RBDMS Data Mining Configuration version 2.0

Coordinate solutions and Virtual Engineering Solutions, Inc.
www.VirtualES.com

Table of Contents

Introduction

RBDMS Data Mining 2.0 provides users the ability to search, review, and visualize RBDMS data via a Web browser. With the 2.0 version of RBDMS Data Mining, the architecture is completely redesigned by utilizing the latest technologies to provide the desired functionality. The application also relies on specific implementations of several base RBDMS.NET projects, like RbdmsBase, RbdmsConfig, RbdmsWebBase and RbdmsWebControls.

RBDMS Data Mining is a presentation layer written in HTML/JavaScript and ASP.NET WEB API. The server side logic is implemented by a set of controllers/models which provide the results in JSON/xml to the JavaScript client.

The map rendering functionality on the client side is implemented by using the high-performance and feature rich OpenLayers JS library, while the map data is served by an OGC compliant server (GeoServer 2.4.4 is used in our test environment) The server must provide OGC compliant implementation of a number of open standards such as Web Feature Service (WFS), Web Map Service (WMS), and Web Coverage Service (WCS).

RBDMS Data Mining 2.0 provides the option to export the configured map in print documents which is implemented by using the MapFish print services. The print server is installed along with GeoServer and use the same java environment at the server side.

The reporting option of RBDMS Data Mining 2.0 is provided by a set of RDLC and XML based reports in an intuitive user interface powered by the jQuery UI. To specify the query criteria, we use the same filtering methodology as in the advanced search tab, which is also the same filter methodology used by RBDMS.Net. There are no required changes to filter collections or reports (XML or RDLC).

In RBDMS Data Mining 2.0 the full text (simple) and advanced search has been redesigned which is no longer based on Nodes.xml, instead we use pre-configured javascript variables (searchNodes and searchNodesAdvanced) to initiate the query from the client side and interact with the controllers and the models to complete a search.

The implementation of the application is divided into common and state specific parts. Regarding to the application configuration, we continue to use the state specific map config file (GWPC_MS.xml in our test environment) with a couple of minor changes described later in this document.

The remainder of this document discusses the applied technologies in more detail and describes the file structure, contents of the configuration files, and other requirements for their use.

Web Map Service (WMS) Configuration (Geoserver)

With RBDMS Data Mining 2.0 we use a GeoServer backend to provide web mapping data for the application. GeoServer is installed as a service application which starts with the OS automatically. We usually configure GeoServer to listen on port 8080, and we should also configure the firewall settings to allow propagating the request to this specific port. Once the Geoserver installation is complete we can enter to the admin page using this url on the server:

http://localhost:8080/geoserver/web

Layer configuration

When configuring the layers in GeoServer we should set up a data source first. We can use either file based data sources (like shapefiles) or database stores (like PostGIS or MS SQL Server). In the latter case the MS SQL driver extension should also be installed with GeoServer.

When adding a new layer we should select a configured data source, specify the common parameters NAME, SRS, EXTENT and configure the layer style by selecting one of the preconfigured styles.

Style configuration

The layer styles can be specified by using an SLD (XML) definition. This configuration provides the option to specify the style or symbols for the features and specify the logical conditions when a specific style of symbol should be applied. For an example see please refer to Appendix I: Styled Layer Descriptor (SLD) example.

Tile caching

The tile caching option in GeoServer is provided by GeoWebCache which can be installed with GeoServer (bundled mode) or in standalone mode. We configure the layers to allow tile caching, which provides the option to store the generated tiles in the file system and then serve the tile requests from this cache when requested in a subsequent map rendering operation. This configuration increases the rendering speed especially for the wells layer, where the number of features is high. The tile cache for the layer can be

reseeded in the GeoServer admin page, which ensures that all the tiles for the specified extent and zoom levels are pre-generated.
When the underlying data store is changing we need to regenerate/reseed the cache manually or by using an automated script.

GeoServer services

GeoServer provides multiple OGC sevices which can be accessed by the HTML/JS client

The web mapping service (WMS) providing map images can be accessed according to the following example:

http://localhost:8080/geoserver/wms

The tile based WMS service (provided by GeoWebCache) can be accessed by:

http://localhost:8080/geoserver/gwc/service/wms

When doing spatial queries, we use web feature service (WFS) provided by GeoServer using this URL:

http://localhost:8080/geoserver/wfs

The external URL-s to GeoServer should be set in the wmsURL and wfsURL parameter of MSOGBOnline.html, in the OGCServer and OGCServerURL of RBDMSGIS_HTML_OL.js and the WFS URL should also be specified in the pUrl setting of proxyReverse.ashx

GIS configuration file

The structure on the GIS configuration (GWPC_MS.xml) file has remained, but a couple of changes made to support the new OGC-based data store. The GIS configuration is used for setting up the layers on the map, the GIS tools and the mapping and printing defaults.

Specifying the map service

In the MapServices section for each MapService entry we need to specify the service type (GS), the server (GeoServer base URL), the port number (8080), the GeoServer workspace and the relative path to access the map service. If we don't specify the workspace section in the MapService node we need to prefix the layer names with the workspace (such as: MS_SHPS:Wells). In this case multiple workspaces can be used in the same map configuration.

```
<MapService>
  <ID>0</ID>
  <ServiceType>GS</ServiceType>
  <Server>http://csgeoserver.coordinatesolutions.com</Server>
  <Port>8080</Port>
  <Workspace>MS_SHPS</Workspace>
  <MapService>/geoserver/gwc/service/wms/</MapService>
</MapService>
```

Specifying the layers

Layers are configured within the Toc section the mapservice is selected according to the MapIndex entry of the Layer node. The ID of the layer corresponds to the layer specified in GeoServer.

```
<Layer>
<Name>Wells</Name>
<MapIndex>0</MapIndex>
<ID>WELLS</ID>
<LayerType>MS</LayerType>
<EntityKeyName>PKEY</EntityKeyName>
<ToolTip>
  <![CDATA[
   <table>
    <tr>
      <td>API</td>
      <td><a href="javascript:window.parent.parent.FillEDOnly('PKey',!-
PKEY-
!,'Integer','WellDetails.xml','ctl00_PageBody_WebPartManager1_gwpPanelD
etails_DetailsFrame');">!-API-!</a></td>
     </tr>
```

```
      </table>
    ]]>
  </ToolTip>
</Layer>
```

The ToolTip section above is also used by the application, which specifies the HTML content to be displayed on the map, when the cursor is dragged over a feature. Currently only one ToolTip entry is taken into account.

GIS tools

The GIS tools can be specified in the TooBar section of the GIS config file to determine which tools should be displayed with the map. The buttons are powered by jQuery UI, and the image can be configured by CSS setting (in styles/default.css) according to the following example:

```
.ui-icon-xy
{
      background-image: url('../images/xy.png') !important;
}
```

The the css enty can be selected via the ImagePath attribute, like:

```
<GISTool>
        <Name>CenterXY</Name>
        <Title>Center by X,Y or Latitude, Longitude</Title>
        <Type>CenterXY</Type>
        <EventType>Dialog</EventType>
        <ToolTip>Center by X,Y or Latitude, Longitude</ToolTip>
        <ApplyTool>true</ApplyTool>
        <ImagePath>ui-icon-xy</ImagePath>
        <HTMLInputNames>
          <string>longlat|+proj=longlat +ellps=GRS80 +no_defs</string>
          <string>MSTM|+proj=tmerc +lat_0=32.500000 +lon_0=-89.750000
+x_0=500000.000000 +y_0=1300000.000000 +datum=NAD83 +ellps=GRS80
+no_defs</string>
        </HTMLInputNames>
      </GISTool>
```

Print options

When printing the map the print options are taken from the PrintOptions node in the GIS config file. Not all of the print options can be specified externally some of the settings should be added to the MapFish print configuration file. Currently the following parameters used:

```
<PrintOption>
  <PrintSize>8.5 X 11 Landscape</PrintSize>
```

```
  <ImageHeight>437</ImageHeight>
  <ImageWidth>678</ImageWidth>
  <ImageOffsetX>57</ImageOffsetX>
  <ImageOffsetY>570</ImageOffsetY>
  <PDFTemplate>landscape.pdf</PDFTemplate>
</PrintOption>
```

Configuring printing

Printing is implemented by utilizing the MapFish print server. This server is started along with GeoServer by adding the print module to the /webapps section of the GeoServer installation. When the print server is properly installed, we get the print test page in the browser by using the URL:

http://localhost:8080/print/

The external URL to the print server should be set in the printURL parameter of MSOGBOnline.html.

The print config file

The print server is configured by the config yaml file in the webapps/print directory of the GeoServer installation. In the print configuration file we can specify the layouts for the printing.

Currently we use 2 layouts: a landscape and a portrait version. Each layouts have 2 variants whether to print the legend or not in the print output.

An example of the portrait layout definition is shown below:

```
#==========================================================================
  8.5 X 11 Portrait:
#==========================================================================
    mainPage:
      rotation: true
      pageSize: LETTER
      backgroundPdf: '${configDir}/${pdfTemplate}'
      items:
        - !map
          spacingAfter: 0
          width: '${imageWidth}'
          height: '${imageHeight}'
          absoluteX: '${imageOffsetX}'
          absoluteY: '${imageOffsetY}'
        - !columns
          absoluteX: 45
          absoluteY: 70
          width: 200
          items:
            - !text
              text: '${scaleText}'
              fontSize: 8
              spacingAfter: 0
        - !columns
          absoluteX: 45
```

```
absoluteY: 60
width: 200
items:
  - !text
    text: '${now MM/dd/yyyy}'
    fontSize: 8
    spacingAfter: 0
```

We can use template parameters in the layout which are specified in the post request to the print server. These are those parameters which can be configured in the GIS config file according to the previous chapter.

The date and the scale section in the print template is specified by using the column sections. At the moment these sections cannot accept template parameters, therefore the offset and the width of these sections should also be configured in config.yaml instead of the GIS config file.

Search configuration

In DataMining 2.0 the search is no longer based on Nodes.xml, instead we use search nodes. The search nodes are JavaScript configuration elements to control the text based and the spatial based queries. We maintain different set of nodes for the full text (simple) and the advanced search according to the following declarations:

```
var searchNodes = {
            entities: [
                {
                    controller: "Well",
                    header: "Wells",
                    keyname: "PKey",
                    keytype: "Integer",
                    dispfield: "DispName",
                    detailxml: "WellDetails.xml",
                    container: {}
                }, {
                    controller: "Entity",
                    header: "Oil and Gas Operators",
                    keyname: "PKey",
                    keytype: "Integer",
                    detailxml: "EntityDetails.xml",
                    container: {}
                }, {
                    controller: "Pool",
                    header: "Pools",
                    keyname: "PoolNumber",
                    keytype: "Integer",
                    detailxml: "PoolDetails.xml",
                    container: {}
                }, {
                    controller: "Field",
                    header: "Fields",
                    keyname: "FieldNumber",
                    keytype: "Integer",
                    detailxml: "FieldDetails.xml",
                    container: {}
                }, {
                    controller: "County",
                    header: "Counties",
                    keyname: "CountyNo",
                    keytype: "Integer",
                    detailxml: "CountyDetails.xml",
                    container: {}
                }, {
                    controller: "Permit",
                    header: "Permits",
                    keyname: "PKey",
                    keytype: "Integer",
                    detailxml: "PermitDetails.xml",
                    container: {}
                }
            ]
        }
```

```
var searchNodesAdvanced = {
            entities: [
                {
                    controller: "Well",
                    header: "Wells",
                    keyname: "PKey",
                    keytype: "Integer",
                    detailxml: "WellDetails.xml",
                    container: {}
                }
            ]
        }
```

Each node is mapped to a server side controller (by the controller parameter) which is used to retrieve data for the corresponding search. We can specify additional parameters for the search and for the ASP.NET page (ED.aspx) which displays the result in the Details section of the user interface. For more information about the search controllers see ASP.NET Web API controllers and models.

ASP.NET Web API controllers and models

The server side of the DataMining 2.0 application is implemented by utilizing the ASP.NET Web API technologies. ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.

Routing and controllers in ASP.NET Web API

In ASP.NET Web API, a *controller* is a class that handles HTTP requests. The public methods of the controller are called *action methods* or simply *actions*. When the Web API framework receives a request, it routes the request to an action.

To determine which action to invoke, the framework uses a *routing table*. The Visual Studio project template for Web API creates a default route:

```
config.Routes.MapHttpRoute(
    name:="DefaultApi",
    routeTemplate:="api/{controller}/{id}",
    defaults:=New With {.id = RouteParameter.Optional}
)
```

Each entry in the routing table contains a route template. The default route template for Web API is "api/{controller}/{id}". In this template, "api" is a literal path segment, and {controller} and {id} are placeholder variables. When the Web API framework receives an HTTP request, it tries to match the URI against one of the route templates in the routing table. If no route matches, the client receives a 404 error. For example, the following URIs match the default route:

/api/contacts

/api/contacts/1

/api/products/gizmo1

However, the following URI does not match, because it lacks the "api" segment:

/contacts/1

Once a matching route is found, Web API selects the controller and the action:

- To find the controller, Web API adds "Controller" to the value of the {controller} variable.
- To find the action, Web API looks at the HTTP method, and then looks for an action whose name begins with that HTTP method name. For example, with a GET request, Web API looks for an action that starts with "Get...", such as "GetContact" or "GetAllContacts". This convention applies only to GET, POST, PUT, and DELETE methods. You can enable other HTTP methods by using attributes on your controller.
- Other placeholder variables in the route template, such as {id}, are mapped to action parameters.

In the following example we specify the Well controller as follows:

```
Public Class WellController
    Inherits ApiController

    Public Function GetAction(<FromUri> search As String) As
IEnumerable(Of SearchResultBase)
        Return Well.Find(search)
    End Function

    Public Function PostAction(<FromBody> f As Filter) As
IEnumerable(Of SearchResultBase)
        Return Well.FindAdvanced(f)
    End Function

End Class
```

The GET requests to the /api/Well route is mapped to the GetAction method, while the POST requests to /api/Well is mapped to the PostAction method. We could also invoke more specific action methods such as /api/Well/id which could be used to retrieve data for a specific item mapped to GetWell(id).

Full text (simple) search routing

In case of the full text (simple) search we map the search nodes to the controller according to the following rule:

/api/[controller]/?search=[search]

where [contoller] is the controller name specified in searchNodes (like Well, County, Pool etc.) and [search] is the text entered on the simple search tab. When performing the search, we initiate the query for each node simultaneously and the search tree is updated with the receive data

immediately. For more detailed information about the simple search refer to the implementation of the search() JavaScript method.

Advanced search routing

In case of the advanced search we retrieve the filter configuration first, by using the following route:

/api/Filter/?collName=DataMining

The controller returns a filter structure according to the model implemented in Filter.vb, which is stored for later use.
Then we fill up an HTML table on the advanced search tab with the filter controls and start retrieving the combo items using the Lookup controller using the route:

/api/Lookup/?lookupName=[lookup name]

When performing the search, the filter structure is updated with the selected values and then sent to the controller as a post data. The controller is specified in the searchNodesAdvanced configuration and we use the following route:

/api/[controller]

Currently only the Well controller is configured to implement the advanced search option.

Models in ASP.NET Web API

A *model* is an object that represents the data in our application. ASP.NET Web API can automatically serialize the model to JSON, XML, or some other format, and then write the serialized data into the body of the HTTP response message. As long as a client can read the serialization format, it can deserialize the object. Most clients can parse either XML or JSON. Moreover, the client can indicate which format it wants by setting the Accept header in the HTTP request message.

Controllers and Models implemented in DataMining 2.0

As we already mentioned the search functionality in DataMining 2.0 is implemented by a set of controllers and the name of the controllers are configured in the search nodes. In this chapter we discuss the implementation of each controller in more detail.

CountyController

The CountyController is used in the simple search and the data is returned as a County model (implemented in County.vb). The controller executes the following database query to retrieve the search results from the database:

```
select distinct c.countyno , c.countyname from RefCounty c where
c.countyname like @Search
```

Where @Search is the specified query text on the simple search tab.

EntityController

The EnityController is used in the simple search and the data is returned as an Entity model (implemented in Entity.vb). The controller executes the following database query to retrieve the search results from the database:

```
select e.pkey , upper(coalesce(e.entityname,'')) + case
isnull(e.firstname, '') when '' then '' else ', ' + upper(e.firstname)
end from entity e inner join entityaddress ea on e.pkey=ea.entitykey
where ea.role='OGO' and       upper(coalesce(e.entityname,'')) + case
isnull(e.firstname, '') when '' then '' else ', ' + upper(e.firstname)
end like @Search order by e.entityname
```

Where @Search is the specified query text on the simple search tab.

FieldController

The FieldController is used in the simple search and the data is returned as a Field model (implemented in Field.vb). The controller executes the following database query to retrieve the search results from the database:

```
select e.fieldnumber, e.fieldname from reffields e where e.fieldname
like @Search order by e.fieldname
```

Where @Search is the specified query text on the simple search tab.

FilterController

The FilterController returns a filter structure according to the model implemented in Filter.vb. This structure is used to fill the table on the advanced search tab with the filter controls.

Since we can maintain multiple filter collections in the database, when invoking the GET action, we should also specify the collection to be used, and the id of the collection is retrieved from the database as follows:

```
select top 1 collid from coll where collname=@CollName
```

Where @CollName is the specified collection name, currently `@CollName = 'DataMining'` is used.

The filter controls are retrieved by using the following query:

```
select cdc.seq, dc.* from devcontrols dc  inner join colldevcontrols
cdc on dc.controlname=cdc.controlname where cdc.collid = @CollId order
by cdc.seq
```

When executing the advanced search, the filter structure is filled with the selected parameters and the entire structure is passed to the controllers specified in searchNodesAdvanced using POST requests. Currently only the WellController implements the advanced search option.

GISConfigController

The GISConfigController is used to dispatch the GIS config data to the JavaScript client. The state dependent location of the GIS Config file is specified in application web.config file by using the GISConfigFile parameter. The GIS config file is deserialized into the GIS model (implemented in GIS.vb) and then the data is returned in JSONJSON format to the JavaScript client. The controller is called from the setupConfig javascript function right before initializing the OpenLayers map configuration. For more information about the map configuration see Map Configuration and spatial queries.

LookupController

The LookupContoller is used to retrieve the data for the selection lists on the advanced search tab. The following GET request is called for each list control by specifying the control name as the parameter:

api/Lookup/?lookupName=[control name]

The SQL query of the selectable values for each control is retrieved from the devcontrols table as:

```
select top 1 SQL, DisplayField, ValueField from devcontrols where
controlname=@LookupName
```

And then the retuned SQL query is executed and the results are returned to the caller by using the Lookup model (implemented in Lookup.vb)

PermitController

The PermitController is used in the simple search and the data is returned as a Permit model (implemented in Permit.vb). The controller executes the following database query to retrieve the search results from the database:

```
select e.pkey, e.permit from permit e where e.permit like @Search order
by e.permit
```

Where @Search is the specified query text on the simple search tab.

PoolController

The PoolController is used in the simple search and the data is returned as a Pool model (implemented in Pool.vb). The controller executes the following database query to retrieve the search results from the database:

```
select e.poolnumber, e.poolname from refpool e where e.poolname like
@Search order by e.poolname
```

Where @Search is the specified query text on the simple search tab.

ReportTreeProviderController

The ReportTreeProviderController is a server component to provide data for filling the report tree in the DataMining reporting page. We can store multiple report configurations in the database and the current configuration is selected according to the RbdmsProfileProviderApplicationName and RbdmsSiteMapReportsMenuName specified in the web.config file, for example:

```
<appSettings>
    <add key="RbdmsProfileProviderApplicationName"
value="MSRBDMS.NET"/>
    <add key="RbdmsSiteMapReportsMenuName" value="ReportsDataMining"/>
  </appSettings>
```

The current XML configuration for the report tree is selected using the following database query:

```
SELECT Menus.XML FROM Menus INNER JOIN aspnet_Applications ON
Menus.ApplicationID = aspnet_Applications.ApplicationId WHERE
aspnet_Applications.LoweredApplicationName = LOWER(@ApplicationName)
AND Menus.MenuName = @MenuName
```

The retrieved XML is returned to the JavaScript client in JSONJSON format which is used to fill the jstree on the reporting page. For more information about the reporting configuration refer to Reporting configuration topic.

ReportProviderController

The ReportTreeProviderController provides the result for the selected report on the DataMining reporting page. To retrieve the result of the selected report a POST request is received which contains the name of the report in the activeReport parameter and the selected filter configuration in the post data.

The controller supports the 2 type of the reports: the RDLCRDLC and the xml reports, the filter collections and the reports hasn't been changed, so the earlier reports will work just fine with the new user interface.

According to the report type the controller works differently. For the RDLCRDLC reports the report definition is loaded from the /Reports subdirectory of the application and the specified parameters are also configured in the report definition which is store in a session variable

(RbdmsReportData). No data is returned to the client at this phase, the report is displayed in a subsequent load of the WebReportRDLC.aspx page.
Conversely, for the XML reports the results are loaded directly from the database and returned to the caller in JSON format

WellController

The WellController is used in both the simple and the advanced searches to query well data from the database.
The GET action implements the query for the simple search, and the following database query is executed:

```
select distinct w.pkey , w.wellid + ' ' + w.wellname, l.x, l.y from
Well w inner join construct c on w.pkey=c.wellkey inner join loc l on
c.pkey=l.constructkey and l.loctype='surf' where w.wellname like
@Search or w.wellid like @Search order by w.wellid + ' ' + w.wellname
```

For the advanced search the POST action is executed with the filter data updated with the filter parameters selected on the advanced search tab.
The base sql query of the advanced search is updated by Filter.GetParametrizedCommand which is a generic implementation to apply the values specified in the Filter model.

With the retrieved data, the Well model (implemented in Well.vb) is filled and then returned to the JavaScript client.

Map Configuration and spatial queries

In DataMining 2.0 we implement web mapping functionalities by using the OpenLayers library. OpenLayers is an open source (provided under the 2-clause BSD License) JavaScript library for displaying map data in web browsers. It provides an API for building rich web-based geographic applications similar to Google Maps and Bing Maps. The library was originally based on the Prototype JavaScript Framework.

OpenLayers is capable to consume data from OGC compliant web mapping services (like GeoServer in our case) and OpenLayers can handle the corresponding WMS/WFS request for the configured layers automatically.

For convenience, the state independent parts if the web mapping implementation is added to a common file (RBDMSGIS_HTML_OL.js) which can be reused in multiple projects.

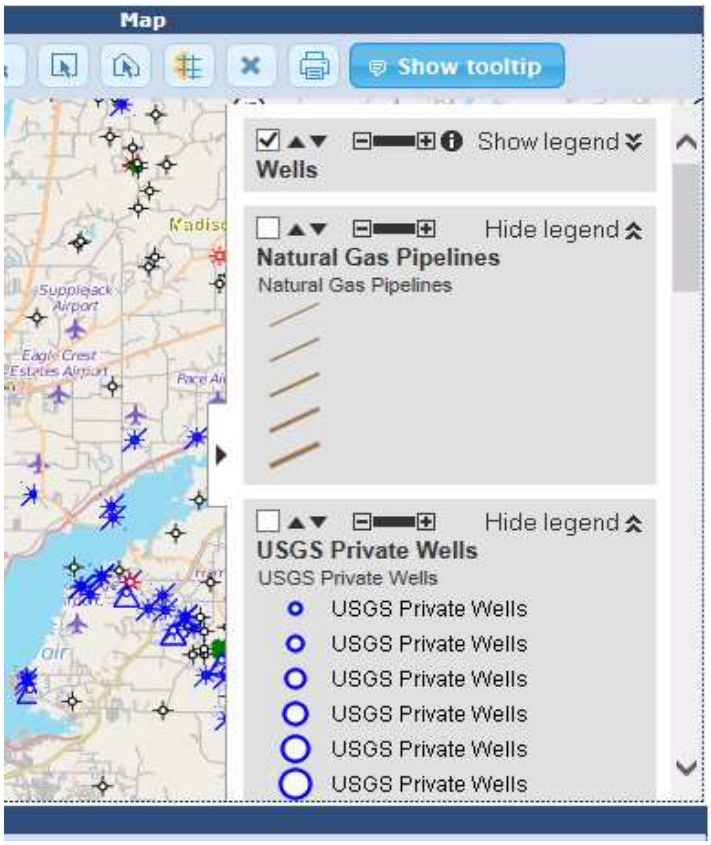
Map Configuration

The map configuration is specified in the GIS configuration file described earlier in this document. The GIS configuration is returned to the JavaScript client by using the GISConfigController.
The layers are created and added in the initMapConfig function. For each layer entry in the GIS config file a New OpenLayers.Layer.WMS is created, configured, and added to the map.
The visibility of the layers is set according to the Visible setting of the GISTOCItem node. The layers for which the Active flag is specified are added as queryable layers and also selected in the selection layers (multiselect listbox) above the map
When the ToolTip entry is specified for a layer, we also configure an OpenLayers GetFeature control which provides to execute the WFS query to GeoServer when the cursor is moved to any position on the map. For more information about the query controls refer to the Spatial queries and OpenLayers controls chapter.

The map legend /TOC

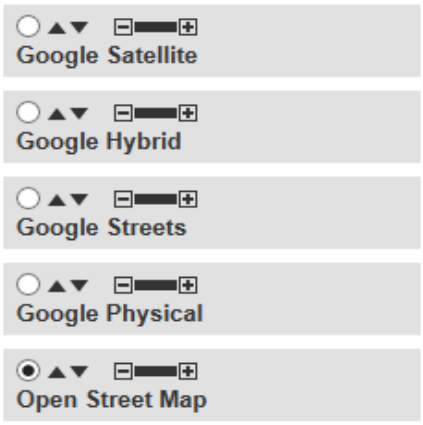
The map legend or TOC displays a list of the layers added to the map configuration. To implement the legend we use a customized version of the LayerSwitcher extension of OpenLayers.

The legend can be shown by clicking on the button on the right side of the map:



The legend supports setting the visibility, the opacity of the layers, changing the layer order and displaying the legend image for each layer. The active layers support the infoClick selection tool which can be activated by the info button.

The legend /TOC also supports changing the base layer of the map. Only one base layer can be displayed in the same time so switching between the base layers can be done using a set of radio buttons:



Spatial queries and OpenLayers controls

The spatial query operations on the selectable layers (like point, rectangle or polygon based selections) are implemented by a set of OpenLayers controls. Most of the implementation is added to the common RBDMS code base (in RBDMSGIS_HTML_OL.js). In most cases only one control should be active at the same time, so each control is added to a central repository (singleUseTools collection). We activate a specific control by calling the activateControl function which automatically deactivates all other controls. If all controls are deactivated (by calling setDefaultTool) OpenLayers will provide the default option which allows panning the map. The controls used with the current implementation are described in more detail in the following chapters.

Displaying the results of a spatial query

According to the state of the query the tree control on the search tab is updated with the query results, which is done in the selectionCallback function implemented in MSOGBOnline.html.

When the selection is starting the callback is called with the "selectionstart" operation. In this case the selection tree is updated according to the search configuration nodes as described earlier. For each search node, a new node is added to the selection tree and the "querying..." text is displayed to show that the query for the node is in progress.

When the selected features are returned from the server, the selection callback is called with the "selected" operation. In his case the features are added to the corresponding nodes as child elements and the "querying..." text is replaced with the actual count of the features.

When clicking on a result node and a keyVal and Detail XML is specified in the search configuration for that node, the EntityDeatils configuration is displayed on the Details section. For more information about the Details configuration see the Detail Configuration chapter later in this document. When the keyVal and Detail XML are not specified, the feature attributes are displayed as a plain table in the Details section.

To display the query results on the map we have configured a couple of overlay layers (selectLayer, selectPolyLayer, bufferLayer etc.) on the top of the map, which are not added to the legend/TOC.

Point selection tool

The point selection tool is implemented with an OpenLayers GetFeature control (called as pointSelectControl) which provides a WFS query to the map server with a specified click tolerance (currently configured as 7 pixels). When the control is active it registers to the mouse click event, initiates a WFS query and the selection callback is called when the results are received.

Rectangle selection tool

The rectangle selection tool is also implemented an OpenLayers GetFeature control (called as wmsGetFeatureControl). The main difference in the operation is that we configure a box selection and tracking the selection rectangle is handled by OpenLayers accordingly.

Polygon selection tool

Implementing the polygon selection is somewhat different than the previous tools. We need to set up a DrawFeature control (polySelectControl) to draw a polygon on the map and register on the "featureadded" event which is called when the drawing is complete. Then we need to configure and initiate the query to retrieve all features which intersects with the search shape. The query results are then displayed in the same way as for the previous tools.

Tooltip selection

The tooltip selection is configured along with the map layer configuration described earlier. When the ToolTip entry is specified for a layer, we configure an OpenLayers GetFeature control which works as a point selection tool with single selection only. In this case we also set the hover option to initiate the

query if the cursor is move to a given position on the map. When we receive a result of the query a popup is displayed on the map with the HTML contents specified in the ToolTip element of the map configuration:

```
  <ToolTip>
  <![CDATA[
   <table>
    <tr>
      <td>API</td>
      <td><a href="javascript:window.parent.parent.FillEDOnly('PKey',!-
PKEY-
!,'Integer','WellDetails.xml','ctl00_PageBody_WebPartManager1_gwpPanelD
etails_DetailsFrame');">!-API-!</a></td>
     </tr>
     </table>
    ]]>
  </ToolTip>
```

The attribute placeholders (like *!-API-!* or *!-PKEY-!*) are automatically replaced with the actual values in the selected feature.

InfoClick selection tool

The layers which are configured as "Active" on the GIS config file will support the infoClick selection tool. For these layers an info icon is displayed in the layer switcher. Clicking on this button will automatically deactivate all selection controls and activate the infoClick control for that layer according to the following example:

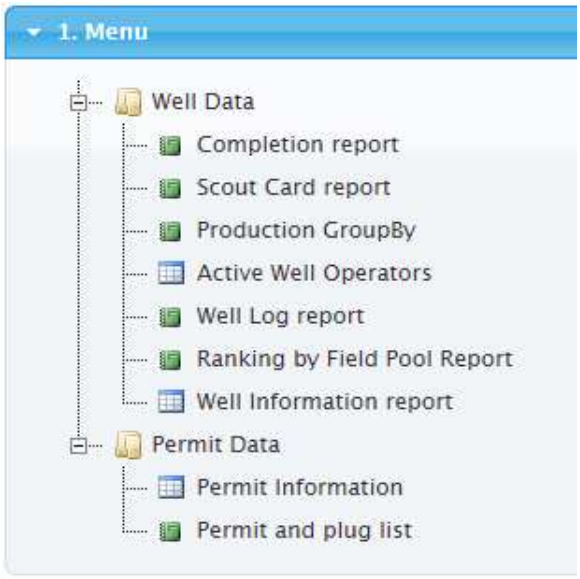
```
layerSwitcherActiveLayer: function (layer) {
        layer.setVisibility(true);
        var control = ME.olMap.getControlsBy("id",
"infoClickControl")[0];
        control.layers = new Array();
        control.layers[0] = layer;
        if ($.inArray(layer.name, ME.WFSCustomPopups) != -1) {
            control.infoFormat = "application/JSON";
        } else {
            control.infoFormat = "text/html";
        }

        ME.activateControl("infoClickControl");
    }
```

The infoClick control is an OpenLayers WMSGetFeatureInfo control, which displays a popup with a feature attribute table for the returned query result.

Reporting configuration

The reporting option of DaraMining 2.0 is implemented in the reporting.html page. The available reports are displayed in the Menu section which is filled by calling the ReportTreeProvider controller.



The content of the tree is stored in XML files in the database, according to the following example:

```
<?xml version="1.0" encoding="utf-8"?>
<mnuRBDMS>
  <siteMapNode description="Home" Create="False" Read="False"
Update="False" Delete="False" rightid="0" rightname="Home"
url="~/WebReportAccordion.aspx" winform="Home.xaml">
    <siteMapNode description="Well Data" Create="False" Read="False"
Update="False" Delete="False" rightid="0" rightname="Home"
url="~/WebReportAccordion.aspx?instance=1" winform="Home.xaml">
      <siteMapNode description="Completion report" Create="False"
Read="True" Update="False" Delete="False" rightid="2002"
rightname="Completions" url="~/WebReportAccordion.aspx?ID=2002"
winform="Completions.RDLC" image="~/images/Report.png" />
      <siteMapNode description="Scout Card report" Create="False"
Read="True" Update="False" Delete="False" rightid="2007"
rightname="Scout Cards" url="~/WebReportAccordion.aspx?ID=2007"
winform="Scout Cards.RDLC" image="~/images/Report.png" />
      <siteMapNode description="Production GroupBy " Create="False"
Read="True" Update="False" Delete="False" rightid="300031"
rightname="ProductionGroup" url="~/Default.aspx"
winform="ProductionSummary.RDLC" image="~/images/Report.png" />
      <siteMapNode description="Active Well Operators" Create="False"
Read="True" Update="False" Delete="False" rightid="2001"
rightname="Active Well Operators"
```

```
url="~/WebReportAccordion.aspx?ID=2001" winform="Active Well
Operators.xml" image="~/images/Table.png" />
      <siteMapNode description="Well Log report" Create="False"
Read="True" Update="False" Delete="False" rightid="2010"
rightname="Well Log" url="~/WebReportAccordion.aspx?ID=2010"
winform="Well Log.RDLC" image="~/images/Report.png" />
      <siteMapNode description="Ranking by Field Pool Report"
Create="False" Read="False" Update="False" Delete="False"
rightid="300204" rightname="ProdRankingByFieldPoolReport"
url="~/Default.aspx?instance=2" winform="RankBYFieldPool.RDLC"
image="~/images/Report.png" />
      <siteMapNode description="Well Information report" Create="False"
Read="True" Update="False" Delete="False" rightid="2009"
rightname="Well Information" url="~/WebReportAccordion.aspx?ID=2009"
winform="WellInformation.RDLC" image="~/images/Table.png" />
    </siteMapNode>
    <siteMapNode description="Permit Data" Create="False" Read="False"
Update="False" Delete="False" rightid="0" rightname="Home"
url="~/WebReportAccordion.aspx?instance=3" winform="Home.xaml">
      <siteMapNode description="Permit Information" Create="False"
Read="True" Update="False" Delete="False" rightid="2011"
rightname="Permit Information" url="~/WebReportAccordion.aspx?ID=2011"
winform="PermitInfo.xml" image="~/images/Table.png" />
      <siteMapNode description="Permit and plug list" Create="False"
Read="True" Update="False" Delete="False" rightid="2004"
rightname="Permit and Plug List"
url="~/WebReportAccordion.aspx?ID=2004" winform="Permit and Plug
List.RDLC" image="~/images/Report.png" />
    </siteMapNode>
  </siteMapNode>
</mnuRBDMS>
```

However, the user should not manually edit the tree in the database, but use RBDMS WinAdmin to modify this menu.

The application supports either the XML based reports or the RDLC reports which are distinguished by 2 different tree icons.
When clicking on a report node, the filter section is opened and filled with the controls supported by the selected report. The filter configuration is retrieved by using the FilterController in exactly the same way as described for the advanced search earlier.

When the Apply Filter button is clicked the filter structure is updated with the actual selection and then a POST request is invoked to the ReportProviderController passing the name of the active report and filter structure as the post data.
When the report data is retrieved the Report section is opened to display the result. If the returned report is an RDLC report, the report is displayed by WebReportRDLC.aspx which hosts the ASP.NET reportviewer control.
The XML based reports are displayed directly by using the jqGrid control and the conversion of the report data to CSV format is also supported in the Report section.

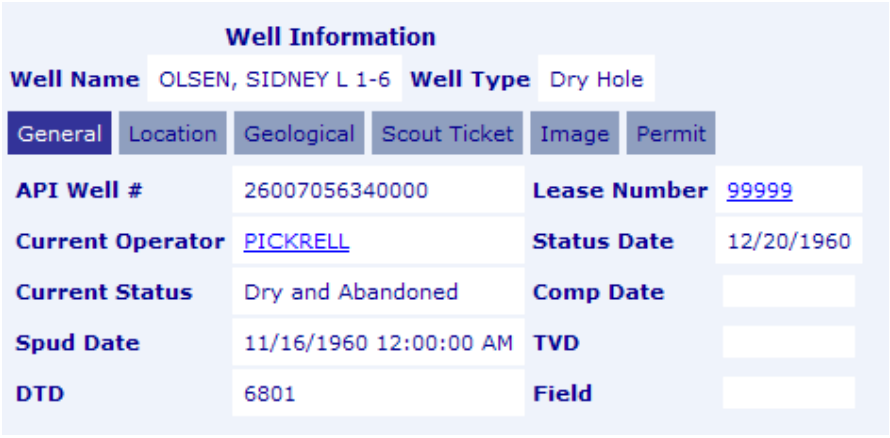
An example of the XML report (PermitInfo.xml) is shown below:

```
<RBDMSReport>
  <CollName>PermitInfo</CollName>
  <Description>Well Permit Information</Description>
  <sql>
    SELECT /*DISTINCT*/ Permit, WellID, WellName, FieldName,
EntityName, CountyName, PermitTypeDescription,
    convert(varchar(10), ApprovedDate, 101) as ApprovedDate, Depth
    FROM rptPermit
    WHERE Permit IS NOT NULL
  </sql>
  <Columns>PermitInfo<Column Name="Permit" Width="100" Caption="Permit
No." /><Column Name="WellID" Width="110" Caption="API No. (Surface)"
/><Column Name="WellName" Width="110" Caption="Well Name" /><Column
Name="FieldName" Width="100" Caption="Field" /><Column
Name="EntityName" Width="150" Caption="Operator" /><Column
Name="CountyName" Width="100" Caption="County" /><Column
Name="PermitTypeDescription" Width="120" Caption="Permit Type"
/><Column Name="ApprovedDate" Width="100" Caption="Date Approved"
/><Column Name="Depth" Width="90" Caption="Permit Depth" /></Columns>
</RBDMSReport>
```

In the XML report definition we specify the filter collection name, the report name, select statement to be executed, and the column information (Name, Width and Caption).

Detail Configuration

To display the information in the Details section we continue to use the RBDMSWebControls.EntityDetails web control hosted by the ED.aspx page. Detail configuration files are used to configure the RBDMSWebControls.EntityDetails web control. The control facilitates standardized, tabbed display of results for individual entities (e.g., wells, permits, operators, etc.).



As other RBDMS configuration files, the details configuration file is a serialized object. The EntityDetails web control is populated with an EntityDetailsPersist object, which has two top-level objects:

- MainTab: Controls title (Well Information) and “always visible” information (Well Name and Well Type).
- Subtabs: The collection of EntityDetailsItem controls.

```
<?xml version="1.0" encoding="utf-8"?>
<EntityDetailsPersist xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <MainTab>
  </MainTab>
  <Subtabs>
    <!-- GENERAL -->
    <EntityDetailsItem>
    </EntityDetailsItem>
  </Subtabs>
</EntityDetailsPersist>
```

Each tab (MainTab or EntityDetailsItem) has the following properties:

- Query: The object specifying the base sql for the tab and how the where clause will be constructed.
    - KeyNameAllowUpdate: Boolean (default = “false”). Can the calling application change the key name?

    - KeyName: The name of the key column.
    - KeyValue: The value in the key column we want to display. The KeyValue will always come from the calling application.
    - KeyType: The data type of the key column.
    - BaseQuery: The base SQL to which the constructed where clause will be appended.
    - OrderBy: The order by clause containing the text "order by."
- RelatedTables: The collection of RelatedTable controls.
- ColumnsWide. The integer specifying how many fields per row are rendered. For example, if there are seven fields selected in the SQL statement and ColumnWide= "3," then there will be 3 rows, the last containing only one field.
- TableTitle. The title of the details page if in MainTab or the tab title text if in EntityDetailsItem.

RelatedTable items are used to display lists of results (e.g., wells in field, leases for an operator, etc.). Each RelatedTable has the following properties:

- Query: See Query under MainTab / EntityDetailsItem).
- Title: The title string.

See Appendix II: Details Samples for more information.

RBDMSWebGIS Configuration

The RBDMSWebGIS Configuration file is discussed in detail in the document *RBDMSWebGIS XML Configuration*.

Appendix I: Styled Layer Descriptor (SLD) example

```
<?xml version="1.0" encoding="iso-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http://www.opengis.net/sld
StyledLayerDescriptor.xsd"
 xmlns="http://www.opengis.net/sld"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <NamedLayer>
    <Name>Counties</Name>
    <UserStyle>
      <Title>Counties</Title>
      <Abstract>SLD for Counties</Abstract>
      <FeatureTypeStyle>


        <!-- Rule 1: Show Counties all scales -->
        <Rule>
          <Name>Counties</Name>
          <Title>Counties</Title>
          <PolygonSymbolizer>
            <!-- Fill>
                              <CssParameter
name="fill">#f7efde</CssParameter>
                              <CssParameter name="fill-
opacity">0.5</CssParameter>
                         </Fill -->
            <Stroke>
              <CssParameter name="stroke">#ad9e8c</CssParameter>
              <CssParameter name="stroke-width">1</CssParameter>
            </Stroke>
          </PolygonSymbolizer>
        </Rule>

        <!-- Rule 2: Show Labels; scale 1 -->
        <Rule>
          <Name>Counties</Name>
          <Title>Counties</Title>
          <MaxScaleDenominator>7000000</MaxScaleDenominator>
          <MinScaleDenominator>4000000</MinScaleDenominator>
          <TextSymbolizer>
            <Label>
              <ogc:PropertyName>CONAME</ogc:PropertyName>
            </Label>

            <Font>
              <CssParameter name="font-family">Arial</CssParameter>
              <CssParameter name="font-size">5</CssParameter>
              <CssParameter name="font-weight">bold</CssParameter>
            </Font>

            <!-- this centers the label on the polygon's centroid-->
            <LabelPlacement>
```

```
      <PointPlacement>
        <AnchorPoint>
          <AnchorPointX>
            0.5
          </AnchorPointX>
          <AnchorPointY>
            0.5
          </AnchorPointY>
        </AnchorPoint>
      </PointPlacement>
    </LabelPlacement>

    <Fill>
      <CssParameter name="fill">#848200</CssParameter>
    </Fill>
  </TextSymbolizer>
</Rule>

<!-- Rule 3: Show Labels; scale  2 -->
<Rule>
  <Name>Counties</Name>
  <Title>Counties</Title>
  <MaxScaleDenominator>4000000</MaxScaleDenominator>
  <MinScaleDenominator>847000</MinScaleDenominator>
  <TextSymbolizer>
    <Label>
      <ogc:PropertyName>CONAME</ogc:PropertyName>
    </Label>

    <Font>
      <CssParameter name="font-family">Arial</CssParameter>
      <CssParameter name="font-size">7</CssParameter>
      <CssParameter name="font-weight">bold</CssParameter>
    </Font>

    <!-- this centers the label on the polygon's centroid-->
    <LabelPlacement>
      <PointPlacement>
        <AnchorPoint>
          <AnchorPointX>
            0.5
          </AnchorPointX>
          <AnchorPointY>
            0.5
          </AnchorPointY>
        </AnchorPoint>
      </PointPlacement>
    </LabelPlacement>

    <Fill>
      <CssParameter name="fill">#848200</CssParameter>
    </Fill>
  </TextSymbolizer>
</Rule>

<!-- Rule 4: Show Labels; scale  3 -->
<Rule>
```

```
            <Name>Counties</Name>
            <Title>Counties</Title>
            <MaxScaleDenominator>847000</MaxScaleDenominator>
            <MinScaleDenominator>350000</MinScaleDenominator>
            <TextSymbolizer>
              <Label>
                <ogc:PropertyName>CONAME</ogc:PropertyName>
              </Label>

              <Font>
                <CssParameter name="font-family">Arial</CssParameter>
                <CssParameter name="font-size">10</CssParameter>
                <CssParameter name="font-weight">bold</CssParameter>
              </Font>

              <!-- this centers the label on the polygon's centroid-->
              <LabelPlacement>
                <PointPlacement>
                  <AnchorPoint>
                    <AnchorPointX>
                      0.5
                    </AnchorPointX>
                    <AnchorPointY>
                      0.5
                    </AnchorPointY>
                  </AnchorPoint>
                </PointPlacement>
              </LabelPlacement>

              <Fill>
                <CssParameter name="fill">#848200</CssParameter>
              </Fill>
            </TextSymbolizer>
          </Rule>

        </FeatureTypeStyle>
      </UserStyle>
    </NamedLayer>
  </StyledLayerDescriptor>
```

Appendix II: Details Samples

Company:

```
<?xml version="1.0" encoding="utf-8"?>
<EntityDetailsPersist xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <MainTab>
    <Query>
      <KeyName>Cono</KeyName>
      <KeyValue xsi:type="xsd:string">01000</KeyValue>
      <KeyType>String</KeyType>
      <BaseQuery>
        <![CDATA[
        select
        [Company Name]=CoName
        from tblRefCompany
        ]]>
      </BaseQuery>
    </Query>
    <RelatedTables />
    <Tabs />
    <ColumnsWide>-1</ColumnsWide>
    <TableTitle>Company Information</TableTitle>
  </MainTab>
  <Subtabs>
    <!-- General -->
    <EntityDetailsItem>
      <Query>
        <KeyName>CoNo</KeyName>
        <KeyValue xsi:type="xsd:string">01000</KeyValue>
        <KeyType>String</KeyType>
        <BaseQuery>
          <![CDATA[
          select
          [Address]=Addr1,
          [Address2]=Addr2,
          [City]=City,
          [State]=State,
          [Zip]=PostalCode,
          [Phone]=Phone,
          [Phone Ext]=PH_Ext,
          [Fax]=fax
          from tblRefCompany
          ]]>
        </BaseQuery>
      </Query>
      <RelatedTables/>
      <Tabs />
      <ColumnsWide>2</ColumnsWide>
      <TableTitle>General</TableTitle>
    </EntityDetailsItem>
    <!-- Company Wells -->
```

```
    <EntityDetailsItem>
      <Query>
        <KeyName>Lease_Unit</KeyName>
        <KeyValue xsi:type="xsd:string">01000</KeyValue>
        <KeyType>String</KeyType>
        <BaseQuery />
      </Query>
      <RelatedTables>
        <RelatedTable>
          <Query>
            <KeyNameAllowUpdate>false</KeyNameAllowUpdate>
            <KeyName>OpNo</KeyName>
            <KeyValue xsi:type="xsd:string">01000</KeyValue>
            <KeyType>String</KeyType>
            <BaseQuery>
              <![CDATA[
              SELECT
              [API WELL NO.] = '<a
href="javascript:parent.FillED(''API_WELLNO'',''' + API_WELLNO +
''','''STRING''','''WellDetails.xml''','''ctl00_PageBody_WebPartManager1_gwp
PanelDetails_DetailsFrame''');">' + API_WellNo + '</a>',
              [WELL NAME] = Well_Nm
              from tblWellMaster
              ]]>
            </BaseQuery>
            <OrderBy>Well_Nm</OrderBy>
          </Query>
        </RelatedTable>
      </RelatedTables>
      <Tabs />
      <ColumnsWide>-1</ColumnsWide>
      <TableTitle>Company Wells</TableTitle>
    </EntityDetailsItem>
  </Subtabs>
</EntityDetailsPersist>
```

Wells (Basic):

```
<?xml version="1.0" encoding="utf-8"?>
<EntityDetailsPersist xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <MainTab>
    <Query>
      <KeyName>API_WELLNO</KeyName>
      <KeyValue xsi:type="xsd:string">26001210010000</KeyValue>
      <KeyType>String</KeyType>
      <BaseQuery>
        <![CDATA[
        select
          [Well Name] = Well_Nm,
          [Well Type] = Well_Typ
        from tblWellMaster
        ]]>
      </BaseQuery>
    </Query>
    <RelatedTables />
```

```
    <Tabs />
    <ColumnsWide>-1</ColumnsWide>
    <TableTitle>Well Information</TableTitle>
  </MainTab>
  <Subtabs>
    <!-- GENERAL -->
    <EntityDetailsItem>
      <Query>
        <KeyName>API_WELLNO</KeyName>
        <KeyValue xsi:type="xsd:string">26001210010000</KeyValue>
        <KeyType>String</KeyType>
        <BaseQuery>
          <![CDATA[
           select
          [API Well #] = API_WellNo,
          [Lease Name] = Lease_Nm,
          [Current Operator] = '<a
href="javascript:parent.FillED(''CoNo'',' + Cast(OpNo as varchar(25)) +
',''STRING'',''CompanyDetails.xml'',''ctl00_PageBody_WebPartManager1_gw
pPanelDetails_DetailsFrame'');">' + (select CoName from tblRefCompany
where CoNo=OpNo)+ '</a>',
          [Status Date]= Convert(char(10),Dt_Status,101),
          [TVD]=TVD,
          [DTD]=DTD
          from tblWellMaster
          ]]>
        </BaseQuery>
      </Query>
      <RelatedTables />
      <Tabs />
      <ColumnsWide>2</ColumnsWide>
      <TableTitle>General</TableTitle>
    </EntityDetailsItem>
  </Subtabs>
</EntityDetailsPersist>
```

Appendix III. Using a WCF for Some or All Data Access

In 2010, two projects necessitated adding the ability to consume WCF services from the Data Mining: Mississippi and Oklahoma. Mississippi's need arose from the desire to serve images to the Internet from their Laserfiche Imaging Service, which is only accessible from their internal network. Additionally, access to the Laserfiche web service from the RBDMS.Net WPF application was accomplished via a WCF service layer, and it made sense to reuse the service in Data Mining. Oklahoma's need arose from their IT requirement that there be no direct database access from any machine accessible from the Internet.

Though numerous modifications were made to the Data Mining to accommodate the use of WCF services, it is fully backward-compatible and uses the same configuration files for the Full Text Search and Details configurations.

The Data Mining application must have a service reference added to the RbdmsWebControls project. The service reference name is used in the configuration.

In version 2.0 of RBDMS Data Mining, the WCF configuration for details is still applicable, but the filter and search capabilities would do not use the WCF by default. Modifications can be made to any controller method to access the WCF instead of making direct database queries as required.

Full Text Search Example

The only difference from the direct data access is in the SQL and SQLAdv elements. The formatting of these elements when using a web service is as follows:

- WebServiceCall|ServiceNamespace|ServiceReferenceName|MethodName |QueryName
- WebServiceCall: Literally says to the Data Mining application that you want it to get the data from a service (i.e., this is not SQL).
- ServiceNamespace: The namespace of the referenced service. This is entered when you add the service reference.
- ServiceReferenceName: Typically the ServiceNamespace & “Client”
- MethodName: The name of the service method to be called (as defined in the service contract).
- QueryName: The parameter to be passed to the service method.

```
<FullTextNode>
    <Name>Wells</Name>
    <FilterName />

<NavURL>javascript:window.parent.FillED('PKey',%NavUrl%,'Integer','WellDetails.xml','ctl00_Pag
eBody_WebPartManager1_gwpPanelDetails_DetailsFrame');</NavURL>
    <?NavRootURL
>javascript:window.parent.FillReport('Filter=&amp;Report=OperatorWells2000.rdl&amp;Key=C
ompanies&amp;Database=NOGCCOnline','ctl00_PageBody_WebPartManager1_gwpPanelDetails_
DetailsFrame');</NavRootURL?>
    <Sql>

<![CDATA[WebServiceCall|DataMiningService|DataMiningServiceClient|FindQuery|Wells]]>
    </Sql>
    <SqlAdv>

<![CDATA[WebServiceCall|DataMiningService|DataMiningServiceClient|AdvancedQuery|Wells]]>
    </SqlAdv>
    <KeyName>w.PKey</KeyName>
    <KeyType>Integer</KeyType>
    <SearchType>FullText</SearchType>
    <ThemeColumn></ThemeColumn>
    <ThemeItems>
      <FullTextThemeItem>
        <Value>*</Value>
        <Image>~/images/Ref1.gif</Image>
      </FullTextThemeItem>
    </ThemeItems>
  </FullTextNode>
```

Details Example

In the same way that the Full Text example only modifies the SQL and SQLAdv elements, the Details.xml file is only altered in the baseQuery elements. The formatting of these elements when using a web service is the same as the Full Text search, as follows:

- WebServiceCall|ServiceNamespace|ServiceReferenceName|MethodName |QueryName

- WebServiceCall: Literally says to the Data Mining application that you want it to get the data from a service (i.e. this isn't SQL).

- ServiceNamespace: The namespace of the referenced service. This is entered when you add the service reference.

- ServiceReferenceName: Typically the ServiceNamespace & “Client.”

- MethodName: The name of the service method to be called (as defined in the service contract).

- QueryName: The parameter to be passed to the service method.

```
<?xml version="1.0" encoding="utf-8"?>
<EntityDetailsPersist xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <MainTab>
  <Query>
   <KeyName>PKey</KeyName>
   <KeyValue xsi:type="xsd:integer">1</KeyValue>
   <KeyType>Integer</KeyType>
   <BaseQuery>

<![CDATA[WebServiceCall|DataMiningService|DataMiningServiceClient|DataMiningQuery|EntityI
nformation]]>
   </BaseQuery>
  </Query>
  <RelatedTables />
  <Tabs />
  <ColumnsWide>-1</ColumnsWide>
  <TableTitle>Entity Information</TableTitle>
 </MainTab>
 <Subtabs>
  <!-- General -->
  <EntityDetailsItem>
   <Query>
    <KeyName>EntityKey</KeyName>
    <KeyNameAllowUpdate>false</KeyNameAllowUpdate>
    <KeyValue xsi:type="xsd:integer">1</KeyValue>
    <KeyType>Integer</KeyType>
    <BaseQuery>

<![CDATA[WebServiceCall|DataMiningService|DataMiningServiceClient|DataMiningQuery|Entity
General]]>
    </BaseQuery>
   </Query>
   <RelatedTables>
    <RelatedTable>
     <Query>
      <KeyNameAllowUpdate>false</KeyNameAllowUpdate>
      <KeyName>EntityKey</KeyName>
      <KeyValue xsi:type="xsd:integer">1</KeyValue>
      <KeyType>Integer</KeyType>
      <BaseQuery>

<![CDATA[WebServiceCall|DataMiningService|DataMiningServiceClient|DataMiningQuery|Entity
Communication]]>
      </BaseQuery>
     </Query>
     <Title>Communication</Title>
    </RelatedTable>
```

```
    </RelatedTables>
    <Tabs />
    <ColumnsWide>2</ColumnsWide>
    <TableTitle>General</TableTitle>
   </EntityDetailsItem>
   <!-- Entity Wells -->
   <EntityDetailsItem>
    <Query>
     <KeyName>EntityKey</KeyName>
     <KeyNameAllowUpdate>false</KeyNameAllowUpdate>
     <KeyValue xsi:type="xsd:integer">1</KeyValue>
     <KeyType>Integer</KeyType>
     <BaseQuery />
    </Query>
    <RelatedTables>
     <RelatedTable>
      <Query>
       <KeyNameAllowUpdate>false</KeyNameAllowUpdate>
       <KeyName>Operator</KeyName>
       <KeyValue xsi:type="xsd:integer">1</KeyValue>
       <KeyType>Integer</KeyType>
       <BaseQuery>

<![CDATA[WebServiceCall|DataMiningService|DataMiningServiceClient|DataMiningQuery|Entity
Wells]]>
       </BaseQuery>
      </Query>
     </RelatedTable>
    </RelatedTables>
    <Tabs />
    <ColumnsWide>2</ColumnsWide>
    <TableTitle>Wells</TableTitle>
   </EntityDetailsItem>
  </Subtabs>
</EntityDetailsPersist>
```

Data Mining WCF Service Contract

```
<ServiceContract()> _
Public Interface IDataMiningService
    <OperationContract()> _
    Function DataMiningQuery(ByVal tokenID As Guid, ByVal name As String, ByVal keys() As
KeyData) As System.Data.DataSet

    <OperationContract()> _
    Function FindQuery(ByVal tokenID As Guid, ByVal name As String, ByVal keys() As KeyData)
As System.Data.DataSet

    <OperationContract()> _
```

```
    Function AdvancedQuery(ByVal tokenID As Guid, ByVal name As String, ByVal where As
String) As System.Data.DataSet

    <OperationContract()> _
    Function GetLookup(ByVal tokenID As Guid, ByVal name As String) As KeyData()
End Interface


KeyData Class Definition (as used in IDataMiningService)

Public Class KeyData
    <DataMember()> _
    Public KeyName As String
    <DataMember()> _
    Public KeyType As String
    <DataMember()> _
    Public KeyValue As Object
End Class
```

Coding the WCF Service

Though the inner workings of the service are immaterial to the Data Mining client (i.e., the ASP.NET website), some discussion of the way the service was implemented for Oklahoma is warranted. Basically, we moved the xml configuration files from the client to the server, so now the server has the SQL statements and direct connections to the database. The App_Data folder of the service contains a StateSpecific folder:

- Nodes.xml
- DetailsXML (folder)
  - EntityDetails.xml
  - WellDetails.xml
  - etc...

Several helper classes are included to assist in locating the correct section in the configuration files to pull the SQL from. The results of the SQL are then formatted and returned to the client as a DataSet with a single Table to be used by the caller.

For a full text or advanced query, the rule is that the QueryName (e.g., Wells) must match the FullTextNode.Name value in the Nodes.xml. In the current implementation, the “full text search” is actually performed with LINQ-to-SQL queries as opposed to full text indexing. Therefore, any SQL present in the <Sql> element of the Nodes.xml is only used as a fallback for advanced queries when the <SqlAdv> element is not present.

For a details type query, the rule is that the QueryName (e.g., EntityInformation) must match the Title or TableTitle value in the Details.xml.