ORACLE
FUSION MIDDLEWARE

An Oracle White Paper
October 2013

Oracle Fusion Middleware MapViewer Primer

ORACLE

Introduction

This whitepaper provides an introduction to Oracle Fusion Middleware MapViewer. MapViewer is robust feature for creating maps from spatial and business data stored in Oracle Database. MapViewer works with Locator and Oracle Spatial and Graph to deliver a rich online mapping facility.

It is a component of Oracle Fusion Middleware and is a mid-tier toolkit that runs as a service inside an instance of Oracle Weblogic or other JEE containers. MapViewer is included with with all editions of Oracle Weblogic Server. MapViewer version 11g is certified on Oracle WebLogic Server version 11g and higher.

What Is MapViewer

Initially, there were desktop GIS applications that created maps from geographical data stored in various file formats. Then in the mid 1990's Oracle introduced Oracle Spatial, which managed geospatial data in the database in regular tables. This brought geospatial data management to a whole new level, and made it a part of enterprise IT infrastructure. Appendix A provides a brief primer for those not familiar with Locator and Oracle Spatial and Graph.

Oracle Fusion Middleware MapViewer is a developer's toolkit; that is, a set of programmable Java components for rendering maps from geospatial application data that is stored and managed in Oracle Database. It transparently queries geospatial data from Locator and Oracle Spatial and Graph, and renders cartographical query results. It provides customizable options for more advanced users. MapViewer deploys on an Internet and Java standards-based platform and integrates maps and spatial analysis into web and standalone applications.

MapViewer includes three sets of APIs. It has an XML request/response protocol for embedding static maps in web pages and a Java API for embedding maps in an application. Both APIs support interactive querying of maps for features within and near an area of interest, and appropriate rendering of features depending on the level of magnification. The JavaScript API in Oracle Maps, a component of MapViewer, takes this interactivity to the next level by adding "slippy" capabilities; that is, the ability to pan, zoom and slide the map image in any direction.

What is Oracle Maps

Oracle Maps is a component of MapViewer 10.1.3.1 or later. It mashes up database content with maps and supports "slippy map" seamless pan and zoom interactions on geospatial data served by Oracle Locator and Oracle Spatial and Graph. It fetches, converts, and transfers geospatial database content to an online mapping service for display.

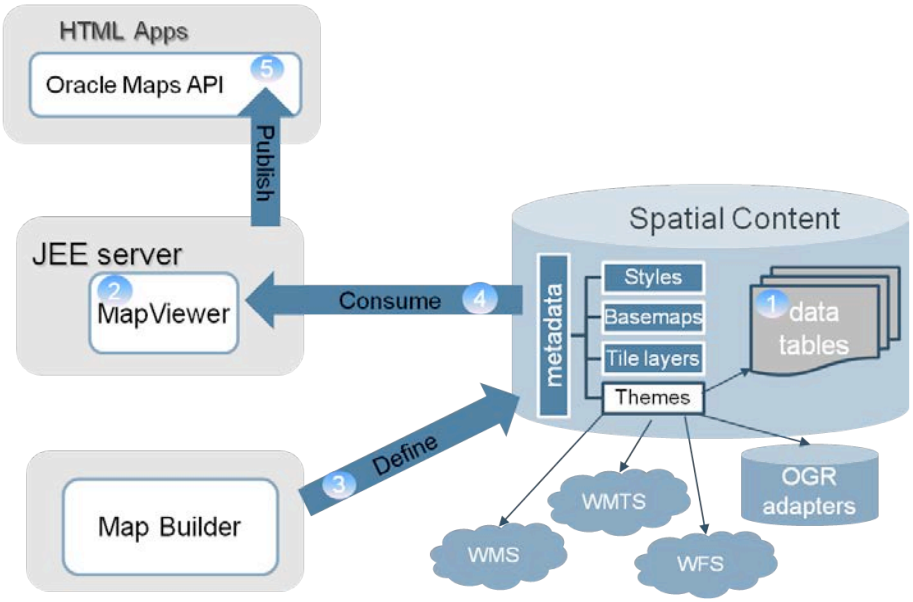
Oracle Maps provides an interactive client side JavaScript API, a server side file system-based image cache for map tiles, and an ad-hoc query module called the feature of interest (FoI) server. The Oracle Maps JavaScript API is also encapsulated in the JDeveloper ADF Faces GeoMap data visualization component. Starting with MapViewer 11.1.1.7 there is a second version of the JavaScript API (aka V2 or HTML5 API) which is based on HTML5 Canvas and/or SVG.

Oracle Maps assumes a general familiarity with MapViewer and an understanding of core concepts, including styles, themes and base maps. It offers many unique advantages. Please refer to the companion Oracle Maps Primer for a more detailed description.

Setting up your MapViewer Environment

This section describes the process to publish your geospatial application data as online maps using MapViewer. The five basic required steps, as illustrated in Figure 1, are:

Figure 1. Steps for setting up your Spatial and Graph/MapViewer environment



1. Load or create geospatial application data in a column of an Oracle Database table.
2. Deploy and configure MapViewer in Oracle Weblogic Server (or Glassfish).
3. Style the geospatial data and create all the necessary mapping metadata for MapViewer using Map Builder, the desktop authoring utility that comes with MapViewer.

4. Define a data source so MapViewer can connect to the database schema containing the geospatial application data and mapping metadata.
5. View your maps by submitting individual map requests, or write a web mapping application that interacts with MapViewer using one of its APIs.

The remainder of this white paper describes each step in further detail.

Loading data into Oracle Spatial and Graph

The data source for MapViewer is geospatial data managed by Oracle Locator or Oracle Spatial and Graph in Oracle Database. There are several options for loading geospatial application data into Oracle Database. You can use the Map Builder utility to import shapefiles supported by ESRI and other vendors into Oracle Database tables. You can also use 3rd party tools, such as FME from Safe Software, or tools from GIS software vendors to load data from any proprietary GIS data format into an Oracle Database. You can also create tables and use SQL to populate them with new records that contain geocoded business data. Once your data is loaded into Locator or Oracle Spatial and Graph, it becomes the single source of truth for MapViewer, as should be the case for all your geospatial applications.

The MapViewer page on the Oracle Technology Network has a simple geospatial data set formatted as an Oracle Database export file for demo purposes. It can be downloaded here:

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

This data set contains simplified data for states, counties, major cities and highways of the United States. It is called the "MVDEMO Sample Data Set". All of the MapViewer built-in tutorials and demos run off this data set so it is highly recommended to import this data set into your Oracle Database, under a new database user MVDEMO (or any other newly created user schema). The detailed instructions are packaged with the demo data set.

Installing MapViewer

While you can deploy MapViewer to Oracle Weblogic Server (release 10.1.3 or later is required), the easiest way to get a MapViewer instance up and running is to use the **MapViewer Quick Start kit**. You can download the kit from the MapViewer OTN page here:

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

The Quick Start kit contains a standalone Glassfish server with MapViewer pre-deployed. Glassfish serves as a web server like Apache, and a JEE container to run your JSPs and Servlets, all rolled into a small-footprint Java bundle. MapViewer is a set of servlets that are initialized when the Glassfish instance is started.

While Oracle WebLogic Server also supports MapViewer, Glassfish is sufficient and simple to manage for all of your MapViewer prototyping needs.

Download the Quick Start kit and unzip it to a directory on your computer. Verify your computer has the Java JDK 1.6 (or later) installed. If you don't have JDK 1.6, go to Oracle's Java SE Downloads page and download it.

http://www.oracle.com/technetwork/java/javase/downloads/

The latest MapViewer (versions 11.1.1.7 and later) requires JDK version 1.6 or later. The readme.txt file, in the Quick Start Kit, contains installation and configuration instructions.

MapViewer is started when you start Glassfish. To start it simply modify and execute the batch file (startServer.bat or startServer.sh) in the Quick Start kit.

Once the server has started, i.e. you see the message domain started successfully, access the MapViewer home page by opening the URL http://localhost:8080/mapviewer in your browser. If the page loads then MapViewer is up and running.

Creating MapViewer metadata

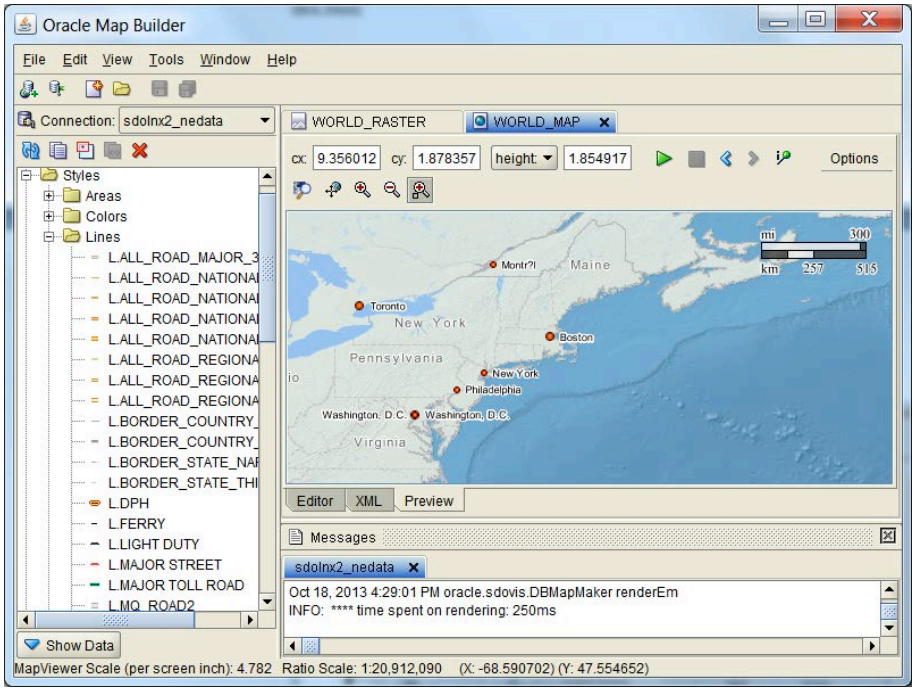
Geospatial application data tables contain geometries but not visualization information. MapViewer relies on mapping metadata to generate quality maps. MapViewer metadata provides the "skins", styling rules, designs or templates that tell MapViewer how to render geospatial application data into different maps for different scales and/or scenarios. The metadata identifies the geospatial application data tables and dictates how to style or render the data in a map with the desired look and feel. The metadata typically include map content and symbology definition, i.e. **style**, **theme, base map,** and **tile cache** definitions.

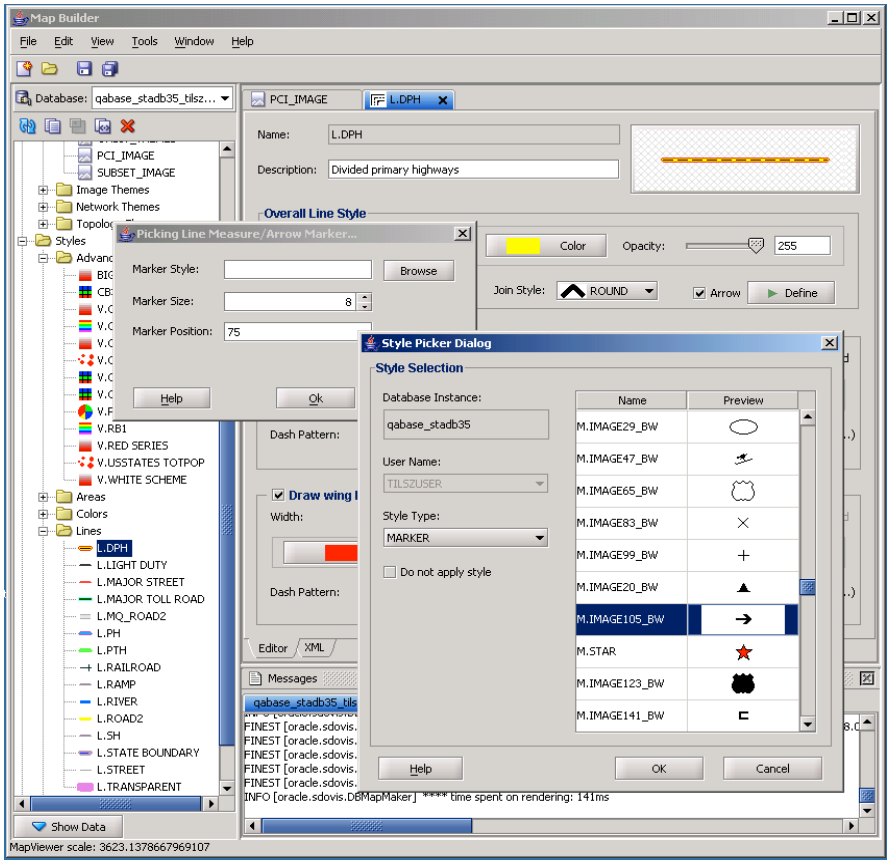
MapBuilder, the desktop authoring utility that comes with MapViewer, creates and maintains all persisted metadata. Mapping metadata is stored in Oracle Database, usually in the same schema as the geospatial application data. Certain types of metadata such as styles and themes can also be programmatically created on the fly and used as part of a map request. This metadata is temporary and will not be stored persistently. MapBuilder supports previewing geospatial data with the chosen style, theme and base map as the metadata is being created, called "styling". MapBuilder does not rely on MapViewer. It connects directly to your database schema(s), and includes the same rendering engine as MapViewer for instant and authentic preview.

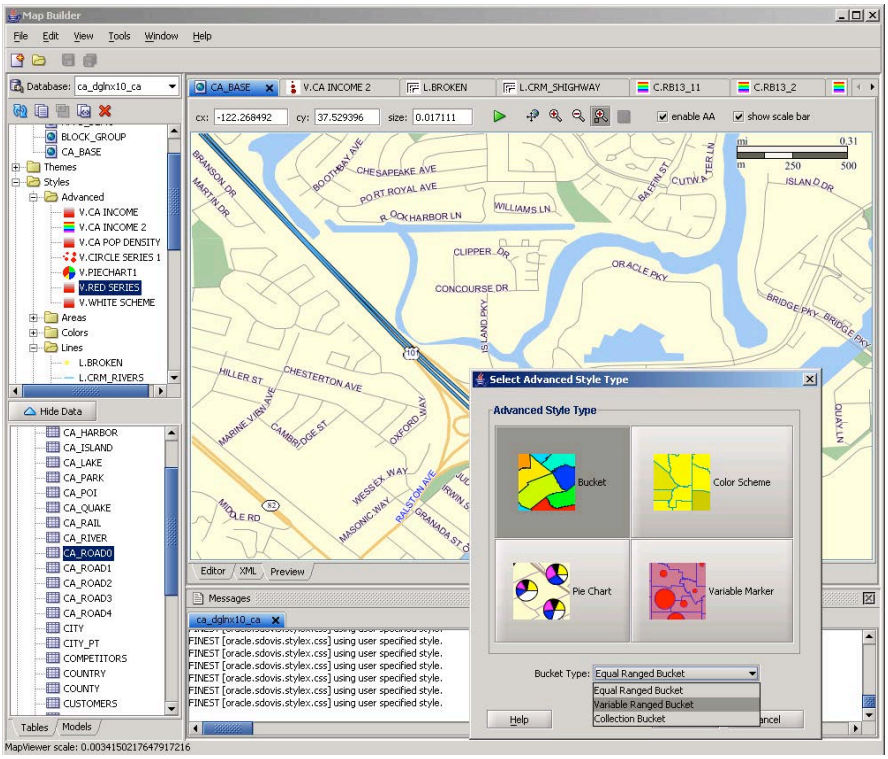
See the **MapViewer metadata** section below for more information.

The following screenshots illustrate Map Builder in action. For more information about Map Builder, consult the MapViewer User's Guide available at,

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


Oracle Map Builder
File Edit View Tools Window Help
Connection: sdolnx2_nedata
WORLD_RASTER
WORLD_MAP
cx: 9.356012 cy: 1.878357 height: 1.854917
Options
Styles
Areas
Colors
Lines
L.ALL_ROAD_MAJOR_3
L.ALL_ROAD_NATIONAL
L.ALL_ROAD_NATIONAL
L.ALL_ROAD_NATIONAL
L.ALL_ROAD_NATIONAL
L.ALL_ROAD_REGIONA
L.ALL_ROAD_REGIONA
L.ALL_ROAD_REGIONA
L.BORDER_COUNTRY_
L.BORDER_COUNTRY_
L.BORDER_STATE_NAI
L.BORDER_STATE_THI
L.DPH
L.FERRY
L.LIGHT DUTY
L.MAJOR STREET
L.MAJOR TOLL ROAD
L.MQ_ROAD2
Show Data
Montr?l
Maine
Toronto
New York
Boston
Pennsylvania
New York
Philadelphia
Washington, D.C.
Washington, D.C.
Virginia
mi 300
km 257 515
Editor XML Preview
Messages
sdolnx2_nedata
Oct 18, 2013 4:29:01 PM oracle.sdovis.DBMapMaker renderEm
INFO: **** time spent on rendering: 250ms
MapViewer Scale (per screen inch): 4.782 Ratio Scale: 1:20,912,090 (X: -68.590702) (Y: 47.554652)


Map Builder
File Edit View Tools Window Help
Database: qabase_stadb35_tilsz...
PCI_IMAGE
L.DPH
Name: L.DPH
Description: Divided primary highways
Overall Line Style
Color
Opacity: 255
Join Style: ROUND
Arrow
Define
Picking Line Measure/Arrow Marker...
Marker Style:
Browse
Marker Size: 8
Marker Position: 75
Help
Ok
Style Picker Dialog
Style Selection
Database Instance: qabase_stadb35
User Name: TILSZUSER
Style Type: MARKER
Do not apply style
Name
Preview
M.IMAGE29_BW
M.IMAGE47_BW
M.IMAGE65_BW
M.IMAGE83_BW
M.IMAGE99_BW
M.IMAGE20_BW
M.IMAGE105_BW
M.STAR
M.IMAGE123_BW
M.IMAGE141_BW
Help
OK
Cancel
Dash Pattern:
Draw wing l
Width:
Editor XML
Messages
FINEST [oracle.sdovis.
INFO [oracle.sdovis.DBMapMaker] **** time spent on rendering: 141ms
Image Themes
Network Themes
Styles
Areas
Colors
Lines
L.DPH
L.LIGHT DUTY
L.MAJOR STREET
L.MAJOR TOLL ROAD
L.MQ_ROAD2
L.PH
L.PTH
L.RAILROAD
L.RAMP
L.RIVER
L.ROAD2
L.SH
L.STATE BOUNDARY
L.STREET
L.TRANSPARENT
V.RB1
V.RED SERIES
V.USSTATES TOTPOP
V.WHITE SCHEME
Show Data
MapViewer scale: 3623.137866796910



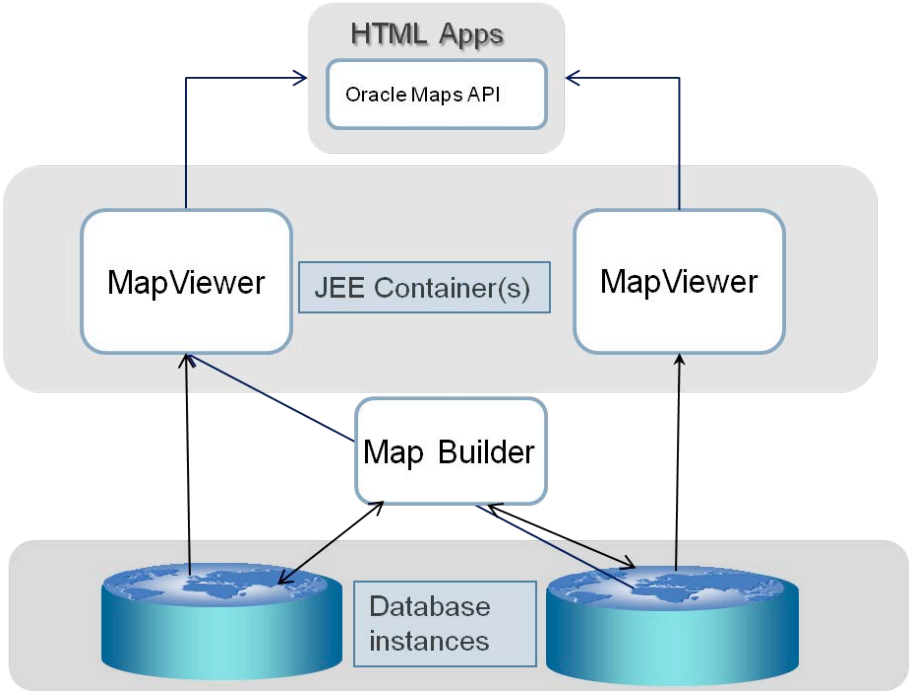
Creating a MapViewer data source

All geospatial application data and corresponding MapViewer metadata can be stored in one or more databases of your choosing; they are not tied to any specific MapViewer instance. MapViewer does not have its own system schema nor does it store any private information about itself.

A MapViewer data source defines the database schema(s) where the required geospatial application data and metadata reside. MapViewer connects to the data source to query the geospatial application data and then uses the associated metadata to identify the styles, themes and base maps it needs to render and serve online maps.

You can have multiple MapViewer instances running inside your organization (or even on a single computer), and they can all connect to the same database to serve maps. A MapViewer instance can also connect to multiple databases or schemas, and aggregate the geospatial application data, across schemas or databases, into a single map. The deployment architecture of MapViewer is flexible, as illustrated in the following figure.

Figure 2. MapViewer deployment architecture



The only connections between MapViewer and database instances (schemas) are the data sources, which are JDBC-based. You can add, remove or change a data source within a MapViewer instance using its web-based Admin page. You can also set up a middle-tier cluster that contains multiple MapViewer instances, each connecting to one or more databases, or databases configured as an Oracle Real Application Cluster.

A data source is always referenced by its name in map requests and application code that interacts with MapViewer. All map requests received by MapViewer include a master or default data source. Internally MapViewer uses this data source's schema as the default when looking up mapping metadata as well as querying data tables. If you have a theme (and its underlying data table) stored in a different schema, you need to define a data source for it and then explicitly specify the data source name in map requests that include the theme. Map requests include **"data source name, theme name"** pairs to aggregate geospatial application data from different schemas into a single map. (There are other ways to achieve data aggregation with MapViewer, but this is the easiest approach.)

A MapViewer instance can have as many data sources as needed. For beginners, it is highly recommended that you create one that connects to the MVDEMO schema mentioned earlier, which contains the MVDEMO sample data set, to experiment with all the built-in demos and tutorials.

MapViewer supports two types of data sources, mapviewer and container managed. A container managed data source is defined using the JEE server's admin console (e.g. as a JDBC Connection Pool and JDBC Resource in Glassfish). A MapViewer data source is defined by specifying the JDBC connection parameters directly in mapViewerConfig.xml via the MapViewer Admin Console. Use a container data source in most cases.

Creating a MapViewer Data Source

MapViewer data sources are defined in the MapViewer configuration file. The configuration file is an XML file located inside the MapViewer deployment directory. Use the MapViewer Admin web page to edit and save this file configuration. After logging in to MapViewer's Admin page, go to `Manage MapViewer > Configuration`. It opens the XML configuration file inside a text area. Scroll all the way down to the end of the file to find a sample MapViewer data source definition like the example below for the sample MVDEMO schema:

```
<!--
<map_data_source name="mvdemo"
                 jdbc_host="my.mapserver.com"
                 jdbc_sid="oradb10gr2"
                 jdbc_port="1521"
                 jdbc_user="mvdemo"
                 jdbc_password="!pwd4mvdemo"
                 jdbc_mode="thin"
                 number_of_mappers="6"
                 allow_jdbc_theme_based_foi="true"
 />
 -->
```

Uncomment it (by removing the XML comment tags in red), and then modify the database connection and login information (the fields in bold text above) to reflect your MVDEMO schema. If this will be your MVDEMO data source, then do not change the name attribute ("mvdemo") since it is the data source name hard-coded in all MapViewer standard tutorials and demos. Also make sure you have the exclamation point "!" in front of the supplied login password value. Next time you restart MapViewer it will automatically obfuscate this password.

Click on the `Save & Restart` button underneath the text area. MapViewer will restart, reload this configuration file, and the data source "mvdemo" will be created (make sure the database and its listener are both up!) To verify, just go to `Manage MapViewer > Datasources`. In the top panel it should list the "mvdemo" data source. If you need to add another data source, simply copy and paste the above definition into the same configuration file, and modify accordingly before saving it and restarting MapViewer.

Creating a Container Data Source

Use the container's admin console to define the data source (or JDBC Connection Pool and Resource in Glassfish) and then add an entry for it in the MapViewer configuration file. For example if a JDBC Resource named jdbc/mvdemo is defined in Glassfish then the corresponding entry will be:

```
<map_data_source name="mvdemo"
                 container_ds="jdbc/mvdemo"
                  number_of_mappers="6"
                  allow_jdbc_theme_based_foi="true"/>
```

MapViewer metadata

MapViewer metadata is crucial in defining a desired look and feel for your map. There are four types of metadata: styles, themes, base maps and map tile layers.

Styles

Styles are the basic graphical attributes that are applied when MapViewer renders a geometry shape or point. For instance, given a table storing information about lakes, you can use Map Builder to define a COLOR style to render all polygon geometries in the table with a solid blue interior fill and a black boundary stroke color. This COLOR style definition is stored the database and is accessible from the USER_SDO_STYLES view. You can give this style a (unique) name, such as 'Lake Color' and reference it in a theme definition.

A style is not tied to a particular theme. (Themes are covered in the next section.) Styles are independent, and stored as individual records in the database's USER_SDO_STYLES view. To change how a theme is rendered, either refer to a different style, or modify the definition of the currently assigned style. When you change the definition of a style, such as changing the fill or stroke color of a COLOR style, you are affecting the appearance of all the themes that currently use this particular style.

MapViewer supports six main types of styles, COLOR, MARKER, LINE, AREA, TEXT, and ADVANCED. TEXT styles are also called Labeling styles, they tell MapViewer how to label geometries. The typical properties of a Text style include font name, font size, font color, and stipulate how to anchor the actual text string to the target geometry.

It is also possible to create impromptu styles in your application. These are called dynamic or temporary styles. Dynamic styles also have a name and are typically referenced in dynamically created themes.

For a more detailed description of each type of style please refer to the MapViewer User's Guide:

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

Themes

Themes are important MapViewer metadata; they allow you to specify which data is displayed in a table and how to render the data. Think of a theme as a map layer. A map is typically composed of multiple themes stacked one on top of the other. For example, a state boundaries theme may be the bottom layer, with other themes rendered on top of it, such as lakes, roads, counties. When you request a map from MapViewer you are telling MapViewer which themes to render and in what order.

A theme tells MapViewer how to render the data stored in a spatial table. A theme called 'Lakes Theme' can be defined to include the previously defined COLOR style 'Lake Color' for a table storing lake geometries. When MapViewer renders the Lake geometries in the table, it looks up the style definition from the database. It creates an instance of the 'Lake Color' COLOR style, and applies 'Lake

Color' to the geometries. The definition and instance of a style are cached in MapViewer's memory after the initial database lookup.

A theme is a piece of metadata that tells MapViewer:

The name of the data table that contains the geospatial records to be rendered;

Optional query conditions for filtering/restricting the records;

The names of the styles to apply when rendering the geospatial records in the query result set; and

The names of the label (TEXT) styles if the theme requires labeling (annotating geometries with text).

This information is captured in a simple XML fragment and stored in a system view (USER_SDO_THEMES) in the database. These themes are called "pre-defined themes**"** because their definition is persisted. When processing such a theme, MapViewer will formulate a complete SQL query for it at run time by combining the various pieces of information in its definition.

A theme is always associated with one table or view. However, a table or view can have multiple themes associated with it. Themes can be created with different combinations of query conditions and style names.

Creating multiple themes with different styles allows the data in the same table to have more than one look and feel. For example, a table might contain information about all the hotels in a country. One could define a "generic" hotel theme that displays all hotels on a map using a certain icon (a MARKER style). However, if you wanted to display on a certain map only those hotels that are ranked 3 stars or above then create a new theme for the same data table, and specify a query condition that says "star_ranking >= 3", assuming the data table has a column "star_ranking". Now whenever you request this new theme to be drawn on a map, MapViewer will include this query condition in the theme query it builds, and the database will only return hotels with a three star or greater rating .

MapViewer automatically builds a SQL query string for each pre-defined theme. You have some means to control certain parts of the final query, for instance by supplying your own query conditions or predicates. If you want complete control of the query, or want MapViewer to only render the geospatial application data returned from your complete custom SQL query, you need to use what is called a Dynamic Theme. A **Dynamic theme** is also called a **JDBC Theme** in the MapViewer documentation. A dynamic theme is created ad-hoc in your application, by supplying a complete, custom query string and associated styling information in a map request. A dynamic theme definition is not stored in the user's metadata table.

Dynamic themes are helpful in the event you want to display certain map layers or features that are based on the results of an ad hoc, dynamically constructed query.

MapViewer's themes (pre-defined or dynamic) are always driven by a SQL query. When asked to render a theme (as part of a map request), MapViewer will always perform a SQL query on the data table associated with the theme, and render only the records returned in the query result set.

It is also important to remember that MapViewer by default will automatically append a spatial filter to the query in any pre-defined theme, in addition to user-specified query conditions. This spatial filtering, natively supported by Oracle Locator and Oracle Spatial through spatial indexing, is usually based on your current map-viewing window. It is a vital piece of the theme query because it restricts the result set to only those records that are within or interacting with the current viewing window. As such it ensures relatively predictable performance and response time, regardless of the total number of records the underlying data table may have, which could be very large. Note that the same spatial filter is even added to the custom query for dynamic themes, although you have the option of disabling it if you so choose. The following is an example of SQL query string constructed by MapViewer for a pre-defined theme, with the spatial filter predicate in bold text:

```
SELECT ROWID, GEOMETRY, 'M.AIRPORT', name, 'T.AIRPORT NAME', 1, 'rule#0',
name, AREA_ID
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'
```

Note the four bind variables in the predicate, :mvbox*, represent the current map viewing window and are filled in at run time.

Creating Themes

This step is necessary when working with your geospatial application data. The MapViewer tutorials and demos include themes that can be accessed by importing mvdemo.dmp and running the SQL script to load the metadata that includes the themes. The MVDEMO data set contains the geospatial application data tables and predefined themes. Feel free to go directly to the MapViewer's home page and try the tutorials and demos. Optionally, you can use your favorite SQL query tool such as SQL Developer or SQL*Plus to login to the MVDEMO schema and view the data tables as well as the theme definitions. The themes are all stored in the system view USER_SDO_THEMES. The following shows the xml definition of the CUSTOMERS theme:

```
SQL> connect mvdemo/pwd4mvdemo
SQL> set long 4000
SQL> select styling_rules from USER_SDO_THEMES where name='CUSTOMERS';

STYLING_RULES
--------------------------------------------------------------------------------
<?xml version="1.0" standalone="yes"?>
<styling_rules >
  <hidden_column>
    <field column="name" name="Name"/>
    <field column="city" name="City"/>
    <field column="sales" name="Sales" />
  </hidden_column>
  <rule >
    <features style="M.SMALL CIRCLE">  </features>
    <label column="NAME" style="T.RED STREET"> 1 </label>
  </rule>
</styling_rules>
```

If you have your own geospatial application data tables, then you must style the data stored in these tables and create themes to give MapViewer the information it needs to render them. This is quite easy to do with Map Builder. The essential steps are:

Launch Map Builder (for instructions, refer to the User's Guide)

Connect to the database schema containing your geospatial application data tables

In the Navigation panel, right click on the Themes node, and then select "Create Geometry Theme".

Use the Wizard to guide you through a few steps to specify the base table name, pick a particular rendering style, and so on.

The Theme editor is presented to allow you to further customize the new theme, or proceed to previewing the theme.

Base maps

The concept of a base map is probably familiar. It is a map that serves as a backdrop, providing contextual information for the map features of interest to the viewer. In MapViewer, a base map is simply an ordered collection of pre-defined themes.

An important step in the base map definition process is to define the map-scale visibility for each individual theme. For instance, if you want certain themes (such as detailed streets) display only when the map has zoomed to certain scale, then assign scale ranges or thresholds to these themes. Such scale range information is stored as part of the base map definition.

To create a new base map, simply launch the Map Builder utility, and use the Base Map wizard to select and arrange a list of themes to be included, and modify the scale range and other properties for each theme.

Note: MapViewer base maps are also the foundation of the (file-system cached) map tile layers used by the Oracle Maps component.

Tile Layers

The fourth type of mapping metadata is Map Tile Layer metadata. This metadata is primarily used by the Oracle Maps JavaScript API. It provides the JavaScript API with information about a draggable map tile layer, including its geographic boundary, coordinate system, number of discrete zoom levels and the size and format of individual map tiles at each zoom level.

A map tile layer is typically associated with a MapViewer base map. This type of map tile layer is often called "Internal Map Tile Layer". Log in to the MapViewer Admin page and open the "Manage Map Tile Layers" tab to create a new internal map tile layer. The tile layer creation wizard will guide you through a series of steps.

If you make changes to any of the themes and styles used to define a base map associated with a map tile layer then the existing map tile files must be deleted from the location on disk specified when creating the tile layer and the user's browser cached files must be purged. The MapViewer Web Admin page or Map Builder can be used to create a map tile layer.

You can create a map tile layer that gets its map tiles from an external source, such as a 3rd party web mapping service. For instance, you can create a map tile layer that gets all of the map tiles by making OGC WMS requests to a 3rd party map server. For more details, please consult the MapViewer User's Guide.

Making map requests and interacting with MapViewer

MapViewer listens for incoming map requests once MapViewer is running and a data source is created. It acts as a web service. The fact that it runs on a JEE platform is transparent to applications and end-users.

Applications and users access MapViewer via one of its APIs. MapViewer's native and lowest-level API is the XML request/response API. Applications construct a complete XML map request document, send it over HTTP to the listening MapViewer server, and wait for a response containing the generated map. This API is described in detail in the MapViewer User's Guide. Clicking on the Requests tab from the MapViewer home page displays a text area with a sample XML map request. Modify it for your data source, base map, and theme definitions to submit a sample XML request using this page. It is a handy tool to quickly test whether things are properly setup.

Most applications, especially hosted services applications, need a more high-level and interactive API. Oracle Maps JavaScript API is appropriate for this purpose. It is an interactive mapping API for embedding maps in your web pages. The MVDEMO sample application (included in the Quickstart kit) comes with over fifty tutorials describing how to use this API. You can find them by following the link "Oracle Maps Tutorials" on the app's welcome page.

If you are developing a Java Swing-based application, then you can use the MapViewer Java API. This API also follows a request/response model. For details, please refer to the MapViewer User's Guide for the Java Bean based API.

There are several demos in the MVDEMO sample application that you can use as basic viewing tools. All are accessible from the welcome page via the Demos tab. The first, "Jview", lets you enter a SQL query that includes a geometry column and visualize the results. To use this demo, pick a data source (the schema you want to query), and then enter up to 3 different SQL queries. A map showing the results of such queries will be displayed with zoom/pan functions. Another demo, "Omaps", can be used to quickly test your map tile layers and is written in the JavaScript API. There are several JSP-based demos that use the Java API, such as "mapclient" and "mapinit" which let you construct and submit map requests. Feel free to view their source code and modify them, or use them for a quick check of your environment.

Appendix A: Primer on Locator and Oracle Spatial AND GRAPH

This section provides a brief introduction to Locator and Oracle Spatial and Graph. For more information, please refer to the Oracle Spatial and Graph Developer's Guide. The Pro Oracle Spatial book is another great resource.

Almost all business applications find Locator, a free feature included in all editions of Oracle Database sufficient to meet their geospatial mapping needs. Locator stores, indexes and provides basic spatial analysis for point, line and polygon geometry data. The Oracle Spatial and Graph option for Oracle Database Enterprise Edition extends Locator with the ability to store, index and analyze complex data models, including persistent topology, networks/graphs with full connectivity information and 3D data. Oracle Spatial and Graph also manages geo-referenced raster data, including imagery, gridded data, satellite and aerial photos.

Details of the functionality in, and differences between Locator and Oracle Spatial and Graph can be found in the Spatial and Graph Developer's Guide or on the Oracle Technology Network here:

http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/

Geospatial application data is stored in Oracle Database tables like any other of data with the addition of a column of SQL type SDO_GEOMETRY (a native Oracle object type). This column stores the coordinates that define the shape or boundary of geospatial features.

Let's look at a simple table, named OFFICE, which stores information about all of the field offices for a company. It is created using the following SQL statement:

```
CREATE TABLE office (ID              number primary key,
                     Name            varchar2(128),
                     Manager         varhcar2(128),
                     NumEmployee     number,
                     Location        MDSYS.SDO_GEOMETRY);
```

This table can store for each office, its unique ID, name, manager's name, and the number of employees in that office. It also contains a geometry object that can store the office's location as a longitude/latitude pair. Alternatively, one can use any of hundreds of spatial reference systems supported by Locator and Oracle Spatial and Graph for geometry objects. The coordinate values and the associated spatial reference system ID will always be present in any SDO_GEOMETRY object.

Beginning with Oracle Database 10g, the prefix "MDSYS" (which is the system schema hosting all the spatial data types) is not needed for spatial data types appearing in your SQL statements.

Now let's insert an actual office record into the above table:

```
INSERT INTO office VALUES
(1001, 'San Francisco Branch', 'John Doe', 45,
 MDSYS. SDO_GEOMETRY(2001, 8307,
                                                    MDSYS.SDO_POINT_TYPE(-
79, 37, NULL),
                                                    NULL, NULL))
);
```

Note that the text in bold is the constructor for creating a point-type geometry object that goes into the sdo_goemetry column. Its coordinate is (-79,37), specified in the WGS84 geodetic reference system as indicated by the SRID value 8307. In other words this office is located at longitude –79 and latitude 37. The value 2001 simply indicates it as a 2-dimensional point object.

This is how geospatial features are stored in regular Oracle Database tables and how records are added to such tables. Once the geospatial data is stored the next step is to create a spatial index on the geometry column in order to perform spatial (and non-spatial) queries on these tables. You can also mix query conditions that involve both spatial and non-spatial predicates in a SQL query, for instance, finding all the restaurants within 5-miles of your current route that serve Chinese food.

Oracle and third party tools can help you import your existing geospatial application data into an Oracle Database. The Oracle Map Builder utility has a built-in wizard to import shapefiles into the database. Oracle partners provide a variety of tools and common data sets as described on OTN here:

http://www.oracle.com/technetwork/database-options/spatialandgraph/community/partners/spatial-partners

This data and more from any other public or commercial sources can be loaded in Oracle Database and served as online maps by MapViewer, after proper styling.

Oracle is committed to developing practices and products that help protect the environment

Oracle Fusion Middleware MapViewer Primer
October 2013
Author: Liujian Qian
Contributing Authors: Jayant Sharma

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com