Sessions S298751 & S298752:

Integrating Maps In Oracle Business Intelligence Enterprise Edition
Introduction
This document describes the methods to integrate Oracle Application Server MapViewer (MapViewer) with Oracle Business Intelligence Enterprise Ed (OBIEE). Specifically, we describe four specific integration scenarios at a conceptual and technical-overview level. This document assumes familiarity with both MapViewer (particularly the Non-Spatial Data Provider, or NSDP, feature), as well as OBIEE. This document is not a tutorial on those technologies but is a cookbook on one method of integration to achieve specific flows. There may be other (and better, simpler, more efficient) ways to achieve the same integration using the published OBIEE and MapViewer Javascript APIs.

The OBIEE product roadmap includes MapViewer integration that may include some of the functionality described in this document. Once available this functionality will remove much of the need for the programmatic methods described here. However not all functionality outlined in this document may be productized in the near term. More importantly, the ability to create highly flexible programmatic integrations of MapViewer with OBIEE will remain a critical requirement, for certain use cases, regardless of specific levels of product integration.

Those familiar with Oracle Spatial and MapViewer should skip to the Hands-on Lab Material section. The next few sections product version dependencies for the lab and provide an overview of key Oracle Spatial & MapViewer concepts.

Product Version Dependencies
The integration described in this document requires:

**AS MapViewer shipped with Oracle Application Server 10.1.3 or greater**
This is the release that includes Oracle Maps, the AJAX API for MapViewer, which supports the NSDP feature for joining MapViewer data to OBIEE analytic results in the mid-tier.

**OBI EE 10.1.3.3**
For the interactive dashboards, ad-hoc querying, reporting and analysis.

**Oracle Database 10g (10.1.0.3) or later**
For managing all the geospatial content (e.g. administrative boundaries and facility locations) and MapViewer metadata (styles, symbology, and map definitions).

**Oracle Spatial & Locator**
Oracle Spatial and Oracle Locator make location a native type within Oracle Database 10g. Oracle Locator is a feature of Oracle Database 10g Express, Standard, Standard One, and Enterprise Editions that provides spatial object type storage, indexing, and operations, to support...
a variety of location-based services (LBS) and 3rd party GIS solutions. Oracle Spatial is an option for Oracle Database 10g Enterprise Edition that provides advanced spatial features to support high-end GIS and LBS solutions. This lab only requires Oracle Locator functions.

**Oracle Fusion Middleware MapViewer**

Oracle FMW MapViewer is a J2EE application that comes with every edition of Oracle Application Server. Once deployed to Oracle AS, it listens and accepts incoming map requests. You can connect it to any Oracle database schema that contains spatial data, and it will be able to render sophisticated maps from those data.

Starting with MapViewer version 10.1.3, it also comes with a standalone desktop tool called **Map Builder**. You can use Map Builder to preview spatial data stored in Oracle Spatial, and define sophisticated styling rules so that professional looking maps can be generated by MapViewer. The styling rules, once created in Map Builder, are stored back into the database. The MapViewer server then picks up those rules and apply them when rendering maps in response to incoming map requests.

The styling rules for the themes (map layers) involved are defined and stored in the MVDEMO schema in the following views:

- **USER_SDOYLES**: this view stores individual map symbols and other types of styles such as: LINE patterns, COLOR definitions, and thematic mapping styles.
- **USER_SDO_THEMES**: this view stores the definitions of individual map layers or themes. A theme is always based on a data table or view, and specifies what rendering and labeling styles to be used.
- **USER_SDO_MAPS**: this view stores the definitions of base maps. Each base map contains a list of themes. It specifies the display order and on/off status of each theme at various map scales.

**Oracle Maps**

In the pre-AJAX era, online mapping application typically worked in a serialized single request/response model. More specifically, the browser/client issues a request for a map, and the server generates a map in response. The map is then displayed to the client. Any interaction on the client side, such as zooming or panning the map, will often result in a new request being sent to the server to produce a new map in response to the client action. In this paradigm, the server is constantly generating, and re-generating, maps on the fly. This often severely affects the scalability and performance of the overall application.

Oracle Maps, similar to Google Maps, introduces the caching of pre-generated map tiles on the server and in the client (via browser cache). As a part of MapViewer 10.1.3.1, Oracle Maps provides map tile caching and many other unique features:

- Caching of map tiles for any base map at the mid-tier
- Support of Features of Interest (FOI). FOIs are JavaScript (geospatial) objects generated at the mid-tier based on dynamic database queries and displayed at the client side with built-in interactions such as mouse over and tool tip.
Highly interactive client side mapping interface. Your application will have built-in capability to drag the map around, instantly zoom or pan to any desired level/location, draw free-hand shapes on the map, add collapsible birds-eye-view map inside the main map area, among many other off-the-shelf functions. All of these functions are available to you through an open JavaScript API.

Oracle Maps provides powerful interactive Web 2.0 mapping while hiding the complexity of spatial data queries and the cartographic rendering process from application developers. Users and developers can customize the appearance and behavior of the map; they can control visual map characteristics—such as the background color, the title, the symbology used to portray features such as roads, store locations and property boundaries, and so on using extensible metadata stored in database tables. It is also possible to incorporate dynamically obtained geospatial data, such as customer locations, and plot these on top of a base map. Thematic mapping portraying the distribution of attributes such as population density, demographic information measuring income, education, etc. is also supported through the MapViewer API, if the user has the needed baseline data.

**Summary of Key MapViewer Concepts**

Some fundamental concepts include feature, style, theme, base map, and map. A feature is an entity with spatial and non-spatial attributes such as cities, rivers and highways. A polygon, a line or a point may represent a feature. Some features displayed on a map essentially provide a spatial context and the user is unlikely to want to click on them and have some action performed as a result. Streets, rivers, and forest may be such features. The user may want to interact with other features, such as store locations, postal code boundaries, or oil pipelines displayed on the map. These are called Features of Interest (FOIs). Within the context of a MapViewer application, FOIs usually display the geographic extent or location of some business object such as customer or store location, sales territory, or delivery zone.

A style defines rendering properties for a feature. For example, if the feature is a polygon showing a county, then the style may define the fill color for the county or it may define a pie chart to be drawn over the county.

A theme is any collection of features that have a common set of attributes and a location. Typically, a theme is associated with a spatial geometry layer. For example, a layer on the map displaying boundaries for US states would be considered a theme. A layer on the map displaying major US highways would also be considered a theme. Other types of themes include, but are not limited to, image themes which may be used to display satellite overlay or a relief map.

A basemap consists of one or more themes. For instance, a base map may be composed of themes that show US state boundaries, county boundaries and markers for cities.

A map can have a basemap 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.

A collection of features of interest is known as a FOI theme. It may be theme based or user-defined. An FOI theme based on a database query, or pre-defined theme in MapViewer parlance, is hence known as a themeBasedFOI. A user-defined FOI is created on the client side by the application. For example, application may create these FOIs from results of an Answers request or some other web service call.
Themes, maps, and styles, and other MapViewer data and metadata are always associated with a **datasource**. A datasource is essentially a database connection and is usually a container (OC4J) managed datasource.

While MapViewer renders spatial data some of the content rendered on a map consists of non-spatial data such as sales volume, crime incident type, or store size. Often this data may come from a datasource that has no geographic information (e.g. Longitude-Latitude of store location, or coordinates defining the extent of a sales region) but does have an identifier, such as a place name or location_id, that links it to a geographic entity. In MapViewer parlance this is known as a **non-spatial data provider**. Examples are CSV files, results of an Answers request output as XML, or results of a SQL query that contains no geographic objects.

**The Hands-on Lab Material**

The hands-on lab walks you through the steps involved in adding a map to an OBIEE dashboard using MapViewer and OBIEE’s Javascript APIs. It describes four integration scenarios: (i) Display a thematic maps of an Answers report, (ii) Drill to a new dashboard page from the map, (iii) Invoke an Answers report by clicking on the map, and (iv) Click on a item in an Table view and highlight corresponding item on the map.

The lab assumes the relevant Oracle Spatial and MapViewer content exists in the database and various map metadata (themes, styles, base maps) have been set up. The following section, however, briefly outlines some of these steps for convenience.

**Oracle Maps Page Described Below**

This section describes the steps required to produce the following interactive map, which will be integrated into OBIEE dashboards. This section is for reference only. Hence the screenshots, database connection, themes, styles, and base map names and code snippets may not match those used in the actual hands-on lab sections.

![Oracle Maps Page](image)

**Step 1:** Open MapBuilder and open or create new database connection
Step 2: Review existing styles and create new styles as needed. Styles define the look of features in a map. This includes area fills and outlines, line styles, marker styles, and label styles.

Step 3: Right click on Geometry Themes to access wizard to create theme to be presented as interactive feature in the map.
Step 4: Click next to select data to used for theme.
Step 5: Click next to select a style for the theme.

Step 6: Click next to select column and style to used for labeling. With Oracle Maps, labels are manifest as tool-tips on mouse-over the feature on the map.
Step 7: Click next to access form for entering a query to filter the data. Enter filtering query if needed, or just click next.

Step 8: Click on the Advanced tab to define attributes to be packaged with the theme. We will use an attribute defined here to create the join to the OBIEE results.
Step 9: Add the attribute to used as the common key for join with OBIEE results. Add any other columns desired; they will be available to appear in pop-up windows and for other interactive uses.

Step 10: The next form displays the metadata that will be created based on the previous form entries. Click finish if ok, otherwise click back to make corrections.
Step 11: Click on the preview tab and then the green arrow to preview how the theme will look on a web page.

Step 12: Review or create base map. Base maps are simply collections of themes used as static backdrops under **interactive themes**.

Step 13: Create basic html to display base map and states theme

```html
<html>
```
The resulting page in a browser:

Step 14: Add NSDP join to analytics data for thematic mapping

The following XML data is a surrogate for the actual data that will be fed by the BI server based on Answers requests:

```xml
<nsdp_xml>
<table>
<tr><th>State</th><th>PriorYearPercentChange</th></tr>
<tr><td>ALABAMA</td><td>-4.7636</td></tr>
<tr><td>ALASKA</td><td>-24.482999767133</td></tr>
</table></nsdp_xml>
We add the following code to our page in order to join the states theme to this XML data and thematically map the attributes from the XML source:

```html
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script language="Javascript" src="/mapviewer/fsmc/jslib/oraclemaps.js"></script>
<script language=javascript>
  var mapview;
  var dynStatesFOI;
  function showMap() {
    var baseURL = "http://"+document.location.host+"/mapviewer";
    var mapCenterLon = -96;
    var mapCenterLat = 37;
    var mapZoom = 0;
    var mpoint = MVSdoGeometry.createPoint(mapCenterLon,mapCenterLat,8265);
    mapview = new MVMapView(document.getElementById("map"), baseURL);
    mapview.addBaseMapLayer(new MVBaseMap("EPA.EPA"));
    mapview.addNavigationPanel("WEST",null,false,null);
    mapview.setCenter(mpoint);
    mapview.setZoomLevel(mapZoom);
    dynStatesFOI = new MVThemeBasedFOI('dynStatesFOI', 'EPA.STATES');
    dynStatesFOI.setRenderingStyle("C.ACTIVE");
    setupNsdp(dynStatesFOI); // Invoke NSDP setup for join to XML data
    setupDynamicStyles(dynStatesFOI, -50, -10);
    mapview.addThemeBasedFOI(dynStatesFOI);
    mapview.display();
  }
  function setupNsdp(dynStatesFOI) {
    //default NSDP is always available with MapViewer 10.1.3.1 production
    var nsdpInfo = new MVNSDP("defaultNSDP");
    nsdpInfo.setTheme("STATES"); //the base theme
    nsdpInfo.setKeyColumn("STATE_NAME"); //match column in the base table of the theme
    ...}
</script>
</head>
</html>
```
nsdpInfo.setRenderStyle("C.ACTIVE"); //default style;
var ps = new Object();
//where the dynamically generated BI data can be found
ps["xml_url"] = "http://"+document.location.host+"/mapviewer/epa/nsdp/states_sample.xml";
nsdpInfo.setParameters(ps);
//this tells mapviewer to join the BI data with the theme
dynStatesFOI.setNSDP(nsdpInfo);
}

//sets up the dynamically created advanced style to
//be used by the theme-based FOI.
function setupDynamicStyles(dynStatesFOI, val1, val2, val3) {
    //create basic colors to be used
    var sc1 = new MVStyleColor("color1", "00FF00", "666666");
    var sc2 = new MVStyleColor("color2", "00FF00", "666666");
    var sc3 = new MVStyleColor("color3", "FFFF00", "666666");
    var sc4 = new MVStyleColor("color4", "FF0000", "666666");
    sc1.setFillOpacity(60);
    sc2.setFillOpacity(60);
    sc3.setFillOpacity(60);
    sc4.setFillOpacity(60);

    //create individual ranged buckets
    var buckets = new Array(4);
    buckets[0] = new MVNumericRangedBucket(-500, val1, "color1", "range1");
    buckets[1] = new MVNumericRangedBucket(val1, val2, "color2", "range2");
    buckets[2] = new MVNumericRangedBucket(val2, val3, "color3", "range3");
    buckets[3] = new MVNumericRangedBucket(val3, null, "color4", "range4");
    var bseries = new MVBucketSeries("SCHEME_CUSTOM");
bseries.setBuckets(buckets);
    var bucketSty = new MVBucketStyle("region_colors", bseries);

    //add all the primitive color styles we just created
dynStatesFOI.addStyle(sc1);
dynStatesFOI.addStyle(sc2);
dynStatesFOI.addStyle(sc3);
dynStatesFOI.addStyle(sc4);

    //now add the bucket style
dynStatesFOI.addStyle(bucketSty);

dynStatesFOI.setRenderingStyle("region_colors");
}
</script>
The resulting page in a browser, with states color coded by XML data attributes:

Mouse click on a state launches info box displaying attributes designed in Step 9 along with NSDP attributes from an XML source

We next describe the integration of Oracle Maps into OBIEE dashboards.

**Integration Scenarios for the Hands-on Lab session**

We will be using the dashboard named Mapping Examples for this session. The EPA Regions dashboard is a working example with all the relevant code integrated into a more elaborate, and usual, OBIEE Dashboard. Screenshots of every step in each of the four mapping examples described here are contained in the PowerPoint presentation file oow08-bi-mv-hol.pps in C:\OracleBI\oe4j_bij2ee\home\applications\mapviewer\web\mapping_examples\Workshop-Code. The code snippets used in this lab (interaction1-snippet1.txt, interaction1-snippet2.txt, interaction1-snippet3.txt, interaction2-snippet1.txt, interaction3-snippet1.txt, interaction4-snippet1.txt) are also in that directory.
Integration 1: Thematic map of an Answers result in Dashboard page

Integration 2: Navigate And Drill Based On Map Click
Interaction 3: Click map to drive Answers Request within current page

Interaction 4: Invoke a MapViewer function within a Dashboard page
Integration 1:  
**Mapviewer Page in Dashboard using Answers Results as NSDP**

**Method Summary**

* Custom jsp to convert Answers filtered XML into NSDP XML  
  (...\obiee_nsdp_xml_direct.jsp. Source code listed in Appendix.)
* Define Answers Request having columns for  
  - join to mapping data (i.e., state name)  
  - analytic value to be rendered (i.e., level of toxic release)
* Use Static Text in Answers to  
  - Define div for map  
  - Get active session’s nQID (authentication) and SID (result sid id)  
  - Launch Mapviewer page passing nQID and SID
* Mapviewer page calls JSP that converts Answers XML to NSDP XML (i.e.,  
  ps["xml_url"]=[jsp]) passing nQID and SID to get filtered XML data as NSDP XML  
* Key to method is passing nQID and SID avoiding dependency on presentation variables.  
* The Answers request that shows the map is same answers request that collects the data.

**Step 1: Define Answers request with having columns for join to MapViewer theme and analytic value to be rendered:**

A separate request is used for the report section of the dashboard. This is so that additional columns and result formatting may be applied without conflicting with the Mapviewer interaction.

Click on the Answers link in the Mapping Examples dashboard.
Then in Answers choose the TRI Data subject area.
Next expand the Dims for Facility and select the TRI Fac State Name dimension. Then expand the Facts for All Releases and choose Fugitive Total Release.

Now change column properties so that the computed value, percent change over prior year, is displayed instead of the Total Release. This involves changing the column formula. Click on Custom Heading. Cut and paste the code in interaction1-snippet1.txt into the column formula and edit the column heading.
Next change the column’s data format so that it displays percentages. Once that’s done add a filter (Reporting Year). Expand Dims for Date on the left and Ctrl-Click on Reporting Year. This opens the filter dialog. Set it to “is prompted”
Step 2: Enter code in Results > Static Text to define map div and invoke MapViewer page:
Now the report is defined add a view object for it on the dashboard. Click on the Results tab and cancel the report execution. Click on Add View and select Static Text. The map image will be displayed in this section of the page. Navigate to the Workshop-Code folder once again. Open interaction1-snippet2.txt and copy and paste the code from that file into the Static Text view definition. Click on the Contains HTML Markup checkbox and Display Results.
example code entered in Static Text

<!--div id must match arg in js call at bottom of code listing -->

<div id="EPAmapNode1"></div>

<script>
// put everything inside a function to separate variable namespace

function readCookie(name) {
  // this pulls authentication out of header
  // which replaces needing to pass user/pwd to authenticate
  // ie, like SSO or trusted sign on
  var nameEQ = name + "=";
  var ca = document.cookie.split(";");
  for(var i=0;i < ca.length;i++) {
    var c = ca[i];
    while (c.charAt(0)==' ') c = c.substring(1,c.length);
    if (c.indexOf(nameEQ) == 0) return c.substring(nameEQ.length,c.length);
  }
  return null;
}

function obiee_mapint_doTheDeed(nodeId) {
  // accepts name of div
  // answers assigned unique ID to result set assoc w/ answers request (called SID)
  // need nQuid/SID to grab results
  // ie look at page source for results, will show SID
  // mechanism to get at SID new in 10.1.3
  // developed for Office integration, we're piggybacking
  var container = document.getElementById(nodeId);
  var sid = null;
  Code to capture nQID and SID, and pass them to MapViewer code
  }

</script>
Save the request (Click on the Floppy icon) in Shared Folders: EPA - TRI Dashboard: Mapping Examples as States Map Request.
Step 3: Add request to dashboard page and define setupNSDP function in MapViewer code to retrieve report XML as NSDP XML:

Now add a section on the dashboard page to display the map. Click on Dashboards and then Page Options->Edit Dashboard. Click on the + sign next to Add a Column then on the Shared Folders, EPA TRI Dashboard, Mapping Examples in the left panel. Drag and drop the States Map Request into the new column.
Then Save.

Now add the setupNsdp() function to the MapViewer code. The relevant code snippet is in interactio1-snippet3.txt. It has to be copied and pasted into states.html, which is in the mapping_examples folder that is one level up from the Workshop-Code folder. Once that is done uncomment the line that invokes this function. Save the states.html file.
function showMap() {
    ...
    dynStatesFOI = new MVThemeBasedFOI('dynStatesFOI', 'EPA.STATES');
    setupNsdp(dynStatesFOI);
    ...
}

function setupNsdp(dynStatesFOI) {
    //default NSDP is always available with MapViewer 10.1.3.1 production
    var nsdpInfo = new MVNSDP("defaultNSDP");
    //the base theme
    nsdpInfo.setTheme("STATES");
    //join column in the base table of the theme
    nsdpInfo.setKeyColumn("STATE_NAME");
    //default style
    nsdpInfo.setRenderStyle("C.ACTIVE");
    //NSDP xml source is call to jsp that invokes java class to convert Answers filtered
    //XML into NSDP XML
    var ps = new Object();
    ps["xml_url"] = "http://"+document.location.host+"/mapviewer/_epa_maps/obiee_nsdp_xml_direct.jsp?nqid=" +
                     urlParamNQID + "&sid=" + urlParamSID;
    nsdpInfo.setParameters(ps);
    //join the BI data with the theme
    dynStatesFOI.setNSDP(nsdpInfo);
}

Result: Dashboard page with States theme color-coded by “Change From Prior Yr”

Integration 2: Navigate And Drill Based On Map Click

Method:
* Invoke the process using MapViewer FOI on-click event listener
* Event listener function
  - captures attributes of FOI that was clicked
  - launches portalNav or portalPageNav in parent (dashboard) passing parameters
* In target page/dashboard, define Answers Request having
  - Prompt(s) for accepting parameter(s) (i.e., State name)
  - Columns and Static Text for NSDP mapping as described in previous section

The description below is for Page navigation using PortalPageNav. Dashboard navigation would be accomplished in similar manner using PortalNav.
Step 1. Define event listener function in Mapviewer to launch portalPageNav (or portalNav)

Open up states.html and uncomment the line that adds a mouse click event listener. Then copy and paste the event handler code from interaction2-snippte1.txt, that is, add the drillDownClick() function. Then save the file and reload the dashboard page.

```javascript
//Mapviewer code in source Dashboard Page
function showMap() { ...
  dynStatesFOI = new MVThemeBasedFOI('dynStatesFOI', 'EPA.STATES');
  ...
  //add event listener to handle FOI click
  dynStatesFOI.addEventListener("mouse_click", drilldownClick);
  ...
}
function drilldownClick(point, foi, evt) {
  ...
  //launch portalPageNav in parent dashboard page
  ...
}

Where
'/shared/EPA = TRI Dashboard/_portal/EPA States = Dashboard
'Dims For Facility' = Presentation table that includes pres field for filtering
'TRI Fac Region' = Presentation field to be used as subject for dashboard filter using foi attr
foi.attrs[2] = FOI attribute to use for filter

The next two steps (2 and 3) are not to be done in this workshop. They are similar to the ones done in the first integration scenario. They are shown here just for reference.

**Step 2:** Define Answers Request having Mapviewer join and analytic value fields, and Prompt to filter by PortalPageNav parameter

**Step 3:** Define NSDP Mapping in target Page
Once the target page accepts the filter value through PortalPageNav, the approach is the same as described above (`Mapviewer Page in Dashboard using Answers Results as NSDP`).
Result: Dashboard page with States theme color-coded by “Change From Prior Yr”

Clicking on state launches PortalPageNav

Resulting page at based on prompted filter passed from PortalPageNav
**Interaction 3: Click map to drive Answers Request within current page**

**Method:**
* MapViewer FOI event listener function
  - captures attributes of FOI that was clicked
  - launches Go URL to open Answers Request in Dashboard passing parameters
* Define placeholder Answers request that defines an iFrame to accept goURL from map click

---

**Step 1: Define Answers Request to be launched by Go URL**
In this case, we wish to click on a facility in the map and drill into facility details within the current Dashboard Page.
Step 2: Define Answers Request that defines iFrame for Go URL results

Copy and paste the text in interaction3-snippet1.txt into the Narrative element.
Save the changes.
Next modify the Facility Detail Request to add the facility id as a report filter. Click on Answers then in the Dashboards panel expand Shared Folders: EPA TRI Dashboard: Mapping Examples: Facility Detail Request.

Step 3: Define MapViewer NSDP function to launch Go URL
Open the sites.html file (in the mapping_examples folder which contains the Workshop-Code folder) and uncomment the code to register the mouse click event listener.
Code example:

```javascript
function showMap() {
  ...
  dynSitesFOI.addEventListener("mouse_click", drilldownClick);
  ...
}

function drilldownClick(point, foi, evt) {
  ...
  /* Define the Go URL, where
  urlParamNQID = nQID passed as url parameter and stored as variable.
  /shared/EPA - TRI Dashboard/Mapping Examples/Facility Detail Request = Answers Request.
  Dims For Facility = Logical Table
  TRI Facility Id = Logical Column (Facility ID in OBIEE Logical model)
  foi.attrs[1] = FOI attribute (Facility ID in MapViewer FOI)
  */
  var drillURL = "http://"+document.location.host+"/analytics/saw.dll?Go&Path=/shared/EPA - TRI Dashboard/Mapping Examples/Facility Detail Request&NQId=" + urlParamNQID + "&Action=Navigate&P0=1&P1=eq&P2=""Dims%20For%20Facility"",&P3=""TRI%20Facility%20Id"",&P4=""foi.attrs[1]"
  //Launch Go URL in parent dashboard and place the results in iFrame named “details”
  //where the iFrame called “details” is defined in a placeholder Answers request
  objWin = window.open(drillURL, "details", "height=600,width=240,scrollbars=yes,resizeable=yes");
}

Note: The Go URL methodology is documented in Chapter 11 of the OBIEE Presentation Services Administration Guide (Version 10.1.3.2)
```
Result: Click on FOI in map drives Answers results in page

Pin mapping function described below
**Interaction 4:**
Invoke MapViewer Javascript function from within Dashboard page

**Method:**
* Define desired Javascript function in MapViewer page
* In Dashboard, Map is contained in an iFrame
  (as described in “Integration 1: Mapviewer Page in Dashboard using Answers Results as NSDP”)
* Set *name* parameter for map iFrame
* Select a Answers Request column that will be used to invoke the MapViewer function
* In Answers, place code in Custom Text Format of Column Properties to invoke iFrame (map) Javascript

<table>
<thead>
<tr>
<th>Dashboard Page</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Column</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Custom Text Format</strong></td>
<td>onclick=&quot;JavaScript:top.frames... @ column value&quot;</td>
</tr>
</tbody>
</table>
| **Map iFrame** | - name param
- Javascript function |
Step1: Define desired MapViewer Javascript function

In this case, our desired function is to create a dynamic pin FOI for the facility clicked in a dashboard table.
We define a theme EPA.SITES_DYNAMIC_BY_ID using a bind variable:

This allows us to define siteID at run time based on value passed from the Dashboard. This method is described in the MapViewer documentation. We then define the following function to support pin mapping for a given facility base on siteID (facility identifier):

```javascript
//function to add pin FOI for site selected in dashboard table
//siteID is the facility ID passed by the Answers request via the top.frames method
// of invoking Javascript in iFrame from parent
function sitePin(siteID) {
    if (mapview.getThemeBasedFOI("sitePinFOI")) {
        mapview.removeThemeBasedFOI(sitePinFOI);
    }
    sitePinFOI = new MVThemeBasedFOI('sitePinFOI','EPA.SITES_DYNAMIC_BY_ID');
    sitePinFOI.setQueryParameters(siteID); //populates bind variable in theme defn
    mapview.addThemeBasedFOI(sitePinFOI);
}
```

The MapViewer code next needs to be invoked from the parent Dashboard.
Step 2: Place map in iFrame with defined name parameter

In the case of map using NSDP
As described above in the section “Integration 1: Mapviewer Page in Dashboard using Answers Results as NSDP”, the code to instantiate the map is located in the Static Text view of Results. The iFrame name parameter should be defined as follows:

```javascript
// create the iframe for content
var nid = readCookie('nQuireID');
if (nid) {
    var iframe = document.createElement('iframe');
    iframe.src = 'http://' + document.location.host + '/mapviewer/mapping_examples/sites.html?sid=' + encodeURIComponent(sid) + '&nqid=' + nid;
    iframe.height = 260;
    iframe.width = 600;
    iframe.frameBorder = 0;
    iframe.marginHeight = 1;
    iframe.marginWidth = 1;
    iframe.name='mapIframe';
}
```

In the case of a simple map not using NSDP
The code to instantiate the map can be placed in the Narrative view of Results. The iFrame name parameter should be defined as follows:

```html
<iframe src="/mapviewer/_epa/Sites.html" name="mapIframe" height="460" width="520" frameborder="0"
    marginheight=1 marginwidth=1>
</iframe>
```
Step 3: Place code to invoke MapViewer Javascript in Custom Text Format of Answers column that will drive the function

In this case we select TRI Facility ID as the Answers column that, when clicked, will drive the MapViewer function sitePin and create a dynamic pin FOI at the selected facility. We will pass the selected TRI Facility ID value to our MapViewer function, and that value will be used to populate the bind variable for our dynamic pin theme.

* Edit the Criteria >> Column Properties for the column we want to click to launch map function (in this case, TRI Facility ID)
* Navigate to Data Format and select Custom Data Format
* Check Override Default data Format and select Custom Text Format (Column Properties >> Data Format >> Override Default >> Custom Data Format)
* Enter code to invoke map iframe javascript using top.frames method, for example:

For example (interaction4-snippet1.txt),

```
[@html]<a href="javascript:void(null)" onclick="JavaScript:top.frames['mapIframe'].sitePin('@')">@</a>
```

Where

* `sitePin` = name of maps javascript function
* `mapIframe` = name of maps iframe
* `@` = placeholder for values coming from database via the logical model

Clicking column value invokes Javascript in map iframe
Result: Clicking value in column launches Javascript function in map iFrame

In this case, the Javascript function, sitePin(), places a pin FOI on the site clicked on in the table report. In addition, we have also added code to launch the Go URL in the sitePin() function, so that clicking the table report column launches the sitePin() function which in turn launches the Go URL to update the details result (as described above in section “Interaction 3: Click map to drive Answers Request within current page”).
Appendix

Installing and configuring MapViewer in oc4j_bi

Download the MapViewer 10.1.3.1 kit (mapviewer.ear) from otn.oracle.com/products/mapviewer/index.html. This should be place in the bi server's oc4j. That is in $BI_HOME/oc4j_bi/j2ee/home/applications.

If you have a Metalink account you should get the latest patch for MapViewer and use that version instead. Log on to your account on metalink.oracle.com. Search for Patch number 7195504. Download it and unzip. Use the mapviewer.ear file which is part of that kit.

A few config files in oc4j_bi/j2ee/home/config/ have to be updated. They are server.xml and default-web-site.xml. Add the following lines in those files.

default-web-site.xml:
<web-app application="mapviewer" name="web" load-on-startup="true" root="/mapviewer" />

server.xml:
<application name="mapviewer" path="../applications/mapviewer.ear" parent="default" start="true" />

Restart the oc4j. MapViewer will be deployed.

If you intend to use the NSDP XML format to get an Answers report in xml format and then join it with map data in the mid-tier, via MapViewer’s NSDP join feature, then you need obiee_nsdp_xml._direct.jsp. This must be placed in the appropriate directory of the MapViewer installation, e.g. j2ee/home/applications/mapviewer/web/_epa_maps/

Setting up MapViewer datasources

Once MapViewer is deployed modify MapViewer config file oc4j_bi/j2ee/home/applications/mapviewer/web/WEB-INF/conf/mapViewerConfig.xml to add map data sources.

This is done either by inserting these lines at the end of the config file using any text editor,

<map_data_source name="datasource_name"
    jdbc_host="database_mc.foo.bar "
    jdbc_sid="oracle_sid"
    jdbc_port="port"
    jdbc_user="username"
    jdbc_password="!password"
    jdbc_mode="thin"
    number_of_mappers="3"
    allow_jdbc_theme_based_foi="true"
or from MapViewer’s Admin console.

It is available at host:port/mapviewer/faces/home.jspx once MapViewer has been deployed and initialized on oc4j startup. Click on the Admin button on the top right of the page. Login using the OC4J admin username and password. Then click on Datasources link that is a menu item under the “Manage MapViewer” menu tab on the left. This lets you add a datasource for the current oc4j and MapViewer session. It will not persist and won’t be available if oc4j is restarted.

Click on the Configuration link instead to edit and add a datasource to the mapViewerConfig.xml file. Add the data source definition in the relevant section of the config file. Click on Save or Save & Restart.

Some hints on debugging
MapViewer outputs its error messages to the oc4j console and the log file specified in mapViewerConfig.xml. The default location of the logfile is $OC4J_HOME/applications/mapviewer/web/WEB-INF/log. Usually the error message on the oc4j console provides enough of a hint to resolve the issue. For example, it states that a map data source was not found, or that a particular theme could not be loaded. Sometimes however further details may be necessary, particularly when the map output is incomplete (basemap was rendered but a dynamic theme is missing), or different (roads displayed as thin black lines instead of 3 pixel wide blue lines). In such cases it may help to set MapViewer’s logging level to Debug, or Finest, in mapViewerConfig.xml. Remember to restart MapViewer after changing the config file.

Source code for obiee_nsdpxml_direct.jsp.
Note: This is sample code and is provided solely for the purpose of this tutorial.

```java
<%@ page language="java" session="true" isThreadSafe="true" contentType="text/html; charset=UTF-8" %>

<%@ page import="javax.xml.parsers.*" %>
<%@ page import="java.net.*" %>
<%@ page import="java.io.*" %>
<%@ page import="org.w3c.dom.*" %>

<% // Sample jsp code file to demonstrate one way to convert Answers xml to // MapViewer’s NSDP xml
    // set the character encoding to UTF-8 prior to accessing // data in the request request.setCharacterEncoding("UTF-8"); %>

    // RETRIEVE THE QUERY STRING PARAMETERS
    String userName = request.getParameter("uid");
    String userPassword = request.getParameter("password");
    String thesid = request.getParameter("sid");
    String nq_id = request.getParameter("nqid");

    if (userName == null) {
        userName = "Administrator";
    }
    if (userPassword == null) {
        userPassword = "Test";
    }

    Document domDoc = null;
```
String fieldName = null;
String urlString = "http://" + request.getRemoteHost() + ":9704/analytics/saw.dll?Go&searchid=" + thesid + "&format=xml&NQId=" + nq_id;

if (thesid != null) {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    URL url = new URL(urlString);
    URLConnection URLconnection = url.openConnection();
    HttpURLConnection httpConnection = (HttpURLConnection)URLconnection;
    int responseCode = httpConnection.getResponseCode();

    if (responseCode == HttpURLConnection.HTTP_OK) {
        InputStream in = httpConnection.getInputStream();
        System.out.println("thesid= " + thesid);
        System.out.println("nq_id=" + nq_id);
        System.out.println("server host=" + request.getRemoteHost());
        System.out.println("server addr=" + request.getRemoteAddr());
        domDoc = db.parse(in);
    } else {
        System.out.println( "HTTP connection response != HTTP_OK" );
    }

    out.println("<nsdp_xml>");
    out.println("<table>");
    //Iterate through metadata object and create the ndsp_xml header
    NodeList rowDef = domDoc.getElementsByTagName("element");
    out.println("<tr>");
    out.println("<th>Column 1</th><th>Column 2</th>");
    out.println("</tr>");
    //Iterate through rows & items and create the ndsp_xml body
    NodeList rows = domDoc.getElementsByTagName("R");
    if (rows != null) {
        for (int i = 0; i < rows.getLength(); i++) {
            out.println("<tr>");
            Node row = rows.item(i);
            if (row != null || row.getNodeType() == Node.TEXT_NODE) { continue; }
            NodeList items = row.getChildNodes();
            if (items == null) { continue; }
            for (int y = 0; y < 2; y++) {
                Node item = items.item(y);
                if (item == null || item.getNodeType() == Node.TEXT_NODE) { continue; }
                if (item.getNodeType() == Node.TEXT_NODE)
                    continue;
                out.println("<td>" +
                    item.getTextContent() + "</td>");
            }
            out.println("</tr>");
        } else {
            out.println("<tr>");
            out.println("<td>No rows.</td>");
            out.println("</tr>");
    }
    out.println("</table>");
}
"