED
               column               CDATA #REQUIRED>

<!ELEMENT node (#PCDATA)>
<!ATTLIST node style      CDATA #REQUIRED
               markersize CDATA #IMPLIED
               column     CDATA #REQUIRED>

<!ELEMENT path (#PCDATA)>
<!ATTLIST path ids    CDATA #REQUIRED
               styles CDATA #REQUIRED
               style  CDATA #REQUIRED
               column CDATA #REQUIRED>

<!ELEMENT hidden_info (field+)>

<!ELEMENT field (#PCDATA)>
<!ATTLIST field column CDATA #REQUIRED
                name   CDATA #IMPLIED>

<!ELEMENT join_table EMPTY>
```

```
<!ATTLIST join_table name          CDATA #REQUIRED
                     start_measure CDATA #IMPLIED
                     end_measure   CDATA #IMPLIED
                     measure       CDATA #IMPLIED>

<!ELEMENT join_columns EMPTY>
<!ATTLIST columns lrs_table_column  CDATA #REQUIRED
                  join_table_column CDATA #REQUIRED>

<!ELEMENT rendering (style+)>

<!ELEMENT style (substyle?)>
<!ATTLIST style name          CDATA #REQUIRED
                value_columns CDATA #IMPLIED>

<!ELEMENT substyle (#PCDATA)>
<!ATTLIST substyle name          CDATA #REQUIRED
                   value_columns CDATA #REQUIRED
                   changes       CDATA #IMPLIED>

<!ELEMENT operations (operation?)>

<!ELEMENT operation (parameter?)>
<!ATTLIST operation name CDATA #REQUIRED>

<!ELEMENT parameters (parameter?)>

<!ELEMENT parameter (#PCDATA)>
<!ATTLIST parameter name  CDATA #REQUIRED
                    value DATA #REQUIRED>

<!ELEMENT bitmap_masks (mask+)>

<!ELEMENT mask (#PCDATA)>
<!ATTLIST mask raster_id    CDATA #REQUIRED
               raster_table CDATA #REQUIRED
               layers       CDATA #REQUIRED
               zeromapping  CDATA #IMPLIED
               onemapping   CDATA #IMPLIED>
```

The `<styling_rules>` element can have a `theme_type` attribute, which is used mainly for certain types of predefined themes. (The default `theme_type` attribute value is `geometry`, which indicates that the theme is based on spatial geometries.) The `theme_type` attribute values for these special types of predefined themes are as follows:

- `annotation` specifies an annotation text theme. Annotation text themes are explained in Annotation Text Themes.
- `geom_custom` specifies a custom geometry theme. You must also specify the `provider_id` and `srid` attributes. Custom geometry themes are explained in Custom Geometry Themes.
- `georaster` specifies a GeoRaster theme. To use specified GeoRaster data (but not if you use a query condition to retrieve the GeoRaster data), you must also specify the `raster_id` and `raster_table` attributes. You can also specify the `raster_pyramid`, `raster_bands`, `polygon_mask`, and `transparent_nodata` attributes. GeoRaster themes are explained in GeoRaster Themes.
- `image` specifies an image theme. You must also specify the `image_format` and `image_column` attributes, and you can specify the `image_resolution` and `image_unit` attributes. Image themes are explained in Image Themes.

- `network` specifies a network theme. You must also specify the `network_name` attribute. You can specify the `network_level` attribute, but the default value (1) is the only value currently supported. Network themes are explained in Network Themes.
- `topology` specifies a topology theme. You must also specify the `topology_name` attribute. Topology themes are explained in Topology Themes.
- `wfs` specifies a WFS theme. You must also specify the `service_url` and `srs` attributes. WFS themes are explained in WFS Themes.

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, the map visualization component 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 the map visualization component 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 the map visualization component to pin all geometry data of this theme entirely in the cache before any viewing request. In contrast to the default value of `NORMAL`, a value of `ALL` caches all geometries from the base table the first time the theme is viewed, and the geometries are not subsequently fetched from the database.

For detailed information about the caching of predefined themes, see Caching of Predefined Themes.

Each `<rule>` element must have a `<features>` element and can have a `<label>` element and a `<rendering>` element. The `<rendering>` element can be used to define multiple render styles, and in this case the render style in the `<features>` element may be undefined. If the render style in the `<features>` element is defined and `<rendering>` element is also defined, the map visualization component will first render the style in the `<features>` element and then render the styles in `<rendering>` element. (The `<rendering>` element is explained later in this section.)

The optional `column` attribute of a `<rule>` element specifies one or more attribute columns (in a comma-delimited list) from the base table to be put in the SELECT list of the query generated by the map visualization component. The values from such columns are usually processed by an advanced style for this theme. The following example shows the use of the column attribute:

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

In the preceding example, the theme's geometry features will be rendered using an advanced style named `V.COUNTY_POP_DENSITY`. This style will determine the color for filling a county

geometry by looking up numeric values in the column named TOTPOP in the base table for this theme.

Each <features> element for a network theme must have a <link>, <node>, or <path> element, or some combination of them. (The <link>, <node>, and <path> elements apply only to network themes, which are explained in Network Themes.) The following example shows the styling rules for a network theme to render links and nodes.

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="network"
          network_name="LRS_TEST" network_level="1">
 <rule>
   <features>
     <link style="C.RED"
           direction_style="M.IMAGE105_BW"
           direction_position="0.85"
           direction_markersize="8"></link>
     <node style="M.CIRCLE" markersize="5"></node>
   </features>
 </rule>
</styling_rules>
```

A <label> element must have a SQL expression as its element value for determining whether or not a label will be applied to a feature. The column attribute specifies a SQL expression for text values to label features, and the style attribute specifies a text style for rendering labels.

The <rendering> element can be used to define multiple rendering styles. The styles are rendered in the order that they appear. Each style in a <rendering> element is defined by a <style> element, which must specify the name attribute and can specify the value_columns attribute. (The value_columns attribute is used with advanced styles, and the column names are added to the list of attributes defined in the column attribute of <rule> element.)

In the <rendering> element, each <style> element can have a <substyle> element that defines the attributes for filling the feature. A <substyle> element must specify the name attribute and can specify the value_columns and changes attributes. For the changes attribute, only the FILL_COLOR value is supported.

The following example shows the styling rules for a geometry theme using the <rendering> element. It defines an advanced style named V.POIVMK to render the feature shape and an advanced substyle named V.POIBKT to fill the feature shape.

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

For more information about using the <rendering> element to apply multiple rendering styles in a single styling rule, see Applying Multiple Rendering Styles in a Single Styling Rule.

The <hidden_info> element specifies the list of attributes from the base table to be displayed when the user moves the mouse over the theme's features. The attributes are specified by a list of <field> elements.

Each <field> element must have a column attribute, which specifies the name of the column from the base table, and it can have a name attribute, which specifies the display name of the column. (The name attribute is useful if you want a text string other than the column name to be displayed.)

The <join_table> element specifies the table name and its one or two columns for the measurements. It may contain one measure column for features of point type or two measure columns for features of line string type. The measure column or columns are used for the linear referencing process when joining with a LRS table, which has an LRS geometry column.

The <join_columns> element specifies one column of the LRS table (which has a LRS geometry column), and one column from the join table (which has one or two measure columns). These two columns are used to join the LRS table and the join table.

The <operations> element specifies the list of image processing operations to be applied on a GeoRaster theme. The operations are specified by a list of <operation> elements.

The <operation> element specifies the image processing operator and its parameters to be applied on a GeoRaster theme. Each <operation> element may have a list of <parameters> elements.

The <parameters> element defines a list of parameters to be used on a specific task. The parameters are specified by a list of <parameter> elements.

The <parameter> element must have the name and value attributes defined.

The <bitmap_masks> element defines the image mask attributes to be used with a GeoRaster theme. The bitmap masks are specified by a list of <mask> elements.

The <mask> element specifies a bitmap mask to be applied on a GeoRaster object. The raster_id, raster_table, and layers attributes must be defined, while the zeromapping and onemapping attributes are optional.

See Styling Rules in Predefined Spatial Geometry Themes for more information about styling rules and for an example.

A.8 Base Maps

A base map definition consists of one or more themes.

The XML definition of a base map is specified in the DEFINITION column of the USER_SDO_MAPS metadata view, using the following DTD:

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

<!ELEMENT theme EMPTY>
<!ATTLIST theme name CDATA #REQUIRED
    name                  CDATA #REQUIRED
    datasource            CDATA #IMPLIED
    template_theme        CDATA #IMPLIED
    max_scale             CDATA #IMPLIED
    min_scale             CDATA #IMPLIED
    label_always_on       (TRUE|FALSE) "FALSE"
    fast_unpickle         (TRUE|FALSE) "TRUE"
    mode                  CDATA #IMPLIED
    min_dist              CDATA #IMPLIED
    fixed_svglabel        (TRUE|FALSE) "FALSE"
    visible_in_svg        (TRUE|FALSE) "TRUE"
    selectable_in_svg     (TRUE|FALSE) "FALSE"
    part_of_basemap       (TRUE|FALSE) "FALSE"
```

```
    simplify_shapes       (TRUE|FALSE) "TRUE"
    transparency          CDATA #IMPLIED
    minimum_pixels        CDATA #IMPLIED
    onclick               CDATA #IMPLIED
    onmousemove           CDATA #IMPLIED
    onmouseover           CDATA #IMPLIED
    onmouseout            CDATA #IMPLIED
    workspace_name        CDATA #IMPLIED
    workspace_savepoint   CDATA #IMPLIED
    workspace_date        CDATA #IMPLIED
    workspace_date_format CDATA #IMPLIED
    fetch_size            CDATA #IMPLIED
    timeout               CDATA #IMPLIED
>
```

The <map_definition> element contains one or more <theme> elements. Themes are rendered on a map on top of each other, in the order in which they are specified in the definition.

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

A.9 Map Tile Layers

An Oracle Maps map tile layer which assembles and displays pregenerated map image tiles from the map tile server,

Tis rpocess is described in Map Tile Layer Configuration. The XML configuration settings of a map tile layer is defined using the following DTD:

```
<!ELEMENT map_tile_layer ((internal_map_source|external_map_source), tile_storage,
coordinate_system, tile_image, , tile_dpi?, tile_meters_per_unit?, zoom_levels,
auto_update?, themes?)>
<!ATTLIST map_tile_layer
          name CDATA #REQUIRED
          image_format CDATA #IMPLIED
          http_header_expires CDATA #IMPLIED
          utfgrid (TRUE|FALSE) "FALSE"
          utfgrid_resolution CDATA #IMPLIED
          concurrent_fetching_threads CDATA #IMPLIED
          fetch_larger_tile (TRUE|FALSE) "TRUE"
          persistent_tiles (TRUE|FALSE) "TRUE">

<!ELEMENT internal_map_source EMPTY>
<!ATTLIST internal_map_source
          data_source CDATA #REQUIRED
          base_map CDATA #REQUIRED
          bgcolor CDATA #IMPLIED
          out_of_bounds_color CDATA #IMPLIED
          antialias (TRUE|FALSE) "TRUE">

<!ELEMENT external_map_source (properties?)>
<!ATTLIST external_map_source
          url CDATA #REQUIRED
          request_method CDATA #REQUIRED
          timeout CDATA #IMPLIED
          adapter_class CDATA #REQUIRED
          proxy_host CDATA #IMPLIED
          proxy_port CDATA #IMPLIED
          clipping_buffer CDATA #IMPLIED>

<!ELEMENT properties (property+) >
```

```
<!ELEMENT property EMPTY >
<!ATTLIST property
          name CDATA #REQUIRED
          value CDATA #REQUIRED>

<!ELEMENT tile_storage EMPTY >
<!ATTLIST tile_storage
          root_path CDATA #REQUIRED
          xyz_storage_scheme (TRUE|FALSE) "FALSE">

<!ELEMENT coordinate_system EMPTY >
<!ATTLIST coordinate_system
          srid CDATA #REQUIRED
          minX CDATA #REQUIRED
          minY CDATA #REQUIRED
          maxX CDATA #REQUIRED
          maxY CDATA #REQUIRED>

<!ELEMENT tile_bound (coordinates)>
<!ELEMENT coordinates (#PCDATA)>

<!ELEMENT tile_image EMPTY >
<!ATTLIST tile_image
          width CDATA #REQUIRED
          height CDATA #REQUIRED>

<!ELEMENT tile_dpi EMPTY >
<!ATTLIST tile_dpi
          value CDATA #REQUIRED>

<!ELEMENT tile_meters_per_unit EMPTY >
<!ATTLIST tile_meters_per_unit
          value CDATA #REQUIRED>

<!ELEMENT zoom_levels (zoom_level+)>
<!ATTLIST zoom_levels
          levels CDATA #REQUIRED
          min_scale CDATA #IMPLIED
          max_scale CDATA #IMPLIED
          min_tile_width CDATA #IMPLIED
          min_tile_height CDATA #IMPLIED>

<!ELEMENT zoom_level (tile_bound?)>
<!ATTLIST zoom_level
          level CDATA #REQUIRED
          level_name CDATA #IMPLIED
          description CDATA #IMPLIED
          scale CDATA #REQUIRED
          tile_width CDATA #REQUIRED
          tile_height CDATA #REQUIRED>

<!ELEMENT auto_update (dirty_mbr_table_name,logtable_name)>
<!ATTLIST auto_update
          finest_level_to_refresh CDATA #REQUIRED
          dirty_mbr_batch CDATA #REQUIRED
          dirty_mbr_cap CDATA #REQUIRED>

<!ELEMENT dirty_mbr_table_name EMPTY>
<!ATTLIST dirty_mbr_table_name
          name CDATA #REQUIRED>
```

```
<!ELEMENT logtable_name EMPTY>
<!ATTLIST logtable_name
          name CDATA #REQUIRED>

<!ELEMENT themes (theme)>
<!ATTLIST auto_update
          finest_level_to_refresh CDATA #REQUIRED
          dirty_mbr_batch CDATA #REQUIRED
          dirty_mbr_cap CDATA #REQUIRED>

<!ELEMENT theme EMPTY>
<!ATTLIST theme
          name CDATA #REQUIRED
          from_level CDATA #REQUIRED
          to_level CDATA #REQUIRED>
```

B

Creating and Registering a Custom Spatial Data Provider

This appendix shows a sample implementation of a spatial data provider, and explains how to register this provider to be used with the map visualization component.

The complete implementation can be found under the map visualization component `web/demo/spatialprovider` directory. The implementation uses then following files:

- `us_bigcities.xml`: sample XML file that the provider parses
- `customSpatialProviderSample.java`: Java implementation of the spatial data provider
- `spatialprovider.jar`: jar file with the compiled version of the customSpatialProviderSample.java source file

The `us_bigcities.xml` file has sections to define the data attributes, the data extents, and the feature information, including the geometry (in GML format) and the attribute values. This file includes the following:

```
<?xml version="1.0" standalone="yes"?>
<spatial_data>

<data_attributes>
  <attribute name="city" type="string" />
  <attribute name="state_abrv" type="string" />
  <attribute name="pop90" type="double" />
</data_attributes>

<data_extents>
   <xmin> -122.49586 </xmin>
   <ymin> 29.45765 </ymin>
   <xmax> -73.943849 </xmax>
   <ymax> 42.3831 </ymax>
</data_extents>

<geoFeature>
  <attributes> New York,NY,7322564 </attributes>
  <geometricProperty>
    <Point>
      <coordinates>-73.943849, 40.6698</coordinates>
    </Point>
   </geometricProperty>
 </geoFeature>

. . .
</spatial_data>
```

- Implementing the Spatial Provider Class
  The spatial provider class interface must be implemented.
- Registering the Spatial Provider with the Map Visualization Component
  To register the spatial provider with the map visualization component, add the following in the spatial provider section of the map visualization component configuration file, and then restart the map visualization component.

- Rendering the External Spatial Data
  To enable you to render the sample external spatial data that comes with the map visualization component kit, create a data source pointing to this data.

B.1 Implementing the Spatial Provider Class

The spatial provider class interface must be implemented.

The provider must implement the class interface shown in Custom Geometry Themes. Example B-1 contains the partial code for the spatial provider in the supplied demo. Note that this sample code is deliberately simplified; it is not optimized, and the provider does not create any spatial indexing mechanism.

After you have implemented the provider code, compile it and generate a jar file with this compiled class. The file `spatialprovider.jar` in the demo directory contains the compiled version of this sample code, and you can use it directly. Copy this jar file to a directory that is part of the map visualization component's CLASSPATH definition, such as the `web/WB-INF/lib` directory.

Example B-1 Implementing the Spatial Provider Class

```
package spatialprovider.samples;

import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.io.File;
import java.util.ArrayList;
import java.util.Properties;
import java.util.StringTokenizer;
import java.util.Vector;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import oracle.mapviewer.share.Field;
import oracle.mapviewer.share.ext.SDataProvider;
import oracle.mapviewer.share.ext.SDataSet;
import oracle.mapviewer.share.ext.SObject;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import oracle.spatial.geometry.JGeometry;
import oracle.spatial.util.GML;

public class CustomSpatialProviderSample implements SDataProvider
{
 ...

 /**
  * Constructor.
  */
 public CustomSpatialProviderSample()
 {
   ...
 }

 /**
  * Returns the initialization parameters for the provider.
  * The "datadir" parameter should be registered in the map visualization component
  * configuration file and can be used to access the data.
  * @return
  */
```

```
public String[] getInitParameterNames()
{
  return new String[]{ "datadir" };
}

/**
 * Returns runtime parameter names. Runtime parameters are additional parameters
 * that the provider may use when retrieving the data objects.
 * @return
 */
public String[] getRuntimeParameterNames()
{
  return new String[]{ "filename" };
}

/**
 * Initializes the provider
 * @param params  init properties
 * @return
 */
public boolean init(Properties params)
{
  dataDirectory = null;
    if(params == null)
    return true;
    dataDirectory = params.getProperty("datadir");
  if(dataDirectory == null || dataDirectory.trim().length() == 0)
  {
    // try upper case
    dataDirectory = params.getProperty("DATADIR");
    if(dataDirectory == null || dataDirectory.trim().length() == 0)
      System.out.println("FINE: Init properties does not define \"datadir\" parameter.");
  }
      return true;
}

/**
 * Returns the data set (geometries plus attributes) that intersects the
 * query window. In this sample the data is parsed just once and
 * there is no spatial index for searching. The search is sequential.
 * @param queryWin  search area
 * @param nonSpatialColumns   attribute columns
 * @param params      runtime properties
 * @return
 */
public SDataSet buildDataSet(Rectangle2D queryWin,
                             String []nonSpatialColumns,
                             Properties params)
{
  if(!dataParsed)
  {
    dataParsed = parseData(params);
    if(!dataParsed)
      return null;
  }
    if(geometries.size() == 0)
    return null;

  SDataSet dataset = new SDataSet();
    boolean fullExtent = isFullExtent(queryWin);
    if(fullExtent)
  {
```

```
    for(int i=0;i<geometries.size();i++)
    {
      JGeometry geom = (JGeometry)geometries.get(i);
      SObject obj = new SObject(geom,getGeometryAttributes(nonSpatialColumns,i));
      dataset.addObject(obj);      }
  }
  else
  {
    for(int i=0;i<geometries.size();i++)
    {
      JGeometry geom = (JGeometry)geometries.get(i);
      double []mbr = geom.getMBR();
      if(mbr == null)
        continue;
      Rectangle2D.Double rect = new Rectangle2D.Double(mbr[0],mbr[1],
                                mbr[2]-mbr[0],
                                mbr[3]-mbr[1]);
            if(rect.getWidth() == 0. && rect.getHeight() == 0.)
      {
        Point2D.Double pt = new Point2D.Double(mbr[0],mbr[1]);
        if(queryWin.contains(pt))
        {
          SObject obj = new SObject(geom,getGeometryAttributes(nonSpatialColumns,i));
          dataset.addObject(obj);                    }
      }
      else if(queryWin.contains(rect) || queryWin.intersects(rect))
      {
        SObject obj = new SObject(geom,getGeometryAttributes(nonSpatialColumns,i));
        dataset.addObject(obj);
      }
    }        }
    if(dataset.getSize() == 0)
    return null;
      return dataset;
}

/**
 * Returns the data provider attribute list.
 * @return
 */
public Field[] getAttributeList(Properties params)
{
  if(!dataParsed)
  {
    dataParsed = parseData(params);
    if(!dataParsed)
      return null;
  }
    if(attributes.size() == 0)
    return null;

  return (Field[])attributes.toArray(new Field[attributes.size()]);
}

/**
 * Returns the data extents.
 * @return
 */
public Rectangle2D getDataExtents(Properties params)
{
  if(!dataParsed)
  {
```

```
    dataParsed = parseData(params);
    if(!dataParsed)
      return null;
  }
    if(extents == null || extents.length < 4)
    return null;
  else
    return new Rectangle2D.Double(extents[0],extents[1],
                                  extents[2]-extents[0],
                                  extents[3]-extents[1]);
}

/**
 * Builds a spatial index for the data. In this sample there is no
 * spatial index. The data is loaded into memory when data is parsed.
 * @return
 */
public boolean buildSpatialIndex(Properties params)
{
  return true;
}

}
```

B.2 Registering the Spatial Provider with the Map Visualization Component

To register the spatial provider with the map visualization component, add the following in the spatial provider section of the map visualization component configuration file, and then restart the map visualization component.

```
<s_data_provider
  id="xmlProvider"
  class="spatialprovider.samples.CustomSpatialProviderSample"
  >
  <parameters>
    <parameter name="datadir" value="/temp/data" />
  </parameters>
</s_data_provider>
```

When you restart the map visualization component, you should see a console message that the spatial provider has been registered. For example:

```
2007-10-01 14:30:31.109 NOTIFICATION Spatial Provider xmlProvider has been registered.
```

B.3 Rendering the External Spatial Data

To enable you to render the sample external spatial data that comes with the map visualization component kit, create a data source pointing to this data.

Example B-2 is an XML request that contains a dynamic custom geometry theme.

Example B-2 Map Request to Render External Spatial Data

```
<?xml version="1.0" standalone="yes"?>
<map_request
 title="Custom Geometry Theme"
 datasource="mvdemo"
 width="500"
```

```
 height="400"
 bgcolor="#a6caf0"
 antialiase="true"
 format="PNG_STREAM"
>
 <center size="40">
   <geoFeature>
     <geometricProperty typeName="center">
       <Point>
         <coordinates>-90,32</coordinates>
       </Point>
     </geometricProperty>
   </geoFeature>
 </center>

 <themes>
   <theme name="custom_theme" >
      <custom_geom_theme
        provider_id="xmlProvider"
        srid="8307"
        render_style="M.CIRCLE"
        label_column="city"
        label_style="T.CITY NAME"
        datasource="mvdemo">
      <parameters>
       <parameter name="filename" value="/lbs/demo/spatialprovider/us_bigcities.xml"/>
      </parameters>
     </custom_geom_theme>
   </theme>
 </themes>
</map_request>
```

In Example B-2, the file name in the `<parameter>` element points to `/lbs/demo/spatialprovider/us_bigcities.xml`. If the file path is not accessible to the map visualization component, the map request may generate error messages in the log file, such as the following:

```
07/09/28 10:26:47 ParseData: Can not access file: /lbs/demo/spatialprovider/
us_bigcities.xml
07/09/28 10:26:47 ParseData: File to be parsed: /temp/data\us_bigcities.xml
07/09/28 10:26:47 ParseData: File can not be accessed on provider data directory. Copy
files there.
```

When the map visualization component searches for the file, it first tries to access the file using the original theme definition parameter; and if that fails, it tries the data directory defined in the map visualization component configuration file (`/temp/data` in the preceding example error messages). Therefore, if the original theme definition data path is not accessible to the map visualization component, copy the data files to the directory defined in the configuration file before you issue the map request.

If the map visualization component can find the data file, the map request inExample B-2 should generate an image like the one in Figure B-1.

Figure B-1 Map Image Using Custom Geometry Theme and External Spatial Data

C

OGC WMS Support in the Map Visualization Component

The map visualization component supports the rendering of data delivered using the Open GIS Consortium (OGC) Web Map Service (WMS) protocol, specifically the WMS 1.1.1 and 1.3.0 implementation specifications.

The map visualization component supports the GetMap, GetFeatureInfo, and GetCapabilities requests as defined in the OGC document 01-068r3 and 06-042.

The map visualization component does not currently support the optional Styled Layer Descriptor capability, and map visualization component will not function as a Cascading Map Server in this release.

- Setting Up the WMS Interface for the Map Visualization Component
  The map visualization component is preconfigured to run as a WMS service.
- WMS Specification and Corresponding Map Visualization Component Concepts
  This section describes the association between, or interpretation of, terms and concepts used in the WMS 1.1.1 and 1.3.0 specifications and the map visualization component.
- Adding a WMS Map Theme
  You can add a WMS map theme to the current map request. The WMS map theme is the result of a GetMap request, and it becomes an image layer in the set of layers (themes) rendered by the map visualization component.

C.1 Setting Up the WMS Interface for the Map Visualization Component

The map visualization component is preconfigured to run as a WMS service.

Internally, the map visualization component translates all incoming WMS requests into proper XML requests to the map visualization component server. For example, the following HTTP request invokes the GetCapabilities service of a the map visualization component server:

```
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&SERVICE=WMS&VERSION=1.1.1
or
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&SERVICE=WMS&VERSION=1.3.0
```

As shown in this example, the URL for the map visualization component WMS service is typically `http://host:port/mapviewer/wms?`, where *`host`* and *`port`* refer to the host and HTTP port of the map visualization component server. The context path `/mapviewer/wms` refers to the WMS interface of map visualization component.

> **Note:**
>
> All WMS requests must be on a single line, so ignore any line breaks that might appear in WMS request examples in this chapter.

C.1.1 Preparing the Data Sources and Editing the wmsConfig.xml File

For a data source to provide WMS services, you may first need to define themes and base maps, because they will become the actual service content that the map visualization server will be providing. You can use the #unique_224 to create the themes and base maps.

After creating themes and base maps, you can edit the `wmsConfig.xml file` in the `WEB-INF/conf/` folder, which is the same folder for the `mapViewerConfig.xml` file. For example, if data sources named `nedata` and `mvdemo` are allowed to provide WMS services, the `wmsConfig.xml` file may contain a section similar to the following:

```
<?xml version="1.0" ?>
<wms_config>
   . . . . . .
  <custom_parameters
            host="localhost" port="80" protocol="http"
            public_datasources="nedata, mvdemo">
    </custom_parameters>
   . . . . . .
</wms_config>
```

For changes to the `wmsConfig.xml` file take effect, you must restart the map visualization server.

To test the server, you can send a getCapabilities request. For example:

```
http://localhost:8080/mapviewer/wms?REQUEST=GetCapabilities&VERSION=1.3.0
```

C.1.2 Data Source Named wms

You must define a map visualization component data source named `wms`, unless every incoming WMS request explicitly specifies a `datasource` CGI parameter. All requests that do not specify the `datasource` parameter are by default directed to the data source named `wms`. For example, the `GetCapabilities` request will by default list all the available themes that are in the `wms` data source. (To configure the information returned by a GetCapabilities request, see Customizing WMS GetCapabilities Responses.)

C.1.3 SDO to EPSG SRID Mapping File

By default, the map visualization component uses the Oracle Spatial (SDO) native SRID (spatial reference ID) values when such information is requested in a WMS request such as GetCapabilities. The EPSG SRID values, however, are more widely used in WMS applications. To have the map visualization component use EPSG SRID values when processing WMS requests and generating responses, specify a mapping file. This mapping file is a text file that tells the map visualization component which SDO SRID values map to which EPSG SRID values. (Each pair of matching SRID values refers to the same spatial reference system.)

The mapping file contains lines where each line defines one pair of equivalent SRID values in the following format:

sdo_srid=epsg_srid

For example, the following lines define SDO SRID 8307 as equivalent to EPSG SRID 4326, and SDO SRID 81922 as equivalent to EPSG SRID 20248:

```
8307=4326
81922=20248
```

After you have created an SDO to EPSG mapping file, you can save it on the server where the map visualization component is running, and specify its location in the map visualization component configuration file using the `<sdo_epsg_mapfile>` element in the `<wms_config>` element, as explained in Customizing WMS GetCapabilities Responses.

C.2 WMS Specification and Corresponding Map Visualization Component Concepts

This section describes the association between, or interpretation of, terms and concepts used in the WMS 1.1.1 and 1.3.0 specifications and the map visualization component.

It also includes some parameters that are specific to the map visualization component but that are not in the WMS 1.1.1 and 1.3.0 specifications.

- Supported GetMap Request Parameters
- Supported GetCapabilities Request and Response Features
- Supported GetFeatureInfo Request and Response Features

C.2.1 Supported GetMap Request Parameters

This section describes the supported GetMap request parameters and their interpretation by the map visualization component. (Parameters that are specific to the map visualization component and not mentioned in the WMS 1.1.1 and 1.3.0 specifications are labeled Map Visualization Component-Only.) The supported parameters are in alphabetical order, with each in a separate subsection. Example C-1 shows some GetMap requests. (Each URL should be entered as a single string.)

Example C-1 GetMap Requests

```
http://localhost:8888/mapviewer/wms?REQUEST=GetMap&VERSION=1.1.1&FORMAT=image/gif&
SERVICE=WMS&BBOX=-121,37,-119,35&SRS=EPSG:4326&LAYERS=theme_demo_states,theme_
demo_counties,theme_demo_highways,theme_demo_cities&WIDTH=580&HEIGHT=500

http://localhost:8888/mapviewer/wms?REQUEST=GetMap&VERSION=1.3.0&FORMAT=image/gif&
SERVICE=WMS&BBOX=35,-121,37,-119&CRS=EPSG:4326&LAYERS=theme_demo_states,theme_
demo_counties,theme_demo_highways,theme_demo_cities&WIDTH=580&HEIGHT=500

http://localhost:8888/mapviewer/wms?request=GetMap&version=1.3.0&crs=none
&bbox=-122,36,-121,37&width=600&height=400&format=image/png&layers=theme_us_
states&mvthemes=<themes><theme%20name="theme_us_counties"/><theme%20name="theme_
us_road1"/></themes>&legend_
request=<legend%20bgstyle="fill:%23ffffff;stroke:%23ff0000"%20profile="medium"%20p
osition="SOUTH_EAST"><column><entry%20style="v.rb1"%20tab="1"/></column></legend>&
```

The default data source for a GetMap request is `WMS`.

The following optional GetMap parameters are not supported in the current release of the map visualization component:

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

The Map Visualization Component-Only parameters must contain valid XML fragments. Because these are supplied in an HTTP GET request, they must be appropriately encoded using a URL encoding mechanism. For example, replace each space ( ) with `%20` and each pound sign (#) with `%23`. The following example shows the use of such encoding:

```
http://localhost:8888/mapviewer/wms?request=GetMap&version=1.1.1&srs=none&bbox=-12
2,36,-121,37&width=600&height=400&format=image/png&layers=theme_us_
states&mvthemes=<themes><theme%20name="theme_us_counties"/><theme%20name="theme_
us_road1"/></themes>&legend_
request=<legend%20bgstyle="fill:%23ffffff;stroke:%23ff0000"%20profile="medium"%20p
osition="SOUTH_EAST"><column><entry%20style="v.rb1"%20tab="1"/></column></legend>&
```

- BASEMAP Parameter (Map Visualization Component-Only)
- BBOX Parameter
- BGCOLOR Parameter
- DATASOURCE Parameter (Map Visualization Component-Only)
- DYNAMIC_STYLES Parameter (Map Visualization Component-Only)
- EXCEPTIONS Parameter
- FORMAT Parameter
- HEIGHT Parameter
- LAYERS Parameter
- LEGEND_REQUEST Parameter (Map Visualization Component-Only)
- MVTHEMES Parameter (Map Visualization Component-Only)
- REQUEST Parameter
- SERVICE Parameter
- SRS (1.1.1) or CRS (1.3.0) Parameter
- STYLES Parameter
- TRANSPARENT Parameter
- VERSION Parameter
- WIDTH Parameter

C.2.1.1 BASEMAP Parameter (Map Visualization Component-Only)

The `BASEMAP` parameter specifies a named base map for the specified (or default) data source. If you specify both the `BASEMAP` and `LAYERS` parameters, all themes specified in the `LAYERS` parameters are added to the base map. Therefore, if you just want to get a map using a named base map, specify the `BASEMAP` parameter but specify an empty `LAYERS` parameter, as in the following examples:

```
REQUEST=GetMap&VERSION=1.1.1&BASEMAP=demo_map&LAYERS=&WIDTH=500&HEIGHT=560&SRS=SDO:8307&B
BOX=-122,36,-120,38.5&FORMAT=image/png
```

```
REQUEST=GetMap&VERSION=1.3.0&BASEMAP=demo_map&LAYERS=&WIDTH=500&HEIGHT=560&CRS=SDO:8307&B
BOX=-122,36,-120,38.5&FORMAT=image/png
```

The base map name can also be part of the `LAYERS` parameter. In this case, the `BASEMAP` parameter does not need to be defined in the URL (see LAYERS Parameter for additional details).

C.2.1.2 BBOX Parameter

The `BBOX` parameter specifies the lower-left and upper-right coordinates of the bounding box for the data from the data source to be displayed. It has the format `BBOX=minX,minY,maxX,maxY`. For example: `BBOX=-122,36,-120,38.5`

For version `1.3.0` and `CRS=EPSG:4326`, this `BBOX` parameter has the format of `BBOX=min_latitude,min_longitude,max_latitude,max_longitude`. For example: `BBOX=36,-122,38.5,-120`

C.2.1.3 BGCOLOR Parameter

The `BGCOLOR` parameter specifies background color for the map display using the RBG color value. It has the format `0xHHHHHH` (where each `H` is a hexadecimal value from 0 to F). For example: `BGCOLOR=0xF5F5DC` (beige).

C.2.1.4 DATASOURCE Parameter (Map Visualization Component-Only)

The `DATASOURCE` parameter specifies the name of the data source for the GetMap or GetFeatureInfo request. The default value is `WMS`. The specified data source must exist prior to the GetMap or GetFeatureInfo request. That is, it must have been created using the `<add_data_source>` MapViewer administrative request or defined in the map visualization component configuration file (`map visualization componentConfig.xml`).

C.2.1.5 DYNAMIC_STYLES Parameter (Map Visualization Component-Only)

The `DYNAMIC_STYLES` parameter specifies a `<styles>` element as part of the GetMap request..

C.2.1.6 EXCEPTIONS Parameter

For the `EXCEPTIONS` parameter, the only supported value is the default: `EXCEPTIONS=application/vnd.ogc.se_xml` for WMS 1.1.1 and `EXCEPTIONS=XML` for WMS 1.3.0. The exception is reported as an XML document conforming to the Service Exception DTD available at the following URLs:

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

`http://schemas.opengis.net/wms/1.3.0/exceptions_1_3_0.xsd`

The `application/vnd.ogc.se_inimage` (image overwritten with Exception message), and `application/vnd.ogc.se_blank` (blank image because Exception occurred) options are not supported.

C.2.1.7 FORMAT Parameter

The `FORMAT` parameter specifies the image format. The supported values are `image/gif`, `image/jpeg`, `image/png`, `image/png8`, and `image/svg+xml`.

The default value is `image/png`.

C.2.1.8 HEIGHT Parameter

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

C.2.1.9 LAYERS Parameter

The `LAYERS` parameter specifies a comma-delimited list of predefined theme names to be used for the display. The specified values are considered to a be a case-sensitive, ordered, comma-delimited list of predefined theme names in a default data source (named `WMS`) or in a named data source specified by the parameter `DATASOURCE=<name>`. For example, `LAYERS=THEME_DEMO_STATES,theme_demo_counties,THEME_demo_HIGHWAYS` translates to the following `<themes>` element in a map visualization component map request:

```
<themes>
<theme name="THEME_DEMO_STATES"/>
<theme name="theme_demo_counties"/>
<theme name="THEME_demo_HIGHWAYS"/>
</themes>
```

The base map name can also be part of the `LAYERS` list. The following example retrieves the image from a base map named DEMO_MAP:

```
http://localhost:7001/mapviewer/wms?
VERSION=1.1.1&REQUEST=GetMap&SRS=EPSG:4326&BBOX=-101.19489559164734,27.0,-78.805104408352
68,37.0&WIDTH=965&HEIGHT=431&FORMAT=image/
gif&BGCOLOR=0xA6CAF0&TRANSPARENT=TRUE&LAYERS=DEMO_MAP&STYLES=&EXCEPTIONS=application/
vnd.ogc.se_xml&datasource=mvdemo
```

See also BASEMAP Parameter (Map Visualization Component-Only).

C.2.1.10 LEGEND_REQUEST Parameter (Map Visualization Component-Only)

The `LEGEND_REQUEST` parameter specifies a `<legend>` element as part of the GetMap request. For information about the `<legend>` element, see Map Legend.

C.2.1.11 MVTHEMES Parameter (Map Visualization Component-Only)

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

C.2.1.12 REQUEST Parameter

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

C.2.1.13 SERVICE Parameter

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

C.2.1.14 SRS (1.1.1) or CRS (1.3.0) Parameter

The `SRS` parameter (WMS 1.1.1) or the `CRS` parameter (WMS 1.3.0) specifies the spatial reference system (coordinate system) for the map visualization component to use. The value must be one of the following: `SDO:srid-value` (where `srid-value` is a numeric Oracle Spatial SRID value), `EPSG:4326` (equivalent to `SDO:8307`), or `none` (equivalent to `SDO:0`).

Except for `EPSG:4326` (the standard WGS 84 longitude/latitude coordinate system), EPSG numeric identifiers are not supported. The namespace `AUTO` (WMS 1.1.1) or `AUTO2` (WMS 1.3.0), for projections that have an arbitrary center of projection, is not supported.

C.2.1.15 STYLES Parameter

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

C.2.1.16 TRANSPARENT Parameter

The `TRANSPARENT=TRUE` parameter (for a transparent image) is supported for PNG images, that is, with `FORMAT=image/png`, or `FORMAT=image/png8` for indexed (8-bit) PNG format. the map visualization component does not support transparent GIF (GIF89) images.

C.2.1.17 VERSION Parameter

The `VERSION` parameter specifies the WMS version number. The value must be `1.1.1` or `1.3.0`.

C.2.1.18 WIDTH Parameter

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

C.2.2 Supported GetCapabilities Request and Response Features

A WMS GetCapabilities request to the map visualization component should specify only the following parameters:

- `REQUEST=GetCapabilities`
- `VERSION=1.1.1` or `VERSION=1.3.0`
- `SERVICE=WMS`

For example:

```
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&VERSION=1.1.1&SERVICE=WMS
or
http://localhost:8888/mapviewer/wms?REQUEST=GetCapabilities&VERSION=1.3.0&SERVICE=WMS
```

The response is an XML document conforming to the WMS Capabilities DTD available at the following, depending on the value of the `VERSION` parameter (1.1.1 or 1.3.0):

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

`http://schemas.opengis.net/wms/1.3.0/capabilities_1_3_0.xsd`

However, the current release of the map visualization component returns an XML document containing the <Service> and <Capability> elements with the following information:

- The <Service> element is mostly empty, with just the required value of OGC:WMS for the <Service.Name> element. Support for more informative service metadata is planned for a future release of the map visualization component.
- The <Capability> element has <Request>, <Exception>, and <Layer> elements.
- The <Request> element contains the GetCapabilities and GetMap elements that describe the supported formats and URL for an HTTP GET or POST operation.
- The <Exception> element defines the exception format. The Service Exception XML is the only supported format in this release. The <Exception> element returns an XML document compliant with the Service Exception DTD, but it does not report exceptions as specified in the implementation specification. The current release simply uses the CDATA section of a <ServiceException> element to return the OMSException returned by the map visualization component server.
- The <Layer> element contains a nested set of <Layer> elements. The first (outermost) layer contains a name (WMS), a title (Oracle WebMapServer Layers by data source), and one <Layer> element for each defined data source. Each data source layer contains a <Layer> element for each defined base map and one entry for each valid theme (layer) not listed in any base map. Each base map layer contains a <Layer> element for each predefined theme in the base map.

  Themes that are defined in the USER_SDO_THEMES view, that have valid entries in the USER_SDO_GEOM_METADATA view for the base table and geometry column, and that are not used in any base map will be listed after the base maps for a data source. These themes will have no <ScaleHint> element. They will have their own <LatLonBoundingBox> and <BoundingBox> elements.

The Content-Type of the response is set to application/vnd.ogc.wms_xml, as required by the WMS implementation specification.

Because the list of layers is output by base map, a given layer or theme can appear multiple times in the GetCapabilities response. For example, the theme THEME_DEMO_STATES, which is part of the base maps named DEMO_MAP and DENSITY_MAP, appears twice in Example C-2, which is an excerpt (reformatted for readability) from a GetCapabilities response.

In Example C-2, the innermost layer describes the IMAGE_LEVEL_2 theme. The <ScaleHint> element lists the min_scale and max_scale values, if any, for that theme in the base map definition. For example, the base map definition for IMAGE_MAP is as follows:

```
SQL> select definition from user_sdo_maps where name='IMAGE_MAP';

DEFINITION
--------------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?>
<map_definition>
  <theme name="IMAGE_LEVEL_2" min_scale="1000.0" max_scale="0.0"/>
  <theme name="IMAGE_LEVEL_8" min_scale="5000.0" max_scale="1000.0"/>
  <theme name="MA_ROAD3"/>
  <theme name="MA_ROAD2"/>
  <theme name="MA_ROAD1"/>
  <theme name="MA_ROAD0"/>
</map_definition>
```

In the innermost layer, the <SRS> and <BoundingBox> elements identify the SRID and the DIMINFO information for that theme's base table, as shown in the following Spatial metadata query:

```
SQL> select srid, diminfo from user_sdo_geom_metadata, user_sdo_themes
  2  where name='IMAGE_LEVEL_2' and
  3  base_table=table_name and
  4  geometry_column=column_name ;

      SRID
----------
DIMINFO(SDO_DIMNAME, SDO_LB, SDO_UB, SDO_TOLERANCE)
--------------------------------------------------------------------------------
     41052
SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X', 200000, 500000, .5), SDO_DIM_ELEMENT('Y', 750
000, 950000, .5))
```

In Example C-2, the <Layer> element for a base map has an <SRS> element and a <LatLonBoundingBox> element. The <SRS> element is empty if all layers in the base map definition do not have the same SRID value specified in the USER_SDO_GEOM_METADATA view. If they all have the same SRID value (for example, 41052), the SRS element contains that value (for example, SDO:41052). The required <LatLonBoundingBox> element currently has default values (-180,-90,180,90). When this feature is supported by map visualization component, this element will actually be the bounds specified in the DIMINFO column of the USER_SDO_GEOM_METADATA view for that layer, converted to geodetic coordinates if necessary and possible.

All layers are currently considered to be opaque and queryable. That is, all layers are assumed to be vector layers, and not GeoRaster, logical network, or image layers.

Example C-2 GetCapabilities Response (Excerpt)

```
<Title>Oracle WebMapServer Layers by data source</Title>
<Layer>
 <Name>mvdemo</Name>
 <Title>Datasource mvdemo</Title>
   <Layer>
     <Name>DEMO_MAP</Name>
     <Title>Basemap DEMO_MAP</Title>
     <SRS>SDO:8307</SRS>
     <LatLonBoundingBox>-180,-90,180,90</LatLonBoundingBox>
. . .
   <Layer>
     <Name>DENSITY_MAP</Name>
     <Title>Basemap DENSITY_MAP</Title>
     <SRS>SDO:8307</SRS>
     <LatLonBoundingBox>-180,-90,180,90</LatLonBoundingBox>
       <Layer>
         <Name>THEME_DEMO_STATES</Name>
         <Title>THEME_DEMO_STATES</Title>
         <SRS>SDO:8307</SRS>
         <BoundingBox SRS="SDO:8307" minx="-180" miny="-90" maxx="180"
                      maxy="90" resx="0.5" resy="0.5"/>
         <ScaleHint min="50.0" max="4.0"/>
       </Layer>
. . .
   </Layer>
   <Layer>
     <Name>IMAGE_MAP</Name>
     <Title>Basemap IMAGE_MAP</Title>
     <SRS>SDO:41052</SRS>
```

```
    <LatLonBoundingBox>-180,-90,180,90</ LatLonBoundingBox>
      <Layer>
        <Name>IMAGE_LEVEL_2</Name>
        <Title>IMAGE_LEVEL_2</Title>
        <SRS>SDO:41052</SRS>
        <BoundingBox SRS="SDO:41052" minx="200000" miny="500000" maxx="750000"
                  maxy="950000" resx="0.5" resy="0.5"/>
        <ScaleHint min="1000.0" max="0.0"/>
      </Layer>
. . .
   </Layer>
```

C.2.3 Supported GetFeatureInfo Request and Response Features

This section describes the supported GetFeatureInfo request parameters and their interpretation by the map visualization component. Example C-3 shows some GetFeatureInfo requests.

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

Most of the following sections describe parameters supported for a GetFeatureInfo request. (Parameters that are specific to MapViewer and not mentioned in the WMS 1.1.1 specification are labeled Map Visualization Component-Only.) Specifying Attributes to Be Queried for a GetFeatureInfo Request explains how to query attributes in a GetFeatureInfo request.

Example C-3 GetFeatureInfo Request

```
http://localhost:8888/mapviewer/wms?REQUEST=GetFeatureInfo&VERSION=1.1.1&BBOX=0,-0
.0020,0.0040&SRS=EPSG:4326&LAYERS=cite:Lakes,cite:Forests&WIDTH=200&HEIGHT=100&INF
O_FORMAT=text/xml&QUERY_LAYERS=cite:Lakes,cite:Forests&X=60&Y=60

http://localhost:8888/mapviewer/wms?REQUEST=GetFeatureInfo&VERSION=1.3.0
&BBOX=0,-0.0020,0.0040&CRS=EPSG:4326&LAYERS=cite:Lakes,cite:Forests&WIDTH=200&HEIGHT=100
&INFO_FORMAT=text/xml&QUERY_LAYERS=cite:Lakes,cite:Forests&I=60&J=60
```

Example C-4 GetFeatureInfo Response

```
<?xml version="1.0" encoding="UTF-8" ?>
<GetFeatureInfo_Result>
  <ROWSET name="cite:Lakes">
    <ROW num="1">
      <ROWID>AAAK22AAGAAACUiAAA</ROWID>
    </ROW>
  </ROWSET>
  <ROWSET name="cite:Forests">
    <ROW num="1">
      <FEATUREID>109</FEATUREID>
    </ROW>
  </ROWSET>
</GetFeatureInfo_Result>
```

- GetMap Parameter Subset for GetFeatureInfo Requests
- EXCEPTIONS Parameter
- FEATURE_COUNT Parameter
- INFO_FORMAT Parameter
- QUERY_LAYERS Parameter
- QUERY_TYPE Parameter (Map Visualization Component-Only)

- RADIUS Parameter (Map Visualization Component-Only)
- UNIT Parameter (Map Visualization Component-Only)
- X and Y or I and J Parameters
- Specifying Attributes to Be Queried for a GetFeatureInfo Request

C.2.3.1 GetMap Parameter Subset for GetFeatureInfo Requests

A GetFeatureInfo request contains a subset of a GetMap request (`BBOX`, `SRS` [1.1.1] or `CRS` [1.3.0], `WIDTH`, `HEIGHT`, and optionally `LAYERS` parameters). These parameters are used to convert the X, Y (1.1.1) or I, J (1.3.0) point from screen coordinates to a point in the coordinate system for the layers being queried. It is assumed all layers are in the same coordinate system, the one specified by the `SRS` parameter.

C.2.3.2 EXCEPTIONS Parameter

The only supported value for the `EXCEPTIONS` parameter is the default: `application/vnd.ogc.se_xml` for WMS 1.1.1 or `xml` for WMS 1.3.0. That is, only Service Exception XML is supported. The exception is reported as an XML document conforming to the Service Exception DTD available at the following, depending on the version (1.1.1 or 1.3.0):

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

`http://schemas.opengis.net/wms/1.3.0/exceptions_1_3_0.xsd`

C.2.3.3 FEATURE_COUNT Parameter

The `FEATURE_COUNT` parameter specifies the maximum number of features in the result set. The default value is 1. If more features than the parameter's value interact with the query point (X, Y), then an arbitrary subset (of the size of the parameter's value) of the features is returned in the result set. That is, a GetFeatureInfo call translates into a query of the following general form:

```
SELECT <info_columns> FROM <layer_table>
  WHERE SDO_RELATE(<geom_column>,
    <query_point>, 'mask=ANYINTERACT')='TRUE'
  AND ROWNUM <= FEATURE_COUNT;
```

C.2.3.4 INFO_FORMAT Parameter

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

C.2.3.5 QUERY_LAYERS Parameter

The `QUERY_LAYERS` parameter specifies a comma-delimited list of layers to be queried. If the `LAYERS` parameter is specified, the `QUERY_LAYERS` specification must be a subset of the list specified in the `LAYERS` parameter.

If the `QUERY_LAYERS` parameter is specified, any `BASEMAP` parameter value is ignored.

C.2.3.6 QUERY_TYPE Parameter (Map Visualization Component-Only)

The `QUERY_TYPE` parameter limits the result set to a subset of possibly qualifying features by specifying one of the following values:

- `at_point`: returns only the feature at the specified point.
- `nn`: returns only the nearest neighbor features, with the number of results depending on the value of the `FEATURE_COUNT` parameter value (see FEATURE_COUNT Parameter). The result set is not ordered by distance.
- `within_radius` (or `within_distance`, which is a synonym): returns only results within the distance specified by the `RADIUS` parameter value (see RADIUS Parameter (Map Visualization Component-Only)), up to the number matching the value of the `FEATURE_COUNT` parameter value (see FEATURE_COUNT Parameter). The result set is an arbitrary subset of the answer set of potential features within the specified radius. The result set is not ordered by distance.

C.2.3.7 RADIUS Parameter (Map Visualization Component-Only)

The `RADIUS` parameter specifies the radius of the circular search area for a query in which the `QUERY_TYPE` parameter value is `within_radius` (see QUERY_TYPE Parameter (Map Visualization Component-Only)). If you specify the `RADIUS` parameter, you must also specify the `UNIT` parameter (see UNIT Parameter (Map Visualization Component-Only)).

C.2.3.8 UNIT Parameter (Map Visualization Component-Only)

The `UNIT` parameter specifies the unit of measurement for the radius of the circular search area for a query in which the `QUERY_TYPE` parameter value is `within_radius` (see QUERY_TYPE Parameter (Map Visualization Component-Only)). The value must be a valid linear measure value from the SHORT_NAME column of the SDO_UNITS_OF_MEASURE table, for example: `meter`, `km`, or `mile`.

If you specify the `UNIT` parameter, you must also specify the `RADIUS` parameter (see RADIUS Parameter (Map Visualization Component-Only)).

C.2.3.9 X and Y or I and J Parameters

The `X` and `Y` (WMS 1.1.1) or `I` and `J` (WMS 1.3.0) parameters specify the x-axis and y-axis coordinate values (in pixels), respectively, of the query point.

C.2.3.10 Specifying Attributes to Be Queried for a GetFeatureInfo Request

In a GetFeatureInfo request, the styling rule for each queryable layer (theme) must contain a `<hidden_info>` element that specifies which attributes are queried and returned in the XML response. The `<hidden_info>` element is the same as the one used for determining the attributes returned in an SVG map request.

An example of such a styling rule as follows:

```
SQL> select styling_rules from user_sdo_themes where name='cite:Forests';

STYLING_RULES
--------------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
          <field column="FID" name="FeatureId"/>
       </hidden_info>
  <rule>
    <features style="C.PARK FOREST">  </features>
    <label column="NAME" style="T.PARK NAME"> 1 </label>
```

```
  </rule>
</styling_rules>
```

This styling rule specifies that if `cite:Forests` is one of the `QUERY_LAYERS` parameter values in a GetFeatureInfo request, the column named FID is queried, and its tag in the response document will be `<FEATUREID>`. The tag is always in uppercase. If no `<hidden_info>` element is specified in the styling rules for the theme's query layer, then the rowid is returned. In Example C-4, the styling rule for the `cite:Lakes` layer has no `<hidden_info>` element; therefore, the default attribute ROWID is returned in the XML response. The `cite:Forests` layer, however, does have a `<hidden_info>` element, which specifies that the attribute column is FID, and that its tag name, in the response document, should be `<FEATUREID>`.

C.3 Adding a WMS Map Theme

You can add a WMS map theme to the current map request. The WMS map theme is the result of a GetMap request, and it becomes an image layer in the set of layers (themes) rendered by the map visualization component.

To add a WMS map theme, use the WMS-specific features of the XML API (see XML API for Adding a WMS Map Theme).

- XML API for Adding a WMS Map Theme
- Predefined WMS Map Theme Definition
- Authentication with WMS Map Themes
- Customizing GetCapabilities Responses: Additional Options

C.3.1 XML API for Adding a WMS Map Theme

To add a WMS map theme to the current map request using the map visualization component XML API, use the `<wms_getmap_request>` element in a `<theme>` element.

For better performance, the `<wms_getmap_request>` element should be used only to request a map image from a Web Map Server (WMS) implementation. That is, the `<service_url>` element in a `<wms_getmap_request>` element should specify a WMS implementation, not a map visualization component instance. If you want to specify a map visualization component instance (for example, specifying `<service_url>` with a value of `http://mapviewer.mycorp.com:8888/mapviewer/wms`), consider using a MapViewer predefined theme or a JDBC theme in the `<themes>` element instead of using a `<wms_getmap_request>` element.

The following example shows the general format of the `<wms_getmap_request>` element within a `<theme>` element, and it includes some sample element values and descriptive comments:

```
<themes>
  <theme>
     <wms_getmap_request isBackgroundTheme="true">
           <!-- The wms_getmap_request theme is rendered in the order it
                 appears in the theme list unless isBackgroundTheme is "true".
           -->
          <service_url> http://wms.mapsrus.com/mapserver </service_url>
          <version> 1.1.1 </version>
            <!-- version is optional. Default value is "1.1.1".
            -->
          <layers> Administrative+Boundaries,Topography,Hydrography </layers>
           <!– layers is a comma-delimited list of names.
              If layer names contain spaces, use '+' instead of a space -->
           <!– styles is optional. It is a comma-delimited list, and it must
```

```
            have the same number of  names as the layer list, if specified.
            If style names contain spaces, use '+' instead of a space -->
        <styles/>
        <srs> EPSG:4326 </srs>
        <format> image/png </format>
        <transparent> true </transparent>
        <bgcolor> 0xffffff </bgcolor>
        <exceptions> application/vnd.ogc.se_inimage </exceptions>
        <vendor_specific_parameters>
             <!-- one or more <vsp> elements each containing
                   a <name> <value> pair  -->
             <vsp>
                  <name> datasource </name>
                  <value> mvdemo </value>
               </vsp>
        <vendor_specific_parameters>
    <wms_getmap_request>
  </theme>
</themes>
```

The following attribute and elements are available with the `<wms_getmap_request>` element:

- The `isBackgroundTheme` attribute specifies whether or not this theme should be rendered before the vector layers. The default value is `false`.
- The `<service_url>` element specifies the URL (without the service parameters) for the WMS service. Example: `http://my.webmapserver.com/wms`
- The `<version>` element specifies the WMS version number. The value must be one of the following: `1.0.0`, `1.1.0`, `1.1.1` (the default), or `1.3.0`.
- The `<layers>` element specifies a comma-delimited list of layer names to be included in the map request.
- The `<styles>` element specifies a comma-delimited list of style names to be applied to the layer names in `layers`.
- The `<srs>` element specifies the coordinate system (spatial reference system) name. The default value is `EPSG:4326`.
- The `<format>` element specifies the format for the resulting map image. The default value is `image/png`.
- The `<transparent>` element specifies whether or not the layer or layers being added should be transparent in the resulting map image. The default value is `false`. To make the layer or layers transparent, specify `true`.
- The `<bgcolor>` element specifies the RGB value for the map background color. Use hexadecimal notation for the value, for example, `0xAE75B1`. The default value is `0xFFFFFF` (that is, white).
- The `<exceptions>` element specifies the format for server exceptions. The default value is `application/vnd.ogc.se_inimage`.
- The `<vendor_specific_parameters>` element contains one or more `<vsp>` elements, each of which contains a `<name>` element specifying the parameter name and a `<value>` element specifying the parameter value.

Example C-5 shows the `<wms_getmap_request>` element in a map request.

Example C-5 Adding a WMS Map Theme (XML API)

```
<?xml version="1.0" standalone="yes"?>
<map_request
  title="Raster WMS Theme and Vector Data"
  datasource="mvdemo" srid="0"
  width="500"
  height="375"
  bgcolor="#a6caf0"
  antialiase="true"
  mapfilename="wms_georaster" format="PNG_URL">
  <center size="185340.0">
    <geoFeature>
      <geometricProperty typeName="center">
        <Point>
          <coordinates>596082.0,8881079.0</coordinates>
        </Point>
      </geometricProperty>
    </geoFeature>
  </center>
  <themes>
    <theme name="WMS_TOPOGRAPHY" user_clickable="false"  >
       <wms_getmap_request isBackgroundTheme="true">
         <service_url> http://wms.mapservers.com:8888/mapserver/wms </service_url>
         <layers> TOPOGRAPHY </layers>
         <srs> EPSG:29190 </srs>
         <format> image/png </format>
         <bgcolor> 0xa6caf0 </bgcolor>
         <transparent> true </transparent>
         <vendor_specific_parameters>
           <vsp>
              <name> ServiceType </name>
              <value> mapserver </value>
           </vsp>
         </vendor_specific_parameters>
       </wms_getmap_request>
    </theme>
    <theme name="cl_theme" user_clickable="false">
       <jdbc_query  spatial_column="geom" render_style="ltblue"
         jdbc_srid="82279" datasource="mvdemo"
         asis="false">select geom from classes where vegetation_type = 'forests'
       </jdbc_query>
     </theme>
  </themes>
  <styles>
    <style name="ltblue">
      <svg width="1in" height="1in">
       <g class="color"
          style="stroke:#000000;stroke-opacity:250;fill:#33ffff;fill-opacity:100">
          <rect width="50" height="50"/>
       </g>
      </svg>
    </style>
  </styles>
</map_request>
```

C.3.2 Predefined WMS Map Theme Definition

The predefined XML definition for a WMS theme uses the same structure of the parameters in XML API for Adding a WMS Map Theme, and adds the optional `capabilities_url` attribute, which is used by Map Builder when editing a WMS theme. If the `capabilities_url` attribute is

defined, Map Builder will issue a `GetCapabilities` request to populate some UI elements in the editor page.

Example C-6 shows how to create a predefined WMS theme in the metadata. The base table and base column names can be any values, and in this example `'WMS'` is used for both.

Example C-6 Creating a Predefined WMS Theme

```
INSERT INTO user_sdo_themes  VALUES (
  'PRED_WMS_THEME',
  'WMS data',
  'WMS',
  'WMS',   '<?xml version="1.0" standalone="yes"?>
   <styling_rules theme_type="wms">
     <service_url> http://sampleserver1b.arcgisonline.com/arcgis/services/Specialty/
ESRI_StateCityHighway_USA/MapServer/WMSServer </service_url>
     <layers> 0,1,2 </layers>
     <version> 1.3.0 </version>
     <srs> CRS:84 </srs>
     <format> image/png </format>
     <bgcolor> 0xA6CAF0 </bgcolor>
     <transparent> false </transparent>
     <styles> +,+,+ </styles>
     <exceptions> xml </exceptions>
     <capabilities_url> http://sampleserver1.arcgisonline.com/ArcGIS/services/Specialty/
ESRI_StateCityHighway_USA/MapServer/WMSServer? </capabilities_url>
   </styling_rules>');
```

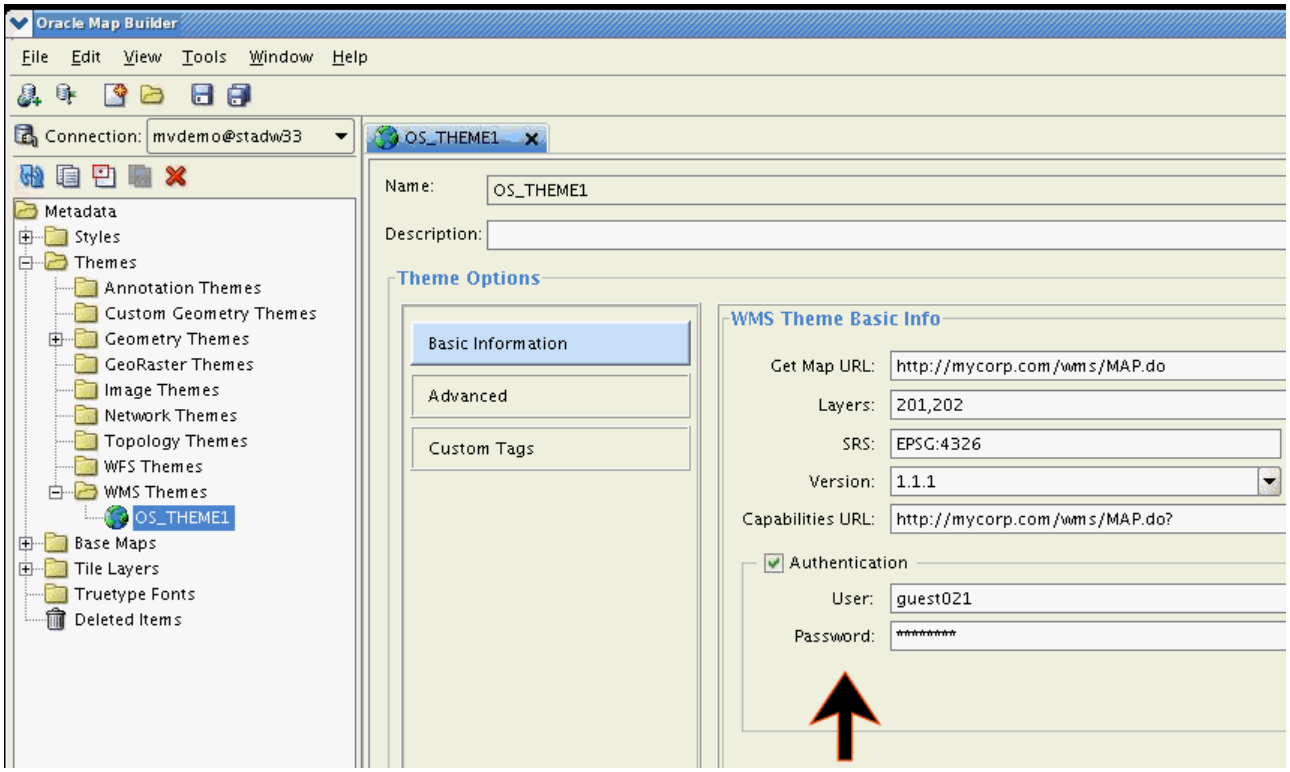
C.3.3 Authentication with WMS Map Themes

For a WMS server that requires authentication for access to the WMS data, the following must be included in the theme definition:

- `<user>` element specifying the user name
- `<password>` element specifying the user password

  If you use the Map Builder tool to create a WMS map theme, the password value will be automatically encrypted. Figure C-1 shows the use of the Map Builder tool to create a WMS theme with authentication information. In this figure, the Authentication option is checked (enabled), and User and Password are specified.

Figure C-1 Using Map Builder to Specify Authentication with a WMS Theme



Example C-7 shows how to create a WMS theme that includes authentication information.

Example C-7 WMS Theme with Authentication Specified

```
<?xml version="1.0" standalone="yes"?>
<styling_rules theme_type="wms">
    <service_url> http://localhost:7001/mapviewer/wms </service_url>
    <user> wmsuser </user>
    <password> ******* </password>
    <layers> THEME_DEMO_STATES </layers>
    <version> 1.1.1 </version>
    <srs> EPSG:4326 </srs>
    <format> image/png </format>
    <bgcolor> 0xA6CAF0 </bgcolor>
    <transparent> true </transparent>
    <exceptions> application/vnd.ogc.se_xml </exceptions>
    <vendor_specific_parameters>
        <vsp>
            <name> datasource </name>
            <value> mvdemo </value>
    </vsp>
  </vendor_specific_parameters>
    <capabilities_url> http://localhost:7001/mapviewer/wms? </capabilities_url>
</styling_rules>
```

C.3.4 Customizing GetCapabilities Responses: Additional Options

The main map visualization component configuration file has a section to customize some information for GetCapabilities responses (see Customizing WMS GetCapabilities Responses). However only a limited number of properties that can be defined in that main map visualization component configuration file, and no customization can be done for several attributes that a layer can have in GetCapabilities response.

Effective with Release 12.1.3, a WMS configuration file (`wmsConfig.xml`) has been added to the map visualization component. Any settings defined in this WMS configuration file will override similar WMS settings defined in the main map visualization component configuration file.

The `wmsConfig.xml` can include the following.

- Custom WMS Capabilities Parameters (<custom_parameters> Element)
- Custom WMS Capabilities Service Attributes (<service_attributes> Element)
- Custom WMS Layer Attributes (<layer_attributes> Element)
- Custom WMS Feature Information (<get_feature_info> Element)

C.3.4.1 Custom WMS Capabilities Parameters (<custom_parameters> Element)

The following attributes are available with the `<custom_parameters>` element:

- The `host` attribute specifies the host part of the service request URL that the client should use for future WMS requests made to this map visualization component server.
- The `port` attribute specifies the port part of the service request URL that the client should use for future WMS requests made to this map visualization component server.
- The `protocol` attribute specifies the protocol part of the service request URL that the client should use for future WMS requests made to this map visualization component server.
- The `default_datasource` attribute specifies the base data source to be used for retrieving the capabilities response. If this attribute is not defined, the data source `WMS` is used, and that data source must exist in this map visualization component server.
- The `public_datasources` attribute specifies which data source contents are to be listed in the getCapabilities response. If this attribute is not defined, all data source contents that exist in server will be listed.
- The `use_text_xml` attribute by default uses standard `text/xml` even for getCapabilities. If this attribute is set to `false`, then the OGC 1.1.1 specification is used (for example, `application/vnd.ogc.wms_xml`).

For example:

```
<custom_parameters
        host="www.my_corp.com" port="80" protocol="http"
        default_datasource="mvdemo"
        public_datasources="mvdemo, wms">
</custom_parameters>
```

C.3.4.2 Custom WMS Capabilities Service Attributes (<service_attributes> Element)

The <service attributes> element overrides any of these values defined in main map visualization component configuration file. For example:

```
<service_attributes>
 <title>
     WMS interface provided by OracleFMW MapViewer
 </title>
 <abstract>
     This WMS service is for demonstration.
 </abstract>
 <keyword_list>
    <keyword>Oracle</keyword>
```

```
    <keyword>MapViewer</keyword>
    <keyword>Spatial and Graph</keyword>
 </keyword_list>
 <online_resource>
     http://localhost:7001/mapviewer/wms
 </online_resource>
 <contact_information>
    <ContactPersonPrimary>
       <ContactPerson>John Doe</ContactPerson>
       <ContactOrganization>My Corp.</ContactOrganization>
    </ContactPersonPrimary>
    <ContactPosition>Sr. Manager</ContactPosition>
    <ContactAddress>
       <AddressType>US Street</AddressType>
       <Address>1 MyCorp drv</Address>
       <City>Nashua</City>
       <StateOrProvince>NH</StateOrProvince>
       <PostCode>03062</PostCode>
       <Country>USA</Country>
    </ContactAddress>
    <ContactVoiceTelephone>18001122333</ContactVoiceTelephone>
    <ContactFacsimileTelephone></ContactFacsimileTelephone>
    <ContactElectronicMailAddress>jdoe@my_corp.com</ContactElectronicMailAddress>
 </contact_information>
 <Fees>None</Fees>
 <AccessConstraints>None whatsoever</AccessConstraints>
</service_attributes>
```

C.3.4.3 Custom WMS Layer Attributes (<layer_attributes> Element)

The WMS OGC specification defines several attributes for Layers. The layer attribute list includes abstract, keyword list, SRS list, style, attribution, and others.

The `<layer_attributes>` element lets you define custom attributes for map visualization component metadata (base maps and themes). WMS Layers are associated with map visualization component themes and base maps. All Layer attributes are optional, and if defined they will complement the GetCapabilities response.

Example C-8 shows a `<layer_attributes>` element with custom parameters for a base map and for a theme. In this example, for the base map DEMO_MAP the custom element `<abstract>` is defined, while for the theme THEME_DEMO_STATES the custom elements `<abstract>`, `<keywordlist>`, `<srs>`, and `<style>` are defined. In the `<metadata>` element, use the value *theme* for map visualization component themes and the value *basemap* for map visualization component base maps in the type attribute. If the `type` attribute is not defined or value is not equal to `theme` or `basemap`, then `theme` is used by default.

Example C-9 shows part of a GetCapabilities response that includes these custom attributes.

Example C-8 Custom WMS Layer Attributes

```
<layer_attributes>
  <datasource name="mvdemo">
    <metadata name="DEMO_MAP"  type="basemap">
           <abstract>Collection of US States</abstract>
    </metadata>
     <metadata name="THEME_DEMO_STATES"  type="theme"
            queryable="1" cascaded="0" opaque="1" noSubsets="0"
            fixedWidth="500" fixedHeight="400">

        <abstract>US States</abstract>
```

```
        <keywordlist>keyword1,keyword2,keyword3</keywordlist>

        <srs>EPSG:4203,EPSG:20248</srs>

        <style>
           <title>Style title</title>
           <name>Style name</name>
           <abstract>Style abstract</abstract>
           <stylesheeturl>
              <format>Stylesheet format</format>
              <onlineresource>
                 <href>http://www.yoururl.com/styledata.html</href>
                 <type>simple</type>
              </onlineresource>
           </stylesheeturl>
           <styleurl>
              <format>Styles format</format>
              <onlineresource>
                 <href>http://www.yoururl.com/style.html</href>
                 <type>simple</type>
              </onlineresource>
           </styleurl>
           <legendurl>
             <format>Legend format</format>
             <width>500</width>
             <height>500</height>
             <onlineresource>
                <href>http://www.yoururl.com/legendurl.html</href>
                <type>simple</type>
             </onlineresource>
           </legendurl>
         </style>
   ...

    </metadata>
  </datasource>
</layer_attributes>
```

Example C-9 GetCapabilities Response with Custom Attributes

```
<Layer>
  <Name>DEMO_MAP</Name>
  <Title>Basemap DEMO_MAP</Title>
  <Abstract>Collection of US States</Abstract>
  <SRS>EPSG:4326</SRS>
  <LatLonBoundingBox maxy="90.0" maxx="180.0" miny="-90.0" minx="-180.0"/>
  <Layer fixedHeight="400" fixedWidth="500" noSubsets="0" opaque="1" cascaded="0"
queryable="1">
    <Name>THEME_DEMO_STATES</Name>
    <Title>THEME_DEMO_STATES</Title>
    <Abstract>US States</Abstract>
    <KeywordList>
      <Keyword>keyword1</Keyword>
      <Keyword>keyword2</Keyword>
      <Keyword>keyword3</Keyword>
    </KeywordList>
    <SRS>EPSG:4326</SRS>
    <SRS>EPSG:4203</SRS>
    <SRS>EPSG:20248</SRS>
    <BoundingBox resy="5.0E-8" resx="5.0E-8" maxy="71.33268128071118" maxx="180.0"
miny="-14.605189123107024" minx="-180.0" SRS="EPSG:4326"/>
    <BoundingBox resy="5.0E-8" resx="5.0E-8" maxy="2.06926736125942E7"
```

```
maxx="5.5812377162013E7" miny="-1.54158038049741E9" minx="-5.74425799715062E8"
SRS="EPSG:4203"/>
    <BoundingBox resy="5.0E-8" resx="5.0E-8" maxy="2.06929693795843E7"
maxx="6.15958436393193E7" miny="-1.74609707025816E9" minx="-6.32398501646291E8"
SRS="EPSG:20248"/>
    <Style>
      <Name>Style name</Name>
      <Title>Style title</Title>
      <Abstract>Style abstract</Abstract>
      <LegendURL height="500" width="500">
        <Format>Legend format</Format>
        <OnlineResource xlink:href="http://www.yoururl.com/legendurl.html"
xlink:type="simple"/>
      </LegendURL>
      <StyleSheetURL>
        <Format>Stylesheet format</Format>
        <OnlineResource xlink:href="http://www.yoururl.com/styledata.html"
xlink:type="simple"/>
      </StyleSheetURL>
      <StyleURL>
        <Format>Stylesheet format</Format>
        <OnlineResource xlink:href="http://www.yoururl.com/style.html"
xlink:type="simple"/>
      </StyleURL>
    </Style>
    <ScaleHint max="1.5E8" min="0.0"/>
  </Layer>
...
</Layer>
```

C.3.4.4 Custom WMS Feature Information (<get_feature_info> Element)

The following attributes, available with the `<get_feature_info>` element, can be used define the default radius (value and unit) for predefined map visualization component themes:

- The `name` attribute specifies the name of the predefined theme.
- The `datasource` attribute specifies map visualization component data source.
- The `radius` attribute specifies the default radius value for theme to be used in `within_radius` GetFeatureInfo requests when the `radius` parameter is not defined.
- The `unit` attribute specifies default radius unit (default is `m` for meters).

For example:

```
<get_feature_info>
  <theme name="theme_demo_states" datasource="mvdemo" radius="500" unit="km" />
</get_feature_info>
```

D

OGC WMTS Support in the Map Visualization Component

The map visualization component supports the rendering of data delivered using the Open GIS Consortium (OGC) Web Map Tile Service (WMTS) protocol, specifically the WMS 1.0.0 implementation specifications

The map visualization component supports the GetCapabilities, GetTile, and GetFeatureInfo requests as defined in the OGC document 07-057r7.

- WMTS Service for MapViewer
  A Web Map Tile Service (WMTS) service in MapViewer is provided through its WMTS server, which is built on MapViewer's map cache server.
- WMTS Operations
  MapViewer supports the GetCapabilities, GetTile, and GetFeatureInfo operations specified in the OGC WMTS 1.0.0 standard.
- Preparing the WMTS Service for MapViewer
  To prepare for using the WMTS Service for MapViewer, follow these major steps:

D.1 WMTS Service for MapViewer

A Web Map Tile Service (WMTS) service in MapViewer is provided through its WMTS server, which is built on MapViewer's map cache server.

That is, this WMTS server delegates a client's WMTS request to MapViewer's map cache server. The OGC operations GetCapabilities, GetTile, and GetFeatureInfo are supported. Two kinds of encoding, KVP and REST, are supported for GetCapabilities and GetTile operations, but for GetFeatureInfo operations only KVP encoding is supported. For a GetFeatureInfo request, the response is encoded in one of the three formats (specified with the `infoformat` attribute): text/xml, text/html, and application/json.

In general, all map cache tile layers in MapViewer can be accessed through its WMTS service. You can customize each map cache tile layer's accessibility by editing a WMTS service policy file, `wmtsConfig.xml`, stored in the same folder as the `mapViewerConfig.xml` file. In WMTS service policy file, you can limit MapViewer's WMTS service by only publishing a subset of all map cache tile layers in MapViewer server (see Customizing WMTS GetCapabilities Responses for details).

Example D-1 and Example D-2 show how to publish a tile layer to MapViewer's WMTS server using a policy file. The first example shows the policy file that specifies a tile layer, and the second shows the tile layer definition.

- Example D-1 is a WMTS policy file. In this file, only one map cache tile layer named TEST_TL in data source MVDEMO is published to MapViewer's WMTS service. This policy file may also contain service provider related information.
- Example D-2 is the map tile layer definition for the TEST_TL tile layer in the schema for data source MVDEMO.

Regardless of whether there are additional data sources in the MapViewer server, or additional map cache tile layers in the specified data source, if the policy file is specified as in

Example D-1, the only published WMTS layer is map cache tile layer TEST_TL in data source MVDEMO (shown in Example D-2).

Example D-1 WMTS Policy File (wmtConfig.xml)

```
<wmts_config>
     <public_datasources>
        <public_datasource name="MVDEMO">
          <tile_layers>
            <tile_layer name="TEST_TL"/>
           </tile_layers>
        </public_datasource>
     </public_datasources>
   <ServiceAttributes>
    <ServiceIdentification>
           <Title>Web Map Tile Service by myCorp</Title>
           <Abstract> U.S. maps for state and county boundaries, highway networks, and
big cities.</Abstract>
           <Keywords>
             <Keyword>Maps</Keyword>
             <Keyword>U.S. State, County Boundaries</Keyword>
             <Keyword>U.S. Interstate Highways</Keyword>
             <Keyword>U.S. Cities</Keyword>
           </Keywords>
           <Fees>none</Fees>
           <AccessConstraints>none</AccessConstraints>
    </ServiceIdentification>
    <ServiceProvider>
      <ProviderName>provider's name</ProviderName>
        <ProviderSite url="http://www.myCorp.com/mySite"/>
        <ServiceContact>
          <IndividualName>my name</IndividualName>
          <PositionName>technical support specialist</PositionName>
          <ContactInfo>
            <Phone>
              <Voice>+1 800 321 1234</Voice>
              <Facsimile>+1 800 321 1235</Facsimile>
            </Phone>
            <Address>
              <DeliveryPoint>123 My Street</DeliveryPoint>
              <City>Nashua</City>
              <AdministrativeArea>New Hampshire</AdministrativeArea>
              <PostalCode>12345-4321</PostalCode>
              <Country>U.S.</Country>
              <ElectronicMailAddress>myname@mycompany.com</ElectronicMailAddress>
            </Address>
          </ContactInfo>
      </ServiceContact>
    </ServiceProvider>
  </ServiceAttributes>
</wmts_config>
```

Example D-2 Tile Layer Definition in the Data Source

```
SQL> select definition from  user_sdo_cached_maps where name='TEST_TL';

DEFINITION
--------------------------------------------------------------------------------
<map_tile_layer name="TEST_TL" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3">
   <internal_map_source base_map="DEMO_MAP" data_source="MVDEMO"/>
   <coordinate_system srid="8307" minX="-180" maxX="180" minY="-90" maxY="90"/>
   <tile_image width="256" height="256"/>
```

```
   <tile_dpi value="90.7142857"/>
   <tile_meters_per_unit value="111319.49079327358"/>
   <zoom_levels levels="19" min_scale="2132.729583849784" max_scale="559082264.0287178">
      <zoom_level tile_width="360.0000" tile_height="360.0000" scale="5.590822640287178E8"/>
      <zoom_level tile_width="180.0000" tile_height="180.0000" scale="2.795411320143589E8"/>
      <zoom_level tile_width="90.0000" tile_height="90.00000" scale="1.3977056600717944E8"/>
      <zoom_level tile_width="45.0000" tile_height="45.00000" scale="6.988528300358972E7"/>
      <zoom_level tile_width="22.5000" tile_height="22.50000" scale="3.494264150179486E7"/>
      <zoom_level tile_width="11.2500" tile_height="11.25000" scale="1.747132075089743E7"/>
      <zoom_level tile_width="5.6250" tile_height="5.62500" scale="8735660.375448715"/>
      <zoom_level tile_width="2.8125" tile_height="2.81250" scale="4367830.1877243575"/>
      <zoom_level tile_width="1.40625" tile_height="1.40625" scale="2183915.0938621787"/>
      <zoom_level tile_width="0.703125" tile_height="0.703125" scale="1091957.5469310894"/>
      <zoom_level tile_width="0.3515625" tile_height="0.3515625" scale="545978.7734655447"/>
      <zoom_level tile_width="0.17578125" tile_height="0.17578125" scale="272989.38673277234"/>
      <zoom_level tile_width="0.087890625" tile_height="0.087890625" scale="136494.693366386"/>
      <zoom_level tile_width="0.0439453125" tile_height="0.0439453125" scale="68247.34668319"/>
      <zoom_level tile_width="0.02197265625" tile_height="0.02197265625" scale="34123.6733415"/>
      <zoom_level tile_width="0.010986328126" tile_height="0.010986328126" scale="17061.8366707"/>
      <zoom_level tile_width="0.0054931640633" tile_height="0.0054931640633" scale="8530.91833539"/>
      <zoom_level tile_width="0.00274658203168" tile_height="0.00274658203168" scale="4265.45916769"/>
      <zoom_level tile_width="0.001373291015841" tile_height="0.001373291015841" scale="2132.72958384"/>
   </zoom_levels>
</map_tile_layer>
```

D.2 WMTS Operations

MapViewer supports the GetCapabilities, GetTile, and GetFeatureInfo operations specified in the OGC WMTS 1.0.0 standard.

It also supports some MapViewer-specific parameters that are not defined in that standard, but which make WMTS operations in MapViewer more flexible.

- GetCapabilities Operation Support
- GetTile Operation Support
- GetFeatureInfo Operation Support

D.2.1 GetCapabilities Operation Support

When the MapViewer server receives a WMTS GetCapabilities request, its WMTS server translates the definition of available map cache tile layers into an XML document according to the OGC WMTS specification. (The accessible cache tile layers are defined by the `wmtsConfig.xml` policy file if it exists; otherwise, all map cache tile layers from all data sources are available to WMTS server are used.) Then the XML document is returned to the client. Within this returned XML document, the information about the service provider, if specified in the `wmtsConfig.xml` policy file, is provided.

This topic includes a GetCapabilities request in KVP encoding and its response, and a GetCapabilities request in REST encoding and its response. Regardless of whether the request is in KVP or REST encoding, the base URL must be in the following format:

`http://<host>:<port>/mapviewer/wmts`

where <*host*> and <*port*> refer to the host and HTTP port of the MapViewer server. The context path `/mapviewer/wmts` refers to the WMTS interface of MapViewer.

Example D-3 is the response from the following example GetCapabilities request in KVP coding:

```
http://localhost:8088/mapviewer/wmts?REQUEST=GetCapabilities&SERVICE=WMTS&VERSION=1.0.0
```

Example D-4 is the response from the following example GetCapabilities request in REST coding:

```
http://localhost:8088/mapviewer/wmts/1.0.0/WMTSCapabilities.xml
```

In Example D-3 and Example D-4, bold text for `<ows:Value>` elements shows differences in their responses when requests are in different encodings. (The encoding to use is an application-specific preference.) In these example responses, the GetTile operations are given in the same encoding as that of the GetCapabilities request's encoding;, but the GetFeatureInfo operations are always in KVP encoding.

With the MapViewer WMTS server, there is no significant performance difference between use of the two encodings.

Example D-3 Response from GetCapabilities Request in KVP Encoding

```
<?xml version="1.0" encoding="UTF-8"?>
<Capabilities xmlns="http://www.opengis.net/wmts/1.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="http://www.opengis.net/wmts/1.0 http://schemas.opengis.net/wmts/1.0/
wmtsGetCapabilities_response.xsd"
version="1.0.0">
<ows:ServiceIdentification>
  <ows:Title>Web Map Tile Service by myCorp</ows:Title>
  <ows:Abstract>U.S. maps for state and county boundaries, highway networks, and big
cities.</ows:Abstract>
  <ows:Keywords>
    <ows:Keyword>Maps</ows:Keyword>
    <ows:Keyword>U.S. State and County Boundaries</ows:Keyword>
    <ows:Keyword>U.S. Interstate Highways</ows:Keyword>
    <ows:Keyword>U.S. Cities</ows:Keyword>
  </ows:Keywords>
  <ows:ServiceType>OGC WMTS</ows:ServiceType>
  <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
  <ows:Fees>none</ows:Fees>
  <ows:AccessConstraints>none</ows:AccessConstraints>
</ows:ServiceIdentification>
<ows:ServiceProvider>
  <ows:ProviderName>provider's name</ows:ProviderName>
  <ows:ProviderSite xlink:href="http://www.myCorp.com/mySite" />
  <ows:ServiceContact>
    <ows:PositionName>technical support specialist</ows:PositionName>
    <ows:ContactInfo>
      <ows:Phone>
        <ows:Voice>+1 800 321 1234</ows:Voice>
        <ows:Facsimile>+1 800 321 1235</ows:Facsimile>
      </ows:Phone>
      <ows:Address>
        <ows:DeliveryPoint>123 My Street</ows:DeliveryPoint>
        <ows:City>Nashua</ows:City>
        <ows:AdministrativeArea>New Hampshire</ows:AdministrativeArea>
        <ows:PostalCode>12345-4321</ows:PostalCode>
        <ows:Country>U.S.</ows:Country>
        <ows:ElectronicMailAddress>myname@mycompany.com</ows:ElectronicMailAddress>
      </ows:Address>
    </ows:ContactInfo>
  </ows:ServiceContact>
```

```
</ows:ServiceProvider>
<ows:OperationsMetadata>
  <ows:Operation name="GetCapabilities">
    <ows:DCP>
      <ows:HTTP>
        <ows:Get xlink:href="http://localhost:8088/mapviewer/wmts?">
          <ows:Constraint name="GetEncoding">
            <ows:AllowedValues>
              <ows:Value>KVP</ows:Value>
            </ows:AllowedValues>
          </ows:Constraint>
        </ows:Get>
      </ows:HTTP>
    </ows:DCP>
  </ows:Operation>
  <ows:Operation name="GetTile">
    <ows:DCP>
      <ows:HTTP>
        <ows:Get xlink:href="http://localhost:8088/mapviewer/wmts?">
          <ows:Constraint name="GetEncoding">
            <ows:AllowedValues>
              <ows:Value>KVP</ows:Value>
            </ows:AllowedValues>
          </ows:Constraint>
        </ows:Get>
      </ows:HTTP>
    </ows:DCP>
  </ows:Operation>
  <ows:Operation name="GetFeatureInfo">
    <ows:DCP>
      <ows:HTTP>
        <ows:Get xlink:href="http://localhost:8088/mapviewer/wmts?">
          <ows:Constraint name="GetEncoding">
            <ows:AllowedValues>
              <ows:Value>KVP</ows:Value>
            </ows:AllowedValues>
          </ows:Constraint>
        </ows:Get>
      </ows:HTTP>
    </ows:DCP>
  </ows:Operation>
</ows:OperationsMetadata>
<Contents>
  <Layer>
    <ows:Title>TEST_TL</ows:Title>
    <ows:Abstract>datasource:MVDEMO,layer:TEST_TL</ows:Abstract>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>-180.0 -90.0</ows:LowerCorner>
      <ows:UpperCorner>180.0 90.0</ows:UpperCorner>
    </ows:WGS84BoundingBox>
    <ows:Identifier>MVDEMO:TEST_TL</ows:Identifier>
    <Style isDefault="true">
      <ows:Identifier>_null</ows:Identifier>
    </Style>
    <Format>image/png</Format>
    <TileMatrixSetLink>
      <TileMatrixSet>MVDEMO:TEST_TL:EPSG:4326</TileMatrixSet>
      <TileMatrixSetLimits>
        <TileMatrixLimits>
        . . . . . <!-- omitted to save space  -->
        <TileMatrixLimits>
          <TileMatrix>MVDEMO:TEST_TL:EPSG:4326:8</TileMatrix>
```

```
          <MinTileRow>0</MinTileRow>
          <MaxTileRow>128</MaxTileRow>
          <MinTileCol>0</MinTileCol>
          <MaxTileCol>256</MaxTileCol>
        </TileMatrixLimits>
        . . . . . <!-- omitted to save space  -->
      </TileMatrixSetLimits>
    </TileMatrixSetLink>
  </Layer>
  <TileMatrixSet>
    <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326</ows:Identifier>
    <ows:SupportedCRS>urn:ogc:def:crs:EPSG:4326</ows:SupportedCRS>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:0</ows:Identifier>
      <ScaleDenominator>5.590822641167623E8</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>2</MatrixWidth>
      <MatrixHeight>2</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:1</ows:Identifier>
      <ScaleDenominator>2.7954113205838114E8</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>3</MatrixWidth>
      <MatrixHeight>2</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:2</ows:Identifier>
      <ScaleDenominator>1.3977056602919057E8</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>5</MatrixWidth>
      <MatrixHeight>3</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:3</ows:Identifier>
      <ScaleDenominator>6.988528301459529E7</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>9</MatrixWidth>
      <MatrixHeight>5</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:4</ows:Identifier>
      <ScaleDenominator>3.494264150729764E7</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>17</MatrixWidth>
      <MatrixHeight>9</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:5</ows:Identifier>
      <ScaleDenominator>1.747132075364882E7</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
```

```
  <TileHeight>256</TileHeight>
  <MatrixWidth>33</MatrixWidth>
  <MatrixHeight>17</MatrixHeight>
</TileMatrix>
<TileMatrix>
  <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:6</ows:Identifier>
  <ScaleDenominator>8735660.37682441</ScaleDenominator>
  <TopLeftCorner>-180.0  90.0</TopLeftCorner>
  <TileWidth>256</TileWidth>
  <TileHeight>256</TileHeight>
  <MatrixWidth>65</MatrixWidth>
  <MatrixHeight>33</MatrixHeight>
</TileMatrix>
<TileMatrix>
  <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:7</ows:Identifier>
  <ScaleDenominator>4367830.188412205</ScaleDenominator>
  <TopLeftCorner>-180.0  90.0</TopLeftCorner>
  <TileWidth>256</TileWidth>
  <TileHeight>256</TileHeight>
  <MatrixWidth>129</MatrixWidth>
  <MatrixHeight>65</MatrixHeight>
</TileMatrix>
<TileMatrix>
  <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:8</ows:Identifier>
  <ScaleDenominator>2183915.0942061027</ScaleDenominator>
  <TopLeftCorner>-180.0  90.0</TopLeftCorner>
  <TileWidth>256</TileWidth>
  <TileHeight>256</TileHeight>
  <MatrixWidth>257</MatrixWidth>
  <MatrixHeight>129</MatrixHeight>
</TileMatrix>
<TileMatrix>
  <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:9</ows:Identifier>
  <ScaleDenominator>1091957.5471030513</ScaleDenominator>
  <TopLeftCorner>-180.0  90.0</TopLeftCorner>
  <TileWidth>256</TileWidth>
  <TileHeight>256</TileHeight>
  <MatrixWidth>513</MatrixWidth>
  <MatrixHeight>257</MatrixHeight>
</TileMatrix>
<TileMatrix>
  <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:10</ows:Identifier>
  <ScaleDenominator>545978.7735515257</ScaleDenominator>
  <TopLeftCorner>-180.0  90.0</TopLeftCorner>
  <TileWidth>256</TileWidth>
  <TileHeight>256</TileHeight>
  <MatrixWidth>1025</MatrixWidth>
  <MatrixHeight>513</MatrixHeight>
</TileMatrix>
<TileMatrix>
  <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:11</ows:Identifier>
  <ScaleDenominator>272989.38677576283</ScaleDenominator>
  <TopLeftCorner>-180.0  90.0</TopLeftCorner>
  <TileWidth>256</TileWidth>
  <TileHeight>256</TileHeight>
  <MatrixWidth>2049</MatrixWidth>
  <MatrixHeight>1025</MatrixHeight>
</TileMatrix>
<TileMatrix>
  <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:12</ows:Identifier>
  <ScaleDenominator>136494.69338788142</ScaleDenominator>
  <TopLeftCorner>-180.0  90.0</TopLeftCorner>
```

```
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>4097</MatrixWidth>
      <MatrixHeight>2049</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:13</ows:Identifier>
      <ScaleDenominator>68247.34669394071</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>8193</MatrixWidth>
      <MatrixHeight>4097</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:14</ows:Identifier>
      <ScaleDenominator>34123.673346970354</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>16385</MatrixWidth>
      <MatrixHeight>8193</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:15</ows:Identifier>
      <ScaleDenominator>17061.836673485177</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>32769</MatrixWidth>
      <MatrixHeight>16385</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:16</ows:Identifier>
      <ScaleDenominator>8530.918336742589</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>65537</MatrixWidth>
      <MatrixHeight>32769</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:17</ows:Identifier>
      <ScaleDenominator>4265.459168371294</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>131073</MatrixWidth>
      <MatrixHeight>65537</MatrixHeight>
    </TileMatrix>
    <TileMatrix>
      <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326:18</ows:Identifier>
      <ScaleDenominator>2132.729584185647</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>262145</MatrixWidth>
      <MatrixHeight>131073</MatrixHeight>
    </TileMatrix>
  </TileMatrixSet>
</Contents>
<ServiceMetadataURL xlink:href="http://localhost:8088/mapviewer/wmts?
```

```
SERVICE=WMTS&amp;VERSION=1.0.0&amp;REQUEST=getCapabilities"/>
</Capabilities>
```

Example D-4 Response from GetCapabilities Request in REST Encoding

```
<?xml version="1.0" encoding="UTF-8"?>
<Capabilities xmlns="http://www.opengis.net/wmts/1.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="http://www.opengis.net/wmts/1.0 http://schemas.opengis.net/wmts/1.0/
wmtsGetCapabilities_response.xsd"
version="1.0.0">
<ows:ServiceIdentification>
  <!-- omitted to save space, the same as in KVP encoding -->
</ows:ServiceIdentification>
<ows:ServiceProvider>
  <!-- omitted to save space, the same as in KVP encoding -->
</ows:ServiceProvider>
<ows:OperationsMetadata>
  <ows:Operation name="GetCapabilities">
    <ows:DCP>
      <ows:HTTP>
        <ows:Get xlink:href="http://localhost:8088/mapviewer/wmts">
          <ows:Constraint name="GetEncoding">
            <ows:AllowedValues>
              <ows:Value>RESTful</ows:Value>
            </ows:AllowedValues>
          </ows:Constraint>
        </ows:Get>
      </ows:HTTP>
    </ows:DCP>
  </ows:Operation>
  <ows:Operation name="GetTile">
    <ows:DCP>
      <ows:HTTP>
        <ows:Get xlink:href="http://localhost:8088/mapviewer/wmts">
          <ows:Constraint name="GetEncoding">
            <ows:AllowedValues>
              <ows:Value>RESTful</ows:Value>
            </ows:AllowedValues>
          </ows:Constraint>
        </ows:Get>
      </ows:HTTP>
    </ows:DCP>
  </ows:Operation>
  <ows:Operation name="GetFeatureInfo">
    <ows:DCP>
      <ows:HTTP>
        <ows:Get xlink:href="http://localhost:8088/mapviewer/wmts?">
          <ows:Constraint name="GetEncoding">
            <ows:AllowedValues>
              <ows:Value>KVP</ows:Value>
            </ows:AllowedValues>
          </ows:Constraint>
        </ows:Get>
      </ows:HTTP>
    </ows:DCP>
  </ows:Operation>
</ows:OperationsMetadata>
<Contents>
```

```
  <Layer>
    <ows:Title>TEST_TL</ows:Title>
    <ows:Abstract>datasource:MVDEMO,layer:TEST_TL</ows:Abstract>
    <ows:WGS84BoundingBox>
      <ows:LowerCorner>-180.0 -90.0</ows:LowerCorner>
      <ows:UpperCorner>180.0 90.0</ows:UpperCorner>
    </ows:WGS84BoundingBox>
    <ows:Identifier>MVDEMO:TEST_TL</ows:Identifier>
    <Style isDefault="true">
      <ows:Identifier>_null</ows:Identifier>
    </Style>
    <Format>image/png</Format>
    <TileMatrixSetLink>
      <TileMatrixSet>MVDEMO:TEST_TL:EPSG:4326</TileMatrixSet>
      <TileMatrixSetLimits>
        . . . . . <!-- omitted to save space  -->
        <TileMatrixLimits>
          <TileMatrix>8</TileMatrix>
          <MinTileRow>0</MinTileRow>
          <MaxTileRow>128</MaxTileRow>
          <MinTileCol>0</MinTileCol>
          <MaxTileCol>256</MaxTileCol>
        </TileMatrixLimits>
        . . . . . <!-- omitted to save space  -->
      </TileMatrixSetLimits>
    </TileMatrixSetLink>
    <ResourceURL format="image/png" template="http://localhost:8088/mapviewer/wmts/
MVDEMO:TEST_TL/{TileMatrixSet}/{TileMatrix}/{TileRow}/{TileCol}.png"
resourceType="tile"/>
  </Layer>
  <TileMatrixSet>
    <ows:Identifier>MVDEMO:TEST_TL:EPSG:4326</ows:Identifier>
    <ows:SupportedCRS>urn:ogc:def:crs:EPSG:4326</ows:SupportedCRS>
    <TileMatrix>
      <ows:Identifier>0</ows:Identifier>
      <ScaleDenominator>5.590822641167623E8</ScaleDenominator>
      <TopLeftCorner>-180.0  90.0</TopLeftCorner>
      <TileWidth>256</TileWidth>
      <TileHeight>256</TileHeight>
      <MatrixWidth>2</MatrixWidth>
      <MatrixHeight>2</MatrixHeight>
    </TileMatrix>
    . . . . . <!-- omitted to save space  -->

  </TileMatrixSet>
</Contents>
<ServiceMetadataURL xlink:href="http://localhost:8088/mapviewer/wmts?
SERVICE=WMTS&amp;VERSION=1.0.0&amp;REQUEST=getCapabilities"/>
</Capabilities>
```

D.2.2 GetTile Operation Support

When the MapViewer server receives a WMTS GetTile request, its WMTS server asks the map cache server for that requested tile. The map cache server first checks the WMTS layer's accessibility; and if the layer is accessible, it translates the requested WMTS tile's tile-row, tile-column, and other parameters into MapViewer map cache tile layer's specification. The map cache server then prepares that tile and returns it to the MapViewer's WMTS server. In other words, a MapViewer WMTS layer published in the WMTS GetCapabilities XML document can be seen as a "virtual" layer physically mapped to the MapViewer map cache tile layer, which is managed by the MapViewer map cache server.

Whenever the WMTS server receives a GetTile request, it requests the MapViewer map cache server to make that tile, and then the WMTS server returns it to the client. When the map cache server receives such requests from the WMTS server, it always generates an image according to the specifications, such as an image's size and bounding box. The map cache server's image generation process can be as easy as getting the needed tiles from cache to "assemble" the requested WMTS tile (by mosaicking and cutting); or if tiles are not available in the cache, the map cache server requests another sever (MapViewer's map server) to generate the tiles.

This topic includes GetTile requests in KVP and REST encodings, and the response.

The following is a GetTile request in KVP coding (note that any WMTS request must be on a single line):

```
http://localhost:8088/mapviewer/wmts?
SERVICE=WMTS&REQUEST=GetTile&VERSION=1.0.0&LAYER=MVDEMO:TEST_TL&STYLE=_null&format=image/
png&TileMatrixSet=MVDEMO:TEST_TL:EPSG:4326&TileMatrix=MVDEMO:TEST_TL:EPSG:4326:8&TileRow=33&TileCol=77
```

Figure D-1 shows the response tile image for the preceding request.

Figure D-1 Image Tile as Response to GetTile Request

The following is the same basic GetTile request but in REST encoding (note that any WMTS request must be on a single line):

```
http://localhost:8088/mapviewer/wmts/MVDEMO:TEST_TL/MVDEMO:TEST_TL:EPSG:4326/8/33/77.png
```

The response image is the same as shown in Figure D-1.

- Map Tiles in WMTS Layer and in Map Cache Tile Layer

D.2.2.1 Map Tiles in WMTS Layer and in Map Cache Tile Layer

The MapViewer map cache server may cache its tile layer's map tiles, but its WMTS server does not cache any WMTS layer's map tiles. In other words, the WMTS server's map tiles do not physically exist, but rather are mapped to a MapViewer map cache tile layer.

For example, if a map cache tile layer's cache has been emptied before you send the WMTS GetTile request in Map Tiles in WMTS Layer and in Map Cache Tile Layer, then the following will occur:

1. WMTS server receives the request, then it passes it to the map cache server.
2. The map cache server converts the request into an internal image map request to generate the tile image (with a spatial dimension defined by the tile's bounding box, and an image dimension defined by the image size, such as 256x256).
3. The needed map cache tile layer's tiles are identified.
4. For each map cache tile layer's tile, the map cache server tries to get it from its cache. If the tile is not available from its cache (which is the case in this scenario because you just emptied the cache), a map request is sent to another server, the MapViewer map server, to generate that tile image.
5. After all the necessary map cache tile layer's map tiles are fetched, the WMTS tile image is then rendered (by mosaicking and cutting if applicable) and returned.

If the client sends a request for the same WMTS tile afterward, the preceding steps will be repeated, and the only difference is in step 4, where the required map cache tile layer's tiles will be available from the cache. Because the most expensive processing in step 4 is avoided, your second request to the same tile should be much faster.

In general, there is no one-to-one match between a tile in the map cache tile layer and a tile in a WMTS layer specified by a GetTile request. This is because a MapViewer map cache tile layer uses a coordinate system with its origin is at the lower left corner (tile_x, tile_y), while WMTS uses a coordinate system that uses the upper left corner as its origin (tile_column, tile_row). For example, a MapViewer's image tile at (tile_x=0, tile_y=0) is the tile at the lower left corner. while the WMTS tile at (tile_column=0, tile_row=0) is located at the upper left corner.

The entire data area is partitioned into tiles starting from its corresponding origin, and the last tile in (tile_x, tile_y) or in (tile_column, tile_row) for each zoom level might not always align exactly with the data's upper bound (for the MapViewer tile coordinate system) or lower bound (for the WMTS tile coordinate system). Therefore, there might be no one-to-one match for tiles between the two systems. However, when the last tile does align exactly with the data boundary, then a one-to-one match does exist.

Using the GetTile request in Map Tiles in WMTS Layer and in Map Cache Tile Layer as an example, if you request a WMTS tile (tile_column=77, tile_row=33) at zoom level 8, that tiles matches exactly with a map cache tile layer's tile at: (tile_x=77, tile_y=94). These two tiles match because:

- The tile layer definition (see Example D-2) ensures that there is a one-to-one match for the WMTS tile and the map cache tile layer's tile because the last tile in either coordinate system aligned with the data boundary exactly.
- WMTS's tile_row 33 complements the map cache server's tile_y 94 to make the total tile rows to be 128 (note that one's origin is the upper-left corner and the other is the lower-left). As shown in Example D-3, the `<MaxTileRow>` element value is 128.

  Note also that the tile_x and tile_column values are the same, because they start from the western boundary and works their way to the right.

When creating a tile layer in MapViewer for the WMTS service, be sure that the `<tile_dpi>` element is given a value of 0.28 mm, which is equivalent in dots per inch (dpi) notation to `<tile_dpi value="90.7142857"/>`. If this element is not provided, then MapViewer's map cache tile layer may use a different value (for example, setting dpi to 96) for the tile layer when generating tiles. This different value may slow the WMTS service and degrade the quality of its map tiles, because WMTS uses a dpi value of 90.7142857.

When creating a tile layer using the WGS84 (longitude/latitude) geodetic reference system in MapViewer for the WMTS service, you need to make sure that the `<tile_meters_per_unit>` element is given this value corresponding to an arc length of 1 degree along the equator: `<tile_meters_per_unit value="111319.49079327358"/>`. If the `<tile_meters_per_unit>` element is not provided, then MapViewer internally calculates a value; however, because the 111319.49079327358 value is commonly used by other WMTS service providers for the WGS84 geodetic reference system, specifying `<tile_meters_per_unit value="111319.49079327358"/>` will ensure that MapViewer's WMTS layer matches others service provider's tile layers.

D.2.3 GetFeatureInfo Operation Support

MapViewer supports the following GetFeatureInfo operation options:

- The standard OGC GetFeatureInfo request
- A MapViewer-specific GetFeatureInfo request at point (x,y)
- A MapViewer-specific GetFeatureInfo request within a bounding box

The feature information returned by these operations is based on the MapViewer base map and theme definitions, and on features selected according to the area of interest. The base map may define the visible scale ranges for each theme; and each theme defines its information columns in the `<field>` child element of `<hidden_info>`, which is a child element in the `<styling_rules>` element. If no information columns are specified in a theme's `<styling_rules>` element, then GetFeatureInfo response will not include any information about the features of that theme.

The area of interest is calculated in a GetFeatureInfo process for each option as follows:

- For a standard OGC GetFeatureInfo request, the pixel's column and row and its tile's location at a given zoom level can be translated into a (x,y) point in the map cache tile layer's coordinate system at the pixel's centroid. Centered at that point, a 5 by 5 pixel window at that zoom level defines the area of interest.
- For a MapViewer-specific (x, y) point request, it is also uses a 5 by 5 pixel window at that level centered at the given (x,y) value as the area of interest.
- For a MapViewer-specific bounding box GetFeatureInfo request option, it uses the specified bounding box (min_x, min_y, max_x, max_y) and their corresponding (column, row) values in the image tile at the zoom level as the area of interest.

MapViewer lets you provide an additional MapViewer-specific parameter in a GetFeatureInfo request: a `childLayer` attribute (for example, `childLayer=myTheme1, myTheme2`). When this parameter is provided, MapViewer only includes the features information in the specified themes (in this example, only myTheme1 and myTheme2). If this parameter is not provided, then all themes in the base map definition are considered for a GetFeatureInfo request.

This topic includes the following subtopics:

These subtopics have examples that use the base map and theme definitions in Example D-5 and Example D-6 for a tile layer named TEST_TL.

Example D-5 Base Map Definition for Tile Layer TEST_TL

```
<?xml version="1.0" standalone="yes"?>
  <map_definition>
    <theme name="THEME_DEMO_STATES" min_scale="" max_scale="0.0" scale_mode="RATIO"/>
    <theme name="THEME_DEMO_COUNTIES" min_scale="8500000.0" max_scale="0.0" scale_mode="RATIO"/>
    <theme name="THEME_DEMO_HIGHWAYS_LINE" min_scale="1.0E8" max_scale="4.5E7" scale_mode="RATIO"/>
    <theme name="THEME_DEMO_HIGHWAYS" min_scale="4.5E7" max_scale="0.0" scale_mode="RATIO"/>
    <theme name="THEME_DEMO_BIGCITIES" min_scale="" max_scale="0.0" scale_mode="RATIO"/>
    <theme name="THEME_DEMO_CITIES" min_scale="7500000.0" max_scale="0.0" scale_mode="RATIO"/>
  </map_definition>
```

Example D-6 Theme Names and Styling Rules for Tile Layer TEST_TL

```
THEME_DEMO_STATES
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="STATE" name="Name"/>
    <field column="STATE_ABRV" name="Abrv."/>
    <field column="TOTPOP" name="Population"/>
  </hidden_info>
  <rule>
    <features style="C.S02_COUNTRY_AREA"> </features>
    <label column="STATE_ABRV" style="T.S02_STATE_ABBREVS"> 1 </label>
  </rule>
</styling_rules>

THEME_DEMO_COUNTIES
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="COUNTY" name="County"/>
    <field column="FIPSSTCO" name="Fips"/>
    <field column="TOTPOP" name="Population"/>
    <field column="STATE_ABRV" name="State"/>
  </hidden_info>
  <rule>
    <features style="L.S06_BORDER_STATE"> </features>
  </rule>
</styling_rules>

THEME_DEMO_CITIES
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
    <field column="CITY" name="City"/>
    <field column="POP90" name="Population"/>
  </hidden_info>
  <rule>
    <features style="M.ALL_CITY_L2"> (pop90 between 200000 AND 1000000 ) </featu
res>
    <label column="city" style="T.S07_CITIES_L2"> 1 </label>
  </rule>
  <rule>
    <features style="M.ALL_CITY_L3"> (pop90 &lt; 200000) </features>
    <label column="city" style="T.S07_CITIES_L3"> 1 </label>
  </rule>
</styling_rules>


THEME_DEMO_HIGHWAYS
<?xml version="1.0" standalone="yes"?>
```

```
<styling_rules>
  <hidden_info>
    <field column="HIGHWAY" name="Name"/>
    <field column="ROUTEN" name="No."/>
  </hidden_info>
  <rule>
    <features style="L.S04_ROAD_INTERSTATE"> </features>
    <label column="routen" style="M.HWY_USA_INTERSTATE_NARROW"> (3-length(routen
)) </label>
  </rule>
</styling_rules>
```

- OGC GetFeatureInfo Request
- MapViewer GetFeatureInfo Request at an (x,y) Point
- MapViewer GetFeatureInfo Request within a Bounding Box

D.2.3.1 OGC GetFeatureInfo Request

The following is a sample GetFeatureInfo request using the OGC standard (note that any WMTS request must be on a single line):

```
http://localhost:8088/mapviewer/wmts?
SERVICE=WMTS&REQUEST=GetFeatureInfo&VERSION=1.0.0&LAYER=MVDEMO:TEST_TL&STYLE=_null&format
=image/
png&TileMatrixSet=MVDEMO:TEST_TL:EPSG:4326&TileMatrix=MVDEMO:TEST_TL:EPSG:4326:8&TileRow=
33&TileCol=76&I=238&J=240&InfoFormat=application/gml%2bxml; version=3.1
```

Example D-7 shows the response from this request.

Example D-7 Response from OGC GetFeatureInfo Request

```
<?xml version="1.0" encoding="UTF-8"?>
<layers>
  <layer name="THEME_DEMO_CITIES">
    <feature id="1">
      <City>Worcester</City>
      <Population>169759</Population>
    </feature>
  </layer>
  <layer name="THEME_DEMO_BIGCITIES">
  </layer>
  <layer name="THEME_DEMO_HIGHWAYS">
  </layer>
  <layer name="THEME_DEMO_HIGHWAYS_LINE">
  </layer>
  <layer name="THEME_DEMO_COUNTIES">
    <feature id="1">
      <County>Worcester</County>
      <Fips>25027</Fips>
      <Population>709705</Population>
      <State>MA</State>
    </feature>
  </layer>
  <layer name="THEME_DEMO_STATES">
    <feature id="1">
      <State>Massachusetts</State>
      <Abrv.>MA</Abrv.>
      <Population>6016424</Population>
    </feature>
  </layer>
</layers>
```

D.2.3.2 MapViewer GetFeatureInfo Request at an (x,y) Point

The following is a sample MapViewer GetFeatureInfo request for information for all themes in text/html format at a point identified by its longitude and latitude (x and y) values, in this case a point in the city of Boston (note that any WMTS request must be on a single line):

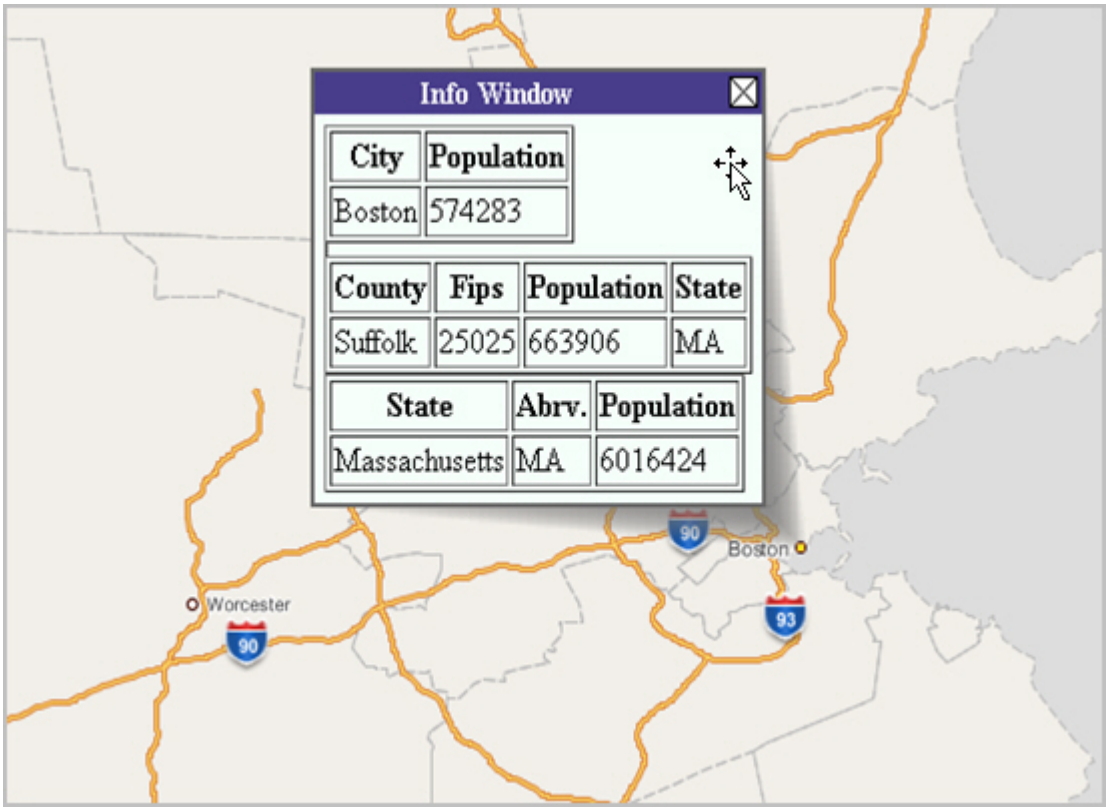
```
http://localhost:8088/mapviewer/wmts?
SERVICE=WMTS&REQUEST=GetFeatureInfo&VERSION=1.0.0&infoformat=text/
html&LAYER=MVDEMO:TEST_TL&zoomlevel=8&x=-71.01522133879438&y=42.33731136863662&SRID=8307
```

In this request, the point's (x,y) coordinates are specified as the parameters. MapViewer translates the point into a 5 by 5 pixel rectangular area at the given zoom level (8 in this example), centered at that point, as an area of interest for getting the feature information. Because there is no `childLayer` parameter in the request, by default all features in the area of interest are considered.

If the (x, y) coordinates are in a different coordinate system (spatial reference system) than that used by the tile layer, then the optional SRID parameter must be specified for converting the coordinates into the tile layer's system. Its default value is the tile layer's SRID value.

Figure D-2 shows the response from this request.

Figure D-2 Response from MapViewer GetFeatureInfo Request at an (x,y) Point



D.2.3.3 MapViewer GetFeatureInfo Request within a Bounding Box

The following is a sample MapViewer GetFeatureInfo request for information for two themes (THEME_DEMO_CITIES and THEME_DEMO_COUNTIES) within an area of interest defined

by a bounding box, in this case a rectangular area including the city of Boston and some surrounding communities (note that any WMTS request must be on a single line):

```
http://localhost:8088/mapviewer/wmts?
SERVICE=WMTS&REQUEST=GetFeatureInfo&VERSION=1.0.0&infoformat=text/
xml&LAYER=MVDEMO:TEST_TL&childLayer=THEME_DEMO_COUNTIES,THEME_DEMO_CITIES&zoomlevel=8&bbo
x=-71.32151786220001,42.19361670333521,-70.96354167846667,42.52386233762441&SRID=8307
```

If the (x, y) coordinates are in a different coordinate system (spatial reference system) than that used by the tile layer, then the optional SRID parameter must be specified for converting the bounding box coordinates into the tile layer's system. Its default value is the tile layer's SRID value.

Figure D-2 shows the bounding box for this request.

Figure D-3 Bounding Box for MapViewer GetFeatureInfo Request

Example D-8 shows the response in XML format from this request.

Example D-8 Response from MapViewer GetFeatureInfo Request within a Bounding Box

```
<layers>
   <layer name="THEME_DEMO_CITIES">
      <feature id="1">
      <City>Boston</City>
      <Population>574283</Population>
      </feature>
   </layer>
   <layer name="THEME_DEMO_COUNTIES">
      <feature id="1">
         <County>Middlesex</County>
         <Fips>25017</Fips>
         <Population>1398468</Population>
         <State>MA</State>
      </feature>
      <feature id="2">
         <County>Norfolk</County>
         <Fips>25021</Fips>
```

```
            <Population>616087</Population>
            <State>MA</State>
         </feature>
         <feature id="3">
            <County>Suffolk</County>
            <Fips>25025</Fips>
            <Population>663906</Population>
            <State>MA</State>
         </feature>
         <feature id="4">
            <County>Essex</County>
            <Fips>25009</Fips>
            <Population>670080</Population>
            <State>MA</State>
         </feature>
      </layer>
</layers>
```

D.3 Preparing the WMTS Service for MapViewer

To prepare for using the WMTS Service for MapViewer, follow these major steps:

- Prepare Predefined Geometry Themes
- Prepare the Base Map
- Prepare Tile Layers
- Publish Tile Layers in the wmtsConfig.xml Policy File
- Verify the MapViewer WMTS Service

D.3.1 Prepare Predefined Geometry Themes

If the getFeatureInfo service must get feature information from a predefined geometry theme , you must use the `<hidden_info>` element with one or more `<field>` child elements. The `column` attribute in the `<field>` element will be returned in a getFeatureInfo response.

The `<hidden_info>` element must be in the `<styling_rules>` element. For example:

```
<?xml version="1.0" standalone="yes"?>
<styling_rules>
  <hidden_info>
        <field column="NAME" name="name"/>
        <field column="MAINPAGE" name="mainpage"/>
  </hidden_info>
  .
  .
  .
</styling_rules>
```

For information using styling rules in a predefined geometry theme, see Styling Rules in Predefined Spatial Geometry Themes

D.3.2 Prepare the Base Map

To prepare a base map (which is just a collection of themes) to use a WMTS theme, add the theme to the base map.

For information about adding themes to a base map, see Maps.

For information about WMTS themes, see WMTS Themes.

D.3.3 Prepare Tile Layers

A MapViewer tile layer becomes a WMTS layer in the MapViewer WMTS server if there are no constraints in the `wmtsConfix.xml` policy file. For example, assume that a base map named DEMO_MAP has been created as in Example D-2, then tile layers can be created using that base map. This topic contains the following examples, each using a well-known scale set found in OGC WMTS 1.0.0 implementation standard Annex E:

- Example D-9
- Example D-10
- Example D-11
- Example D-12

In these examples, the `<tile_dpi>` and `<tile_meters_per_unit>` elements are specified, to comply with the OGC WMTS 1.0.0 implementation standard where pixel size is 0.28mm. If the coordinate (spatial reference) system is WGS84, then 1 degree on the equator equals 111319.49 meters.

Example D-9 Tile Definition of Scale Set GlobalCRS84Scale

```
insert into user_sdo_cached_maps values(
'WMTS_E1',
'wmts annex e1 scale set for GlobalCRS84Scale',
'',
'YES',
'YES',
'<map_tile_layer name="WMTS_E1" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3">
   <internal_map_source base_map="DEMO_MAP" data_source="MVDEMO" />
   <coordinate_system srid="8307" minX="-180" maxX="180" minY="-90" maxY="90"/>
   <internal_map_source base_map="DEMO_MAP" data_source="MVDEMO" db_tile_table=""/>
   <cache_storage root_path=""/>
   <coordinate_system srid="8307" minX="-180" maxX="180" minY="-90" maxY="90"/>
   <tile_image width="256" height="256"/>
   <tile_dpi value="90.7142857"/>
   <tile_meters_per_unit value="111319.49079327358"/>
   <zoom_levels levels="21" min_scale="100" max_scale="5.0E8">
      <zoom_level scale="5.0E8" tile_width="321.9561978791382" tile_height="321.9561978791382"/>
      <zoom_level scale="2.5E8" tile_width="160.9780989395691" tile_height="160.9780989395691"/>
      <zoom_level scale="1.0E8" tile_width="64.39123957582764" tile_height="64.39123957582764"/>
      <zoom_level scale="5.0E7" tile_width="32.19561978791382" tile_height="32.19561978791382"/>
      <zoom_level scale="2.5E7" tile_width="16.09780989395691" tile_height="16.09780989395691"/>
      <zoom_level scale="1.0E7" tile_width="6.439123957582764" tile_height="6.439123957582764"/>
      <zoom_level scale="5.0E6" tile_width="3.219561978791382" tile_height="3.219561978791382"/>
      <zoom_level scale="2.5E6" tile_width="1.609780989395691" tile_height="1.609780989395691"/>
      <zoom_level scale="1.0E6" tile_width="0.6439123957582763" tile_height="0.6439123957582763"/>
      <zoom_level scale="5.0E5" tile_width="0.32195619787913815" tile_height="0.32195619787913815"/>
      <zoom_level scale="2.5E5" tile_width="0.16097809893956908" tile_height="0.16097809893956908"/>
      <zoom_level scale="1.0E5" tile_width="0.06439123957582764" tile_height="0.06439123957582764"/>
      <zoom_level scale="5.0E4" tile_width="0.03219561978791382" tile_height="0.03219561978791382"/>
      <zoom_level scale="2.5E4" tile_width="0.01609780989395691" tile_height="0.01609780989395691"/>
      <zoom_level scale="1.0E4" tile_width="0.006439123957582764" tile_height="0.006439123957582764"/>
      <zoom_level scale="5.0E3" tile_width="0.003219561978791382" tile_height="0.003219561978791382"/>
      <zoom_level scale="2.5E3" tile_width="0.001609780989395691" tile_height="0.001609780989395691"/>
      <zoom_level scale="1.0E3" tile_width="6.439123957582763E-4" tile_height="6.439123957582763E-4"/>
      <zoom_level scale="5.0E2" tile_width="3.2195619787913813E-4" tile_height="3.2195619787913813E-4"/>
      <zoom_level scale="2.5E2" tile_width="1.6097809893956906E-4" tile_height="1.6097809893956906E-4"/>
```

```
      <zoom_level scale="1.0E2" tile_width="6.439123957582763E-5" tile_height="6.439123957582763E-5"/>
   </zoom_levels>
</map_tile_layer>',
'DEMO_MAP','');
```

Example D-10 Tile Definition of Scale Set GlobalCRS84Pixel

```
insert into user_sdo_cached_maps values(
'WMTS_E2',
'wmts annex e2 scale GlobalCRS84Pixel',
'',
'YES',
'YES',
'<map_tile_layer name="WMTS_E2" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3">
   <internal_map_source base_map="DEMO_MAP" data_source="MVDEMO"/>
   <coordinate_system srid="8307" minX="-180" maxX="180" minY="-90" maxY="90"/>
   <tile_image width="256" height="256"/>
   <tile_dpi value="90.7142857"/>
   <tile_meters_per_unit value="111319.49079327358"/>
   <zoom_levels levels="19" min_scale="1104.360027711047" max_scale="795139219.9519541">
      <zoom_level scale="795139219.9519541" tile_width="512.000000" tile_height="512.000000"/>
      <zoom_level scale="397569609.9759771" tile_width="256.000000" tile_height="256.000000"/>
      <zoom_level scale="198784804.9879885" tile_width="128.000000" tile_height="128.000000"/>
      <zoom_level scale="132523203.3253257" tile_width="85.333333" tile_height="85.333333"/>
      <zoom_level scale="66261601.66266284" tile_width="42.666666" tile_height="42.666666"/>
      <zoom_level scale="33130800.83133142" tile_width="21.333333" tile_height="21.333333"/>
      <zoom_level scale="13252320.33253257" tile_width="8.5333333" tile_height="8.5333333"/>
      <zoom_level scale="6626160.166266284" tile_width="4.2666666" tile_height="4.2666666"/>
      <zoom_level scale="3313080.083133142" tile_width="2.1333333" tile_height="2.1333333"/>
      <zoom_level scale="1656540.041566571" tile_width="1.0666666" tile_height="1.0666666"/>
      <zoom_level scale="552180.0138555236" tile_width="0.3555555" tile_height="0.3555555"/>
      <zoom_level scale="331308.0083133142" tile_width="0.2133333" tile_height="0.2133333"/>
      <zoom_level scale="110436.0027711047" tile_width="0.0711111" tile_height="0.0711111"/>
      <zoom_level scale="55218.00138555237" tile_width="0.0355555" tile_height="0.0355555"/>
      <zoom_level scale="33130.80083133142" tile_width="0.0213333" tile_height="0.0213333"/>
      <zoom_level scale="11043.60027711047" tile_width="0.00711111" tile_height="0.00711111"/>
      <zoom_level scale="3313.080083133142" tile_width="0.00213333" tile_height="0.00213333"/>
      <zoom_level scale="1104.360027711047" tile_width="7.111111E-4" tile_height="7.111111E-4"/>
   </zoom_levels>
</map_tile_layer>',
'DEMO_MAP',
'');
```

Example D-11 Tile Definition of Scale Set GoogleCRS84Quad

```
insert into user_sdo_cached_maps values(
'WMTS_E3',
'wmts annex e3 scale set GoogleCRS84Quad',
'',
'YES',
'YES',
'<map_tile_layer name="WMTS_E3" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3">
   <internal_map_source base_map="DEMO_MAP" data_source="MVDEMO"/>
   <coordinate_system srid="8307" minX="-180" maxX="180" minY="-90" maxY="90"/>
   <tile_image width="256" height="256"/>
   <tile_dpi value="90.7142857"/>
   <tile_meters_per_unit value="111319.49079327358"/>
   <zoom_levels levels="19" min_scale="2132.729583849784" max_scale="559082264.0287178">
      <zoom_level tile_width="360.0000" tile_height="360.0000" scale="5.590822640287178E8"/>
      <zoom_level tile_width="180.0000" tile_height="180.0000" scale="2.795411320143589E8"/>
      <zoom_level tile_width="90.0000" tile_height="90.00000" scale="1.3977056600717944E8"/>
```

```
      <zoom_level tile_width="45.0000" tile_height="45.00000" scale="6.988528300358972E7"/>
      <zoom_level tile_width="22.5000" tile_height="22.50000" scale="3.494264150179486E7"/>
      <zoom_level tile_width="11.2500" tile_height="11.25000" scale="1.747132075089743E7"/>
      <zoom_level tile_width="5.6250" tile_height="5.62500" scale="8735660.375448715"/>
      <zoom_level tile_width="2.8125" tile_height="2.81250" scale="4367830.1877243575"/>
      <zoom_level tile_width="1.40625" tile_height="1.40625" scale="2183915.0938621787"/>
      <zoom_level tile_width="0.703125" tile_height="0.703125" scale="1091957.5469310894"/>
      <zoom_level tile_width="0.3515625" tile_height="0.3515625" scale="545978.7734655447"/>
      <zoom_level tile_width="0.17578125" tile_height="0.17578125" scale="272989.38673277234"/>
      <zoom_level tile_width="0.087890625" tile_height="0.087890625" scale="136494.693366386"/>
      <zoom_level tile_width="0.0439453125" tile_height="0.0439453125" scale="68247.34668319"/>
      <zoom_level tile_width="0.02197265625" tile_height="0.02197265625" scale="34123.6733415"/>
      <zoom_level tile_width="0.010986328126" tile_height="0.010986328126" scale="17061.8366707"/>
      <zoom_level tile_width="0.0054931640633" tile_height="0.0054931640633" scale="8530.91833539"/>
      <zoom_level tile_width="0.00274658203168" tile_height="0.00274658203168" scale="4265.45916769"/>
      <zoom_level tile_width="0.001373291015841" tile_height="0.001373291015841" scale="2132.72958384"/>
   </zoom_levels>
</map_tile_layer>',
'DEMO_MAP',
'');
```

Example D-12 Tile Definition of Scale Set GoogleMapsCompatible

```
insert into user_sdo_cached_maps values(
'WMTS_E4',
'wmts annex e4 scale set',
'',
'YES',
'YES',
'<map_tile_layer name="WMTS_E4" image_format="PNG" http_header_expires="168.0"
concurrent_fetching_threads="3">
   <internal_map_source base_map="DEMO_MAP" data_source="MVDEMO"/>
     <coordinate_system srid="3857" minX="-20037508" maxX="20037508" minY="-20037508" maxY="20037508"/>
       <tile_image width="256" height="256"/>
       <tile_dpi value="90.7142857"/>
       <tile_meters_per_unit value="1"/>
       <zoom_levels levels="19" min_scale="2132.729583849784" max_scale=" 559082264.0287178">
         <zoom_level tile_width="4.0075016692E7" tile_height="4.0075016692E7" scale="5.5908226403E8"/>
         <zoom_level tile_width="2.0037508346E7" tile_height="2.0037508346E7" scale="2.7954113201E8"/>
         <zoom_level tile_width="1.0018754173E7" tile_height="1.0018754173E7" scale="1.3977056601E8"/>
         <zoom_level tile_width="5009377.086486" tile_height="5009377.086486" scale="6.9885283004E7"/>
         <zoom_level tile_width="2504688.543243" tile_height="2504688.543243" scale="3.4942641502E7"/>
         <zoom_level tile_width="1252344.27162" tile_height="1252344.27162" scale="1.747132075089E7"/>
         <zoom_level tile_width="626172.135810" tile_height="626172.135810" scale="8735660.3754487"/>
         <zoom_level tile_width="313086.067905" tile_height="313086.067905" scale="4367830.1877243"/>
         <zoom_level tile_width="156543.033952" tile_height="156543.033952" scale="2183915.0938621"/>
         <zoom_level tile_width="78271.5169763" tile_height="78271.5169763" scale="1091957.5469311"/>
         <zoom_level tile_width="39135.7584882" tile_height="39135.7584882" scale="545978.77346554"/>
         <zoom_level tile_width="19567.8792441" tile_height="19567.8792441" scale="272989.38673277"/>
         <zoom_level tile_width="9783.93962204" tile_height="9783.93962204" scale="136494.69336638"/>
         <zoom_level tile_width="4891.96981102" tile_height="4891.96981102" scale="68247.346683193"/>
         <zoom_level tile_width="2445.98490551" tile_height="2445.98490551" scale="34123.673341596"/>
         <zoom_level tile_width="1222.99245275" tile_height="1222.99245275" scale="17061.836670798"/>
         <zoom_level tile_width="611.496226378" tile_height="611.496226378" scale="8530.9183353991"/>
         <zoom_level tile_width="305.748113189" tile_height="305.748113189" scale="4265.4591676995"/>
         <zoom_level tile_width="152.874056594" tile_height="152.874056594" scale="2132.7295838498"/>
       </zoom_levels>
</map_tile_layer>',
'DEMO_MAP',
'');
```

D.3.4 Publish Tile Layers in the wmtsConfig.xml Policy File

To publish tile layers, you can edit and add information in the `<public_datasources>` element in the `wmtsConfig.xml` policy file, which is stored in the same location as `mapViewerConfig.xml`. Example D-13 shows the entries for publishing five tiles in data source MVDEMO (TEST_EL, WMTS_E1, WMTS_E2, WMTS_E3, and WMTS_E4) and all tile layers in data source TILSMENV.

Example D-13 Publishing Tile Layers

```
<public_datasources>
  <public_datasource name="MVDEMO">
    <tile_layers>
      <tile_layer name="TEST_TL"/>
      <tile_layer name="WMTS_E1"/>
      <tile_layer name="WMTS_E2"/>
      <tile_layer name="WMTS_E3"/>
      <tile_layer name="WMTS_E4"/>
    </tile_layers>
  </public_datasource>
  <public_datasource name="TILSMENV" include_all_tile_layers="TRUE">
</public_datasources>
```

D.3.5 Verify the MapViewer WMTS Service

To verify that the MapViewer WMTS service is working properly, follow these steps:

1. Restart the MapViewer server to let MapViewer retrieve the tile layer definitions from the data source and apply the WMTS policy file.
2. Issue a GetCapabilities request in the following format, to check that the tile layers are published and that all scale sets are shown as expected:

   `http://<url>:<port>/mapviewer/wmts?REQUEST=GetCapabilities&SERVICE=WMTS&VERSION=1.0.0`

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y