

An Oracle MapViewer primer

LJ.Qian@oracle.com

Summer 2008

What Is MapViewer

Initially, there were desktop GIS applications that created maps from geographical data stored in various file formats. Then in the late 90's Oracle introduced Oracle Spatial that lets you store a variety of geospatial data inside the database, as regular tables. This brought spatial data management to a whole new level, and made it a part of enterprise IT infrastructure. If you are not familiar with Oracle Spatial at all, now is a great time to check out appendix A that provides a mini primer on this subject.

MapViewer is first introduced around 2002, as a simple reporting tool to satisfy the internal needs for creating business related maps by combining spatial and business data stored in a database. Since then MapViewer has gone through several iterations, and together with Oracle Spatial they form a powerful online mapping platform. The full name is now Oracle Fusion Middleware MapViewer. But we will simply refer to it as MapViewer in the remainder of this document.

MapViewer is a component of the Oracle Application Server (or Fusion Middleware), in other words, it is a mid-tier component. More specifically, MapViewer runs as a service inside an instance of Oracle Container for J2EE (OC4J). If you don't know what OC4J is, don't worry we will explain it in a bit. MapViewer itself is actually a generic J2EE application, and can even be deployed to other 3rd party J2EE containers. The latest MapViewer version 10.1.3.3 is also certified on WebLogic Server version 9 and higher.

What is Oracle Maps

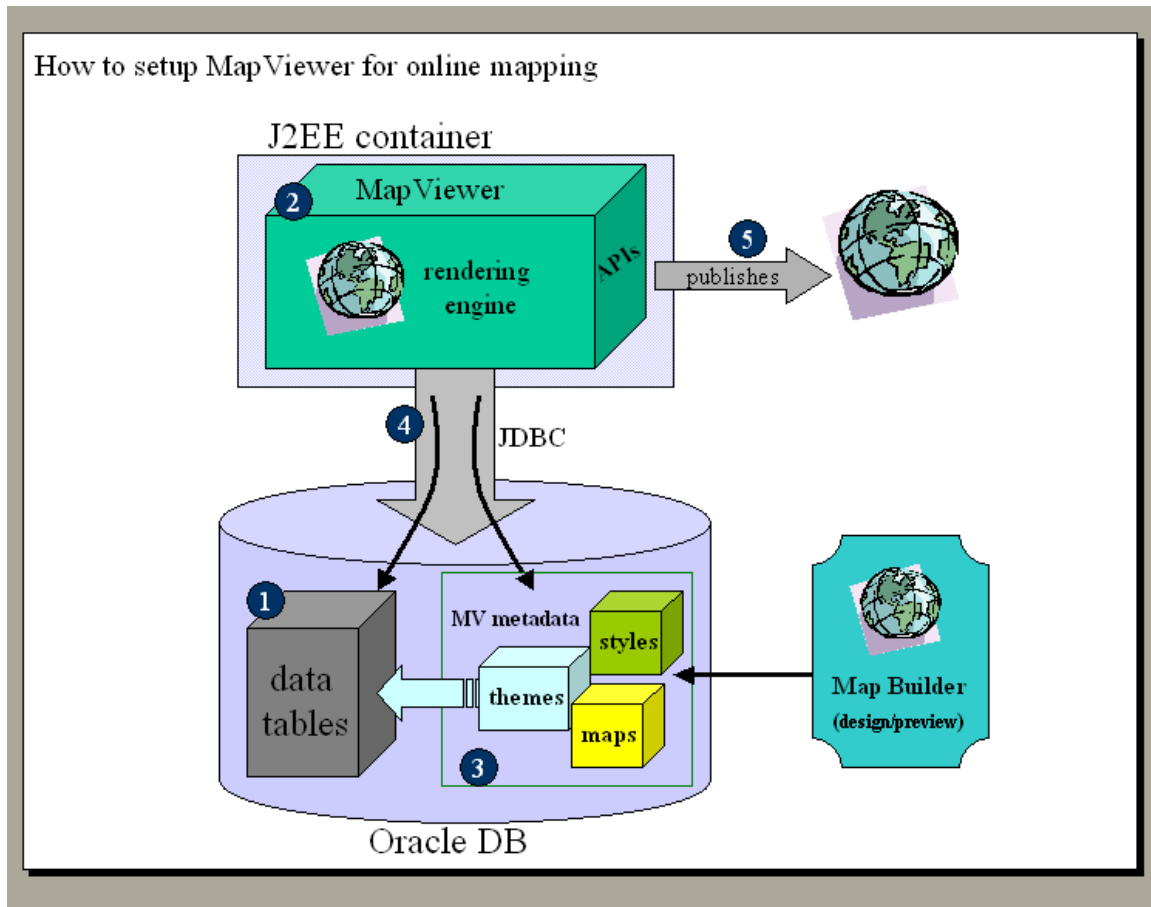
If you are interested in MapViewer, you no doubt have heard about the exciting new feature called Oracle Maps. So what is Oracle Maps? It is actually a feature set of MapViewer that provides a highly interactive JavaScript/AJAX API, and on the server side, a file system based image cache for map tiles, as well as an ad hoc query and pre-defined theme based feature server. The whole purpose of Oracle Maps is to support Google Maps like interactions entirely driven by your Oracle Spatial database. It offers many unique advantages (but we will not list all of them as they are beyond the scope of this document.) Please refer to the Oracle Maps Primer (to be published later) for a more detailed description. Suffice it is to know the following facts:

1. Oracle Maps is part of MapViewer; when you install MapViewer 10.1.3.1 or later version you have access to all the Oracle Maps features.
2. The core concepts of MapViewer, such as styles, themes and base maps, and how MapViewer works in general, are crucial even if you intend to work only with the Oracle Maps JavaScript API.

Setting up your MapViewer Environment

So what does it take to publish your own spatial data as online maps using MapViewer? Well it takes several general steps as illustrated in Figure 1.

Figure 1 Steps for setting up your Spatial/MapViewer environment



As you can see from the above diagram, the essential steps are:

1. Load or create spatial data in your Oracle database as regular data tables
2. Install MapViewer in your Oracle Application Server (or standalone OC4J).
3. Use Map Builder (a desktop authoring tool that comes with MapViewer) to style your data and create all the necessary mapping metadata for MapViewer.
4. Define a data source so that MapViewer can connect to the database schema containing the spatial 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 order above is not a strict one. For instance you can perform 1 and 2 in parallel. Steps 2 and 3 can also be done independent of each other. Note however you must have completed all 4 steps before you can attempt step 5.

Also note that the above steps are performed in their entirety only if you are starting from scratch. If you already have a database with spatial data stored in Oracle Spatial data types, then obviously you will start with step 2. What if you already have an existing MapViewer instance running but just want to hook it up with a newly populated or existing database schema? Well in this case you start with step 3, making sure you have created the necessary mapping metadata in that schema. This is then followed by step 4 where you define a data source in MapViewer so that it can connect to the schema.

Now that we have the big picture covered, lets go over each step in more detail.

Loading data into Oracle Spatial

The primary data source for MapViewer is always the spatial data managed by Oracle Spatial. If you are not familiar with Oracle Spatial, now is a great time to check out appendix A that provides a mini primer on this subject.

So how do you get data into Oracle Spatial? There are several options. For starters, you can use the Map Builder tool to import existing shapefiles into Oracle Spatial tables. Or, you can use 3rd party tools (such as the FME engine from Safe.com) to load data from any proprietary GIS data format into Oracle Spatial. In addition to loading your existing data into Oracle Spatial, you can always create tables from scratch and populate them with new records that contain your business related spatial data using Oracle SQL. Once your data is loaded into Oracle Spatial, it becomes the single source of truth as far as MapViewer is concerned (and should also be for all your other applications).

MapViewer itself comes with a simple spatial data set (in the form of an Oracle database export file) for demo purposes. This data set contains simplified data for states, counties, major cities and highways of the United States. It is commonly known as the “mvdemo” data set, and is included with your MapViewer kit downloaded from OTN. All of MapViewer’s built-in tutorials and demos run off this data set so it is highly recommended that you 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 inside the demo data set itself.

Make sure to import the MVDEMO sample data set that comes with MapViewer into your database!

Installing MapViewer

While you can deploy MapViewer to a fully installed Oracle Application Server (10.1.3 or later versions are recommended), the easiest way to get a MapViewer instance up and

running is use the **MapViewer Quick Start kit**. You can download the kit from the MapViewer OTN page here:
<http://www.oracle.com/technology/software/products/mapviewer/index.html>

The Quick Start kit contains a standalone OC4J (Oracle Container for J2EE) with MapViewer pre-deployed. What is OC4J? You can think of it as the guts of your Oracle Application Server (very overly simplified), a web server (like Apache), and a J2EE container able to run your JSPs and Servlets, all rolled into a small-footprint Java bundle. MapViewer, by the way, is just a set of servlets that will be brought up when the OC4J instance is started.

Note that going forward, OC4J is probably on the way out as Oracle has acquired BEA's WebLogic Server. But for all of your development needs, a standalone OC4J is perfect and very simple to manage, as far as MapViewer is concerned.

So download the Quick Start kit, unzipped it to a directory on your computer. And make sure your computer has the Java JDK 1.5 (or later) installed. If you don't have JDK 1.5, go to Sun's Java web site and download it. The latest MapViewer (versions 10.1.3.1 and later) requires JDK version 1.5 or later.

To start MapViewer (actually you start OC4J, which then brings up MapViewer automatically), simply modify and execute a simple batch file that comes with the Quick Start kit. For instance on Windows, the kit has a batch command file (start.bat) that contains commands like the following:

```
cd C:\mv10131qs\oc4j\j2ee\home  
"C:\Program Files\Java\jdk1.5.0_08\bin\java" -server -Xmx512M -jar oc4j.jar
```

The first line changes into the directory where you unzipped the Quick Start kit. The second line uses the Java command to start up OC4J. Note you will need to change the directories (in bold red texts) to match the actual locations of the unzipped kit and JDK. The option `-Xmx512M` in the second command line tells Java to make a maximum of 512 MB memory available to the OC4J process. You can use a smaller or larger number if you want. Basically it will depend on how big/detailed your typical maps are, and how many concurrent users will be requesting maps from MapViewer.

When you run the above commands for the first time, you will be prompted to enter a password. This password is for the OC4J admin account (and also used to log into MapViewer admin page). Remember this password, as you will need it to log into MapViewer's admin page. The admin user's name is always **oc4jadmin**.

When the commands run successfully, you will see a bunch of messages from OC4J as it starts up, and then finally messages from MapViewer itself. These messages are quite verbose, but the essential ones are:

```
INFO [oracle.lbs.mapserver.oms] *** Oracle MapViewer started. ***
```

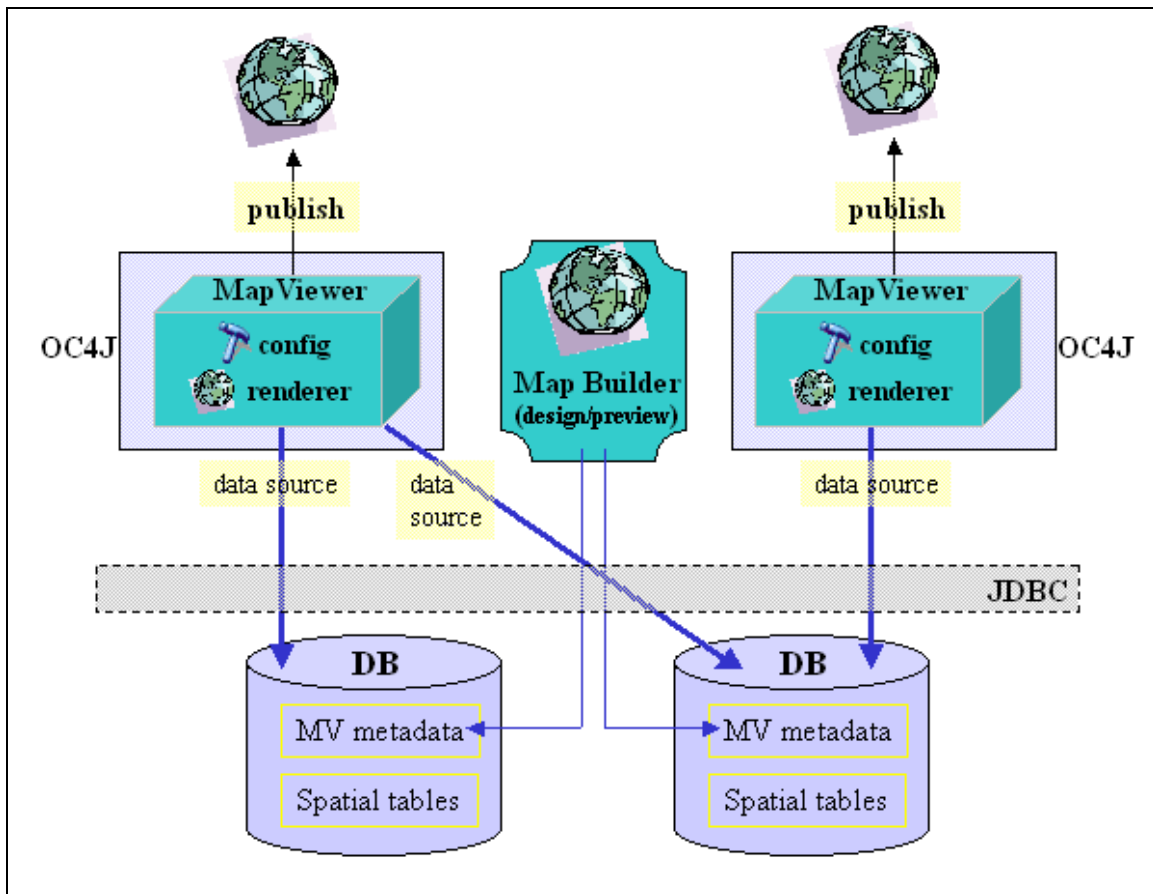
The above will be buried in the middle of other MapViewer messages. And finally:

```
INFO [oracle.lbs.mapcache.mcservlet] *** Oracle MapTileServer started. ***
```

If you see these, then your MapViewer is up and running, and you should be able to visit the MapViewer home page by opening the URL <http://localhost:8888/mapviewer> in your browser.

It is worth noting that because all the spatial data as well as their accompanying MapViewer metadata are stored in databases, they are never tied to any specific MapViewer instance (or Map Builder). 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 and serve maps out of it. Any particular MapViewer instance can also connect to multiple databases or schemas, and aggregate the spatial data, cross schema or cross database, into a single map. The deployment architecture of MapViewer is indeed very flexible, as illustrated in the following figure.

Figure 2 MapViewer deployment architecture



As you can see the only connections between MapViewer instances and database instances (schemas, actually) are the data sources, which are JDBC based bridges linking the two sides. You can add, remove or change a data source within a MapViewer instance using its web-based Admin page (as described later). You can also setup a middle-tier cluster that contains multiple MapViewer instances, each connecting to one or multiple databases, even databases configured as RAC.

Finally, note that MapViewer does not have its own system schema nor does it store any private information about itself; your spatial data and mapping metadata can be stored in any database schema.

Creating MapViewer metadata

With the spatial data tables alone MapViewer cannot produce nice looking maps out of blue; it can draw them as plain points, lines and polygons in a few default colors, but that is probably not what you want (other than for a quick viewing of your data). To generate professional looking maps, MapViewer relies on mapping metadata that tells it which data tables to get data from, and more importantly, how to style or render the data into a map that conforms to the look and feel you (or your organization) desire. Such metadata typically include map symbologies (called **Styles** in MapViewer's term), **Theme** and **Base map** definitions. If you think of the spatial data tables as providing the raw data, then the MapViewer metadata provide "skins", styling rules, designs or templates that tell MapViewer how to render the same raw data into different maps for different scales and/or scenarios.

The mapping metadata are essential to the functioning of MapViewer. They are always stored in the database, usually in the same schema as the spatial data themselves. Most of the metadata are created and maintained using the desktop authoring tool Map Builder that comes with MapViewer. Note that Map Builder also lets you preview your spatial data as you are "styling" them. Specifically, Map Builder lets you visualize how a style, theme or base map will look like when rendered by MapViewer. Map Builder itself, however, does not rely on MapViewer for its operation. In fact Map Builder connects directly to your database schema(s), and it embeds the same rendering engine that MapViewer uses in order to provide instant and authentic preview of your data.

Because of the importance of the MapViewer metadata, we will actually come back and deep dive into this topic once we have covered all the steps illustrated in Figure 1. Or you can go directly to section titled "**MapViewer metadata**" now.

Creating a MapViewer data source

In your MapViewer instance, before you can do practically anything with it, you must define a data source that tells it which database schema to connect to. Once a data source is defined, MapViewer will look for the metadata that tells it what styles are available, which themes and base maps have been created and can be served as online maps, et al.

You always refer to a data source by its name in your map requests and also in your application code that interacts with MapViewer. Basically in all map requests received by MapViewer there is always a master/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, all you need to do is define a data source for that schema, then explicitly specify the data source name when incorporating the theme in your map

requests. In other words, to aggregate spatial data from different schemas into a single map, you use the (**data source name, theme name**) pairs in your map request (there are other ways to achieve this kind of 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, just so you can play with all the built-in demos and tutorials.

Before you go ahead and create a data source, it is important to know that MapViewer supports two types of data sources: dynamic and persistent. A dynamic data source, once defined, is only available in the current MapViewer session. As soon as you restart MapViewer or its OC4J container the data source definition is gone. A persistent data source on the other hand will always be available even if you restart OC4J/MapViewer (unless of course if the database itself is down or gone). In most cases you want your data source to be persistent; only use dynamic data sources when you are doing some quick testing or just want to connect to a db schema, take a quick look at some of the spatial data and be done with it.

To create a dynamic data source (if you insist), go to the MapViewer home page, then click on the Admin button, log in using oc4jadmin and the password you provided when you first started OC4J. After logged in, go to Manage MapViewer > Datasources. It will list all existing data sources you may have. Below the list is a panel where you define a new data source by providing a name and the database connection information.

Persistent data sources are defined in MapViewer's configuration file. The configuration file is an XML file located inside the MapViewer deployment directory, which is itself deep inside the unzipped OC4J directory tree. Fortunately you don't have to manually locate this file and edit it; you can use the same MapViewer admin web page to edit and save it. After logging in to MapViewer's Admin page, just go to Manage MapViewer > Configuration. It opens the xml configuration file inside a text area. Now scroll all the way down to the end of the file. There you will see a sample MapViewer data source definition like the following (for the sample MVDEMO schema):

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

-->

Make sure you 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 that of the MVDEMO schema. If this will be your MVDEMO data source, then do not change the name attribute (“mvdemo”), as 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.

Now 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 if everything goes smoothly (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 other data sources, simply copy and paste the above definition into the same configuration file, and modify accordingly before saving it and restarting MapViewer.

MapViewer metadata

Earlier we mentioned that mapping metadata are crucial in defining a desired look & feel for your map. Lets now take a deeper look at these metadata. There are 4 types of metadata: styles, themes, base maps and map tile layers.

MapViewer Styles

Styles represent basic graphical attributes that can be applied when MapViewer renders a geometry shape or point. For instance, if you want MapViewer to render all the polygon geometries stored in a table in such a way, that each polygon’s interior is filled with solid blue while its boundary is drawn in black, you can define a Color style to achieve this. This color style, which can be created in Map Builder, will specify solid blue as its fill color and black as its stroking color. You can then store this Color style definition in the database. More specifically, it is stored in the schema’s USER_SDO_STYLES view. You can now reference this style with its (unique) name.

So how do you actually use a style? You simply mention its name when defining a theme. We will describe what is a theme in the next section, suffice it is to say that a theme tells MapViewer how to render the data stored in a spatial table. Lets say the above Color style is called ‘My Color 127’. And you have created a theme based on a table that contains all the lakes from your home state. This theme now can assign ‘My Color 127’ as its rendering style. What happens next, is that when MapViewer starts rendering the geometries for this theme, it will look up the definition of this particular style from the database. It then creates an instance of this style, and applies this style to the geometries belonging to this theme. ‘Applying’ here means MapViewer will first fill the lake boundaries with a solid blue, and then draw their boundaries using black color. The

definition and instance of a style will be cached in MapViewer's memory after the initial database lookup.

You will soon realize that a style is not tied to any particular theme at all. In other words, styles themselves are quite independent, and are in fact stored as individual records in the database in the above-mentioned USER_SDO_STYLES view. To change how a theme is rendered, all you need to do is change to a different style, or modify the definition of the currently assigned style. Note that when you change the definition of a style, such as changing the fill or stroke color of a Color style, you are potentially impacting the appearance of all the themes that currently use this particular style.

Note: anytime you change the definition of a style, theme or basemap, you will need to purge MapViewer's cached metadata definitions. To do this, go to MapViewer's Admin page, click **Datasources** under the **Manage Mapviewer** tab, select the proper data source in the existing data sources list, then click the **Purge cached metadata** button.

MapViewer supports 6 main types of styles, COLOR, MARKER, LINE, AREA, TEXT, and ADVANCED. TEXT styles are also called Labeling styles, as they tell MapViewer how to label geometries. For instance the typical properties of a Text style include font name, font size, font color, how to anchor the actual text string with regard to a target geometry, and so forth.

It is also possible to create styles on the fly in your application. These are often 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 check out the MapViewer User's Guide.

MapViewer Themes

Among all of the MapViewer metadata, probably the most important ones are **Themes**. Conceptually you can think of a theme as a map layer. A map is typically composed of multiple themes stacked on top of each other, for instance a state boundaries theme will be the bottom layer, with the other themes rendered on top of it. When you ask for a map from MapViewer, you really are telling MapViewer which themes to render and in what order.

So what exactly is a theme? It is a piece of metadata that tells MapViewer these things:

1. The name of the data table that contains the geospatial records to be rendered;
2. (Optionally) some query conditions for filtering/restricting the records; and
3. The names of the styles to apply when rendering the geospatial records in the query result set.
4. The names of the labeling (Text) styles if the theme requires labeling (annotating geometries with texts).

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

From the above description, you may get the impression that each data table can have only one theme defined for it. This is not true. Yes any pre-defined theme will always be associated with one data table (or view), but the same data table can have many themes defined on top of it. If you think about it it's quite logical. For instance, say you have a table that contains all the hotels in a country. It is natural to define a "generic" hotel theme that displays these hotels on a map using a certain icon (a Marker style in MapViewer terms). But what if you want to display on certain map only those hotels that are ranked 3 stars or above? Easy, you just 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 always include this query condition in the theme query it builds, and the database will return only such hotels. Similarly, you can create different themes for the same data table just to achieve different looks by assigning different styles. In other words you can completely customize what and how you want to render the data stored in a table by creating themes with different combination of query conditions and style names.

As mentioned earlier 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 just render the geospatial data returned from your complete custom SQL query, you need to use what is called a **Dynamic Theme**. Sometimes a Dynamic theme is also called a **JDBC Theme** in MapViewer's documentation. A dynamic theme is one that you create on the fly in your application, by supplying a complete, custom query string and associated styling information in a map request. These themes' definitions are never stored in the user's metadata table.

In most cases pre-defined themes are all you need; but 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, then the dynamic themes are a great option.

It is critical to realize that MapViewer's themes (pre-defined or dynamic) are always query driven. 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 know that MapViewer by default will automatically append a spatial filter in any pre-defined theme's query (in addition to those user specified query conditions). This spatial filtering, natively supported by Oracle Spatial through its spatial index, is usually based on your current map-viewing window. It is a vital piece of the

theme query as it restricts the result set to only those records that are within or interacting with the current viewing window. As such it ensures a relatively predictable performance and response time, regardless of the total number of records the underlying data table may have (which can be in millions or even billions). Note that the same spatial filter is even added to the custom query for dynamic themes, although there you do have the option of disabling it if you so choose. If you are interested, 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 binding variables in the predicate, **:mvbox***, represent the current map viewing window and are filled in at run time.

MapViewer always attaches a spatial filter predicate to a theme query unless explicitly told not to.

Creating Themes

This step is only necessary if you are working with your own spatial data. If all you want to do, initially anyway, is to view the standard tutorials and demos of MapViewer, then you are all set. The MVDEMO data set not only contains the spatial data tables, it also contains half dozen predefined themes. Feel free to go directly to the MapViewer's home page and try out the tutorials/demos. If you are curious enough, you can use your favorite SQL query tool such as SQL Developer or SQL*Plus to login to the MVDEMO schema and check out the data tables as well as the themes directly. The themes are all stored in the system view USER_SDO_THEMES. For instance, the following shows how to view the xml definition of the theme CUSTOMERS:

```
SQL> conenct mvdemo/mvdemo
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 do have your own spatial data tables, then you must style the data stored in these tables by creating various themes so that MapViewer knows how to render them. This is quite easy with Map Builder. The essential steps are:

1. Launch Map Builder (for instructions, check out the User's Guide)
2. Connect to the database schema containing your spatial data tables
3. In the Navigation panel, right click on the Themes node, then select "Create Geometry Theme".
4. A Wizard will now guide you through a few steps where you specify the base table name, pick a particular rendering style, and so forth.
5. You will now be presented with a Theme editor, where you can further customize the new theme, or proceed to previewing this theme.

MapViewer Base maps

The concept of base map should be familiar to most people. It is basically a map that serves as backdrop, providing contextual information for the really important map features that the viewer is interested in. In MapViewer, a base map is simply an ordered collection of pre-defined themes. One of the most important things you can do in the context of a base map is defining map-scale based visibility for each individual theme. For instance, if you want certain themes (such as detailed streets) to not display until the map has zoomed to certain scale, then you can assign proper scale limits or thresholds to these themes. Such scale range information are stored as part of the base map definition.

To create a new base map, simply launch the Map Builder tool, and follow its New Base Map wizard where you can select and arrange a list of themes to be included, and modify each theme's scale ranges and other properties.

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

MapViewer Tile Layers

The forth type of mapping metadata is Map Tile Layer. This type of mapping metadata is mainly used by the Oracle Maps JavaScript mapping API, as it tells the JavaScript API

various pieces of information about a draggable map tile layer, such as its geographic boundary, coordinate system, number of discrete zoom levels as well as 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”. You create a new internal map tile layer by logging into the Admin page of MapViewer, then opening the “Manage Map Tile Layers” tab. The tile layer creation wizard will guide you through a series of steps.

Note that since a map tile layer is based on a base map, which is itself based on individual themes and styles, any change to those dependant mapping metadata will require you to delete the existing map tile files (from the disk where MapViewer is running) associated with the tile layer, and purge your browser of its cached files.

You can also 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 OpenGIS WMS requests to a 3rd party map server. For more details, please check out the MapViewer User’s Guide.

Making map requests and interacting with MapViewer

So that your MapViewer is up and running. And you have created a data source for MapViewer to connect to the database schema containing your enterprise geospatial data and mapping metadata. At this point, MapViewer patiently listens for incoming map requests. In other words MapViewer simply acts as a web service, and the fact that it runs on a Java J2EE platform is of little relevance as far as your application or end user is concerned.

So how do you (or your application) interact with a running MapViewer? Through one of its APIs. MapViewer’s native (and lowest-level) API is the REST-ful XML request/response API. With this API, your application constructs a complete XML map request document, sends it over HTTP to the listening MapViewer server, and waits for a response containing the generated map. For details of this API, please check out the MapViewer User’s Guide. If you go to the MapViewer home page, click on the Requests tab, it displays a big text area where you will see a sample XML map request. Feel free to modify it accordingly or submit your very own XML request using this page. It is a very handy tool to do some quick testing on whether things have been setup properly.

Most applications, especially those of Web 2.0 type, will want to use a more high-level and interactive API. This is where the Oracle Maps JavaScript API fits right in. It is an AJAX-based interactive mapping API that can be embedded in your web pages. MapViewer itself comes with 50+ tutorials on how to use this API. You can find them by following the link “Oracle Maps Tutorials” at the MapViewer home page.

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

Finally, there are also several demos in your MapViewer that you can use as basic viewing tool. They are all accessible from the MapViewer home page after you click the Demos tab. The first demo, JView, is a great little tool that allows you to type a custom SQL query and visualize the query results immediately. To use this JView demo, you simply pick a data source (for the schema you want your queries executed against), 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 itself written in the JavaScript API. There are also several JSP-based demos that use the Java API, such as "mapclient" and "mapinit". Feel free to view their source code or just use them for a quick testing of your environment.

Appendix A: Mini primer on Oracle Spatial

This section provides a very brief and overly simplified introduction to Oracle Spatial. For more information, please refer to the Oracle Spatial User's Guide that comes with your Oracle database. Another great resource is the Pro Oracle Spatial book.

In an Oracle database, spatial data are stored in tables just like any other type of data. The only notable difference from a 'regular' table is that tables with spatial data will have a column with SQL type SDO_GEOMETRY (which is considered an object type in Oracle's term). This column is used to store the actual coordinates that define the shape or boundary of your geospatial features.

Lets look at a simple table, OFFICE, which can be used to store the information of all the field offices of a fictional company. It can be created in an Oracle database using the following SQL statement:

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

The above table lets you store, for each office, its unique ID, name of the office, manager's name, and number of employees at the office. It also contains a geometry object that captures the office's location in terms of longitude/latitude. Well it does not always have to be in longitude/latitude; you can use any spatial reference system (such as any of the myriad of projected map coordinate systems) for such geometry objects. The important thing to know is that both the coordinate values and the associated spatial reference system ID will always be present in any SDO_GEOMETRY object.

Note that beginning with Oracle database 10g, you can safely drop the prefix "MDSYS" (which is the system schema that hosts all the Oracle Spatial data types) from the spatial data types appearing in your SQL statements.

Now lets 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 Location column). It's coordinate is (-79,37), specified in the WGS84 geodetic reference system as indicated by the 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.

So there you have it: you now know how geospatial features (of simple vector type anyway) are stored in the database (in regular tables), and how to add records to such tables. The next step is to create a spatial index on the geometry column and you will be able to run a host of spatial (and non-spatial) queries on such 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 my current route that serves Chinese food.

A deeper introduction of Oracle Spatial is beyond the scope of this document, so you are kindly referred to the Oracle Spatial User's Guide for further reading. Suffice it is to mention here that you can store different geometry types in Oracle Spatial (not just points!). You can even store complex data models such as persistent Topology, as well as Networks/Graphs (with full connectivity information) and 3D data in Oracle Spatial database. Oracle Spatial can also manage all of your raster data (imageries, grid data, satellite and aerial photos et al).

Finally, there are 3rd party tools that help you import your existing spatial data (stored in various file formats) into Oracle Spatial. For starters, the Oracle Map Builder tool (which will be described a bit later in this doc) does have a built-in wizard that can import your shapefiles into an Oracle Spatial database. Note that MapViewer (or Oracle Spatial) does not come with any exhaustive, commercial usage data, although some sample data sets do come with Spatial and MapViewer. For commercial data, such as street networks or high-quality zip code boundaries, you will have to go to data providers, such as Navteq, and buy the data from them. Such data can then be loaded into your Oracle database and served as online maps by MapViewer (after proper styling). Of course you can also get data from any other public or commercial sources.

Note that when we say Oracle Spatial database, we really mean just any regular Oracle database with the Oracle Spatial option installed. Actually, for your typical geospatial applications, you may even be able to rely solely on Oracle Locator, a free feature of any Oracle database editions. Oracle Locator contains an essential subset of what Oracle Spatial provides. Suffice it is to say that basic storage and querying of geometry data are fully provided in Oracle Locator. Again, please refer to the Spatial User's Guide for more information.