July 2009

Developer's Guide to the Avitek Sample Portal

Built on WebCenter 11g

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Introduction

The Avitek Sample Portal is a custom portal built using WebCenter 11g and WebCenter Services. Avitek's source code is distributed to help customers and partners understand how best to design WebCenter portals. This document describes the portal, highlights the design themes behind it, provides installation instructions and gives a tour of major features and their implementation.

Figure 1 - The Avitek Sample Portal

Installation Instructions

Avitek is packaged as a JDeveloper project. You will need JDeveloper 11gR1 11.1.1.1.0 or later.

1. Download and unzip avitek_sample_*version*.zip to your local drive. That archive contains this document, plus two additional archives.
2. Unzip avitek_workspace_*version*.zip to any convenient location. This archive contains the Avitek JDeveloper project.
3. Locate the drive or partition which JDeveloper use for the systems* directory and unzip avitek_content_*version*.zip into the root (e.g. C:\ or /) of that drive or partition. This will create a directory like C:\scratch\avitek with the content repository of the Avitek application.
4. Start JDeveloper and make sure that the WebCenter Extension is installed. If not, install it from Help | Check for Updates.
5. Open Avitek/Avitek.jws in JDeveloper.

6. To run the sample portal, right-click on the ViewController project and select Run.
7. The context root of the portal is /avitek, so the home page URL will resemble: http://server:7101/avitek/

Focus of the Sample Portal

The focus of Avitek is on core portal features, such as navigation, look-and-feel, content integration and personalization. Avitek illustrates how WebCenter approaches these areas of functionality. Readers will notice that other areas of portal functionality, like search, are not the main focus of the Avitek sample.

Design Themes

Separation of Content, Skin and Navigation

A portal's purpose is to display information, with a particular look and feel, in an organized way. In robust portal architectures these three areas--content, "skin" and navigation--are modular. Page templates, for example, are designed with only general consideration of content. Look and feel is applied with only a general knowledge of page structure and navigation. Ongoing management of content can occur independent of the portal.

WebCenter makes this kind or architecture possible. The Avitek sample portal separates the definition of page templates and navigation from the identification of content. Content resides in a CMS and as remote portlets. Integration with identity management allows delegation of administrative rights. Skins are applied to the whole.

This kind of separation is impossible without standards. Whenever possible, standards like JCR (content) and WSRP (remote portlets) are supported in WebCenter.

Personalization and Page Customization

Today's portal users constantly create and customize pages. The Avitek sample makes heavy use of WebCenter's Page Service, a powerful utility for provisioning pages. The sample also uses Composer, which allows end users to customize pages, either by adding and removing content or by changing look and feel.

Portlet Consumption

In Avitek, remote portlets can be configured at run time. Avitek consumes WSRP portlets, but WebCenter is not limited to WSRP. In addition to WSRP 1.0 and 2.0 portlets, WebCenter also consumes JPDK portlets and is fully integrated with JDeveloper for development using ADF.

Useable, Linkable URLs

Avitek demonstrates how "pretty" URLs can be used to clearly reflect the taxonomy of a site and allow direct access to any page. All page URLs in Avitek are bookmarkable.

Delegated Administration

Avitek demonstrates how various administrative roles can be defined and applied.

Tour of the Portal

Public Pages

Avitek is an intranet for a telecommunications company. For the purposes of the sample, all users are considered employees. Anonymous users are shown public pages, i.e. pages that are of general interest to employees and that do not require authentication.

Employees see a typical set of public pages in the left-hand navigation of the landing page. Navigation and page model are described later.

Figure 2 – Pages available to anonymous employees.

Authentication & Security

The sample portal includes two users, Monty and Vicki. To log in as Monty, click on Login in the upper right hand corner of the page and enter username: monty, password: welcome1. Vicki uses vicki/welcome1.

Avitek uses JAZN as a user and policy store. JAZN is Oracle's JAAS provider. In Avitek, all user and policy information is stored in `jazn-data.xml`. JDeveloper includes a special editor for making changes to this file.

Customer 360 Dashboard

When logged in, Monty and Vicki see an additional set of pages. Some of these pages are secure employee content, like the customer dashboard.

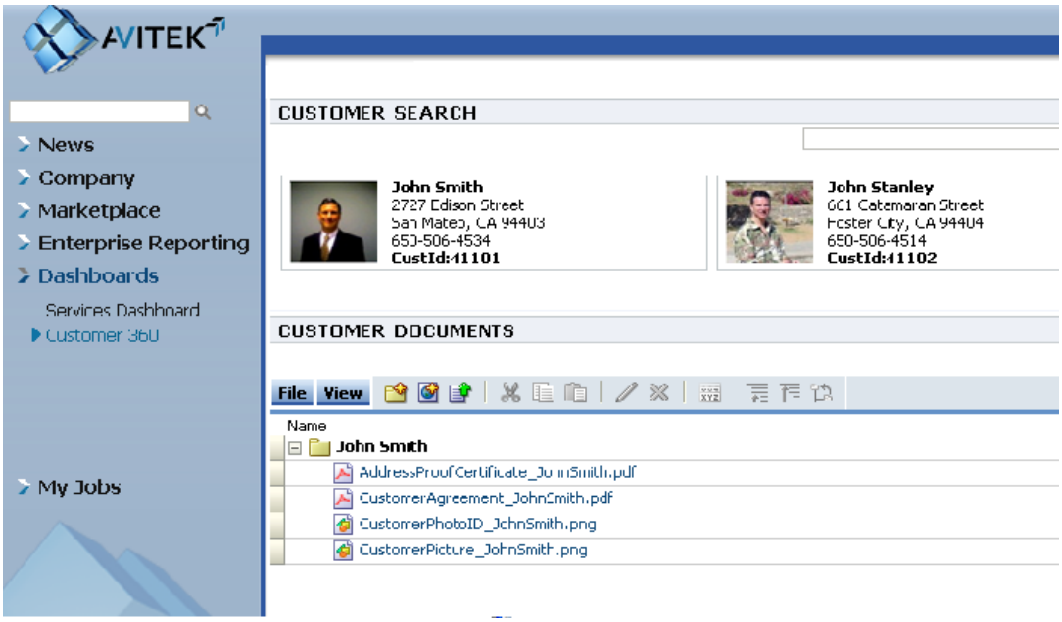


Figure 3 – The Customer 360 Dashboard.

The customer dashboard includes three taskflows. The customer viewer taskflow displays basic customer information, like contact info. It includes a customer search box. If a particular customer is selected, the customer documents and customer details taskflows respond with relevant content. The customer documents taskflow is an instance of the Document Library, a WebCenter Service that can be backed by any JSR-170 repository. In the case of Avitek, it is a file-based repository. The customer details taskflow is a taskflow that would normally be backed by CRM data, such as from Siebel.

Like all pages in Avitek, the Customer 360 Dashboard can be accessed using a well-defined URL: `avitek/pages/Dashboards/Customer 360`. Note that the URL is case sensitive and can take spaces as arguments.

Administration Pages

Monty also happens to be a portal administrator, so he sees a "Manage Site" button. Manage Site allows him to administer the portal. The management console has three sub-tabs:

The Site Tab

The Site tab allows Monty to define the navigation of the site. If Monty clicks on the "Company" node, and then on "Create" he will create a new page in the Company menu.

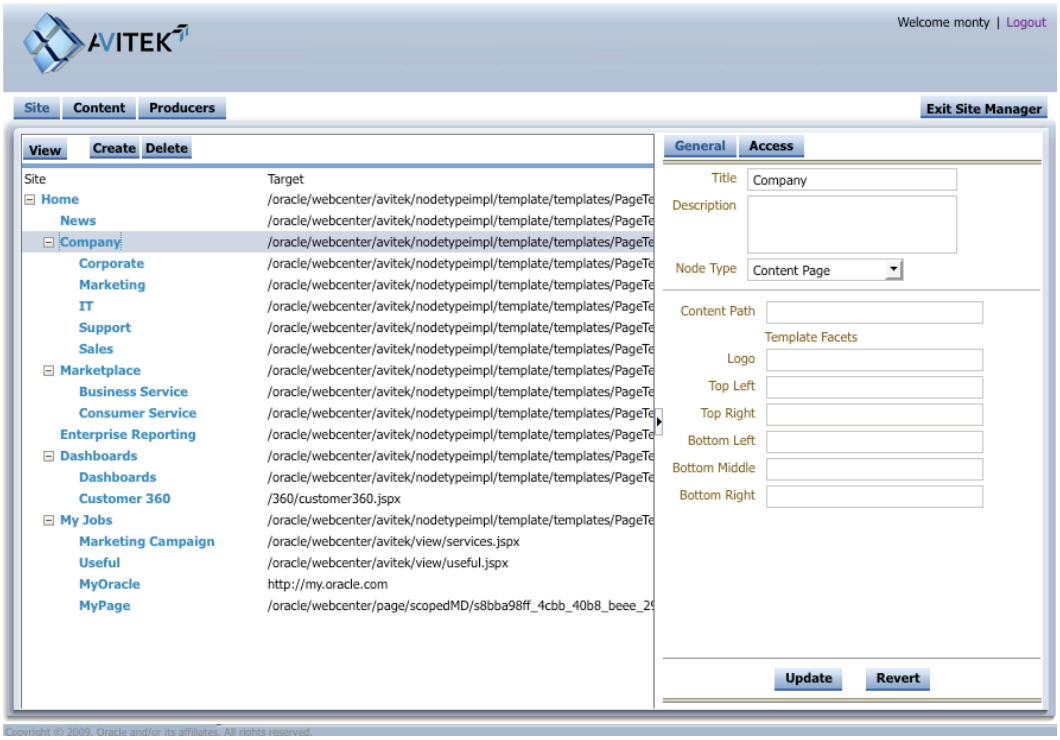


Figure 4 – The Site Tab.

The new page will be of a certain *page type*. If the type is "content page", the default type, it will also require a content *page template* and it will be mapped to a directory in the content repository.

Avitek includes five *page types*. Page types define attributes of the page, like whether or not the page will have content and whether or not the page is customizable. Avitek's types are:

- Content Page: A page that will be mapped to content in the content repository, and driven by that content.
- Editable Content Page: A page that will be mapped to content in the content repository, and that will also be editable by end users.
- Editable Page: A page that will be editable by end users. Instances of editable pages are created by the Page Service.
- Application Page: A page included in the portal application at design time, but addable to the navigation in various places at run time. And example is the Customer 360 Dashboard page.
- URL: A link to some other URL.

Avitek includes two *page templates* for content pages, content template 1 and content template 2. These pages are defined in xml and describe the structure of the page and where content will be placed. Content Pages and Editable Content Pages both take a path to a content source. Their templates expect to render content, so they must be mapped to content in the repository that meets their expectations. The content path can be determined by navigating the content repository, described in the next section.

Finally, the Site Tab allows administrators like Monty to control access for any page. Click on the Access tab, next to the General tab for any page. A list of all available roles and users is presented. View, Edit and Manage privileges can be assigned. For Avitek, all possible users, roles and privileges are defined using JAZN.

The Content Tab

The Content tab allows Monty to manage site content. In the case of Avitek, the content is stored on the file system. Monty can upload, replace and delete content from the Content tab.

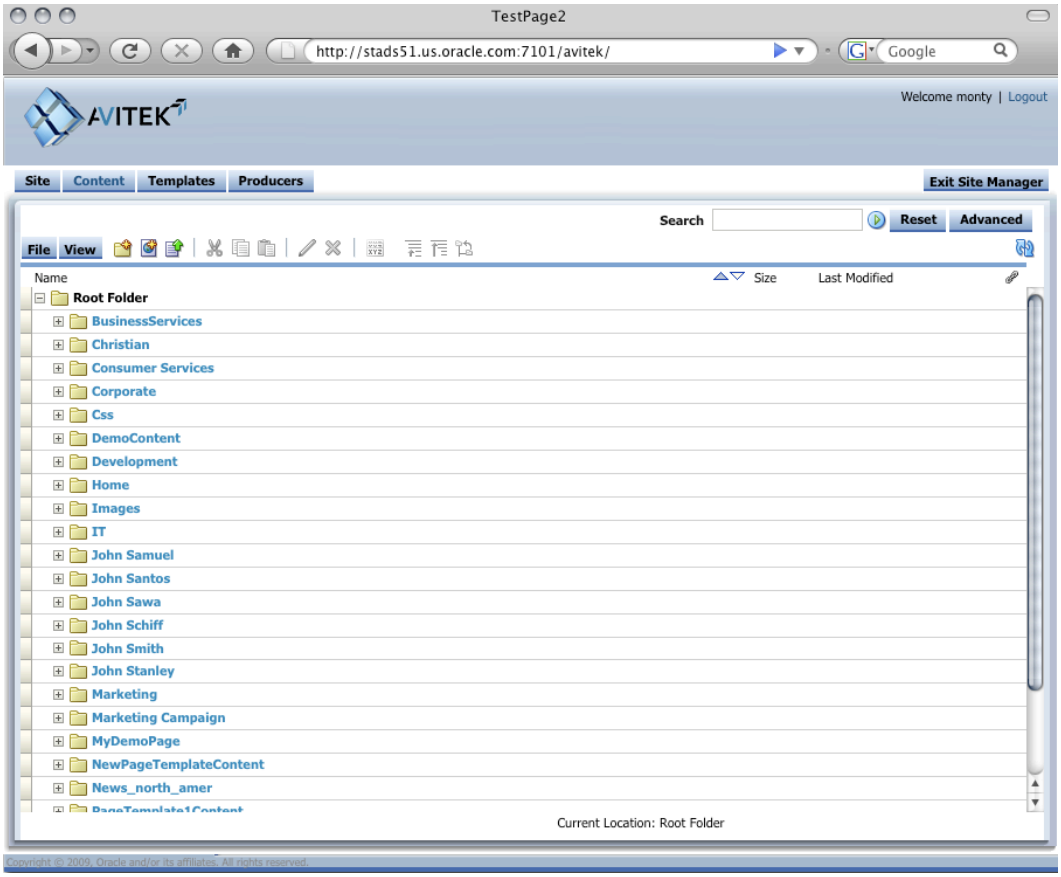


Figure 5 – The Content Tab.

Monty can determine the path for any piece of content by clicking on the properties icon.

For example the DemoContent folder has the path /DemoContent. This path can be mapped to any content page and the content will be displayed. The structure under /DemoContent matches what the content page templates expect.

The Producers Tab

The Producers tab allows Monty to register remote portlet producers. Monty could register a new WSRP producer, for example, and portlets for that producer would appear in the Resource Catalog, described below.

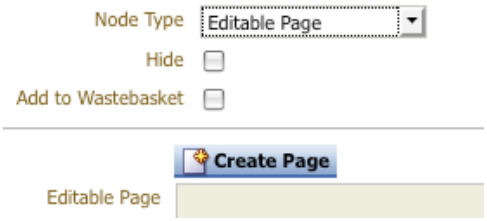
Site Navigation and Skins

The Exit Site Manager button returns Monty to the main site. Notice the "Change Layout" button at the top right of the home page. When this button is clicked, the left-hand navigation becomes a header navigation. This occurs because the application-level template surrounding the page has been changed. As described above, templates are first-order objects in ADF. They are defined at design-time and they reside in the application's `.ear` file. In Avitek, they are included in the Templates project. Typically, they are defined in `.jspx` files. More on application-level templates can be found below, in the Implementation Highlights section.

Skins, likewise, are defined at design time and can be switched at run time. Skins are defined using CSS. In Avitek, the Skin project contains `CollabSkin.css`, which defines the skin.

Editable Pages, Composer & the Resource Catalog

Monty can create end-user editable pages in the Site Manager. He selects the My Jobs node on the Site Tab. Then he clicks "Create". He changes the page name from newNode# to "MyPage" and he selects a Node Type of Editable Page, then clicks Update. He will see another button, Create Page:



Clicking on create page brings up a dialog provided by the Page Service. This dialog asks for a page template to be selected—these templates are Page Service page templates and are different from the content page templates described above. Monty selects the Avitek Flex template and also names this page "MyPage". Once the dialog is complete, the new editable page is created. Any end user can now visit this page and customize it. Monty, for example, could click on the page name in the site hierarchy to visit it. He could then click the edit page button.

The page is now viewed in Composer, a run-time page editor.

Figure 6 – An editable page in Composer.

In Composer, end users can edit the look and feel of a page by clicking on the edit icon

at the top-right corner of the page. The Change Layout button allows users to alter the layout of the page. The Add Content button allows users to add content from the Resource Catalog. The Resource Catalog is a list of all available Page Layout Items, Portlets and WebCenter Services.

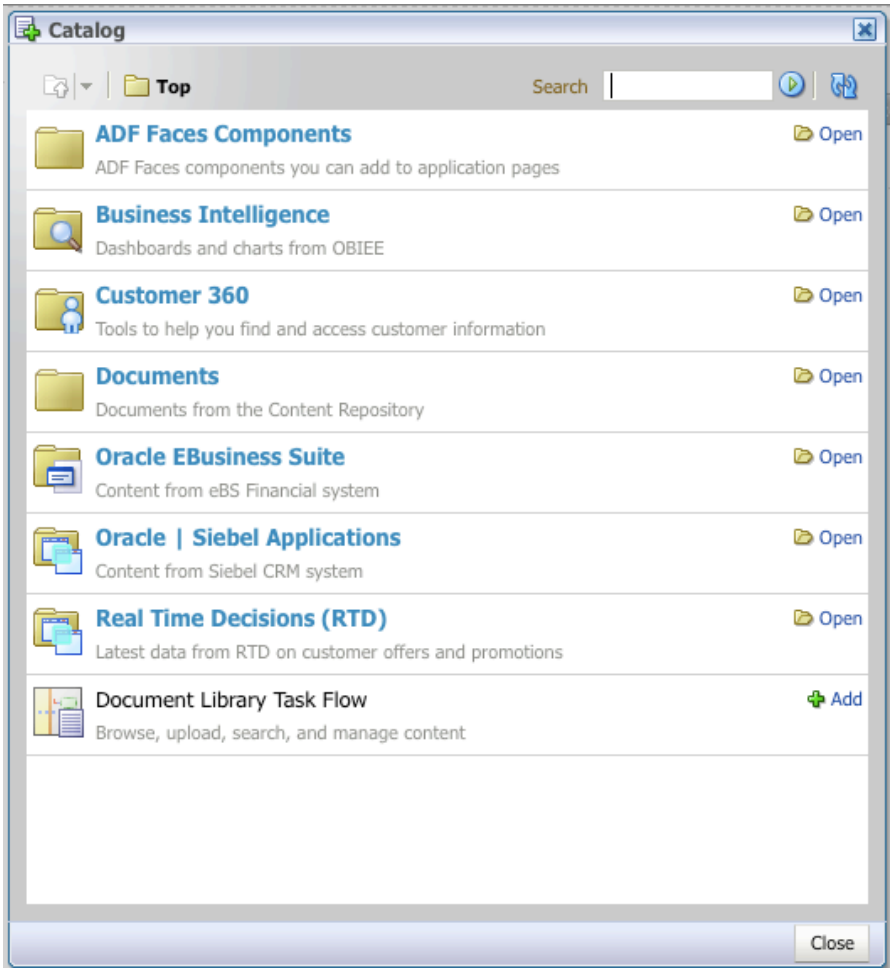


Figure 7 – The Resource Catalog or Business Dictionary.

Implementation Highlights

Application Anatomy

The Avitek application consists of the several modules listed below. Each module is organized as a JDeveloper project. The ViewController project serves as the main webapp. The other projects generate libraries that are used by the main webapp.

Avitek Application

- ViewController – serves as the main webapp
- SiteModel – provides the Java API to manage site hierarchy
- NodeType – contains a generic interface for the node type and implementation of various out-of-box node types. Node type is the implementation name for Page Type.
- SiteManager – provides the user interface for the Administration Pages (Site Tab, Content Tab and Producers Tab).
- Skin – contains stylesheet and relevant files for the application skin.
- Templates – implementation of application-level portal templates (e.g. site and navigation templates)

Rebuilding the Application

The application can be rebuilt by following these instructions:

1. Open the application in JDeveloper
2. Click Build → Clean All
3. Click Yes in the confirmation dialog
4. In the application navigator, right-click on the SiteModel project, select Deploy → avitek-custom-navigation → to JAR file
5. Deployment Log shows the creation of jar file
6. In the application navigator, right-click on the SiteModel project, select Deploy → avitek-site-model → to ADF Library JAR
7. In the application navigator, right-click on the NoteType project, select Deploy → avitek-nodetype → to ADF Library JAR
8. In the application navigator, right-click on the Skin project, select Deploy → avitek-skin → to ADF Library JAR
9. In the application navigator, right-click on the Templates project, select Deploy → avitek-templates → to ADF Library JAR
10. In the application navigator, right-click on the SiteManager project, select Deploy → avitek-site-manager → to ADF Library JAR
11. In the application navigator, right-click on the ViewController project, select Rebuild ViewController.jpr

Templates

The Templates project contains application level templates, namely, site and navigation templates. These templates are different from the "page" templates mentioned above (both the content page templates and the Page Service templates). The application level templates are the outermost templates used by all pages in the application. They apply objects such as site logo, footer, and navigation to all pages.

The site level template is located in the Templates project under Web Content → oracle/ webcenter/avitek/templates/siteTemplate.jspx. The template pulls into the page site logo, site footer, buttons to change layout and manage site, login/logout link, and other objects common across all pages.

There are two navigation templates used by the app, sidebarNavigationTemplate.jspx for the left-hand navigation and dropDownMenuNavigationTemplate.jspx for the header navigation. They can be found under Web Content → oracle/ webcenter/avitek/templates. These templates render the site model in their specific format.

The application logic behind the UI components of the templates resides mostly in oracle.webcenter.avitek.templates.beans.TemplateBean under the Templates project. This bean also serves as interface with other modules of the application such as SiteModel to translate UI gestures to operations on the site model.

This project builds the library /Avitek/Templates/deploy/avitek-templates.jar.

Site Model

The SiteModel project exposes the Java API to manage site hierarchy. The site hierarchy consists simply of nodes in a tree structure. Behind the API, Site Model uses MDS to persist and change the metadata that represents the site hierarchy.

The metadata of the site hierarchy can be found under the directory /Avitek/mds/oracle/ webcenter/avitek/metadata. Each node of the site has a corresponding XML file. The name of the file is of the format node[X].xml where [X] is an integer sequence that is incremented each time a new node is created. The root of the hierarchy is node0.xml and that is the first node displayed for each new application session.

Each node has references to its children as well as its parent. The parent is referenced with the parentRef attribute of the <node> element. The children references are represented using the <child> elements. For example, the metadata for the Company node looks like this:

```
<?xml version = '1.0' encoding = 'UTF-8'?>

<node id="id_company" xmlns="http://xmlns.oracle.com/webcenter/avitek"

                xmlns:avitek="http://xmlns.oracle.com/webcenter/avitek"

                avitek:parentRef="/oracle/webcenter/avitek/metadata/node0.xml#id_root">

 <property id="deleted" value="false"/>

 <property id="includeAsFolder" value="false"/>
```

```
<property id="type" value="template"/>
<property id="hidden" value="false"/>
<property id="target" value="/oracle/webcenter/avitek/nodetypeimpl/template/templates/PageTemplate1.jspx"/>
<property id="title" value="Company"/>
<child id="id_corporate" avitek:childRef="/oracle/webcenter/avitek/metadata/node3.xml#id_corporate"/>
<child id="id_marketing" avitek:childRef="/oracle/webcenter/avitek/metadata/node4.xml#id_marketing"/>
<child id="id_it" avitek:childRef="/oracle/webcenter/avitek/metadata/node5.xml#id_it"/>
<child id="id_support" avitek:childRef="/oracle/webcenter/avitek/metadata/node6.xml#id_support"/>
<child id="id_sales" avitek:childRef="/oracle/webcenter/avitek/metadata/node7.xml#id_sales"/>
</node>
```

The properties of a node such as type, target, title are represented as generic property elements in the metadata. The property elements are basically name value pairs represented with the id and value attribute respectively.

The main Java API of the Site Model includes the classes oracle.webcenter.avitek.model.MetadataPageService and oracle.webcenter.avitek.model.MetadataPageServiceNode under SiteModel project. MetadataPageService provides the tree level operations such as getRootNode(), getSelectedNode(), createNode(), deleteNode(), and etc. MetadataPageServiceNode provides node specific operations such as getProperties(), getPrettyURI(), and etc.

The model representing the tree of nodes can be easily converted to TreeModel suitable for use by ADF Faces Components. The code snippet below shows how to obtain an oracle.adf.view.faces.model.TreeModel from a root node obtained from MetadataPageService.

```
...
List nodeList = new ArrayList();
Node root = MetadataPageService.getInstance().getRootNode();
nodeList.add(root);
TreeModel tree = new ChildPropertyTreeModel(nodeList, "childNodes");
...
```

SiteModel also contains oracle.webcenter.avitek.customization.NodeCC class that plays a key role in storing customizations of pages. Whenever customizations are made to a page, NodeCC provides the currently selected node id to the metadata service layer (MDS) of the ADF framework. MDS then stores the customization against the provided node id. This enables, for example, content driven pages to run using the same template jspx but store customizations against different node ids.

This project builds the library /Avitek/SiteModel/deploy/avitek-site-model.jar and /Avitek/SiteModel/deploy/avitek-custom-navigation.jar.

Node Types

One of the Avitek application's main features is the ability to incorporate heterogeneous types of pages into the same site. The out-of-box page types available are content driven pages, editable page service pages, external URL pages, and seeded application pages. This is possible because the different types of pages are simply nodes in the site hierarchy tree with different value for the type property. Depending on the value of the type property, the application can show different page property editors in the site manager and also perform different navigation actions.

The generic interface for node type and implementation of various out-of-box node types can be found in the NodeType project. The generic interface is the oracle.webcenter.avitek.nodetype.NodeTypeHandler class. The implementation of the various out-of-box types can be found under the oracle.webcenter.avitek.nodetypeimpl package.

The various node types are registered with the application in oracle.webcenter.avitek.nodetype.NodeTypeManager. To create a new node type, one would implement a type handler based on the oracle.webcenter.avitek.nodetype.NodeTypeHandler abstract class or an existing node type handler. The new type handler can then be registered with the application in the initNodeTypes() method of NodeTypeManager.

Each node type can also supply its own UI for the Site Manager's property inspector. The type specific property inspector is supplied as a bounded task flow. The property inspector task flow for the out-of-box node types can be found under the package /oracle/webcenter/avitek/nodetypeimpl in Web Content. Depending on the node type, Site Manager pulls in the relevant property inspector task flow.

This project builds the ADF library /Avitek/SiteModel/deploy/avitek-nodetype.jar.

Security

The Avitek sample portal provides an additional layer of security on top of existing security to access the content pages. This additional layer secures access and modifications to the nodes in the tree hierarchy. Each node of the tree is provisioned with permission scheme that controls which users/role can perform view, edit, and grant operations on the node.

A custom permission class is used in the policy store to denote permissions on a node. This permission class is oracle.webcenter.avitek.sitemodel.security.NodePermission in the SiteModel project. For example, a permission entry in the application's jazn-data.xml looks like this:

```
<permission>
   <class>oracle.webcenter.avitek.sitemodel.security.NodePermission</class>
   <name>id_news</name>
   <actions>view,edit,grant</actions>
</permission>
```

Node id is used to denote the name of secured resources.

The application also provides UI in SiteManager for delegated security administration. Both the UI as well as the logic can be found in the SiteManager project. This UI is incorporated into SiteManager's property inspector via the access-manager bounded task flow. The task flow is mainly based on /oracle/webcenter/avitek/siteadmin/access-manager.jsff and the logic resides in oracle.webcenter.avitek.siteadmin.security.NodePolicyBean and oracle.webcenter.avitek.siteadmin.securty.NodeSecurityBean.

Pretty URLs

The Pretty-URL capability of the application is implemented via the custom JSF view handler class oracle.webcenter.avitek.sitemodel.navigation.CustomViewHandler in the SiteModel project. The basic idea is to use this view handler to create a separation between the pretty URL used by the browser and the viewId used internally. When a request for the page comes in the form of a pretty URL, the createView(FacesContext ctx, String url) and restoreView(FacesContext ctx, String url) intercept the request. The parameter url will be in the form of pretty-URL. It's the responsibility of createView() and restoreView() to lookup the internal viewId given the pretty URL and create and return the UIViewRoot for the internal viewId.

Partial Page Request (PPR) Navigation

Avitek application employees a feature in ADF Faces that allows all requests between the browser and the application to be carried out in the form of partial page request (PPR). The end result is the page is more responsive.

In order to take advantage of PPR Navigation, the application's web.xml in the ViewController project contains the following flag that tells ADF Faces to use PPR for all requests.

```
<context-param>
  <param-name>oracle.adf.view.rich.pprNavigation.OPTIONS</param-name>
  <param-value>onWithForcePPR</param-value>
</context-param>
```

Other possible values are:

- on - enable PPR Navigation
- onWithForcePPR - enable PPR Navigation and force partialSubmit="true" on all action events
- off - disable PPR Navigation.

The other required task is to implement an ADFc custom navigation handler. An ADFc custom navigation handler allows dynamic creation of ADFc View Activities at runtime. Since a user can create and add pages to the site at runtime, one cannot pre-create all the required flows between the newly created pages at design-time. Likewise, since users can define at runtime where a node navigates to, there is no way to create the corresponding View Activity to represent the node in JDeveloper. The closest one could get would be to use a URL View Activity.

To get around these constraints, ADFc allows definition of the destination View Activity on the fly. ADFc will first check for any valid pre-defined ActivityID and if it can't find one, will call a custom handler to see if it can create the corresponding ActivityID. This handler is registered with the file META-INF/services/oracle.adf.controller.internal.AdfcNavigationHandler inside

the avitek-custom-navigation.jar. In the Avitek application, this custom handler is oracle.webcenter.avitek.sitemodel.navigation.CustomNavigationhandler in the SiteModel project.

Developer's Guide to Avitek, a Sample Custom Portal Built on WebCenter 11g
July 2009

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

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

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

0109