BEA WebLogic Workshop™ Help

Version 8.1 SP4
December 2004

Table of Contents

Developing Portal Applications....................1
Updating Portal Libraries with New Service Packs....................3
Integrating Existing Applications into Portals....................4
Integrating Web Applications....................6
Integrating Java Page Flow Applications....................7
Integrating Struts Applications....................9
Overview of Content Management....................12
Unified User Profiles Overview....................16
Setting up Unified User Profiles....................20
Adding WebLogic Portal Functionality to an Application....................32
Enabling Desktop Selection....................34
Adding Visitor Tools to Portals....................36
Creating URLs to Portal Resources....................39
Developing a New Portal Application....................41
Creating a Portal Application and Portal Web Project....................43
Building Different Types of Applications....................46
Developing Web Applications....................47
Building a Java Page Flow Application....................49
Adding Portal Controls to Java Page Flows....................50
Using Portal Controls....................52
Portal Control Properties....................57
Portal Control Declaration....................59
Portal Control Security....................60

Table of Contents

Group Provider Control....61

Profile Control....62

Property Control....63

Rules Executor Control....64

Rules Manager Control....65

User Info Control....66

User Login Control....67

User Provider Control....68

Click Content Event Control....70

Display Content Event Control....71

Generic Event Control....72

Generic Tracking Control....73

Rule Event Control....74

Session Login Event Control....75

User Registration Event Control....76

Building a Struts Application....77

Building a Commerce Application....78

Adding Commerce Services to an Application....79

Enabling Catalog Management....81

Creating Catalog Structure Properties....82

Creating Discounts....84

Creating Portals for Mobile Devices....86

Developing Personalized Applications....91

Table of Contents

Using Portal JSP Tags....92
Overview of Content Management....95
Unified User Profiles Overview....99
Setting up Unified User Profiles....103
Enabling Desktop Selection....115
Adding Visitor Tools to Portals....117
Creating URLs to Portal Resources....120
Building Portlets....122
Using Portlets from the Portlet Library....123
Login to Portal Portlet....125
Login Director Portlet....127
Targeted Menu Portlet....129
dev2dev Portlet....131
RSS News Feed Portlet....133
Portal Search Portlet....135
My Mail Portlet....137
My Task List Portlet....141
My Calendar Portlet....144
My Contacts Portlet....148
Discussion Forums Portlet....151
Discussion Forum Administration Portlet....155
My Content Portlet....159
Content Management Portlet....161

Table of Contents

Creating Portlets....162

Implementing WSRP−compliant Portlets....165

Building a Remote Portlet....167

Modifying a Remote Portlet....177

Customizing a Remote Portlet....180

Disabling A Producer....181

Building JSP/HTML Portlets....182

Building Java Portlets....183

Building Java Page Flow Portlets....187

Building Struts Portlets....189

Creating a Web Service Portlet....191

How Do I: Create a Personalized Portlet?....192

Adding a Portlet to a Portal....193

Customizing Portlets....194

Setting Portlet Modes and States....195

Setting Portlet Height and Scrolling....198

How Do I: Establish Inter−Portlet Communication?....200

Developing Portal Applications

Developing portal applications involves using the WebLogic Portal framework and tools to surface applications in a portal user interface. It also involves adding personalization, campaigns, and behavior tracking to your applications.

You can quickly and easily integrate your own applications into WebLogic Workshop and apply WebLogic Portal's framework, tools, and services to them, or you can create new portal applications in WebLogic Workshop.

When you have integrated your existing applications into the portal framework or created new portal applications, you can create portlets to surface your application functionality in a portal interface.

Updating Portal Libraries with New Service Packs

Provides instructions on updating the WebLogic Portal libraries in existing portal applications and Web projects when product service packs are released.

Integrating Existing Applications into Portals

Explains how to integrate many types of applications into the WebLogic Workshop development environment and add the portal framework and services to them.

Developing a New Portal Application

Provides instructions on creating a portal framework and building different types of applications that you can surface in a portal interface using the portal framework. This section also includes instructions on managing content and users and adding special features to your portal desktops.

Building Portlets

Provides instructions on creating and customizing portlets, adding portlets from Sample Portal to your portal application, and establishing inter−portlet communication.

Related Topics

Guide to Development with WebLogic Workshop

Developing Personalized Applications

Developing Portal User Interfaces

Assembling Portal Applications

Securing Portal Applications

Deploying Portal Applications

Portal Reference

Updating Portal Libraries with New Service Packs

After you install a new service pack that includes portal library updates, you must update the libraries in the applications you have developed. Updating overwrites the existing libraries. To update your application libraries:

1. Shut down your server if it is running. In WebLogic Workshop, choose ***Tools* −−> *WebLogic Server* −−> *Stop WebLogic Server***.
2. In WebLogic Workshop, open the portal application you want to update.
3. In the Application window, right−click the application directory and choose ***Install* −−> *Update Portal Libraries***.
4. If the service pack includes Commerce or Pipeline updates, right−click the application directory and choose ***Install* −−> *Commerce Services*** and ***Install* −−> *Pipeline Services***.
5. After the portal application libraries are updated, a dialog box appears that lets you select Web projects in the application to update. Select the Web projects whose libraries you want to update, and click ***OK***.
6. In the Application window, right−click the Web project directory and choose ***Install* −−> *Update Portal Libraries***.
7. In the Application window, right−click the Libraries directory and choose ***Add Library***. In the Add Library window, switch to the <app>/APP−INF/lib directory, select all the files in that directory, and click ***Open***. If you are prompted to overwrite existing files, click ***Yes***.
8. If the service pack includes updates to Commerce or Webflow JSP tag libraries, right−click the Web project directory in the Application window and choose ***Install* −−> *Commerce Taglibs*** and ***Install* −−> *Webflow Taglibs***.
9. Restart the server.

Integrating Existing Applications into Portals

The following topics provide instructions on bringing different types of existing applications into the WebLogic Workshop development environment where they can use the portal framework and portal services. If you want to create a new portal application from the ground up, see Developing a New Portal Application.

This section includes the following topics:

Integrating Web Applications

Provides instructions for integrating existing Web applications into the portal framework in the WebLogic Workshop development environment.

Integrating Java Page Flow Applications

Provides instructions for integrating existing Java Page Flow applications into the portal framework in the WebLogic Workshop development environment.

Integrating Struts Applications

Provides instructions for integrating existing Struts applications into the portal framework in the WebLogic Workshop development environment.

Overview of Content Management

Provides instructions and links for setting up content management for use by your applications.

Setting up Unified User Profiles

Shows you how to set up Unified User Profiles, which provide the capability to leverage user data from external sources such as LDAP servers, legacy systems and databases.

Adding WebLogic Portal Functionality to an Application

Describes the WebLogic Portal functionality you can add to your portal−enabled applications, such as personalization and campaigns.

Related Topics

Building Portlets

Developing Portal User Interfaces

Assembling Portal Applications

Securing Portal Applications

Deploying Portal Applications

Portal Reference

Integrating Web Applications

You can integrate, or import, an existing Web application into an enterprise application in WebLogic Workshop. Once in WebLogic Workshop, you can quickly and easily give the Web application a portal user interface, add personalization and campaign functionality to it, and take advantage of WebLogic Portal's content and user management services.

If you want to be able to create portlets out of resources in your Web application, your Web application must have one of the following types of resources out of which to create portlets. If you do not have any of these types of resources at the time you integrate your Web application into WebLogic Workshop, you can create them after you integrate:

- Java Page Flow application (developed in WebLogic Workshop)
- Struts application
- Java Portlet (JSR 168 compliant)
- Java Server Pages (JSPs)

To integrate an existing Web application into WebLogic Workshop:

1. In WebLogic Workshop, open your application (.work file).
2. In the Application window, right−click the application directory and choose ***Import*** **−−>** ***Import Project***. The Import Project window appears.
3. In the right pane of the window, select ***Web Project***.
4. Click the ***Browse*** button in the Directory field and select the Web application's root directory.
5. Make sure the ***Copy into Application directory*** option is selected.
6. You can change the directory name of the Web application in the ***Name*** field. The name you use is part of the URL used to access the Web application.
7. Click ***Import***.

   After you import your Web application directory into WebLogic Workshop, a dialog box may appear that asks you if you want to add missing files to the Web project. Click ***No***.
8. Install portal in the application and Web application:
   a. In the Application window, right−click the application directory and choose ***Install*** **−−>** ***Portal***.
   b. Right−click the Web application directory and choose ***Install*** **−−>** ***Portal***.

Your Web application is now a portal Web project. You can give it a portal interface, add portal functionality, and build portlets (assuming you have one of the resources types listed at the beginning of this topic).

Related Topics

Developing Web Applications

Building Portlets

Integrating Java Page Flow Applications

Java Page Flows are a native feature of WebLogic Platform. Page Flows provide an event−driven flow through an application. Page Flows let you separate the user interface code from navigational control and other business logic.

This topic shows you how to integrate Page Flows into a portal application so that you can surface them in a portal interface.

There are two scenarios for integrating Page Flows:

- Build a Page Flow in a non−portal application in WebLogic Workshop
- Build a Page Flow in a portal application in WebLogic Workshop

To build a Page Flow in a non−portal application in WebLogic Workshop

To add portal functionality to your Page Flow application or surface Page Flows in portlets, you must install portal in the application and project containing the Page Flow application (unless the application is already a portal application and the project is already a portal Web project).

To use Page Flows in a portal environment:

1. Install Portal in the application and project. See Creating a Portal Application and Portal Web Project.
2. In order for URLs in the Page Flows to resolve correctly, Page Flow support must be enabled in the portal Web project's WEB−INF/netuix−config.xml file, as shown in the following example. Notice the <enable> element is set to true.
   ```
   <!-- Enable or disable Pageflow support -->
   <pageflow>
       <enable>true</enable>
   </pageflow>
   ```

   If this block is not present in netuix−config.xml, do not add it. Without the block, the setting defaults to true.

You can now give your Page Flow application a portal interface, build portlets for it, and add portal functionality to it.

To build a Page Flow in a portal application in WebLogic Workshop

In an existing portal application, you already have the necessary files and services to surface your Page Flows in a portal interface. All you must do is build Page Flows in the portal application and take the necessary steps to surface them in portlets.

1. In any portal Web project in the application, create a Page Flow. See Getting Started with Page Flows.
2. In order for URLs in the Page Flows to resolve correctly, Page Flow support must be enabled in the portal Web project's WEB−INF/netuix−config.xml file, as shown in the following example. Notice the <enable> element is set to true.
   ```
   <!-- Enable or disable Pageflow support -->
   <pageflow>
       <enable>true</enable>
   </pageflow>
   ```

If this block is not present in netuix−config.xml, do not add it. Without the block, the setting defaults to true.

You can now give your Page Flow application a portal interface, build portlets for it, and add portal functionality to it.

Related Topics

Building Java Page Flow Portlets

How Do I: Add Portal Functionality to an Existing Page Flow Application?

Integrating Struts Applications

You can integrate, or import, a Struts 1.1 application into an enterprise application in WebLogic Workshop. Once in WebLogic Workshop, you can quickly and easily give the Struts application a portal user interface, add personalization and campaign functionality to it, and take advantage of WebLogic Portal's content and user management services.

This topic contains the following sections:

Integrating a Struts Application into a Portal

Best Practices and Development Issues

Struts and Page Flows

Integrating a Struts Application into a Portal

1. Create a portal application and portal Web project in which to add the Struts application. See Creating a Portal Application and Portal Web Project. Struts support is added automatically.
2. In order for URLs to resolve correctly, Java Page Flow support must be enabled in the portal Web project's WEB−INF/netuix−config.xml file, as shown in the following example. Notice the <enable> element is set to true.

   ```
   <!-- Enable or disable Pageflow support -->
   <pageflow>
       <enable>true</enable>
   </pageflow>
   ```

   If this block is not present in netuix−config.xml, do not add it. Without the block, the setting defaults to true.
3. Add the Struts application to the portal Web project.
   a. Copy any JSP, HTML, or image files into the portal Web project following the standard Struts module directory structure (the module path is the directory path relative to the Web application root).
   b. Copy any supporting Java source used by the Struts application into the project's WEB−INF/src.
   c. Copy any necessary custom JARs for the Struts application into WEB−INF/lib.
   d. Copy the Struts application's struts−config.xml or module configuration file into WEB−INF, but rename it struts−auto−config−<module−path>.xml, where <module−path> is the module path to the Struts application relative to the Web application root, with all instances of '/' or '\' changed to '−'.

      For example, if the module path is /struts/my/module, struts−config.xml should be renamed to struts−auto−config−struts−my−module.xml. Naming the module configuration file in this manner enables the PageFlowctionServlet used as the Action Servlet to automatically register the module without explicitly registering it with an init−param in web.xml. If you don't want to take advantage of this functionality, you can rename struts−config.xml arbitrarily, but you must manually register the module in web.xml as usual for a Struts 1.1 module.
   e. In the module configuration file, add the following line to configure the RequestProcessor that is required for portal integration:

   <controller processorClass="com.bea.struts.adapter.action.AdapterRequestProcessor"/>

   (unless the Struts application requires a custom RequestProcessor).
4. Create a portlet that contains a StrutsContent control that specifies the module and the default action for the Struts application. See Building Portlets.
5. Add the new portlet to the portal using the WebLogic Workshop Portal Designer. See Adding a Portlet to a Portal.

Best Practices and Development Issues

Use the following guidelines for integrating Struts applications in portals:

- It is highly recommended that you fully develop and test a Struts application before attempting to host it within a portal. This will help to separate the complexities of simply developing a working Struts application from the additional issues involved in putting the Struts application into a portlet.
- Any Struts applications that are intended for use in a Portal *must* be developed as Struts modules, including the usage of the html:link tag for any URLs used in JSPs. Without this, it is impossible for the portal framework to perform the necessary URL rewriting that is required to transparently modify links when the Struts application is used within a portlet.
- If you encounter stack traces or messages in the Struts application portlet showing that an action cannot be found, ensure that the module is correctly configured, named correctly, and registered in web.xml. This can be tested by running the Struts application stand−alone
- If you encounter resource not found exceptions or class not found exceptions for dependent classes:
    - Make sure that all dependent Java source exists in WEB−INF/src, and that it has successfully been built into the corresponding class files in WEB−INF/classes.
    - If more than one message−resource element is specified in the Struts configuration file for the module, any module files that reference a non−default message bundle must append the module path to the bundle key. For example, if the bundle key is alternate, and the module is /my/module, any users of the bundle will have to fully qualify it as alternate/my/module.
- If following action links in a Struts portlet results in full−screen, stand−alone Struts pages, make sure that struts−adapter JSP tag libraries are in the project's WEB−INF/lib directory and that they are registered in web.xml.
- If the"No ActionResult returned for action" error is returned when the action attribute of an html:form element contains a query parameter, use a hidden html:text input field.
- Some Struts applications use of a custom RequestProcessor. Portal Struts integration support requires that certain behaviors of a RequestProcessor be overridden. The class com.bea.struts.adapter.action.AdapterRequestProcessor, located in struts−adapter.jar, provides this standard behavior and must be used in all Struts applications used within a portal. Any custom RequestProcessors must either extend this class or use a utility class to perform the same required operation that this RequestProcessor performs. When extending this class, overrides of doForward() *must* call the superclass doForward() and also must not attempt to write to the response. Custom RequestProcessors that do not extend AdapterRequestProcessor must call com.bea.struts.adapter.action.AdapterRequestProcessorUtil.forwardUsingRequest() to perform any forwarding operations. (This method replaces an actual RequestDispatcher forward request with an operation that simply captures the forward URI for later use in including the URI into the portal output.)
- If a Struts application depends on the use of a custom Action servlet, it must be refactored to use a custom RequestProcessor instead, as outlined above, and as recommended by the Struts 1.1 implementation. Since the Page Flow functionality in WebLogic Portal uses a custom Action servlet,

and since there can be only one Action servlet in a portal Web project, portal Struts integration requires that the Action servlet not be customized. For more information on refactoring an Action servlet customization into a RequestProcessor customization, see the Struts 1.1 documentation at http://jakarta.apache.org/struts/.

- Module switching − The StrutsContent control supports module switching using Action forwards, as described in the Struts documentation found on http://jakarta.apache.org/struts/. If the Action forward returned by an invoked Action results in a content URI that resides in another module, the current module will be switched to the corresponding new module, and all further requests to the Struts portlet containing the control will be performed using the new module. Module switching should only be done using Action forwards, *not* by using the <html:link> tag to directly link to a JSP in another module; doing so may prevent the portal and Struts frameworks from correctly setting up and selecting the module.
- If a Struts application used within a Portal also needs to support stand−alone operation, JSPs referenced by Action forwards must be authored to use several optional tags in the HTML tag library found in struts.jar and struts−adapter.jar. The first of these, <html:html>, is found in both Struts and the Struts−adapter. The Struts−adapter version overrides the Struts version of the tag and adds support for detecting whether or not to inhibit rendering of the tag output text if it is used from within a portal, where outputting the HTML text would result in non−well−formed HTML. Two additional tags are provided in the Struts−adapter version of the HTML tag library, and should be used in JSPs that also need to be used stand−alone: <html:head> and <html:body>. These two tags have the same portal−aware rendering behavior as the <html:html> tag.

Struts and Page Flows

WebLogic Workshop provides interchangeable support for Struts modules and Page Flow controller classes working together in the same Web project. See Interoperating With Struts and Page Flows.

Related Topics

Advantages of Using Page Flows

Building Portlets

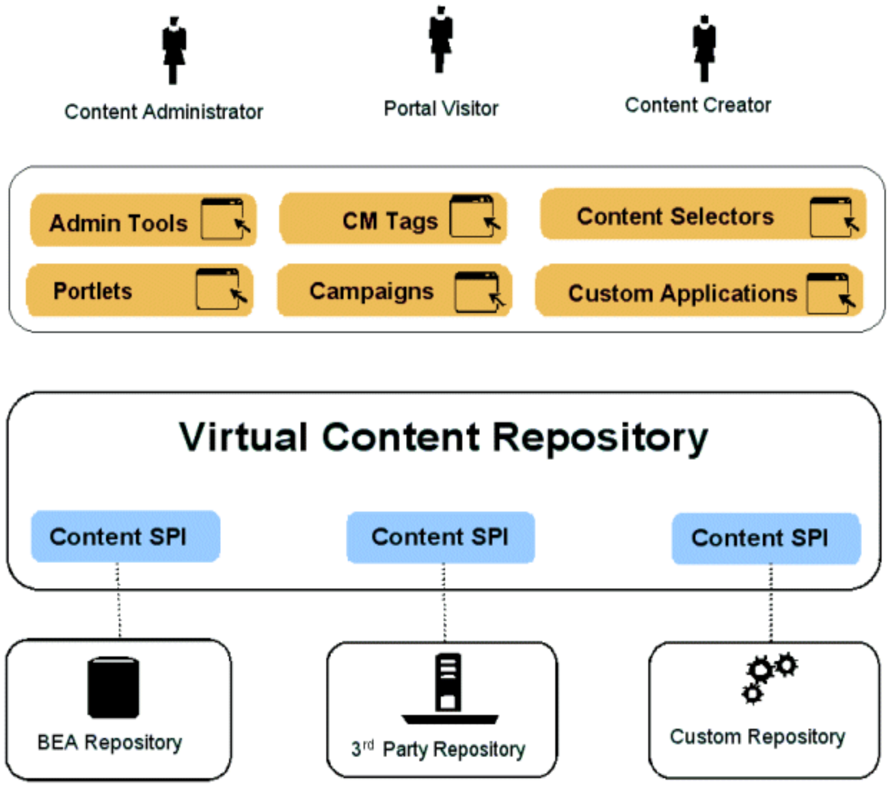
Overview of Content Management

The content you want to show users, whether it is a single line of text, an HTML file, a graphic, or an animation file can be stored in a content repository. BEA's Virtual Content Repository, included with WebLogic Portal, provides a single interface that lets you store content in BEA repositories as well as seamlessly incorporate BEA−compatible third−party content management systems. This overview provides information on the following subjects:

- The Virtual Content Repository
- Content Hierarchy
- Content Types
- Creating and Modifying Content
- Using Content in Personalized Applications

The Virtual Content Repository

The Virtual Content Repository can contain multiple content repositories. It provides services such as federated search (a search that returns a result set from all the relevant content across the plugged in repositories), content lifecycle management, Delegated Administration and content type management. Many Portal subsystems interact with the Virtual Content Repository. Content Management tags execute queries to deliver dynamic content to end users. Content Selectors and Campaigns deliver dynamic, personalized content to user based upon personalization rules or conditions.



The Content Hierarchy

WebLogic Portal Content Management is organized hierarchically. The Virtual Content Repository (VCR) is the top−level node in the content management system. Repositories are the immediate children of the VCR. These repositories can be made up of multiple BEA Systems repositories, multiple third−party repositories, or custom content repositories.

Hierarchy Nodes and Content Nodes comprise the next level of the hierarchy tree and are organized much like a file system. Hierarchy Nodes can contain both Hierarchy Nodes and Content Nodes. Content Nodes can only contain other Content Nodes. Nodes can be created based upon Content Types. For example:

Virtual Content Repository

Repository 1

Hierarchy Node

ContentNode (index.htm)

ChildContent1 (logo.gif)
ChildContent2 (photo.jpg)

Content Repositories provide the storage mechanism for content, and they comprise the second−level of the Virtual Content Repository hierarchy. Content Repositories may include multiple instances of BEA repositories, 3rd party repositories, or customer repositories. To plug into the Virtual Content Repository, you must implement the BEA Content Management Service Provider Interface   the CM SPI.

Hierarchy Nodes are organizational mechanisms that help you organize and group content in the hierarchy, much like folders in a file system. Hierarchy Nodes can contain other Hierarchy Nodes as well as Content Nodes. They can also be typed so that they function similarly to Content Nodes.

Content Nodes represent content stored in the repository. A complete content node comprises a set of data property values defined by a content type. This data structure may include files such as a word processing document, HTML file, spreadsheet or image. It may also include metadata such as the author, version number or summary. Content Nodes can also have child Content Nodes. For example, The Content Node for an HTML document may have child Content Nodes for the images used by the HTML document.

Content Types

Content Types define the set of properties that make up a Content Node or Hierarchy Node. This may include any combination of the supported data types, such as date and time, number, text (string), Boolean (true/false), or binary (file).

For example, the Content Type for image content may have a number property "width" and a number property "height," while the Content Type for news article content my have a text property "Author", a text property "Summary", a date property "Published Date", and a binary property "Article" for a file containing the formatted article. Types do not have to include a binary, although a common example of a type is a single binary with a set of non−binary properties that describe the document.

Repository 1

  Content Type 1

    Property 1 = Binary
    Property 2 = String

  Content Type 2

Content Types also define the available values for a given property, including whether it can contain multiple values. For example, a property called "Priority" may only allow a single choice among the values "High", "Medium", and "Low", while a property called "Favorite Color" may allow multiple pre−defined values to be chosen.

Each repository has its own set of content types. You can create types in BEA repositories and third−party repositories that support this feature.

Creating and Modifying Content

After you connect a BEA−compatible content management system to the Virtual Content Repository you can continue to add and modify content directly in your BEA−compatible content management system. Changes appear automatically in the Virtual Content Repository. You can create and manage content in the Administration Portal, in the My Content Portlet, or with the bulkloader. For more information, see "Creating Content."

Using Content in Personalized Applications

WebLogic Workshop extensions support development of personalized applications, while the WebLogic Administration Portal enables portal administrators to adapt site interaction to fit the needs of the audience. The core of the Personalization system is the underlying rules engine that matches users with appropriate content. Content Selectors, Placeholders and Campaigns are the aspects of content management visible to administrators. Also, User Segments contain the criteria that define the target visitor, such as gender or browser type.

The Content Management component provides the run−time API by which content is queried and retrieved. The functionality of this component is accessible via tags. The content retrieval functionality is provided using either the provided reference implementation or third−party content retrieval products.

Related Topics

Creating Content

Setting up Users and User Properties

Designing Interaction Management

Creating Personalization Conditions

Personalizing Portal Applications

Unified User Profiles Overview

If you have an existing store of users, groups, and additional properties (such as address, e−mail address, phone number, and so on), unified user profiles are a necessary part of bringing those user properties into the WebLogic Portal environment, where they can be used for retrieving and editing property values and setting up personalization, delegated administration, and visitor entitlements.

This topic describes the unified user profile, when to use it, and when not to use it.

Note: This topic contains the terms "user store" and "data store." A user store can contain users and groups, as well as additional properties. A data store implies that the store does not have to contain users and groups. It can simply contain properties.

What is a Unified User Profile?

Here is an example that explains what a unified user profile is and does:

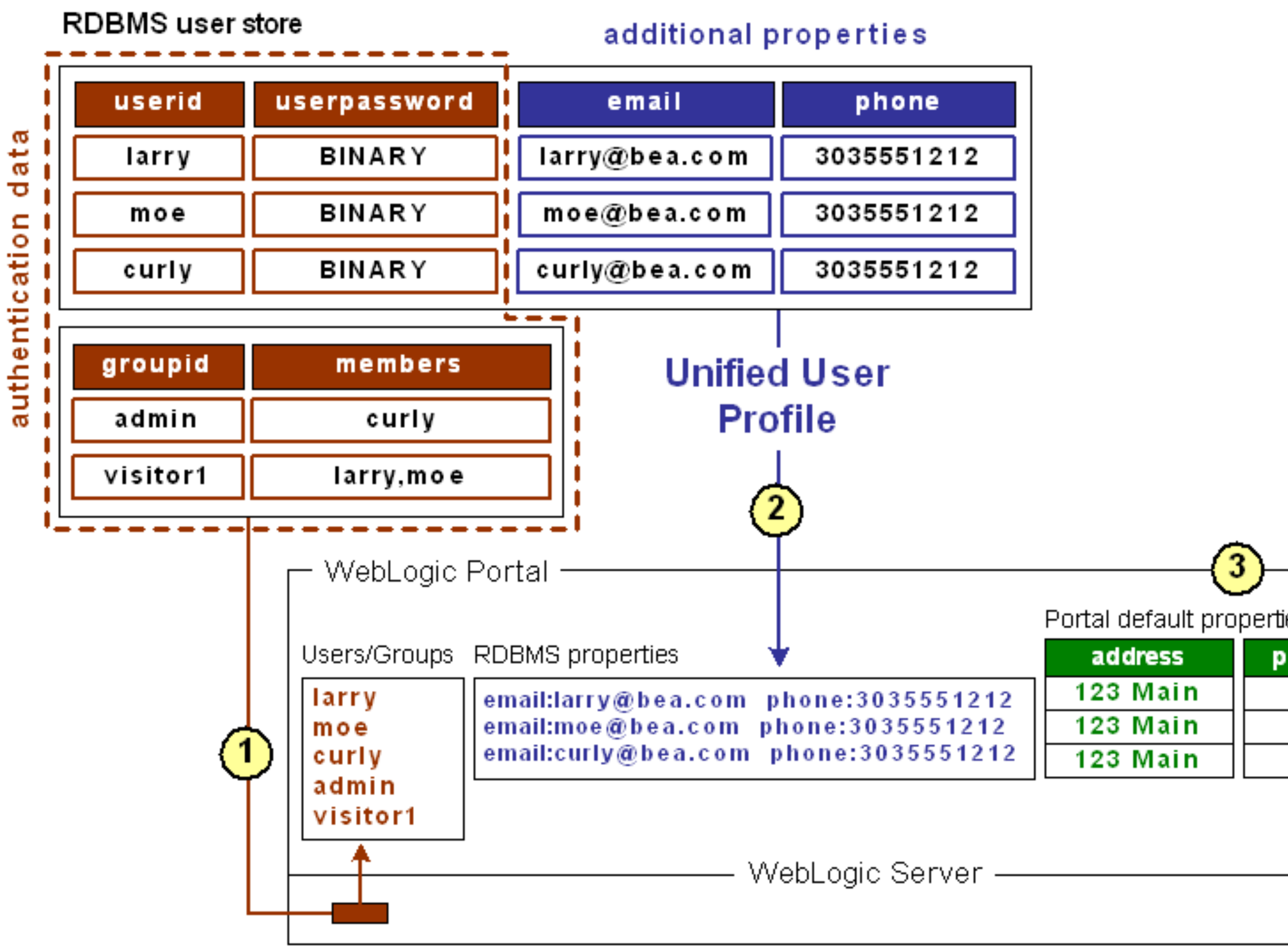
Let's say you're creating a new portal application that you want users to be able to log in to. Let's also say your users are stored in an RDBMS user store outside of the WebLogic environment. You could connect WebLogic Server (your portal application's domain server instance) to your RDBMS system, and your users could log in to your portal application as if their usernames and passwords were stored in WebLogic Server. If authentication was all you wanted to provide through your RDBMS user store, you could stop here without needing a unified user profile.

However, let's say you also stored e−mail and phone number information (properties) for users in your RDBMS user store, and you wanted to be able to access those properties in your portal applications. In this case, you need to create a unified user profile for your RDBMS user store that lets you access those additional properties from your code.

Technically speaking, a unified user profile is a stateless session bean you create (with associated classes) that lets WebLogic Portal read property values stored in external data stores, such as LDAP servers and databases. Once connected to an external data store with a unified user profile, you can use portal JSP tags, controls, and the WebLogic Portal API to retrieve user property values from that store. You can also take the extra step of surfacing these external properties in the WebLogic Administration portal, where the properties can be used to define rules for personalization, visitor entitlements, and delegated administration.

Whether or not you have additional properties stored in your external user store, the external users and groups you connect to WebLogic Server are automatically assigned the default user property values you have set up in WebLogic Portal, without the use of a unified user profile. With the WebLogic Administration Portal, you can change the default WebLogic Portal property values for those users. These values are stored in WebLogic Portal's RDBMS data store using the Portal schema.

The following figure shows where a unified user profile fits between an external user store and the WebLogic environment.



1	This external RDBMS user store, which supports authentication, contains users (principals) and passwords in one database table and groups (principals) in another. Giving a user store authentication capabilities (as an authentication provider or identity asserter) involves configuration steps not associated with the unified user profile configuration process. (See Developing Security Providers for WebLogic Server.) Unified user profile configuration is not dependent on the authentication provider configuration and vice versa. Once the RDBMS authentication provider is connected to WebLogic Server, WebLogic Server (and WebLogic Portal) can see those users and groups. Those users can log in to your portal applications, and you can include those users and groups in your rules for personalization, delegated administration, and visitor entitlements. Also, WebLogic Portal's ProfileWrapper maps the principals to properties kept in the Portal schema, thereby establishing the user profile.
2	***Unified User Profile*** − The same external table that contains users and passwords also contains additional properties (email and phone) for each user. These additional properties are not part of authentication; but they are

	part of each user's profile. If you want to access these properties in your portal applications (with the WebLogic Portal JSP tags, controls, or API), you must create a unified user profile for the RDBMS user store. When you create the unified user profile, the ProfileWrapper includes the external properties in the user profile. The unified user profile consists of a stateless session bean and associated classes that you create. If you want to surface any of these properties in the WebLogic Administration Portal to be used in defining rules for personalization, delegated administration, or visitor entitlements, create a user profile property set for the external user store in addition to implementing your unified user profile session bean. The property set provides metadata about your external properties so that WebLogic Workshop and the WebLogic Administration Portal know how to display them. Properties from an external data store are typically read only in the WebLogic Administration Portal.
3	WebLogic Portal lets you create user/group properties and set default values for those properties. Any user or group in WebLogic Server, whether created in the default LDAP store or brought in through a connection to an external user store, is automatically assigned those default property values; and you can change the default values for each user or group, programmatically or in the WebLogic Administration Portal. This does not involve unified user profiles, because the properties to be retrieved are local, not stored in an external data store. In the illustration, after the authentication provider or identity asserter provides the principals, the ProfileWrapper combines the principals with the external properties of email and phone (retrieved by the unified user profile) and the default WebLogic Portal properties of address and postal code, all of which make up the full user profile.

What a Unified User Profile is Not

A user profile is not a security realm, and it does not provide authentication. It is not even the external user store itself. It is the connection (stateless session bean with associated classes) that lets you read properties in the external user store.

When Should You Create a Unified User Profile?

Create a unified user profile for an external data store if you want to do any of the following:

- Use WebLogic Portal's JSP tags, controls, or API to retrieve property values from that external store.
- Surface external properties in the WebLogic Administration Portal for use in defining rules for personalization, delegated administration, or visitor entitlements. Users and groups are not considered properties.

When Don't You Need a Unified User Profile?

You do not need to create a unified user profile for an external data store if you only want to:

- Provide authentication for users in the external user store.
- Define rules for personalization, delegated administration, or visitor entitlements based only on users or groups in an external user store, not on user properties.
- Define rules for personalization, delegated administration, or visitor entitlements based on the WebLogic Portal user profile properties you create in WebLogic Workshop, which are kept in the Portal schema.

Setting up a Unified User Profile

See Setting up Unified User Profiles.

Related Topics

Using Multiple Authentication Providers in Portal Development (external user stores)

Setting up Unified User Profiles

This topic provides guidelines and instructions on creating a unified user profile to access user/group properties from an external user store. (See Unified User Profiles Overview for overview information.)

Best Practices: When possible, use WebLogic Portal's user profile functionality (default UserProfileManager) to assign properties to users and groups. Given the choice between creating and storing additional properties in an external user store (which requires write access to that external store, which must be implemented) and creating and storing them in WebLogic Portal, doing so in WebLogic Portal can greatly improve performance on accessing property values. If you are storing users and groups in an external store, the ideal configuration is storing only users, groups, and passwords in the external store and creating and setting additional properties in WebLogic Portal. With that configuration, performance is optimal and you do not have to create a unified user profile.

To create a UUP to retrieve user data from external sources, complete the following tasks:

Create an EntityPropertyManager EJB to Represent External Data

Deploy a ProfileManager That Can Use the New EntityPropertyManager

If you have an LDAP server for which you want to create a unified user profile, WebLogic Portal provides a default unified user profile you can modify. See Retrieving User Profile Data from LDAP.

Create an EntityPropertyManager EJB to Represent External Data

To incorporate data from an external source, you must first create a stateless session bean that implements the methods of the com.bea.p13n.property.EntityPropertyManager remote interface. EntityPropertyManager is the remote interface for a session bean that handles the persistence of property data and the creation and deletion of profile records. By default, EntityPropertyManager provides read−only access to external properties.

In addition, the stateless session bean should include a home interface and an implementation class. For example:

```
MyEntityPropertyManager
extends com.bea.p13n.property.EntityPropertyManager

MyEntityPropertyManagerHome
extends javax.ejb.EJBHome
```

Your implementation class can extend the EntityPropertyManagerImpl class. However the only requirement is that your implementation class is a valid implementation of the MyEntityPropertyManager remote interface. For example:

```
MyEntityPropertyManagerImpl extends
com.bea.p13n.property.internal.EntityPropertyManagerImpl
```

or

```
MyEntityPropertyManagerImpl extends
javax.ejb.SessionBean
```

Recommended EJB Guidelines

We recommend the following guidelines for your new EJB:

- Your custom EntityPropertyManager is not a default EntityPropertyManager. A default EntityPropertyManager is used to get/set/remove properties in the Portal schema. Your custom EntityPropertyManager does not have to support the following methods. It can throw java.lang.UsupportedOperationException instead:
    - getDynamicProperties()
    - getEntityNames()
    - getHomeName()
    - getPropertyLocator()
    - getUniqueId()
- If you want to be able to use the portal framework and tools to create and remove users in your external data store, you must support the createUniqueId() and removeEntity() methods. However, your custom EntityPropertyManager is not the default EntityPropertyManager so your createUniqueId() method does not have to return a unique number. It must create the user entity in your external data store and then it can return any number, such as −1.
- The following recommendations apply to the EntityPropertyManager() methods that you must support:
    - getProperty() − Use caching. You should support the getProperties() method to retrieve all properties for a user at once, caching them at the same time. Your getProperty() method should use getProperties().
    - setProperty() − Use caching.
    - removeProperties(), removeProperty() − After these methods are called, a call to getProperty() should return null for the property. Remove properties from the cache, too.
- Your implementations of the getProperty(), setProperty(), removeProperty(), and removeProperties() methods must include any logic necessary to connect to the external system.
- If you want to cache property data, the methods must be able to cache profile data appropriately for that system. (See the com.bea.p13n.cache package in the WebLogic Portal Javadoc.)
- If the external system contains read−only data, any methods that modify profile data must throw a java.lang.UnsupportedOperationException. Additionally, if the external data source contains users that are created and deleted by something other than your WebLogic Portal createUniqueId() and removeEntity() methods can simply throw an UnsupportedOperationException.
- To avoid class loader dependency issues, make sure that your EJB resides in its own package.
- For ease of maintenance, place the compiled classes of your custom EntityPropertyManager bean in your own JAR file (instead of modifying an existing WebLogic Portal JAR file).

Before you deploy your JAR file, follow the steps in the next section.

Deploy a ProfileManager That Can Use the New EntityPropertyManager

A "user type" is a mapping of a ProfileType name to a particular ProfileManager. This mapping is done in the UserManager EJB deployment descriptor.

To access the data in your new EntityPropertyManager EJB, you must do ***one*** of the following:

- Modifying the Existing ProfileManager Deployment Configuration − In most cases you will be able to use the default deployment of ProfileManager, the UserProfileManager. You will modify the UserProfileManager's deployment descriptor to map a property set and/or properties to your custom EntityPropertyManager. If you support the createUniqueId() and removeEntity() methods in your

custom EntityPropertyManager, you can use WebLogic Administration Portal to create a user of type "User" with a profile that can get/set properties using your custom EntityPropertyManager.
- Configuring and Deploying a New ProfileManager − In some cases you may want to deploy a newly configured ProfileManager that will be used instead of the UserProfileManager. This new ProfileManager is mapped to a ProfileType in the deployment descriptor for the UserManager. If you support the createUniqueId() and removeEntity() methods in your custom EntityPropertyManager, you can use the WebLogic Administration Portal (or API) to create a user of type "MyUser" (or anything else you name it) that can get/set properties using the customized deployment of the ProfileManager that is, in turn, configured to use your custom EntityPropertyManager.

ProfileManager is a stateless session bean that manages access to the profile values that the EntityPropertyManager EJB retrieves. It relies on a set of mapping statements in its deployment descriptor to find data. For example, the ProfileManager receives a request for the value of the "DateOfBirth" property, which is located in the "PersonalData" property set. ProfileManager uses the mapping statements in its deployment descriptor to determine which EntityPropertyManager EJB contains the data.

Modifying the Existing ProfileManager Deployment Configuration

If you use the existing UserProfileManager deployment to manage your user profiles, perform the following steps to modify the deployment configuration.

Under most circumstances, this is the method you should use to deploy your UUP. An example of this method is the deployment of the custom EntityPropertyManager for LDAP property retrieval, the LdapPropertyManager. The classes for the LdapPropertyManager are packaged in p13n_ejb.jar. The deployment descriptor for the UserProfileManager EJB is configured to map the "ldap" property set to the LdapPropertyManager. The UserProfileManager is deployed in p13n_ejb.jar.

1. Back up the p13n_ejb.jar file in your enterprise application root directory.
2. From p13n_ejb.jar, extract META−INF/ejb−jar.xml and open it for editing.
3. In ejb−jar.xml, find the <env−entry> element, as shown in the following example:
   ```
   <!−− map all properties in property set ldap to ldap server −−>
   <env−entry>
       <env−entry−name>PropertyMapping/ldap</env−entry−name>
       <env−entry−type>java.lang.String</env−entry−type>
       <env−entry−value>LdapPropertyManager</env−entry−value>
   </env−entry>
   ```

   and add an <env−entry> element after this to map a property set to your custom EntityPropertyManager, a shown in the following example:
   ```
   <!−− map all properties in UUPExample property set to MyEntityPropertyManager −−>
   <env−entry>
       <env−entry−name>PropertyMapping/UUPExample</env−entry−name>
       <env−entry−type>java.lang.String</env−entry−type>
       <env−entry−value>MyEntityPropertyManager</env−entry−value>
   </env−entry>
   ```
4. In ejb−jar.xml, find the <ejb−ref> element shown in the following example:
   ```
   <!−− an ldap property manager −−>
   <ejb−ref>
       <ejb−ref−name>ejb/LdapPropertyManager</ejb−ref−name>
       <ejb−ref−type>Session</ejb−ref−type>
       <home>com.bea.p13n.property.EntityPropertyManagerHome</home>
       <remote>com.bea.p13n.property.EntityPropertyManager</remote>
   </ejb−ref>
   ```

and add an <ejb−ref> element after this to map a reference to an EJB that matches the name from the previous step with ejb/ prepended as shown in the following example:

```
<!-- an example property manager -->
<ejb-ref>
    <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>examples.usermgmt.MyEntityPropertyManagerHome</home>
    <remote>examples.usermgmt.MyEntityPropertyManager</remote>
</ejb-ref>
```

The home and remote class names match the classes from your EJB JAR file for your custom EntityPropertyManager.

5. If your EntityPropertyManager implementation handles creating and removing profile records, you must also add Creator and Remover entries. For example:

```
<env-entry>
    <env-entry-name>Creator/Creator1</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>

<env-entry>
    <env-entry-name>Remover/Remover1</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
```

   This instructs the UserProfileManager to call your custom EntityPropertyManager when creating or deleting user profile records. The names "Creator1" and "Remover1" are arbitrary. All Creators and Removers will be iterated through when the UserProfileManager creates or removes a user profile. The value for the Creator and Remover matches the ejb−ref−name for your custom EntityPropertyManager without the ejb/ prefix.
6. From p13n_ejb.jar, extract META−INF/weblogic−ejb−jar.xml and open it for editing.
7. In weblogic−ejb−jar.xml, find the elements shown in the following example:

```
<weblogic-enterprise-bean>
    <ejb-name>UserProfileManager</ejb-name>
    <reference-descriptor>
        <ejb-reference-description>
            <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
            <jndi-name>${APPNAME}.BEA_personalization. EntityPropertyManager</jndi-name>
        </ejb-reference-description>
```

   and add an ejb−reference−description to map the ejb−ref for your custom EntityPropertyManager to the JNDI name. This JNDI name must match the name you assigned in weblogic−ejb−jar.xml in the JAR file for your customer EntityPropertyManager. It should look like the following example:

```
<weblogic-enterprise-bean>
    <ejb-name>UserProfileManager</ejb-name>
    <reference-descriptor>
        <ejb-reference-description>
            <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
            <jndi-name>${APPNAME}.BEA_personalization. EntityPropertyManager</jndi-name>
        </ejb-reference-description>
        <ejb-reference-description>
            <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
            <jndi-name>${APPNAME}.BEA_personalization. MyEntityPropertyManager</jndi-nam
        </ejb-reference-description>
```

Note the ${APPNAME} string substitution variable. The WebLogic EJB container automatically substitutes the enterprise application name to scope the JNDI name to the application.

8. Update p13n_ejb.jar for your new deployment descriptors. You can use the jar uf command to update the modified META−INF/ deployment descriptors.
9. Edit your application's META−INF/application.xml to add an entry for your custom EntityPropertyManager EJB module as shown in the following example:
   ```
   <module>
       <ejb>UUPExample.jar</ejb>
   </module>
   ```
10. If you are using an application−wide cache, you can manage it from the WebLogic Administration Console if you add a <Cache> tag for your cache to the META−INF/application−config.xml deployment descriptor for your enterprise application like this:
    ```
    <Cache Name="UUPExampleCache" TimeToLive="60000"/>
    ```
11. Verify the modified p13n_ejb.jar and your custom EntityPropertyManager EJB JAR archive are in the root directory of your enterprise application and start WebLogic Server.
12. Use the WebLogic Server Administration Console to verify your EJB module is deployed to the enterprise application and then use the console to add your server as a target for the EJB module. You need to select a target to have your domain's config.xml file updated to deploy your EJB module to the server.
13. Use the WebLogic Workshop Property Set Designer to create a User Profile (property set) that matches the name of the property set that you mapped to your custom EntityPropertyManager in ejb−jar.xml for the UserProfileManager (in p13n_ejb.jar). You could also map specific property names in a property set to your custom EntityPropertyManager, which would allow you to surface the properties and their values in the WebLogic Administration Portal for use in creating rules for personalization, delegated administration, and visitor entitlements.

Your new Unified User Profile type is ready to use. You can use the WebLogic Administration Portal to create a user, and it will use your UUP implementation when the "UUPExample" property set is being modified. When you call createUser("bob", "password") or createUser("bob", "password", null) on the UserManager, several things will happen:

- A user named "bob" is created in the security realm.
- A WebLogic Portal Server profile record is created for "bob" in the user store.
- If you set up the Creator mapping, the UserManager will call the default ProfileManager deployment (UserProfileManager) which will call your custom EntityPropertyManager to create a record for Bob in your data source.
- Retrieving Bob's profile will use the default ProfileManager deployment (UserProfileManager), and when you request a property belonging to the "UUPExample" property set, the request will be routed to your custom EntityPropertyManager implementation.

Configuring and Deploying a New ProfileManager

If you are going to deploy a newly configured ProfileManager instead of using the default ProfileManager (UserProfileManager) to manage your user profiles, perform the following steps to modify the deployment configuration. In most cases, you will not have to use this method of deployment. Use this method only if you need to support multiple types of users that require different ProfileManager deployments deployments that allow a property set to be mapped to different custom EntityPropertyManagers based on ProfileType.

An example of this method is the deployment of the custom CustomerProfileManager in customer.jar. The CustomerProfileManager is configured to use the custom EntityPropertyManager (CustomerPropertyManager) for properties in the "CustomerProperties" property set. The UserManager EJB in p13n_ejb.jar is configured to map the "WLCS_Customer" ProfileType to the custom deployment of the ProfileManager, CustomerProfileManager.

To configure and deploy a new ProfileManager, use this procedure.

1. Back up the p13n_ejb.jar file in your enterprise application root directory.
2. From p13n_ejb.jar, extract META−INF/ejb−jar.xml, and open it for editing.
3. In ejb−jar.xml, copy the entire <session> tag for the UserProfileManager, and configure it to use your custom implementation class for your new deployment of ProfileManager.
   In addition, you could extend the UserProfileManager home and remote interfaces with your own interfaces if you want to repackage them to correspond to your packaging (for example., examples.usermgmt.MyProfileManagerHome, examples.usermgmt.MyProfileManager).
   However, it is sufficient to replace the bean implementation class:
   You must create an <env−entry> element to map a property set to your custom EntityPropertyManager. You must also create a <ejb−ref> element to map a reference to an EJB that matches the name from the PropertyMapping with ejb/ prepended. The home and remote class names for your custom EntityPropertyManager match the classes from your EJB JAR file for your custom EntityPropertyManager.

   Also, if your EntityPropertyManager implementation handles creating and removing profile records, you must also add Creator and Remover entries. This instructs your new ProfileManager to call your custom EntityPropertyManager when creating or deleting user profile records.

   Note: The name suffixes for the Creator and Remover, "Creator1" and "Remover1", are arbitrary. All Creators and Removers will be iterated through when your ProfileManager creates or removes a user profile. The value for the Creator and Remover matches the <ejb−ref−name> for your custom EntityPropertyManager without the ejb/ prefix.
4. In ejb−jar.xml, you must add an <ejb−ref> to the UserManager EJB section to map your ProfileType to your new deployment of the ProfileManager, as shown in the following example:
   ```
   <ejb−ref>
       <ejb−ref−name>ejb/ProfileType/UUPExampleUser</ejb−ref−name>
       <ejb−ref−type>Session</ejb−ref−type>
       <home>com.bea.p13n.usermgmt.profile.ProfileManagerHome</home>
       <remote>com.bea.p13n.usermgmt.profile.ProfileManager</remote>
   </ejb−ref>
   ```

   The <ejb−ref−name> must start with ejb/ProfileType/ and must end with the name that you want to use as the profile type as an argument in the createUser() method of UserManager.
5. From p13n_ejb.jar, extract META−INF/weblogic−ejb−jar.xml and open it for editing.
6. In weblogic−ejb−jar.xml, copy the <weblogic−enterprise−bean> tag, shown in the following example, for the UserProfileManager and configure it for your new ProfileManager deployment:
   ```
   <weblogic−enterprise−bean>
       <ejb−name>MyProfileManager</ejb−name>
       <reference−descriptor>
           <ejb−reference−description>
               <ejb−ref−name>ejb/EntityPropertyManager</ejb−ref−name>
               <jndi−name>${APPNAME}.BEA_personalization. EntityPropertyManager</jndi−name>
           </ejb−reference−description>
           <ejb−reference−description>
               <ejb−ref−name>ejb/PropertySetManager</ejb−ref−name>
               <jndi−name>${APPNAME}.BEA_personalization. PropertySetManager</jndi−name>
           </ejb−reference−description>
           <ejb−reference−description>
               <ejb−ref−name>ejb/MyEntityPropertyManager</ejb−ref−name>
   ```

```
            <jndi-name>${APPNAME}.BEA_personalization. MyEnitityPropertyManager</jndi-na
        </ejb-reference-description>
    </reference-descriptor>
    <jndi-name>${APPNAME}.BEA_personalization. MyProfileManager</jndi-name>
</weblogic-enterprise-bean>
```

You must create an <ejb−reference−description> to map the <ejb−ref> for your custom EntityPropertyManager to the JNDI name. This JNDI name must match the name you assigned in weblogic−ejb−jar.xml in the JAR file for your custom EntityPropertyManager.
Note the ${APPNAME} string substitution variable. The WebLogic Server EJB container automatically substitutes the enterprise application name to scope the JNDI name to the application.

7. In weblogic−ejb−jar.xml, copy the <transaction−isolation> tag for the UserProfileManager, shown in the following example, and configure it for your new ProfileManager deployment:
```
<transaction-isolation>
    <isolation-level>TRANSACTION_READ_COMMITTED</isolation-level>
    <method>
        <ejb-name>MyProfileManager</ejb-name>
        <method-name>*</method-name>
    </method>
</transaction-isolation>
```
8. Create a temporary p13n_ejb.jar for your new deployment descriptors and your new ProfileManager bean implementation class. This temporary EJB JAR archive should not have any container classes in it. Run ejbc to generate new container classes.
9. Edit your application's META−INF/application.xml to add an entry for your custom EntityPropertyManager EJB module, as shown in the following example:
```
<module>
    <ejb>UUPExample.jar</ejb>
</module>
```
10. If you are using an application−wide cache, you can manage it from the WebLogic Server Administration Console if you add a <Cache> tag for your cache to the META−INF/application−config.xml deployment descriptor for your enterprise application as shown in the following example:
```
<Cache Name="UUPExampleCache" TimeToLive="60000"/>
```

    Verify the modified p13n_ejb.jar and your custom EntityPropertyManager EJB JAR archive are in the root directory of your enterprise application and start your server.
11. Use the WebLogic Server Administration Console to verify your EJB module is deployed to the enterprise application and add your server as a target for the EJB module. You must select a target to have your domain's config.xml file updated to deploy your EJB module to the server.
12. Use the WebLogic Workshop Property Set Designer to create a User Profile (property set) that matches the name of the property set that you mapped to your custom EntityPropertyManager in ejb−jar.xml for the UserProfileManager (in p13n_ejb.jar). You could also map specific property names in a property set to your custom EntityPropertyManager, which would allow you to surface the properties and their values in the WebLogic Administration Portal for use in creating rules for personalization, delegated administration, and visitor entitlements.

Your new Unified User Profile type is ready to use. You can use the WebLogic Administration Portal to create a user, and it will use your UUP implementation when the "UUPExample" property set is being modified. That is because you mapped the ProfileType using an <ejb−ref> in your UserManager deployment descriptor, ejb/ProfileType/UUPExampleUser.

Now, when you call createUser("bob", "password", "UUPExampleUser") on the UserManager, several things will happen:

- A user named "bob" is created in the security realm.
- A WebLogic Portal Server profile record is created for "bob" in the WebLogic Portal RDBMS repository.
- If you set up the Creator mapping, the UserManager will call your new ProfileManager deployment, which will call your custom EntityPropertyManager to create a record for Bob in your data source.
- Retrieving Bob's profile will use your new ProfileManager deployment, and when you request a property belonging to the "UUPExample" property set, the request will be routed to your custom EntityPropertyManager implementation.

Retrieving User Profile Data from LDAP

WebLogic Portal provides a default unified user profile for retrieving properties from an LDAP server. Use this procedure to implement the LDAP unified user profile for retrieving properties from your LDAP server.

The LdapRealm security realm and the LdapPropertyManager unified user profile (UUP) for retrieving user properties from LDAP are independent of each other. They do not share configuration information and there is no requirement to use either one in conjunction with the other. A security realm has nothing to do with a user profile. A security realm provides user/password data, user/group associations, and group/group associations. A user profile provides user and group properties. A password is not a property.

In order to successfully retrieve the user profile from the LDAP server, ensure that you've done the following:

1. If you have already deployed the application on a WebLogic Portal instance, stop the server.
2. Extract p13n_ejb.jar from your application root to a temporary directory.
3. In the temporary directory, open META−INF/ejb−jar.xml, which contains a commented block called "Ldap Property Manager." Uncomment and reconfigure this section using the following steps:
    a. Remove the closing comment mark (−−>) from the end of the "Ldap Property Manager" block, just before the "Property Set Web Service EJB" block, and add it to the end of the first paragraph of the Ldap Property Manager block, like this:

    ```
    <!-- Ldap Property Manager
         To use this, uncomment it here as well as in weblogic−ejb−jar.xml.
         Configure the LDAP connection and settings using the env−entry values
         Do not forget to uncomment the ejb−link and method−permission tags for
         An easy way to ensure you don't miss anything is to search for "ldap"
         weblogic−ejb−jar.xml. Search from the beginning to the end of the file
    −−>
    ```

    b. In the "Ldap Property Manager" block, look for the following default settings and replace them with your own:

ldap://server.company.com:389	Change this to the value of your LDAP server URL.
uid=admin, ou=Administrators, ou=TopologyManagement, o=NetscapeRoot	Change this to the value of your LDAP server's principal.
<env−entry−value>weblogic</env−entry−value>	Change "weblogic" to your LDAP server's principalCredential.

ou=People,o=company.com	Change this to your LDAP server's UserDN.
ou=Groups,o=company.com	Change this to your LDAP server's GroupDN.
<env−entry−value>uid</env−entry−value>	Change "uid" to your LDAP server's usernameAttribute setting.
<env−entry−value>cn</env−entry−value>	Change "cn" to your LDAP server's groupnameAttribute setting.

c. In the "User Profile Manager" and "Group Profile Manager" sections, find the following lines:

```
<!−− <ejb−link>LdapPropertyManager</ejb−link> −−>
<ejb−link>EntityPropertyManager</ejb−link>
```

Uncomment the LdapPropertyManager line and delete the EntityPropertyManager line in both sections.

d. In the <method−permission> and <container−transaction> sections, find and uncomment the following:

```
<!−−
<method>
    <ejb−name>LdapPropertyManager</ejb−name>
    <method−name>*</method−name>
</method>
−−>
```

e. Check to see that you have uncommented all Ldap configurations by doing a search for "Ldap" in the file.

f. Save and close the file.

4. In the temporary directory, open META−INF/weblogic−ejb−jar.xml and perform the following modifications:

a. Uncomment the "LdapPropertyManager" block:

```
LdapPropertyManager
<weblogic−enterprise−bean>
    <ejb−name>LdapPropertyManager</ejb−name>
    <enable−call−by−reference>True</enable−call−by−reference>
    <jndi−name>${APPNAME}.BEA_personalization.LdapPropertyManager</jndi−nam
</weblogic−enterprise−bean>
```

b. In the "Security configuration" section of the file, uncomment the LdapPropertyManager method:

```
<method>
    <ejb−name>LdapPropertyManager</ejb−name>
    <method−name>*</method−name>
</method>
```

   c. Check to see that you have uncommented all Ldap configurations by doing a search for "Ldap" in the file.
   d. Save and close the file.
5. Replace the original p13n_ejb.jar with the modified version.

   a. Rename the original p13n_ejb.jar to use it as a backup. For example, rename it to p13n_ejb.jar.backup.
   b. JAR the temporary version of p13n_ejb.jar to which you made changes. Name it p13n_ejb.jar.
   c. Copy the new JAR to your application's root directory.
6. Start the server and re−deploy the application.
7. The properties from your LDAP server are now accessible through the WebLogic Portal API, JSP tags, and controls.

   If you want to surface the properties from your LDAP server in the WebLogic Administration Portal (for use in defining rules for personalization, delegated administration, and visitor entitlements), create a user profile property set called ldap.usr, and create properties in the property set that exactly match the names of the LDAP properties you want to surface.

Enabling SUBTREE_SCOPE Searches for Users and Groups

The LdapPropertyManager EJB in p13n_ejb.jar allows for the inspection of the LDAP schema to determine multi−valued versus single−value LDAP attributes, to allow for multiple userDN/groupDN, and to allow for SUBTREE_SCOPE searches for users and groups in the LDAP server. Following are more detailed explanations:

The determination of multi−value versus single−value LDAP attributes allows a developer to configure the ejb−jar.xml deployment descriptor for the LdapPropertyManager EJB to specify that the LDAP schema be used to determine if a property is single− or multi−value.

To enable SUBTREE−SCOPE for users and groups:

1. Stop the server.
2. Extract p13n_ejb.jar from your application root directory to a temporary directory and edit the temporary META−INF/ejb−jar.xml by setting the following env−entries.

```
<!−− Flag to specify if LDAP attributes will be determined to be single value
or multi−value via the schema obtained from the attribute. If false,
then the attribute is stored as multi−valued (a Collection) only if it has
more than one value. Leave false unless you intend to use multi−valued LDAP
attributes that may have only one value. Using true adds overhead to check
the LDAP schema. Also, if you use true beware that most LDAP attributes are
multi−value. For example, iPlanet Directory Server 5.x uses multi−value for
givenName, which you may not expect unless you are familiar with LDAP schemas.
This flag will apply to property searches for all userDNs and all groupDNs. −−>

<env−entry>
    <env−entry−name>config/detectSingleValueFromSchema</env−entry−name>
    <env−entry−type>java.lang.Boolean</env−entry−type>
    <env−entry−value>true</env−entry−value>
</env−entry>

<!−− Value of the name of the attribute in the LDAP schema that is used
to determine single value or multi−value (RFC2252 uses SINGLE−VALUE).
This attribute in the schema should be true for single value and false
```

```
or absent from the schema otherwise. The value only matters if
config/detectSingleValueFromSchema is true. -->

<env-entry>
    <env-entry-name>config/singleValueSchemaAttribute</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>SINGLE-VALUE</env-entry-value>
</env-entry>
```

It is not recommended that true be used for config/detectSingleValueFromSchema unless you are going to write rules that use multi−valued LDAP attributes that have a single value. Using config/detectSingleValueFromSchema = true adds the overhead of checking the LDAP schema for each attribute instead of the default behavior (config/detectSingleValueFromSchema = false), which only stores an attribute as multi−valued (in a Collection) if it has more than one value.

This feature also implements changes that allow you to use SUBTREE_SCOPE searches for users and groups. It also allows multiple base userDN and groupDN to be specified. The multiple base DN can be used with SUBTREE_SCOPE searches enabled or disabled.

A SUBTREE_SCOPE search begins at a base userDN (or groupDN) and works down the branches of that base DN until the first user (or group) is found that matches the username (or group name).

To enable SUBTREE_SCOPE searches you must set the Boolean config/objectPropertySubtreeScope env−entry in the ejb−jar.xml for p13n_ejb.jar.jar to true and then you must set the config/userDN and config/groupDN env−entry values to be equal to the base DNs from which you want your SUBTREE_SCOPE searches to begin.

For example, if you have users in ou=PeopleA,ou=People,dc=mycompany,dc=com and in ou=PeopleB,ou=People,dc=mycompany,dc=com then you could set config/userDN to ou=People,dc=mycompany,dc=com and properties for these users would be retrieved from your LDAP server because the user search would start at the "People" ou and work its way down the branches (ou="PeopleA" and ou="PeopleB").

You should not create duplicate users in branches below your base userDN (or duplicate groups below your base groupDN) in your LDAP server. For example, your LDAP server will allow you to create a user with the uid="userA" under both your PeopleA and your PeopleB branches. The LdapPropertyManager in p13n_ejb.jar.jar will return property values for the first userA that it finds.

It is recommended that you do not enable this change (by setting config/objectPropertySubtreeScope to true) unless you need the flexibility offered by SUBTREE_SCOPE searches.

An alternative to SUBTREE_SCOPE searches (with or without multiple base DNs) would be to configure multiple base DNs and leave config/objectPropertySubtreeScope set to false. Each base DN would have to be the DN that contains the users (or groups) because searches would not go any lower than the base DN branches. The search would cycle from one base DN to the next until the first matching user (or group) is found.
The new ejb−jar.xml deployment descriptor is fully commented to explain how to set multiple DNs, multiple usernameAttributes (or groupnameAttributes), and how to set the objectPropertySubtreeScope flag.

3. Save and close the file.
4. Replace the original p13n_ejb.jar with the modified version:

   a. Rename the original p13n_ejb.jar to use it as a backup. For example, rename it to p13n_ejb.jar.backup.
   b. JAR the temporary version of p13n_ejb.jar to which you made changes. Name it p13n_ejb.jar.
   c. Copy the new JAR to your application's root directory.
5. Start the server and re−deploy the application.

Related Topics

Using Multiple Authentication Providers in Portal Development

Adding WebLogic Portal Functionality to an Application

After you integrate an existing application into WebLogic Workshop and create a portal application, you can use the WebLogic Portal tools, services, and framework to add powerful features to your application and give it portal user interface.

This topic highlights the tools and services you can use to add WebLogic Portal functionality to your applications.

Portal User Interface Framework

WebLogic Portal's flexible, powerful framework lets you create portal interfaces independently of your application logic or Web pages, and WebLogic Portal's integration into WebLogic Workshop lets you surface the applications and Web services you develop in WebLogic Workshop seamlessly and easily in your portal interfaces.

Reusable Sample Portlets

WebLogic Portal provides many sample portlets that you can reuse in your portal applications.

Personalization and Campaigns

WebLogic Portal provides integrated tools in WebLogic Workshop and the WebLogic Administration Portal for adding personalization and campaigns to your portal applications

Mobile Device Support

WebLogic Portal includes a multichannel framework for fast, flexible development of portals for mobile devices. You can develop portals that simultaneously serve multiple devices.

Content Management

BEA's Virtual Content Repository lets you combine and manage multiple BEA−compatible content management systems in a single interface. Any content in the Virtual Content Repository can be used in your portals. WebLogic Portal also provides a content repository and a reusable My Content portlet for uploading, managing, viewing, and searching content stored in the Virtual Content Repository.

Page Flows

Page Flows control the navigational flow through an application and give you the flexibility and extensibility to separate the user interface code from navigational control and other business logic.

Portal Controls

Built−in portal Java controls let portal developers quickly add Java code to portal applications for functionality such as user creation, authentication, and property set management.

Commerce

You can build robust commerce applications in portals by using WebLogic Portal's commerce API, JSP tags, discount service, and catalog management service.

JSP Tags

WebLogic Portal provides a full library of JSP tags to help you with user and group management, content management, reliable URL generation to portal resources across different servers, internationalization, and other development tasks.

WebLogic Portal API

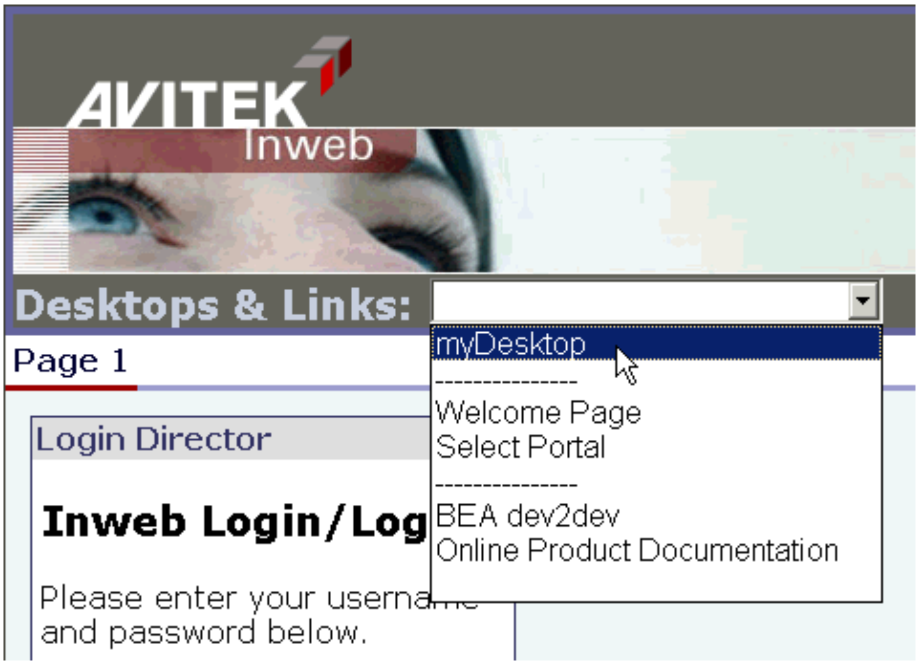
WebLogic Portal provides a full Java API for application development.

Enabling Desktop Selection

Oftentimes users are entitled to view multiple desktops in your portals. This topic shows you how to let users select from a list of the specific desktops to which they are entitled.

The desktop selection feature is a JSP used by the shell that provides a drop−down list of desktops and links to other resources. Because the desktop selector lets users switch between multiple desktops, it must run in streaming mode where multiple desktops exist. When viewing the feature in single file mode (development), only one desktop is ever available at a time.

The following figure shows the desktop selector in action.



To add the desktop selector to your desktops

The following procedure for adding Visitor Tools assumes you are adding them to a custom portal application (not the portalApp sample). If your application an/or project is not portal−enabled, install portal in both. See Creating a Portal Application and Portal Web Project.

1. Set up some form of authentication for your portal desktop. Authentication allows visitor entitlements to take effect. See Login Portlet, Login Director, or Implementing Authentication for information on adding authentication to your desktops.
2. In WebLogic Workshop create a new portal file.
3. Import the following files from Sample Portal into your application:

Import or copy this	***to this directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/header/header.jsp`	<PORTAL_APP>/<project>/portlets/header/
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/images/`	<PORTAL_APP>/<project>/images/

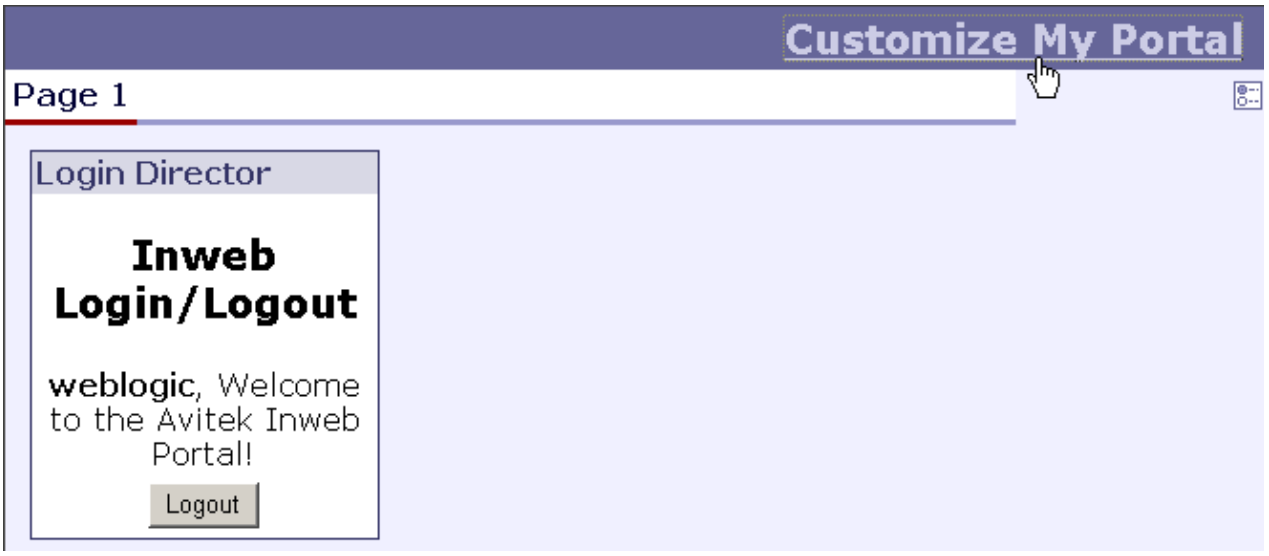
4. Open <PORTAL_APP>/<project>/portlets/header/header.jsp in WebLogic Workshop and replace the string sampleportal with the name of your project.
5. Create a shell and make <PORTAL_APP>/<project>/portlets/header/header.jsp the header content.
6. In a .portal file open in the Portal Designer, select the new shell for the desktop.
7. Save the portal file.

When portal administrators create desktops in the WebLogic Administration Portal and select that shell for the desktop, the desktop selector appears in the rendered desktops.

Adding Visitor Tools to Portals

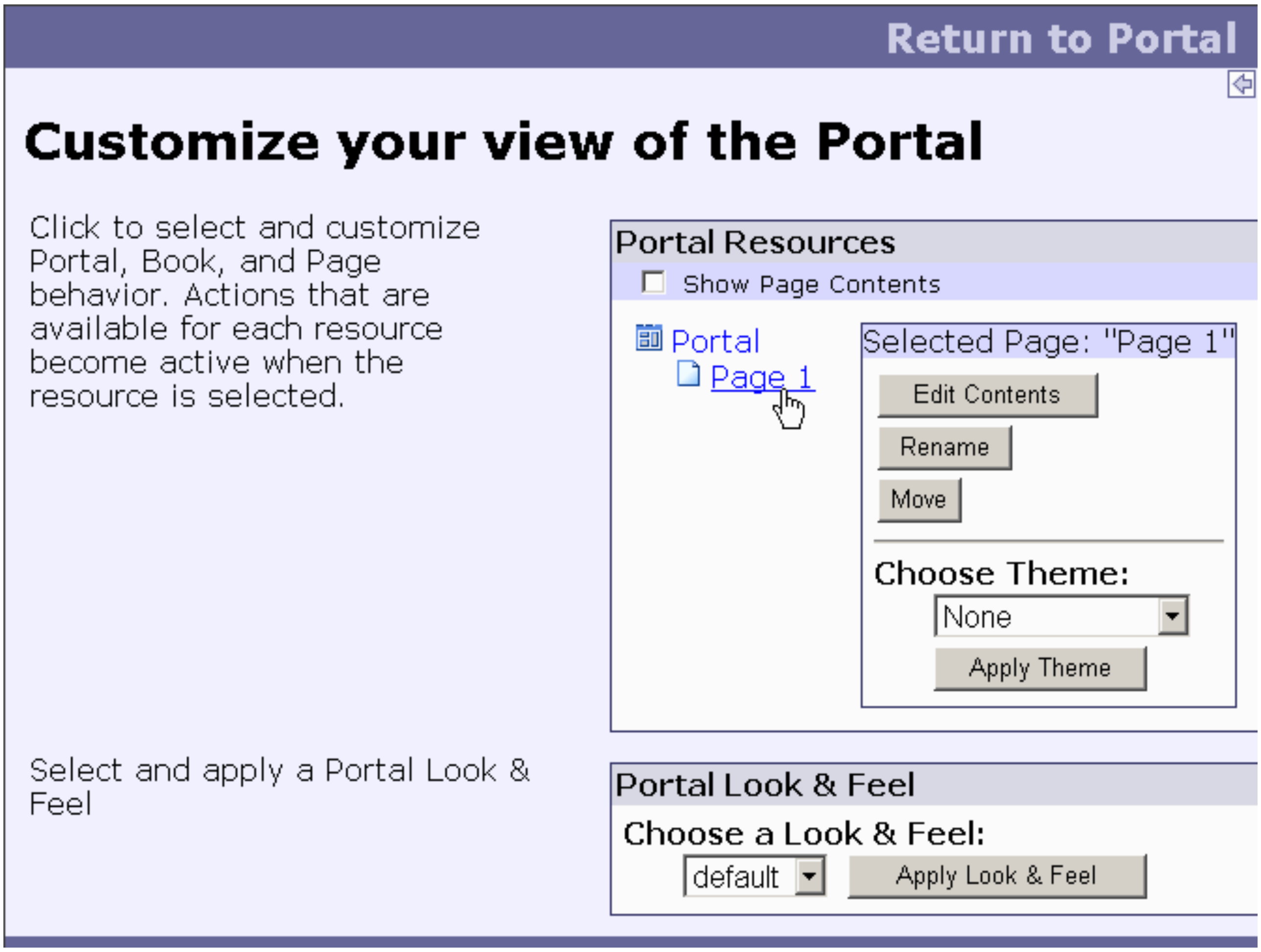
You can add functionality to your portal desktops that lets visitors modify their desktops, books, and pages. In order to use these Visitor Tools, visitors must be logged in to a desktop that is running in streaming mode.

Visitors access the visitor tools by clicking a text link or an icon in the desktop menu bar, as shown in the following illustration.



In this example, visitors can click on either the ***Customize My Portal*** link or the icon below it to access the Visitor Tools. The Customize My Portal link is supplied by the JSP used in the shell (described later in this topic), and the Edit icon is inserted by the menu skeleton JSP. Notice that the visitor must be logged in to access the Visitor Tools.

The following figure shows the Visitor Tools.


Return to Portal
Customize your view of the Portal
Click to select and customize Portal, Book, and Page behavior. Actions that are available for each resource become active when the resource is selected.
Portal Resources
Show Page Contents
Portal
Page 1
Selected Page: "Page 1"
Edit Contents
Rename
Move
Choose Theme:
None
Apply Theme
Select and apply a Portal Look & Feel
Portal Look & Feel
Choose a Look & Feel:
default
Apply Look & Feel

To add the Visitor Tools to Your Portals

The following procedure for adding Visitor Tools assumes you are adding them to a custom portal application (not the portalApp sample). If your application an/or project is not portal−enabled, install portal in both. See Creating a Portal Application and Portal Web Project.

1. Set up some form of authentication for your portal desktop. See Login Portlet, Login Director, or Implementing Authentication for information on adding authentication to your desktops.
2. In WebLogic Workshop create a new portal file.
3. In the Portal Designer, select the Main Page Book.
4. In the Property Editor window, set either of the following combinations of property values:

   Navigation: Single Level Menu or Multi Level Menu
   Editable: Edit in Menu

   or

   Navigation: No Navigation
   Editable: Edit in Titlebar
5. In the Mode Properties that appear, click the ellipsis icon [...] in the ***Content URI*** field, select <project>/visitorTools/visitorTools.portion, and click ***Open***.
6. Set the ***Visible*** property to false.

7. In the Portal Designer, select the Desktop.
8. In the Property Editor window, set the ***Shell*** to "Visitor Tools Shell."

   Now you must create a streaming desktop using the .portal file as a template to use the Visitor Tools.
9. If the server is not running, start it. Choose ***Tools −−> WebLogic Server −−> Start WebLogic Server***.
10. When the server is running, choose ***Portal −−> Portal Administration*** to start the WebLogic Administration Portal.
11. Log in to the WebLogic Administration Portal (the default username and password is ***weblogic***/***weblogic***).
12. Create a new desktop using your .portal file as a template. See Create a New Portal and Create a Desktop in the WebLogic Administration Portal online help posted on e−docs.
13. Select the new desktop in the Portal Resources tree, and go to the Desktop Properties page. At the bottom of the page, click ***View Desktop***.
14. When the desktop appears, log in and access the Visitor Tools.

You will notice that you can access the Visitor Tools by clicking the Customize My Portal link or the Edit icon. You do not have to use both ways to access the Visitor Tools.

- To use the Customize My Portal link only, use the Visitor Tools Shell and set the Main Page Book's ***Editable*** property to "Not Editable."
- To use the Edit icon, leave the ***Editable*** and ***Content URI*** property values in place and choose a shell other than Visitor Tools Shell.

The main page book in your .portal file can be used as the main page book when creating a desktop in the WebLogic Administration Portal to enable Visitor Tools. This will provide the desktop with Visitor Tools.

Note: You can also use the default New Blank Desktop template in the WebLogic Administration Portal to create a desktop that has Visitor Tools enabled.

Related Topics

Creating Shells

Creating URLs to Portal Resources

WebLogic Portal provides a convenient, extensible mechanism for creating URLs to your portal resources in a portal Web project that can transfer from domain to domain without breaking, especially when server names and port numbers change. This URL−creation mechanism also lets you switch between secure and non−secure URLs (http and https).

The two pieces involved in creating portable URLs are:

- The <render:*Url> JSP tags in the Portal Rendering JSP tag library.
- A portal Web project's WEB−INF/url−template−config.xml file.

The url−template−config.xml file contains multiple URL "templates," each with a unique name. Those template URLs contain variables such as url:domain and url:port that are read in from the active server. The <render:*Url> JSP tags have a "template" attribute in which you can specify the name of a URL template in url−template−config.xml.

The following examples show how the JSP tags use the templates to create URLs.

url−template−config.xml	***<render:resourceUrl>***

The following is a sample URL template in url−template−config.xml.
 	
`<url−template name="secure−url">`
`    https://{url:domain}:{url:securePort}/{url:path}?{url:queryString}`
`</url−template>`
	The following is how the <ren
	template.
	
	`<% String reportpath = "`
	
	`<a href="<render:resourc`
	`View the Report`
	`</a>`

You can use any of the URL templates in url−template−config.xml provided by WebLogic Portal, and you can add as many templates as you want to the file.

The following variables are available for use in URL template building:

{url:domain} − Reads the name of the server from the current request.

{url:port} − Reads the listen port number of the server from the current request. (See Troubleshooting below.)

{url:securePort} − Reads the SSL port number of the server from the current request. (See Troubleshooting below.)

{url:path} − Reads the name of the Web application. The URLs to all resources in a Web application are relative to the Web application directory.

{url:queryString} − Reads a queryString variable for the URL.

Troubleshooting

If you are using a proxy server or switching back and forth between non−secure and secure ports, you may find that URLs do not resolve if you use the {url:port} or {url:securePort} variables. This is because the variables for those values are read from the request. For example, if a user in a non−secure URL (port number 80) clicks a secure https link that was created with a URL template that uses the {url:securePort} variable, the port number of the request (80) is used for the {url:securePort} variable, which would create a secure request (https) on an non−secure port. The same could happen if a user on a proxy server (port 80) clicks a link to a resource outside the proxy server (port 443).

In both of those cases, you need to hard code port numbers in the URL templates to get URLs to resolve correctly.

Web Services for Remote Portlets (WSRP)

The url−template−config.xml file automatically created in a portal Web project also contains URL templates and variables for WSRP portlets. These templates must remain in the file if you are going to be a WSRP producer.

Related Topics

Portal Rendering JSP Tags

Developing a New Portal Application

Use the procedures in this section when you want to build a new portal application from the ground up. If you have an existing application you want to integrate into WebLogic Workshop, see Integrating Existing Applications into Portals.

This section includes the following topics:

Creating a Portal Application and Portal Web Project

Shows you how to lay the foundation of portal development by creating a portal application and portal Web project or installing Portal in existing applications and projects.

Building Different Types of Applications

Describes the many types of applications in WebLogic Workshop you can surface in portals, including Web applications, Java Page Flow applications, Struts applications, Web services, commerce applications, and applications that can be accessed by mobile devices.

Overview of Content Management

Provides instructions and links for setting up content management for use by your applications.

Setting up Unified User Profiles

Shows you how to set up Unified User Profiles, which provide the capability to leverage user data from external sources such as LDAP servers, legacy systems and databases.

Enabling Desktop Selection

Shows you how to let users access any of the portal desktops to which they are entitled.

Adding Visitor Tools to Portals

Shows you how to let users customize their portal desktops.

Related Topics

Building Portlets

Developing Portal User Interfaces

Assembling Portal Applications

Securing Portal Applications

Deploying Portal Applications

Portal Reference

Creating a Portal Application and Portal Web Project

To create the necessary resources for portal development, you must do one of two things:

- Option 1: Create a new portal application and add a Portal Web Project to it

  or
- Option 2: Install Portal into an existing application and add a Portal Web Project to it

Following are the procedures for each option.

Option 1: To create new portal application and add a Portal Web project to it

Use this procedure to create a new portal application.

You do not need to perform these steps if you are developing on a shared domain and the portal application has already been created and stored in a version−control system. Simply synchronize to the current version of the domain to put the portal application on your machine.

1. If you have not yet created a portal domain on your development machine, create one with the Configuration Wizard. For instructions, see the Overview of Platform Configuration on the BEA's e−docs Web site.

   Performing this step ensures you have a server (config.xml) for your portal application to use, as described later in this procedure.
2. Create a new portal application. In WebLogic Workshop Platform Edition, choose ***File −−>New −−>Application***.
3. In the New Application window, select ***Portal Application*** in the right pane.
4. In the ***Directory*** field, click ***Browse*** to set the location of the new application. The application will be created in a subdirectory of the directory you select.
5. Make sure the ***Name*** field contains the name of the application. This name will be the application directory.
6. In the ***Server*** field, click ***Browse*** and select the config.xml file for the server (domain) you want to use.

   The config.xml file is in the portal domain directory you created.
7. Click ***Create***. The application directory appears in the Application window. The application contains the WebLogic Administration Portal (contained in Modules/adminPortal.war), a datasync directory (data) for interaction management development, and application−level EJBs and APIs.
8. Create a portal Web project for your application. Right−click the **<*app_name*>** directory in the Application window, and choose ***New −−>Project***.
9. In the New Project window, select ***Portal Web Project*** in the right pane.
10. In the ***Project name*** field, enter the name for the portal Web project. This will be the name of a Web application directory.
11. Click ***Create***. The project folder appears in the Application window. The portal Web project contains WebLogic Portal JSP tags, Web−application−level APIs, and default portal framework files.
12. If you have any external projects or files you want to include in your portal application, perform any of the following steps:
    - To import a project, right−click the **<*app−name*>** directory in the Application window and choose ***Import Project***. In the Import Project window, select the type of project to import,

browse to select the project folder, and click ***Import***.

- To import files, such as existing datasync files (User Segments, Campaigns, Placeholders, and so on) or the Workshop Portal Extensions sample portlets to use in your portals , right−click the appropriate directory in the Application window and choose ***Import***. In the Import Files window, select the directory or files you want to import, and click ***Import***.

  The sample portlets are located in `<BEA_HOME>\<WEBLOGIC_HOME>\samples\portal\portalApp\sampleportal\port` There are other useful sample files throughout the `<BEA_HOME>\<WEBLOGIC_HOME>\samples` directory. See the instructions in Portal Samples for more information.

You now have the resources and directories for developing personalized applications and creating portals to surface applications.

13. Start your development server. In WebLogic Workshop, choose ***Tools−−>WebLogic Server−−>Start WebLogic Server***. The server you assigned to your application in the previous steps starts. All your work is deployed automatically on your machine as you develop.

Option 2: To install portal in an existing application and add a Portal Web project to it

Use this procedure to add portal services to an existing application.

You do not need to perform these steps if you are developing on a shared domain and the portal−enabled application has already been created and stored in a version−control system. Simply synchronize to the current version of the domain to put the portal application on your machine.

1. In WebLogic Workshop Platform Edition, open the application in which you want to install portal.
2. In the Application window, right−click the ***<app_name>*** directory and choose ***Install−−>Portal***. WebLogic Workshop adds the WebLogic Administration Portal (contained in Modules/adminPortal.war), a datasync directory (data) for interaction management development, and application−level EJBs and APIs.
3. Create a portal Web project for your application. Right−click the ***<app_name>*** directory in the Application window, and choose ***New−−>Project***.
4. In the New Project window, select ***Portal Web Project*** in the right pane.
5. In the ***Project name*** field, enter the name for the portal Web project. This will be the name of a Web application directory.
6. Click ***Create***. The project folder appears in the Application window. The portal Web project contains WebLogic Portal JSP tags, Web−application−level APIs, and default portal framework files.
7. If you have any external projects or files you want to include in your application, perform any of the following steps:
   - To import a project, right−click the ***<app−name>*** directory in the Application window and choose ***Import Project***. In the Import Project window, select the type of project to import, browse to select the project folder, and click ***Import***.
   - To import files, such as existing datasync files (User Segments, Campaigns, Placeholders, and so on) or the Workshop Portal Extensions sample portlets to use in your portals , right−click the appropriate directory in the Application window and choose ***Import***. In the Import Files window, select the directory or files you want to import, and click ***Import***.

     The sample portlets are located in `<BEA_HOME>\<WEBLOGIC_HOME>\samples\portal\portalApp\sampleportal\port` There are other useful sample files throughout the `<BEA_HOME>\<WEBLOGIC_HOME>\samples` directory. See the instructions in Portal

Samples for more information.

You now have the resources and directories for developing personalized applications and creating portals to surface applications.

8. Start your development server if it is not already running. In WebLogic Workshop, choose ***Tools−−>WebLogic Server−−>Start WebLogic Server***. All your work is deployed automatically on your machine as you develop.

Related Topics

Portal Samples

Developing Portal Applications

How Do I: Create a New Application?

The WebLogic Workshop Development Environment

Building Different Types of Applications

You can develop many different types of applications with WebLogic Workshop, all of which you can surface in a portal interface.

Following are the different types of applications you can develop:

Developing Web Applications

Provides overview information and procedures for developing Web applications in WebLogic Workshop.

Building a Java Page Flow Application

Provides overview information and proceduers for developing Java Page Flow applications. Also provides portal−specific considerations for using Page Flows in portals.

Building a Struts Application

Provides overview information and links to relevant Struts development topics.

Building a Commerce Application

Provides instructions on adding commerce services to your applications, managing catalog services, and creating discounts.

Creating Portals for Mobile Devices

Describes how to use the multichannel framework to develop portals for mobile devices.

Developing Personalized Applications

Describes the pieces involved in and provides instructions for adding personalization and campaigns to your portal applications.

Using Portal JSP Tags

Provides guidance and tips on using Portal JSP tags in your JSPs.

Developing Web Applications

Enterprise web applications today can easily contains hundreds, if not thousands of pages. These pages should not only look great visually, but also offer services to customers that require the implementation of complex business logic. Managing a complex web site can be a daunting task, especially where the business logic is implemented directly in the web pages, and changing the logic requires many edits in many locations.

WebLogic Workshop provides you with the tools to manage complex web applications using JavaServer Pages (JSPs) and Page Flows. Separation of presentation and business logic allows for modularity of business logic implementation, such that the impact of changing business logic can be minimal. Furthermore, this separation allows the application developer to concentrate on implementing the business process using Java controls and EJBs, while the web developer can focus on the presentation. Page flows provide the navigational control, allowing a web application architect to easily design the flow between the JSP pages in the web application.

Topics Included in This Section

Guide to Building Page Flows

This development guide explains the key concepts involved in developing web applications using page flows, JavaServer Pages (JSPs), and WebLogic Workshop.

Exception Handling and Validating User Input

Explains how to handle errors at arise in a web application and how to validate data submitted by users.

Working with Struts Applications

Explains how to use your existing Struts applications with Page Flow and Portal applications.

Web Application Reference

The reference section provides details on page flow annotations, JSP tag syntax, and Flow View icons.

Related Topics

Getting Started Tutorial: Web Applications

This tutorial provides an entry−level introduction into the WebLogic Workshop environment for developing web applications that contain page flows and JSPs.

Tutorial: Page Flow

This tutorial provides more advanced tour of the WebLogic Workshop environment for developing web applications. This tutorial will teach you the basics of Page Flow technology, as well as more advanced features such as data binding, Java controls and security.

Page Flow and JSP Samples

This section discusses a number of pre−built sample web applications that demonstrate key concepts in page flow and JSP technology.

How Do I... topics for Page Flows and JSPs

This 'How Do I...?' section presents a number of hands−on examples to building page flows and data binding.

Building a Java Page Flow Application

Page Flows provide an event−driven flow through an application. Page Flows let you separate the user interface code from navigational control and other business logic. For instructions on developing Page Flows, see Developing Page Flow Applications.

Portal−Specific Setup

In order for URLs to resolve correctly, Java Page Flow support must be enabled in the portal Web project's WEB−INF/netuix−config.xml file, as shown in the following example. Notice the <enable> element is set to true.

```
<!-- Enable or disable Pageflow support -->
<pageflow>
    <enable>true</enable>
</pageflow>
```

If this block is not present in netuix−config.xml, do not add it. Without the block, the setting defaults to true.

Using Page Flows in Portlets

If you are retrieving information from the request within a portlet that uses Page Flow, you may need to get the information from the outer request.

For example, if you use regular HTML tags within Netui form tags:

```
<netui:form action="myAction">
    <input type="checkbox" name="test"/>
    <netui:button value="myAction"></netui:button>
</netui:form>
```

You need to do the following to retrieve that HTML input value:

```
<%@page import="com.bea.wlw.netui.pageflow.scoping.ScopedServletUtils"%>
    <%
    HttpServletRequest outerRequest = ScopedServletUtils.getOuterRequest( request );
    %>
    test: <%=outerReq.getParameter("test")%>
```

Related Topics

How Do I: Add Portal Functionality to an Existing Page Flow Application?

Adding Portal Controls to Java Page Flows

Page Flows let you control a user's path through an application. The actions (such as a button click) and conditions (such as whether the user's login succeeded or failed or whether or not the user is a manager) determine which JSP the user it taken to next in the application. Portal Controls provide a variety of actions and ways to incorporate conditions that give you precise control over both a user's path through your applications and what occurs on that path.

This section provides descriptions of and guidance on using Portal Controls in your Page Flows.

Topics Included in This Section

Using Portal Controls

This section provides general guidance on adding and configuring Portal Controls in your Page Flows.

Group Provider Control

This control lets you use group management actions in your Page Flows.

Profile Control

This control lets you manage user profiles in your Page Flows.

Property Control

This control lets you manage and retrieve

Rules Executor Control

This control lets you evaluate objects in working memory against a set of rules.

Rules Manager Control

This control lets you look up information about the rule sets that can be used by the Rules Executor Control.

User Info Control

This control lets you retrieve information about users in your Page Flows.

User Login Control

This control lets you add login and logout to your Page Flows.

User Provider Control

This control provides user management actions in your Page Flows.

Click Content Event Control

This control dispatches a ClickContentEvent to the event service for use in campaigns and behavior tracking.

Display Content Event Control

This control dispatches a DisplayContentEvent to the event service for use in campaigns and behavior tracking.

Generic Event Control

This control dispatches an event to the event service for use in campaigns.

Generic Tracking Control

This control dispatches a tracking event to the event service for use in campaigns and behavior tracking.

Rule Event Control

This control dispatches a RuleEvent to the event service for use in campaigns and behavior tracking.

Session Login Event

This control dispatches a SessionLoginEvent to the event service for use in campaigns and behavior tracking.

User Registration Event

This control dispatches a UserRegistrationEvent to the event service for use in campaigns and behavior tracking.

Note: WebLogic Portal also supports the legacy use of deprecated controls from the initial product release. If you have deprecated controls in your Page Flows, and you want to upgrade to the replacement controls to take advantage of new features, go to your Page Flow's Source View, click in the name of the control, and press F1. The Javadoc that appears tells you which control to use instead of the deprecated control.
Related Topics

Using Portal Controls

Working with Java Controls

Guide to Building Page Flows

Using Portal Controls

Portal Controls are collections of actions (Java methods) you can drag and drop into your Page Flows, making Java development easier and more automated. You can add actions in a graphical interface and configure the actions with the Property Editor, insulating your from working directly with Java code (though you can still work directly with code in Source View). Even if you want to work directly with code, working initially with the graphical interface (Flow View and Action View) automates code entry and makes it more syntax error free.

For example, Portal Controls provide built−in forms on some methods. If you want an action that creates a user, you can use the createUser method in the User Provider control. In the Page Flow's Action View, if you drag the createUser method from the Data Palette into the control's action area, the control provides a CreateUserForm bean that can be added to a JSP and linked to the action automatically. (That process is described in To use controls that provide forms.)

To add a control to a Page Flow:

1. Open an existing Page Flow (.jpf file) or create a new "basic" Page Flow.
2. Select the Action View tab.
3. In the Data Palette, on the Controls bar, click ***Add −−> [Portal Controls or Portal Event Controls] −−> [control]***.
4. In the Insert Control dialog box, enter a name for the new control and click ***Create***. The control and all its available methods appear in the Action View, as well as in the Data Palette under the Controls bar.

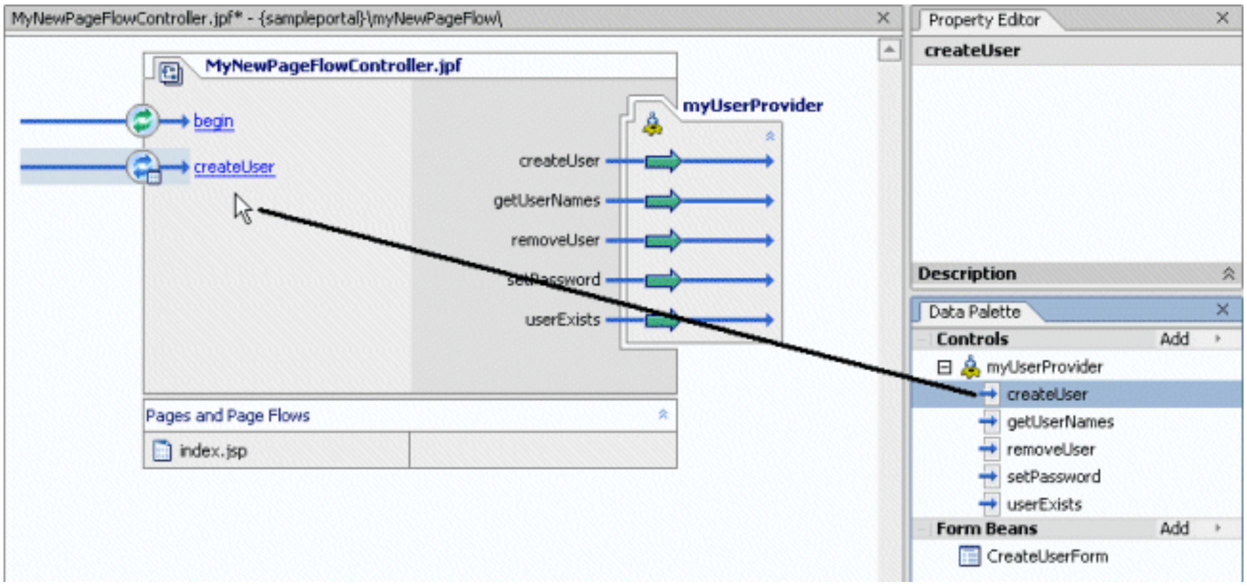
   All the methods in the control are now available to your Page Flow.
5. Add a method (action) to your page flow by dragging a method from the Data Palette into the action area of the Page Flow (the left side of the window in Action View).
6. Switch to Flow View, where you can connect the action to the appropriate location in the Page Flow.

To use controls that provide forms:

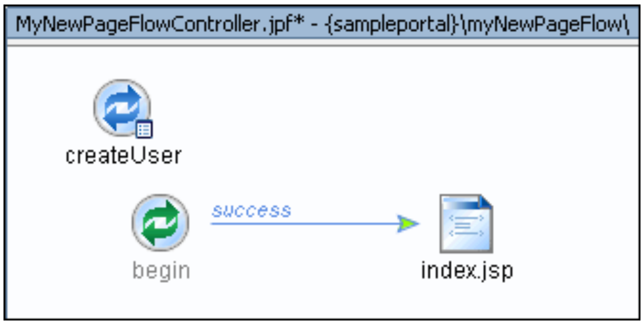
Some methods (actions) on controls lend themselves to form entry, such as the createUser method on the User Provider control, where you can, for example, have a new user enter a username and password for self−registration. This procedure walks you through the createUser scenario to highlight the basics of adding such a form to a Page Flow.

The basic scenario is providing a user self−registration form. When the user enters his username and password, the new user is created and the user is taken to the next JSP.

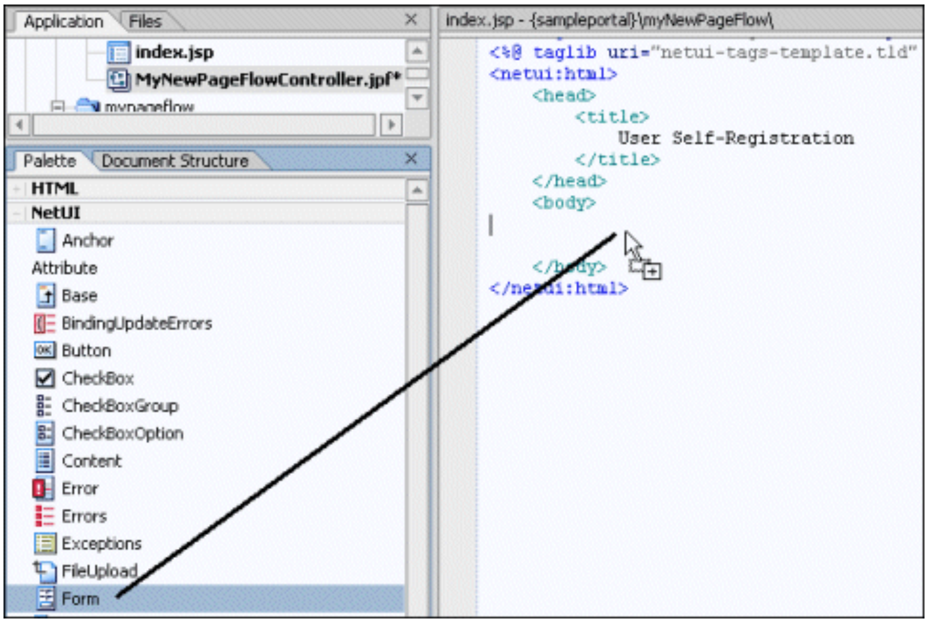
1. To use a createUser action, you must add the User Provider control to the Page Flow, as described in To add a control to a Page Flow, above.
2. With the control in the Page Flow, select Action View in the Page Flow editor, and drag the createUser method from the Data Palette to the action area of the Page Flow, as shown in the following illustration. Notice in the action area of the Page Flow editor that the createUser icon is different than the begin icon, because the action provides a form. The form is visible in the Data Palette under the Form Beans bar.



3. The following illustration shows the Page Flow in Flow View, where the createUser action now appears. Again, the createUser icon shows there is a form associated with the action.


MyNewPageFlowController.jpf* - {sampleportal}\myNewPageFlow\
createUser
success
begin
index.jsp

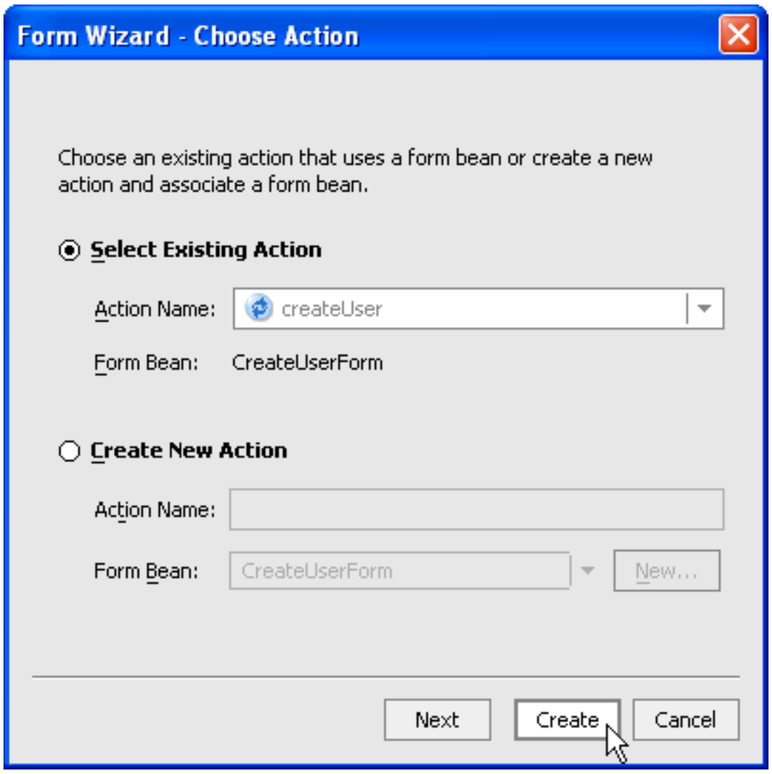
4. Now you can add the createUser form to the JSP where users will self−register. Open the JSP and select the Source View tab.
5. With the JSP open, drag the Form tag from the Palette into the JSP, as shown in the following illustration.



```
<%@ taglib uri="netui-tags-template.tld"
<netui:html>
    <head>
        <title>
            User Self-Registration
        </title>
    </head>
    <body>


    </body>
</netui:html>
```

6. In the Form Wizard window that appears, as shown in the following illustration, select the createUser action and click Create.


Form Wizard - Choose Action

Choose an existing action that uses a form bean or create a new action and associate a form bean.

(●) Select Existing Action

Action Name: createUser

Form Bean: CreateUserForm

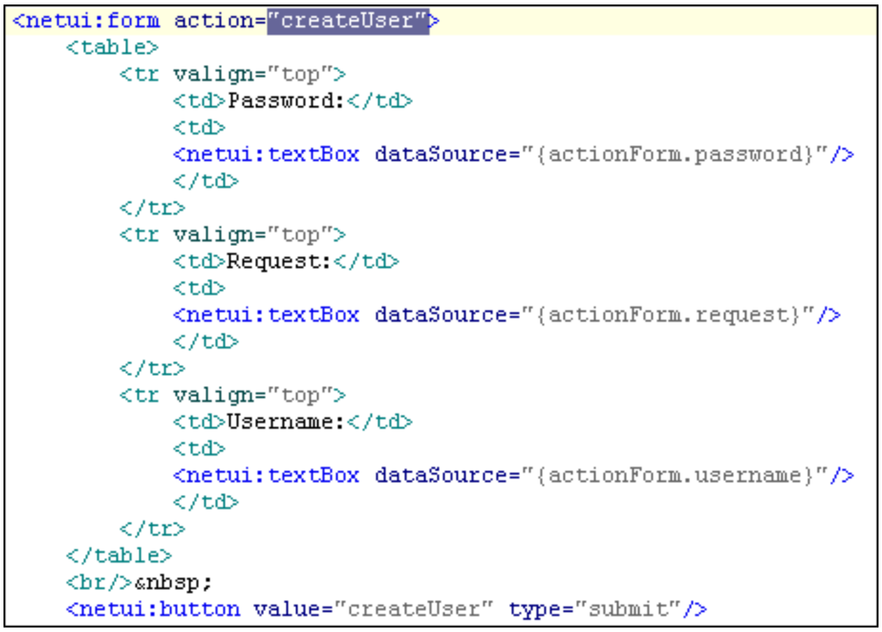
( ) Create New Action

Action Name:

Form Bean: CreateUserForm

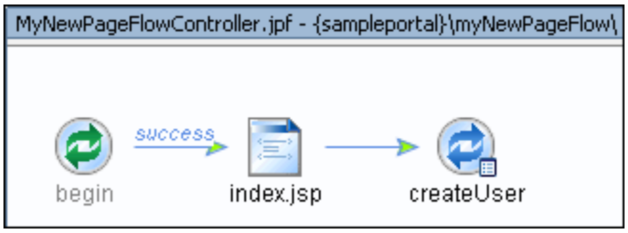
7. The following illustration shows the form that is added to the JSP. Notice the form is tied to the createUser action. When the user fills in the fields and clicks the "createUser" button on the form, the form data is sent to the createUser action, and the new user is added to the authentication provider you specify in the Property Editor (with the control selected in Action View or Source View).

   Note: In this example, the authentication provider must support write access for the user to be created. See the WebLogic Administration Portal online help for more information on using multiple authentication providers.



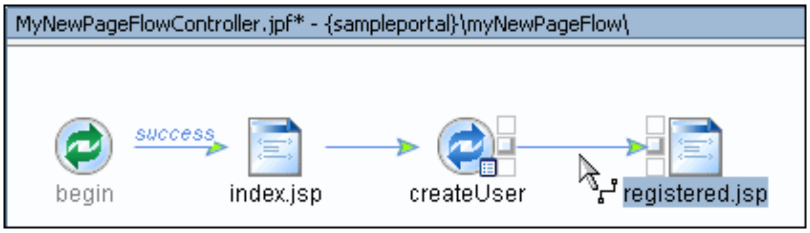
```
<netui:form action="createUser">
    <table>
        <tr valign="top">
            <td>Password:</td>
            <td>
            <netui:textBox dataSource="{actionForm.password}"/>
            </td>
        </tr>
        <tr valign="top">
            <td>Request:</td>
            <td>
            <netui:textBox dataSource="{actionForm.request}"/>
            </td>
        </tr>
        <tr valign="top">
            <td>Username:</td>
            <td>
            <netui:textBox dataSource="{actionForm.username}"/>
            </td>
        </tr>
    </table>
    <br/>&nbsp;
    <netui:button value="createUser" type="submit"/>
```

8. Save and close the JSP.
9. Select the Flow View tab on the Page Flow. Now that the form has been used for the createUser action, the Page Flow is automatically updated, as shown in the following illustration. Notice the arrow from the login.jsp to the createUser action. (The developer creating this Page Flow has rearranged the icons in Flow View.)



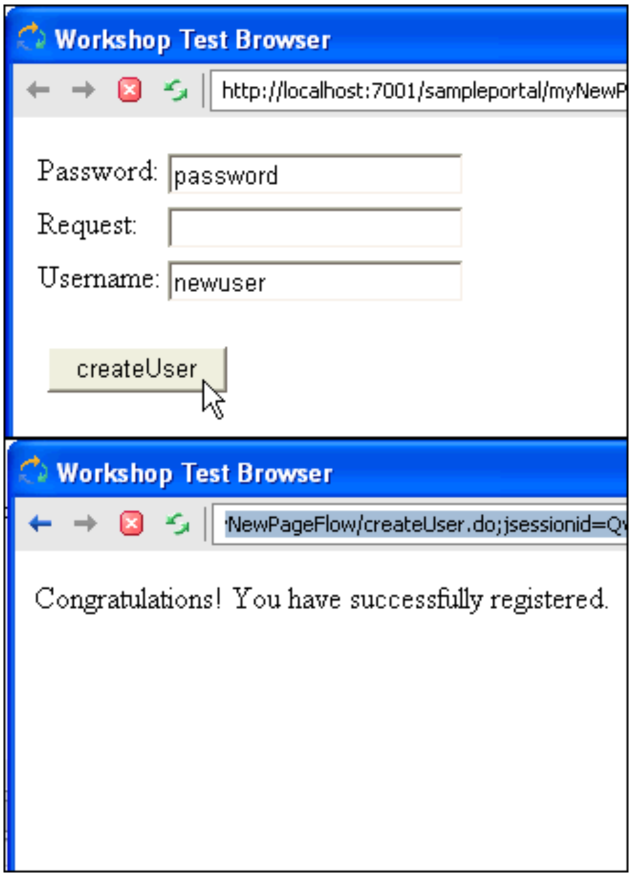
10. Now you have a form in a JSP that users can fill out to self−register, and you have an action that creates a new user with the user's form data. Now all you need is a JSP that will be displayed after the createUser action succeeds.

    In the Page Flow directory, create a new JSP and give it a text message such as, "Congratulations! You have successfully registered."
11. In Flow View, connect the createUser action to the new JSP, as shown in the following illustration.


MyNewPageFlowController.jpf* - {sampleportal}\myNewPageFlow\
begin
success
index.jsp
createUser
registered.jsp

12. Save the Page Flow. You can view the results by clicking the Start button in the toolbar or pressing Ctrl+F5.

    The user experience is shown in the following illustration.



Congratulations! You have successfully registered.

13. As the previous illustration shows, you may need to modify the default form by rearranging/removing input fields and renaming buttons.

In a real−world Page Flow, you would create JSPs to handle action failures and exceptions, such as if a user entered a username that was already in use.

For detailed information on forms, see Using Data Binding in Page Flows.

Related Topics

Getting Started with Page Flows

Adding Portal Controls to Java Page Flows

Building Java Page Flow Portlets

Tutorial: Creating a Login Portlet Using Portal Controls

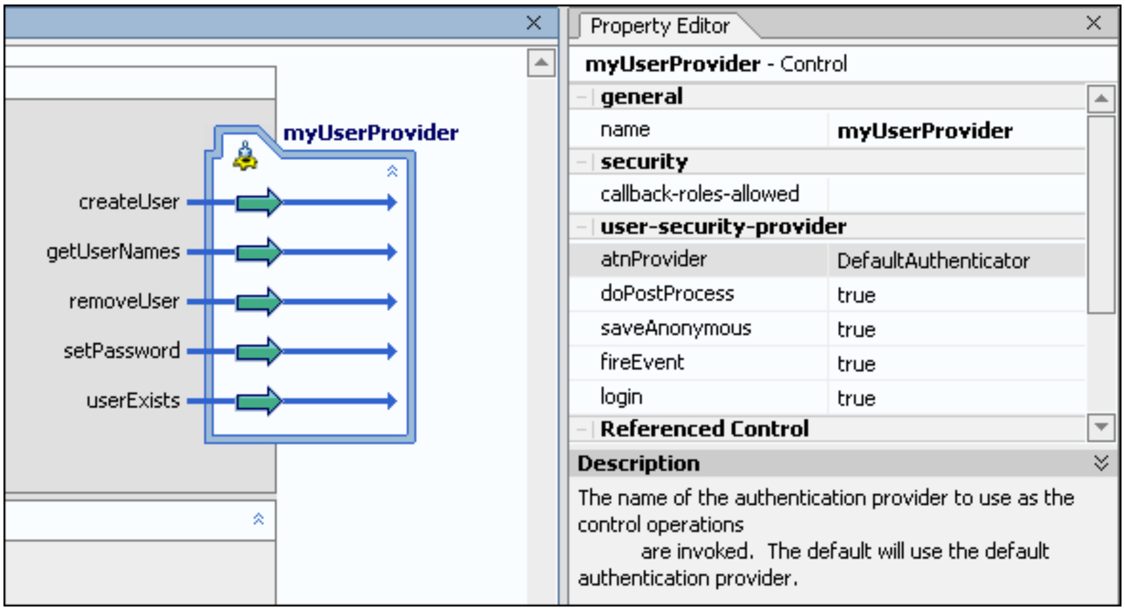
Portal Control Properties

Portal Control Security

Portal Control Declaration

Portal Control Properties

Portal Controls have properties you can edit in the Property Editor. Control properties−−also called annotations−−provide a convenient way to pass parameters to the underlying API at run time without having to manually pass the properties in your own Java code. For example, the User Provider Control has an atnProvider property that lets you enter the authentication provider that should be used for any of the control's actions, as shown in the following illustration.



myUserProvider - Control

general	
name	**myUserProvider**
security	
callback-roles-allowed	
user-security-provider	
atnProvider	DefaultAuthenticator
doPostProcess	true
saveAnonymous	true
fireEvent	true
login	true
Referenced Control	

Description

The name of the authentication provider to use as the control operations
are invoked. The default will use the default authentication provider.

The following code in Source View shows how the Page Flow uses the antProvider property.

```
/**
* @common:control
* @jc:user−security−provider atnProvider="DefaultAuthenticator"
*/
private com.bea.p13n.controls.securityProvider.UserProviderControl myUserProvider;
```

To see which properties are available on a Portal Control, click in the control name in Source View and press F1 to see the control's Javadoc (the com.bea.p13n.controls.* packages).

Related Topics

Adding Portal Controls to Java Page Flows

Control Security

Portal Control Declaration

Working with Java Controls

Building Custom Java Controls

Portal Control Declaration

When you use WebLogic Workshop to drag and drop a Portal Control onto the Pageflow design view, the following code is automatically placed in the Pageflow controller source:

```
/**
* @common:control
*/
private com.bea.p13n.controls.login.UserLoginControl myControl;
```

If you are creating a control in the Page Flow's Source View, or outside of WebLogic Workshop, be sure to include this control declaration in this form.

Related Topics

Adding Portal Controls to Java Page Flows

Control Security

Portal Control Properties

Working with Java Controls

Portal Control Security

Many Portal Controls have secured methods, meaning that any control attempting to execute such a method would need to be in an authorized security role. You can specify security roles in a Page Flow on each action. A user must be a member of the designated role(s) for the action to be fired. For example, the User Provider Control has a removeUser() method that requires the caller to be in the role of "PortalSystemAdministrator" or "Admin." See Portal Control Properties for more information.

For user and group management actions, the roles you specify in the WebLogic Administration Portal Authentication Security Provider Service determine whether or not the user can perform the action.

You can add security roles to a domain using the WebLogic Server Administration Console.

Related Topics

Security Roles (WebLogic Server e−docs topic)

Adding Portal Controls to Java Page Flows

Portal Control Properties

Portal Control Declaration

Working with Java Controls

Group Provider Control

The Group Provider control provides a convenient way to incorporate group management actions into your Page Flows, such as creating groups and getting a list of users in a group.

Use the atnProvider attribute to specify which authentication provider contains the groups you want to manage with the control.

Security: For user and group management actions, the roles you specify in the WebLogic Administration Portal Authentication Security Provider Service determine whether or not the user can perform the action.

If you use the createGroup action, also use the Profile Control to create a group profile for the new group.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Adding Portal Controls to Java Page Flows

Using Portal Controls

User Provider Control

Profile Control

Tutorial: Creating a Login Control Page Flow Using the Wizard

User/Group Management JSP Tags

Profile Control

The Profile control provides a convenient way to incorporate user and group profile management actions into your Page Flows, such as creating profiles for users and groups and getting profile information. For example, use this control to retrieve a user's profile, use the Property Control to put properties in working memory, then use the Rules Executor Control to evaluate and filter the user's profile properties in order to trigger actions based on that evaluation.

You can use this tag to create a standalone profile for a user in an external authentication provider that does not provide read access to its users and groups. Then, when that user logs in, the profile is automatically associated with the user.

Note: It is possible (but not recommended) to store an identical username or group name in more than one authentication provider. For example, user "foo" can reside in the default WebLogic Server LDAP provider and in an external RDBMS provider. In that case, WebLogic Portal uses only one user profile for user "foo."

Security: For profile management actions, the roles you specify in the WebLogic Administration Portal Authentication Security Provider Service determine whether or not the user can perform the action for users and groups. For example, to let users modify their own user profile, make sure the "Self" role is in "Roles That Can Update Users" (which is already a default setting).

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Adding Portal Controls to Java Page Flows

Using Portal Controls

Property Control

User Provider Control

Group Provider Control

Tutorial: Creating a Login Control Page Flow Using the Wizard

User/Group Management JSP Tags

Property Control

The Property control provides a convenient way to incorporate user profile property management actions into your Page Flows, such as creating, setting, and getting profile properties on users and groups. For example, use the Profile control to retrieve a user's profile, use this control to put properties in working memory, then use the Rules Executor Control to evaluate and filter the user's profile properties in order to trigger actions based on that evaluation.

Security: For profile management actions, the roles you specify in the WebLogic Administration Portal Authentication Security Provider Service determine whether or not the user can perform the action for users and groups. For example, to let users modify their own user profile, make sure the "Self" role is in "Roles That Can Update Users" (which is already a default setting).

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Adding Portal Controls to Java Page Flows

Using Portal Controls

Profile Control

User Provider Control

Group Provider Control

Tutorial: Creating a Login Control Page Flow Using the Wizard

User/Group Management JSP Tags

Rules Executor Control

The Rules Executor control gives you fine−grained control of your Page Flows using factors such as user profile values and other objects in working memory.

The control provides a convenient way to evaluate objects in working memory against the set of rules you designate. The control, which executes rule sets using the underlying rules engine, also lets you filter the results of the rule evaluation.

It is assumed you know which rules exist in the rules repository.

The rulesetUri property is required. This property is the path to the rule set you want to use in evaluating the objects in in working memory. The path is relative to your portal application's META−INF/data directory. For example, if the rule set you want to use is located in META−INF/data/rulesets/myRuleSet.rls, the rulesetUri would be /rulesets/myRuleSet.rls.

For information on rule sets, the rules engine, and putting objects into working memory, see Using Rules in Portal Applications on e−docs.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Rules Manager Control

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

Rules Manager Control

The Rules Manager control lets you access and managing rules and rule sets for the Portal rules manager. It is intended to be used only by Portal system administrators. You can use this control as a development tool to look up rule sets you want to use with the Rules Executor Control.

Rule sets, which must be stored in your portal application's META−INF/data directory, are loaded automatically when the server starts.

Because this control requires the caller be in an authorized role, it cannot be used from a Java Web Service.

Security: The caller must be in PortalSystemAdministrator role to invoke all of the control's methods.

For information on rule sets and the rules engine, see Using Rules in Portal Applications on e−docs.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Rules Executor Control

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

User Info Control

The User Info control queries information about a particular user, such as which groups the user belongs to, which roles the user is in, which roles are available to the user. To perform user maintenance such as creating and removing users and setting passwords, use the User Provider Control.

Security: For retrieving user information, the roles you specify in the WebLogic Administration Portal Authentication Security Provider Service determine whether or not the user can perform the action. To invoke the getAllGroupNames method, the caller must be in the PortalSystemAdministrator role.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Adding Portal Controls to Java Page Flows

Using Portal Controls

User Provider Control

Profile Control

Tutorial: Creating a Login Control Page Flow Using the Wizard

User/Group Management JSP Tags

User Login Control

The User Login Control provides a convenient way to add login and logout functionality to a Page Flow.

This control provides a login form that you can add to a JSP. An example of adding a form to a JSP is described in Using Portal Controls.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Adding Portal Controls to Java Page Flows

Using Portal Controls

User Provider Control

Profile Control

Implementing Authentication

Tutorial: Creating a Login Control Page Flow Using the Wizard

User/Group Management JSP Tags

User Provider Control

The User Provider control provides a convenient way to incorporate user management actions into your Page Flows, such as creating users, getting a list of users, and setting passwords.

Use the atnProvider attribute to specify which authentication provider contains the users you want to manage with the control.

Security: For user and group management actions, the roles you specify in the WebLogic Administration Portal Authentication Security Provider Service determine whether or not the user can perform the action.

If you use the createUser action, a user profile is automatically created for the user if you perform post−user−creation−processing with the attributes below. Otherwise, you can use the Profile Control to create a user profile for the new user; or have a profile created for the user automatically when the user next logs in.

Following are descriptions of properties on the control:

doPostProcess

Optional − Works in conjunction with the fireEvent, login, and saveAnonymous properties. If this property and the fireEvent, login, and saveAnonymous properties are set to true, all three events occur after user creation. If the doPostProcess property is set to false, the individual settings on the other three properties are used.

fireEvent

Optional − If set to true, a UserRegistrationEvent is dispatched to the event service during the post−user−creation process. Defaults to true. If this property is set to false, doPostProcess is ignored.

login

Optional (Boolean) − If set to true, the user is logged in during the post−user−creation process. Defaults to true. If true, a user profile is created automatically for the user at login. If this attribute is set to false, doPostProcess is ignored, and you must either create a profile for the user or have a profile created for the user automatically when the user next logs in.

saveAnonymous

Optional − If set to true, any properties the user may have set during the session before registering are added to the new user's properties during the post−user−creation process. Defaults to true. If this property is set to false, doPostProcess is ignored.

See Using Portal Controls for an example use of the User Provider control.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Adding Portal Controls to Java Page Flows

Using Portal Controls

Group Provider Control

Profile Control

Tutorial: Creating a Login Control Page Flow Using the Wizard

User/Group Management JSP Tags

Click Content Event Control

The Click Content Event control provides a convenient way to dispatch a ClickContentEvent to the event service. Use this control in a Page Flow for one of two primary purposes:

- To trigger a campaign action when the event occurs.
- To persist information about the event for use in behavior tracking and analytics.

This control may not be used in a Java Web Service, because the request and session objects it requires are unavailable from a Java Web Service.

You can obtain the Session and Request objects from a Page Flow with the following code:

```
HttpServletRequest request = this.getRequest();
```

The control's dispatch action lets you pass (HttpServletRequest *request*, String *documentType*, String *documentId)* to the event service for the clicked content.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Using Session, Request, and Event Properties in Campaigns

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

Interaction Management JSP Tags

Display Content Event Control

The Display Content Event control provides a convenient way to dispatch a DisplayContentEvent event to the event service. Use this control in a Page Flow for one of two primary purposes:

- To trigger a campaign action when the event occurs.
- To persist information about the event for use in behavior tracking and analytics.

This control may not be used in a Java Web Service, because the request and session objects it requires are unavailable from a Java Web Service.

You can obtain the Session and Request objects from a Page Flow with the following code:

HttpServletRequest request = this.getRequest();

The control's dispatch action lets you pass (HttpServletRequest *request*, String *documentType*, String *documentId)* to the event service for the displayed content.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Using Session, Request, and Event Properties in Campaigns

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

Interaction Management JSP Tags

Generic Event Control

The Generic Event control provides a convenient way to dispatch a non−tracked event (whose name you specify) to the event service. A non−tracked event is not designed to be persisted (unless you have implemented your own event listener and persistence classes). Use this control in a Page Flow to trigger a campaign action when the event occurs.

For the eventType property, enter the name of the event you want to dispatch.

This control may not be used with a Java Web Service, because the request object it requires is unavailable from a Java Web Service.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Using Session, Request, and Event Properties in Campaigns

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

Generic Tracking Control

The Generic Tracking control provides a convenient way to configure and dispatch a tracked event to the event service. Tracked events are designed to be persisted, such as in a database. Use this control in a Page Flow for one of two primary purposes:

- To trigger a campaign action when the event occurs.
- To persist information about the event for use in behavior tracking and analytics.

For the eventType property, enter the name of the event you want to dispatch.

This control may not be used with a Java Web Service, because the request object it requires is unavailable from a Java Web Service.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Using Session, Request, and Event Properties in Campaigns

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

Rule Event Control

The Rule Event Control control provides a convenient way to dispatch a RuleEvent to the event service. Use this control in a Page Flow for one of two primary purposes:

- To trigger a campaign action when the event occurs.
- To persist information about the event for use in behavior tracking and analytics.

This control may not be used in a Java Web Service, because the request and session objects it requires are unavailable from a Java Web Service.

You can obtain the Session and Request objects from a Page Flow with the following code:

HttpServletRequest request = this.getRequest();

The control's dispatch action lets you pass (HttpServletRequest *request*, String *rulesetName*, String *ruleName*) to the event service.

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

Session Login Event Control

The Session Login Event control provides a convenient way to dispatch a SessionLoginEvent event to the event service. Use this control in a Page Flow for one of two primary purposes:

- To trigger a campaign action when the event occurs.
- To persist information about the event for use in behavior tracking and analytics.

This control may not be used in a Java Web Service, because the request and session objects it requires are unavailable from a Java Web Service.

You can obtain the Session and Request objects from a Page Flow with the following code:

HttpServletRequest request = this.getRequest();

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Using Session, Request, and Event Properties in Campaigns

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

User Registration Event Control

The User Registration Event control provides a convenient way to dispatch a UserRegistrationEvent event to the event service. Use this control in a Page Flow for one of two primary purposes:

- To trigger a campaign action when the event occurs.
- To persist information about the event for use in behavior tracking and analytics.

This control may not be used in a Java Web Service, because the request and session objects it requires are unavailable from a Java Web Service.

You can obtain the Session and Request objects from a Page Flow with the following code:

```
HttpServletRequest request = this.getRequest();
```

Javadoc

For property, method, and other details on this control, see the control's Javadoc.

Related Topics

Using Session, Request, and Event Properties in Campaigns

Adding Portal Controls to Java Page Flows

Using Portal Controls

Tutorial: Creating a Login Control Page Flow Using the Wizard

Building a Struts Application

Struts is a Java−based navigation framework that is part of the Apache Jakarta Project. Any Struts applications that are intended for use in a portal *must* be developed as Struts modules, including the usage of the html:link tag for any URLs used in JSPs. Without this, it is impossible for the portal framework to perform the necessary URL rewriting that is required to transparently modify links when the Struts application is used within a portlet.

For information on Struts and Struts development (especially with regard to developing Struts modules), see http://jakarta.apache.org/struts/.

After you build a Struts application you can integrate it into WebLogic Workshop and surface it in a portal user interface. See Integrating Struts Applications.

Using Page Flows

WebLogic Platform provides an event−driven navigation framework called Page Flows that is built on top of the Struts framework and provides many advantages that Struts does not. See the following topics for more information.

- Advantages of Using Page Flows
- Interoperating With Struts and Page Flows

Related Topics

Building Different Types of Applications

Building a Commerce Application

This section provides instructions on creating the necessary framework on which to build commerce applications.

Adding Commerce Services to an Application

Provides instructions on making your portal application commerce−enabled by installing commerce services.

Enabling Catalog Management

Provides instructions in installing the tools necessary to build and manage catalogs.

Creating Catalog Structure Properties

With catalog management enabled, this topic provides instructions on creating the catalog properties that are used to describe items in your catalog.

Creating Discounts

Provides instructions on creating discounts for stand−alone use or for use by campaigns.

Related Topics

Building Different Types of Applications

Adding Commerce Services to an Application

You can add commerce functionality to your portal application, which adds commerce services, a commerce API, and a set of JSP tags to the portal application and portal Web project. Commerce services include catalog, order, shopping cart, tax, payment, and shipping. For technical details on the commerce services, see the WebLogic Portal Javadoc com.bea.commerce.* and com.beasys.commerce.* packages.

To add commerce to your application

1. Open your portal application in WebLogic Workshop Platform Edition.
2. Stop the server.
3. In the Application window, right−click the portal application folder and choose ***Install−−>Commerce Services***. The following files, APIs, and JSP tags are added to your portal application:

    - <PORTAL_APP>\Modules\commerce.jar
    - <PORTAL_APP>\Modules\toolSupport.war Web application

      Contains services for campaign cleanup, ad clickthrough behavior, cache management, non−text binary content viewing, campaign e−mail previewing, placeholder previewing, and adds catalog browsing. Replaces wps−toolSupport.war.
    - <PORTAL_APP>\Libraries\commerce_util.jar

3. Install the commerce JSP tags in a portal Web Project. In the Application window, right click the portal Web project folder and choose ***Install−−>Commerce Taglibs***. The following JSP tag libraries are added:
    - <project>\WEB−INF\lib\cat_taglib.jar
        - getProperty Tag
        - iterateViewIterator Tag
        - iterateThroughView Tag
        - catalogQuery Tag
        - catalogSelector Tag
    - <project>\WEB−INF\lib\eb_taglib.jar
        - smnav Tag
    - <project>\WEB−INF\lib\productTracking_taglib.jar
        - displayProductEvent Tag
        - clickProductEvent Tag
4. Restart the server.

To develop commerce functionality

Use the commerce APIs and JSP tags to develop commerce functionality in your portal application. See the Portal Javadoc for API details.

Related Topics

Creating Discounts

Creating Campaigns

Portal JSP Tags

Enabling Catalog Management

Creating Catalog Structure Properties

Creating User Profile Properties

Registering Custom Events

Creating Session Properties

Creating Request Properties

Enabling Catalog Management

You can add WebLogic Portal catalog administration functionality to your portal applications. The following procedure allows you to create and manage catalog categories and content items. You can also create and manage catalog property sets in the WebLogic Workshop Portal Extensions and manage them in the online catalog administration tools that you add with this procedure.

1. Copy tools700.war into the enterprise application root folder.

   From the <WEBLOGIC_HOME>/portal/lib directory, copy the tools700.war file into your enterprise application directory. For example, to add this functionality to the sample portal application, place this file in <WEBLOGIC_HOME>/samples/portal/portalApp. The next time you open this application in WebLogic Workshop, the catalog administration tools are deployed automatically.

   The location, as well as the Web application name, are changed in the new release of WebLogic Portal.

   Note: If your server's ports are different than 7501/7502, unzip tools700.war, open WEB−INF/web.xml, change the port numbers to match your server's ports, save web.xml, and re−war the Web application.
2. If commerce services haven't been installed in the application, right−click the enterprise directory in WebLogic Workshop and choose ***Install −−> Commerce Services***.
3. Copy the WebFlow files supporting the commerce administration tools into the new application:
   a. In the /samples/portal/portalApp/META−INF/data directory in your WebLogic 8.1 installation directory, create a directory called webapps, and within that directory, create one called tools700.
   b. From the /beaApps/sampleportal−project/application−sync/webapps/tools/ directory in the WebLogic Portal 7.0 SP2 domain, copy the tools' webflow files into the newly−created /webapps/tools700 directory.
   c. If the context root of your tools is named anything other than tools700, rename the tools700 directory to the context root of your tools.
4. To use these tools, open a Web brower and navigate to http://<hostname>:<port>/tools700 and click Catalog Management.

   Be sure to log in as a user that is in the Administrators user group.
5. In WebLogic Workshop, you can create catalog structure properties to structure the properties in your catalog.
6. Since the catalog will be used with commerce functionality, be sure to add commerce services to your application. See Adding Commerce Services to an Application.

Related Topics

Creating Catalog Structure Properties

Adding Commerce Services to an Application

Creating Catalog Structure Properties

The Catalog Structure Properties designer lets you add properties to your catalog. To use catalog structure properties, you must enable the WebLogic Portal Administration Tools that support catalog management.

Catalog Structure Properties are name/value pairs that lets you define the information you want to enter about items in your catalog, such as "SKU," "Description," and "Price."

To create a Catalog Structure Property Set

1. Enable catalog management, which provides the catalog tools that use the catalog structure properties. See Enabling Catalog Management.
2. In the Application window, right−click the `data\catalog` folder and choose ***File−−>New−−>Other File Types*** .
3. In the New File window, select ***Catalog Structure Property Set*** in the right pane.
4. In the ***File name*** field, enter a name for the property set. Make sure you keep the file extension.
5. Click ***Create***. The Property Set designer appears.
6. Use the next procedure to add properties to the property set.

To add properties to a property set

After you create a property set, you add the properties you want to it.

1. In the Palette window, drag one of the types of properties into the Property Set Designer.

   The type defines the number of values that can be entered for the property. Following are descriptions of each type.

   Single Unrestricted − A single unrestricted property can have only one value, but you can enter any value.

   Single Restricted − A single restricted property can have only one value, and you are restricted to selecting that value from a predefined list.

   Multiple Unrestricted − A multiple unrestricted property can have multiple values, and you can enter any values.

   Multiple Restricted − A multiple restricted property can have multiple values, and you are restricted to selecting the values from a predefined list.
2. In the Property Editor window:
   - ♦ Enter a name and description for the property
   - ♦ Select the ***Data Type*** for the property value. For example, if you select Boolean, your property value can be only true or false. (Properties with a Boolean data type are automatically set to "single restricted.")
   - ♦ In the ***Selection Mode*** and ***Value Range*** fields, you can change the type of property. For example, you can change a property from "single unrestricted" to "multiple restricted." ***Note***: Any change to Data Type, Selection Mode, or Value Range removes anything previously entered in the Values field.

- Use the ***Values*** field to enter values for "restricted" types or to set the default value(s) for "unrestricted types." Click the ellipsis icon (...) to enter values. (In the Enter Property Value dialog box that appears, click ***Add*** after each entry, and click ***OK*** when all values are entered.)

3. Save the file after you have added all the properties you want.

To modify properties and their values

To modify properties and their values, double−click the property set file in the the Application window, click the property you want to modify, and change the values in the Property designer window.

To delete properties

You can delete individual properties from a property set, and you can delete property sets.

To delete a property from a property set, open the property set file, select the property, and press the ***Delete*** key.

To delete a property set, select the property set file in the Application window and press the ***Delete*** key.

Related Topics

Property Set Designer window

Building a Commerce Application

Creating Segments

Creating Content Selectors

Creating Campaigns

Creating User Profile Properties

Creating Session Properties

Creating Request Properties

Registering Custom Events

Creating Discounts

Use the WebLogic Workshop Portal Extensions Discount Designer to create discounts that can be used on the items in your catalog or shopping cart.

You can create discounts to globally apply to all users or to be used exclusively in campaigns.

Before You Create Discounts

To create discounts that can be incorporated into your commerce application, you must enable commerce functionality. Also, you are going to create discounts based on product categories or items, you must set up a product catalog against which discounts can be applied. Use the following topics for guidance:

Adding Commerce Services to an Application

Enabling Catalog Management

Creating Catalog Structure Properties

To create a Discount

1. If your server is not running, start it by choosing ***Tools−−>WebLogic Server−−>Start WebLogic Server***.
2. In the Application window, right−click the ***data\discounts\DefaultDiscountSet*** folder and choose ***New−−>Other File Types***.
3. In the New File window, select the ***Discount*** in the right pane.
4. In the ***File name*** field, enter a name for the Discount. Make sure you keep the file extension.
5. Click ***Create***. The Discount Designer appears.
6. In the Discount Designer, click the ***Wizard*** icon to use the Discount Wizard to create the discount.

   OR
7. If you create the discount manually (without the wizard), select the ***Discount type*** option you want (discount on a single item, on a set of items, or on an entire order. If you select ***Set−based discount***, determine whether there is a limit to the number of discounts in the order.
8. In the ***Discount terms*** pane, click ***Add a Trigger*** to set the items or order properties for which the discount will be used.
9. Click ***Add a Discount*** to define a discount percentage or dollar amount.
10. Click ***Add a Target*** to set the items to which the discount is applied.
11. In the ***Property Editor***, set the following properties:

Description	For campaign discounts. The text you enter can be displayed in the shopping cart when the discount is applied.
Usage	Select whether the discount will apply to all users (Stand−alone) or whether it will be used in campaigns (Campaign)
Explanation	For stand−alone discounts only. The text you enter can be displayed in the shopping cart when the discount is applied.
Priority	

	The number that determines which discount is used if there are competing discounts (1 is the highest priority).
Overall limit	You can set the number of times a customer can receive the discount. Enter 0 if there is no limit.

Required

Start Date and ***Stop Date***	Set the range of time for the discount to be in effect. Start and stop date must be set to be able to finalize the discount.

12. When the discount is final and ready to use, select the ***Finalize this discount*** option.
13. Save the discount.

If you created a campaign discount, the discount will be available for use when you create campaigns.

Related Topics

Discount Designer window

Building a Commerce Application

Creating Campaigns

Creating Portals for Mobile Devices

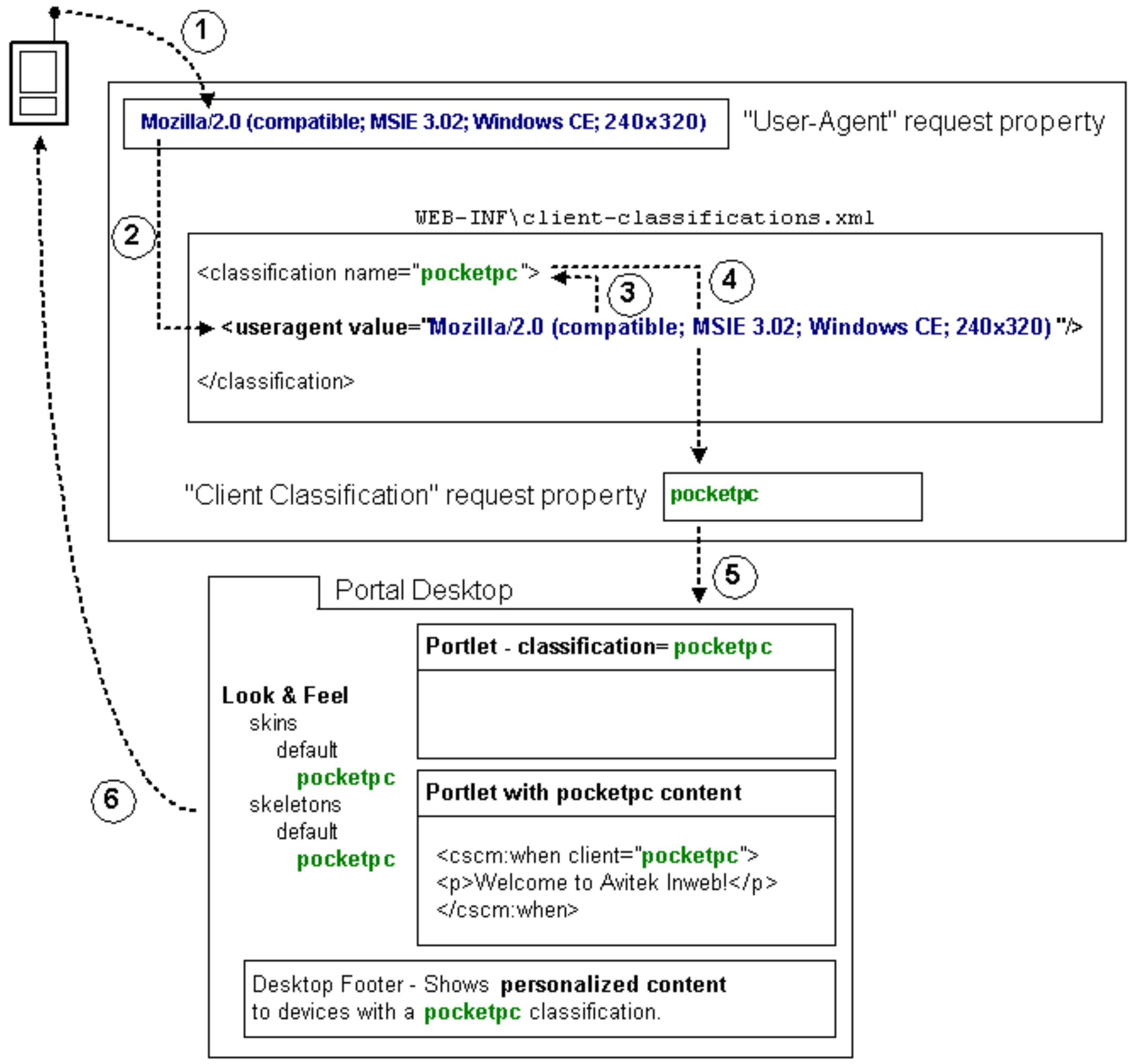
There are many types of Web−enabled mobile devices that can access your portals. Since these devices have different interfaces and different−sized viewing areas, each has a unique requirement for the type of content they display.

With the multichannel framework provided in WebLogic Workshop Portal Extensions, you can extend your portals to include support for mobile device access. This flexible framework lets you create a single portal that serves content to Web−capable devices seamlessly and simultaneously. You can also serve different content to different browsers, such as Mozilla, Netscape, Opera, and Internet Explorer.

The multichannel framework allows the following processes to occur: You can build specific content and look and feel elements for specific devices. When a device accesses a portal, the portal knows what kind of device it is and automatically serves the device the content you created for it.

When a device (whether it's a PC or a handheld) accesses a portal, it sends information about itself to the portal in the HTTP header information such as the type of browser being used and the type of device. This combination of information defines a "client," which is equivalent to the model of a device. Clients, in turn, can be grouped into "classifications." For example, there are many models of Palm handheld devices, but they all fall under the classification of "Palm." Classifications are the key element in enabling multichannel support in portals.

The following illustration and table describe the multichannel framework and provide instructions for building content and presentation for mobile devices.


1
Mozilla/2.0 (compatible; MSIE 3.02; Windows CE; 240x320)
"User-Agent" request property
2
WEB-INF\client-classifications.xml
<classification name="pocketpc">
3
4
<useragent value="Mozilla/2.0 (compatible; MSIE 3.02; Windows CE; 240x320)"/>
</classification>
"Client Classification" request property
pocketpc
5
Portal Desktop
Portlet - classification= pocketpc
Look & Feel
skins
default
pocketpc
skeletons
default
pocketpc
6
Portlet with pocketpc content
<cscm:when client="pocketpc">
<p>Welcome to Avitek Inweb!</p>
</cscm:when>
Desktop Footer - Shows personalized content to devices with a pocketpc classification.

1	When a device accesses a portal−enabled server with a URL, the device sends a user−agent string in the HTTP header that tells what kind of client it is. The server stores this user−agent string in the "User−Agent" request property for the portal application. The "User−Agent" request property is automatically included with any portal application you create in WebLogic Workshop Platform Edition. To view this property, open the following file in WebLogic Workshop: <PORTAL_APP>\data\request\DefaultRequestPropertySet.req. ***Portal developer tasks***: None. This happens automatically.
2	To enable multichannel support for devices, a portal Web project must be able to map the user−agent string stored in the "User−Agent" property to a classification. This mapping must be created before portals are accessed by mobile devices. ***Portal developer tasks***: You must map clients to classifications in your portal Web project

WEB−INF\client−classifications.xml file. The default client−classifications.xml file contains default client mappings.

For each client entry that maps to a classification, you can enter either an explicit user−agent string that maps exactly to what a device sends, or you can enter a regular expression that can encompass multiple user−agent strings.

The following example of a client classification mapping in client−classifications.xml shows explicit mappings (with the <useragent> tag) and a regular expression mapping (with the <useragent−regex> tag).

```
<classification name="pocketpc" description="For the PocketPC">
        <useragent value="Mozilla/2.0 (compatible; MSIE 3.02; Windows CE; 240x320)"/>
        <useragent value="Mozilla/2.0 (compatible; MSIE 3.02; Windows CE; PPC; 240x320)"/>
        <useragent-regex value=".*PDA; Windows CE.*NetFront/3.*" priority="1"/>
</classification>
```

An explicit <useragent> value can be used for only one classification. If you use more than one <useragent−regex> tag to map with regular expressions, it is possible that a device accessing a portal could map to more than one classification. To determine which classification the device is mapped to, use the priority attribute, as shown above. The value "1" is the highest priority. Enter any whole number for the priority value.

Note: For portlets that are assigned client classifications, the classification "description" value is used in the WebLogic Administration Portal to show which classifications the portlet is assigned to. Write descriptions that are easily understood by portal administrators.

For information on user−agent strings and values for different devices, perform a Web search for "user−agent."

3 Because of the client−classification.xml mappings you defined, the user−agent string stored in the "User−Agent" property is mapped to the classification name you provided. In the example mapping above, the name is "pocketpc".

Portal developer tasks: None. This happens automatically.

4 After the client is successfully mapped to a classification, the classification name is stored in the "Client Classification" property in the DefaultRequestPropertySet.

Portal developer tasks: None. This happens automatically.

5 The portal uses that client classification name stored in the DefaultRequestPropertySet throughout the portal framework to identify the content and presentation tailored to the device.

Portal developer tasks: The portal is where you develop and enable specific content and presentation to be used for different mobile devices. The portal framework includes the following touchpoints for creating device−specific content and presentation:

- ***Portlet Development*** − When you create a portlet with the WebLogic Workshop Portal Extensions, you can assign the portlet to be used by different devices (client classifications). With the portlet open in the Portlet Designer, in the Property Editor window, do the following:
    1. Click the ellipsis icon [...] in the ***Client Classifications*** field to launch the Manage Portlet Classifications dialog box.
    2. In the dialog box, select whether you want to enable or disable classifications for the portlet. (If you disable classifications, the portlet is automatically enabled for the classifications you do not select for disabling.)

	3. Move the classifications you want to enable/disable from the Available Classifications list to the Selected Classifications list, and click ***OK***. The list of classifications is read from the client−classifications.xml file. • ***JSP Tags*** − The WebLogic Workshop Portal Extensions include a set of JSP tags for creating device−specific inline content in JSPs. Only the content that meets the device criteria defined by the JSP tag is delivered to the device. The JSP tags have a required "client" attribute for mapping the JSP content to classifications. For that client value in the JSP tag, you must use the exact value used for the name in the client−classification.xml file (the value being stored in the "Client Classification" property in the DefaultRequestPropertySet). See the Mobile Devices JSP Tags for more information. • ***Look & Feel Development*** − The Look & Feels (skins and skeletons) provided with the WebLogic Workshop Portal Extensions include support for a few mobile devices (nokia, palm, and pocketpc). These skins and skeletons are included as subdirectories of the main skins and skeletons in your portal Web projects. For example, a pocketpc skin is included as part of the "default" skin in <project>\framework\skins\default\pocketpc. You can also develop your own skins and skeletons to support different devices. When a Look & Feel is selected for a desktop, the portal framework reads the "Client Classification" property in the DefaultRequestPropertySet and uses the Look & Feel logic to find skin and skeleton directories matching the name of the client classification. For instructions on creating skins and skeletons for Look & Feels, see Creating Skins and Skin Themes and Creating Skeletons and Skeleton Themes. • ***Interaction Management Development*** − With the client classification name being stored in the "Client Classification" property of the DefaultRequestPropertySet, you can build and trigger personalization and campaigns for devices based on that property value. For information on developing personalization and campaigns, see Developing Personalized Applications.
6	Based on the mapping you set up to match user−agent (client) strings in the HTTP request to classification names, the portal sends the device−specific content and presentation you developed to the different devices that access the portal. ***Portal developer tasks***: None. This happens automatically.

Samples

The Tutorial Portal, one of the Portal Samples provided with the WebLogic Workshop Portal Extensions, includes examples of multichannel functionality. Also, when you create a portal Web project, a WEB−INF\client−classifications.xml file is created automatically with default settings.

Any portal Web project you create also includes a default set of multichannel Look & Feels located in skin and skeleton subdirectories (<project>\framework\skins and <project>\framework\skeletons).

Related Topics

Mobile Devices JSP Tags

Creating Look & Feels

Creating Skins and Skin Themes

Creating Skeletons and Skeleton Themes

Creating Portlets

Developing Personalized Applications

WebLogic Portal provides powerful tools for building personalized portal applications. These interaction management tools let you develop personalization and campaigns.

Personalization and Campaigns − With personalization and campaigns, you can target users with personalized content and actions. Based on conditions such as user profile properties, user segment membership, HTTP session or request data, date/time conditions, or events, each user is dynamically served personalized Web content, automatic e−mails, and discounts with pinpoint accuracy.

Steps for Adding Interaction Management to Your Applications

Interaction management development involves setting up interrelated pieces. The following sections describe the steps needed to implement interaction management functionality. Each section contains links to different implementation tasks depending on your needs. Use this topic as an overall roadmap for developing personalized applications.

1. Overview of Content Management
2. Setting up Users and User Properties
3. Designing Interaction Management
4. Creating Personalization Conditions
5. Personalizing Portal Applications

Related Topics

Portal JSP Tags

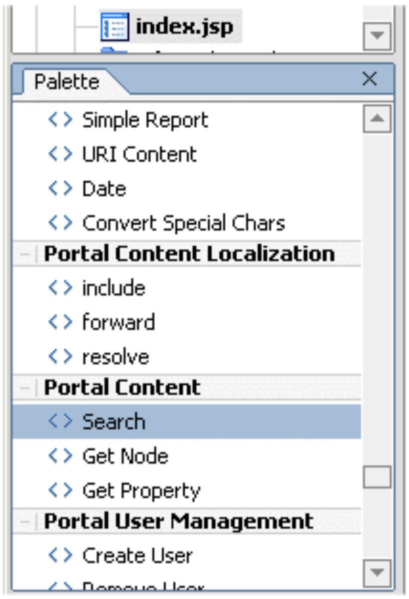
Using Portal JSP Tags

This section introduces tools to assist you in creating the most complete functionality with the least effort possible.

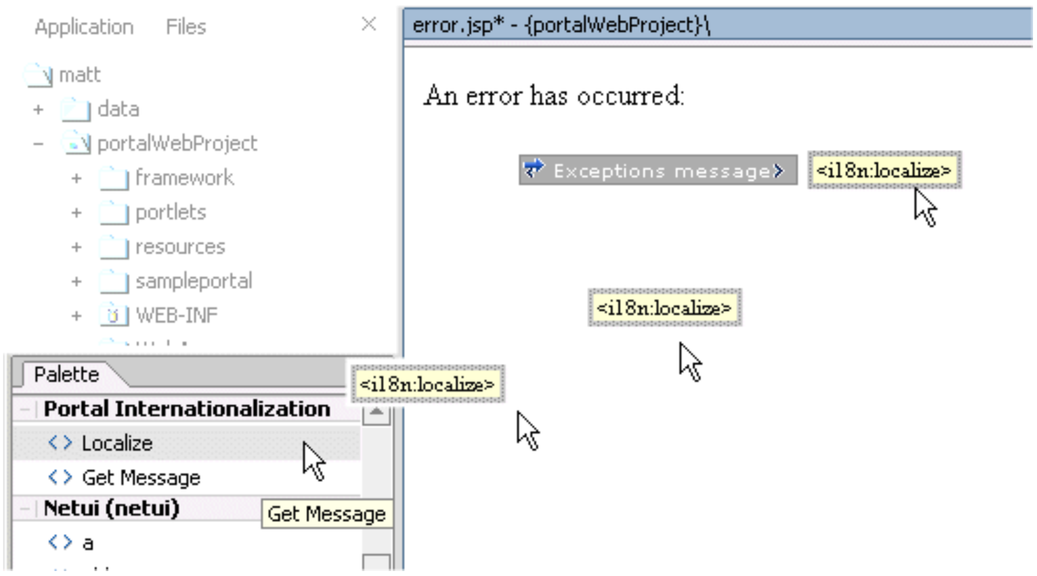
WebLogic Workshop includes many powerful features to help you create Web applications − from custom tag libraries to visual editors with color−coding and strong syntax and error checking. Portal Extensions also includes a lightweight browser to preview your portlets, as well as a Content Preview window to view the results of content queries.

Using JSP Tags

When a JSP is opened in Workshop, the Palette displays all the JSP tags currently loaded and available.

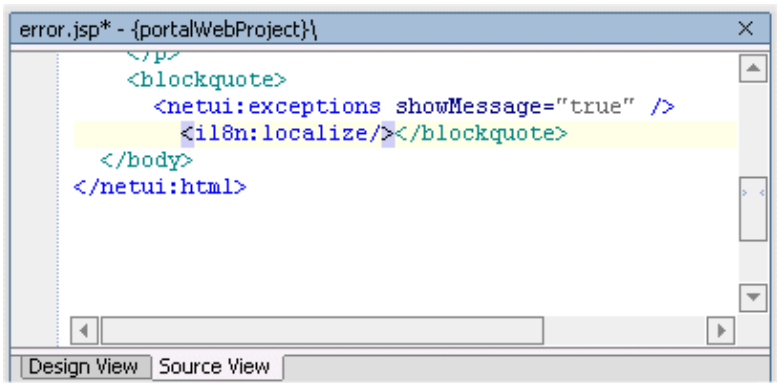


To use a tag, simply drag it into the JSP designer, use the Source View to edit the code directly, and use the Properties Designer to set properties.



Edit the JSP, toggling back and forth between the Source View and Design View tab.

You can also manually start to type the JSP tag in Source View and in many cases the auto−complete feature provides a drop−down selection of tags you can choose from. You can also press Alt−Enter to automatically add the tab library import statement to the top of the JSP.



```
    </p>
    <blockquote>
      <netui:exceptions showMessage="true" />
        <i18n:localize/></blockquote>
  </body>
</netui:html>
```

NOTE: Much of the functionality exposed by the Portal JSP tags has been conglomerated into even simpler objects called Controls. This means that most user management functionality, for example, can be easily exposed with a User Manager Control on a Page Flow. For more fine−grained customization, JSPs are extremely powerful.

JSPs in Portlets

Portlets use JSPs as their content nodes, enabling reuse and facilitating personalization and other programmatic functionality. JSPs are created by WebLogic Workshop and provide a structure for other elements to be added to a portlet.

For instance, using the Palette Window for JSP Tags, you can drag portal tags into the design or the source view of your JSP, and use the Property Designer to make edits to exposed elements of the code.

About JSP Tags and Portlets

The following topics offer overviews and links to more detailed reference material:

- JSP Tag Wizards: When dragged into the Portal Designer , certain Portal JSP tags invoke wizards that automatically populate important tag attributes in your JSP.
- JSP Tags Reference: The tag libraries provided for developing Web applications on WebLogic Platform are documented extensively.
- Pageflow Reference: To use Pageflows effectively, familiarize yourself with annotations, icons, exception handling, and data binding. Actions defined in a Pageflow can be called fom within a JSP, and vice−versa. These calls can be invoked by dragging action icons into the design view of your JSP.

Related Topics

Developing JSPs

How Do I Create a Portlet?

How Do I: Start Using Portals?

How Do I: Debug an Application?

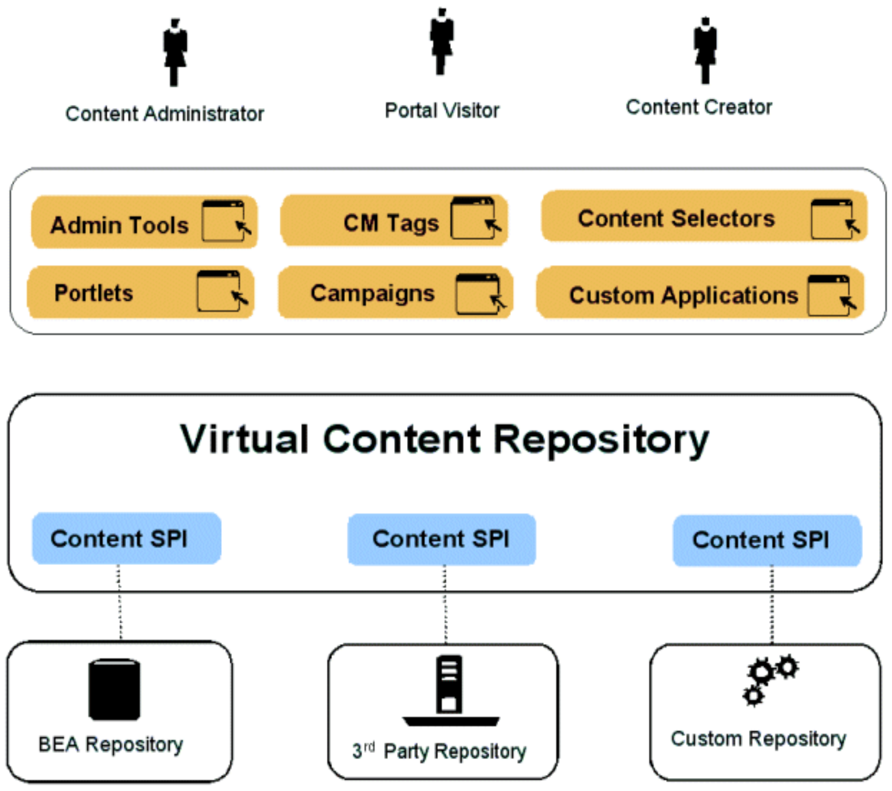
Overview of Content Management

The content you want to show users, whether it is a single line of text, an HTML file, a graphic, or an animation file can be stored in a content repository. BEA's Virtual Content Repository, included with WebLogic Portal, provides a single interface that lets you store content in BEA repositories as well as seamlessly incorporate BEA−compatible third−party content management systems. This overview provides information on the following subjects:

- The Virtual Content Repository
- Content Hierarchy
- Content Types
- Creating and Modifying Content
- Using Content in Personalized Applications

The Virtual Content Repository

The Virtual Content Repository can contain multiple content repositories. It provides services such as federated search (a search that returns a result set from all the relevant content across the plugged in repositories), content lifecycle management, Delegated Administration and content type management. Many Portal subsystems interact with the Virtual Content Repository. Content Management tags execute queries to deliver dynamic content to end users. Content Selectors and Campaigns deliver dynamic, personalized content to user based upon personalization rules or conditions.



The Content Hierarchy

WebLogic Portal Content Management is organized hierarchically. The Virtual Content Repository (VCR) is the top−level node in the content management system. Repositories are the immediate children of the VCR. These repositories can be made up of multiple BEA Systems repositories, multiple third−party repositories, or custom content repositories.

Hierarchy Nodes and Content Nodes comprise the next level of the hierarchy tree and are organized much like a file system. Hierarchy Nodes can contain both Hierarchy Nodes and Content Nodes. Content Nodes can only contain other Content Nodes. Nodes can be created based upon Content Types. For example:

```
Virtual Content Repository

        Repository 1

                Hierarchy Node

                        ContentNode (index.htm)

                                ChildContent1 (logo.gif)
                                ChildContent2 (photo.jpg)
```

Content Repositories provide the storage mechanism for content, and they comprise the second−level of the Virtual Content Repository hierarchy. Content Repositories may include multiple instances of BEA repositories, 3rd party repositories, or customer repositories. To plug into the Virtual Content Repository, you must implement the BEA Content Management Service Provider Interface   the CM SPI.

Hierarchy Nodes are organizational mechanisms that help you organize and group content in the hierarchy, much like folders in a file system. Hierarchy Nodes can contain other Hierarchy Nodes as well as Content Nodes. They can also be typed so that they function similarly to Content Nodes.

Content Nodes represent content stored in the repository. A complete content node comprises a set of data property values defined by a content type. This data structure may include files such as a word processing document, HTML file, spreadsheet or image. It may also include metadata such as the author, version number or summary. Content Nodes can also have child Content Nodes. For example, The Content Node for an HTML document may have child Content Nodes for the images used by the HTML document.

Content Types

Content Types define the set of properties that make up a Content Node or Hierarchy Node. This may include any combination of the supported data types, such as date and time, number, text (string), Boolean (true/false), or binary (file).

For example, the Content Type for image content may have a number property "width" and a number property "height," while the Content Type for news article content my have a text property "Author", a text property "Summary", a date property "Published Date", and a binary property "Article" for a file containing the formatted article. Types do not have to include a binary, although a common example of a type is a single binary with a set of non−binary properties that describe the document.

Repository 1

    Content Type 1

        Property 1 = Binary
        Property 2 = String

    Content Type 2

Content Types also define the available values for a given property, including whether it can contain multiple values. For example, a property called "Priority" may only allow a single choice among the values "High", "Medium", and "Low", while a property called "Favorite Color" may allow multiple pre−defined values to be chosen.

Each repository has its own set of content types. You can create types in BEA repositories and third−party repositories that support this feature.

Creating and Modifying Content

After you connect a BEA−compatible content management system to the Virtual Content Repository you can continue to add and modify content directly in your BEA−compatible content management system. Changes appear automatically in the Virtual Content Repository. You can create and manage content in the Administration Portal, in the My Content Portlet, or with the bulkloader. For more information, see "Creating Content."

Using Content in Personalized Applications

WebLogic Workshop extensions support development of personalized applications, while the WebLogic Administration Portal enables portal administrators to adapt site interaction to fit the needs of the audience. The core of the Personalization system is the underlying rules engine that matches users with appropriate content. Content Selectors, Placeholders and Campaigns are the aspects of content management visible to administrators. Also, User Segments contain the criteria that define the target visitor, such as gender or browser type.

The Content Management component provides the run−time API by which content is queried and retrieved. The functionality of this component is accessible via tags. The content retrieval functionality is provided using either the provided reference implementation or third−party content retrieval products.

Related Topics

Creating Content

Setting up Users and User Properties

Designing Interaction Management

Creating Personalization Conditions

Personalizing Portal Applications

Unified User Profiles Overview

If you have an existing store of users, groups, and additional properties (such as address, e−mail address, phone number, and so on), unified user profiles are a necessary part of bringing those user properties into the WebLogic Portal environment, where they can be used for retrieving and editing property values and setting up personalization, delegated administration, and visitor entitlements.

This topic describes the unified user profile, when to use it, and when not to use it.

Note: This topic contains the terms "user store" and "data store." A user store can contain users and groups, as well as additional properties. A data store implies that the store does not have to contain users and groups. It can simply contain properties.

What is a Unified User Profile?

Here is an example that explains what a unified user profile is and does:

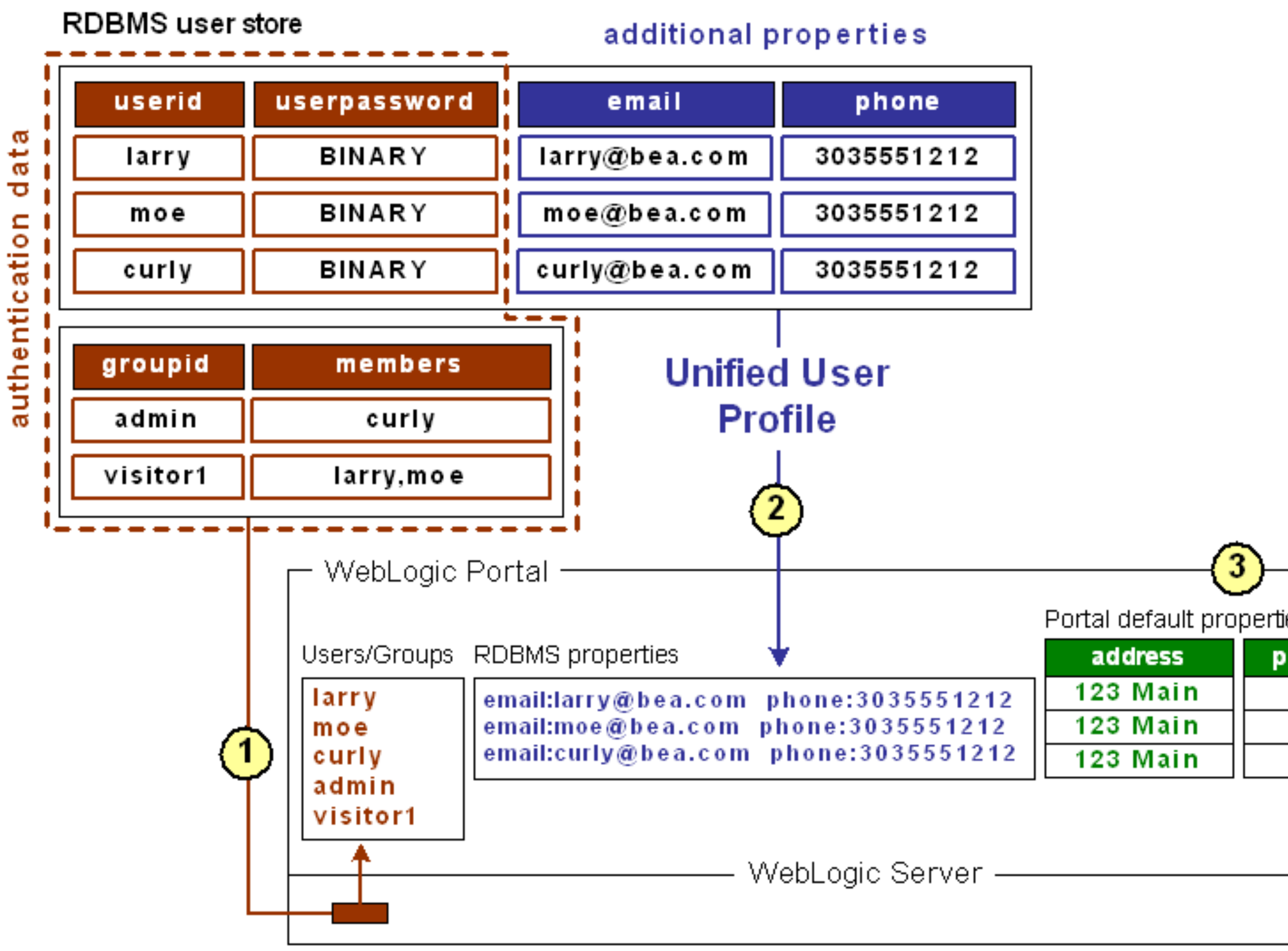
Let's say you're creating a new portal application that you want users to be able to log in to. Let's also say your users are stored in an RDBMS user store outside of the WebLogic environment. You could connect WebLogic Server (your portal application's domain server instance) to your RDBMS system, and your users could log in to your portal application as if their usernames and passwords were stored in WebLogic Server. If authentication was all you wanted to provide through your RDBMS user store, you could stop here without needing a unified user profile.

However, let's say you also stored e−mail and phone number information (properties) for users in your RDBMS user store, and you wanted to be able to access those properties in your portal applications. In this case, you need to create a unified user profile for your RDBMS user store that lets you access those additional properties from your code.

Technically speaking, a unified user profile is a stateless session bean you create (with associated classes) that lets WebLogic Portal read property values stored in external data stores, such as LDAP servers and databases. Once connected to an external data store with a unified user profile, you can use portal JSP tags, controls, and the WebLogic Portal API to retrieve user property values from that store. You can also take the extra step of surfacing these external properties in the WebLogic Administration portal, where the properties can be used to define rules for personalization, visitor entitlements, and delegated administration.

Whether or not you have additional properties stored in your external user store, the external users and groups you connect to WebLogic Server are automatically assigned the default user property values you have set up in WebLogic Portal, without the use of a unified user profile. With the WebLogic Administration Portal, you can change the default WebLogic Portal property values for those users. These values are stored in WebLogic Portal's RDBMS data store using the Portal schema.

The following figure shows where a unified user profile fits between an external user store and the WebLogic environment.



1	This external RDBMS user store, which supports authentication, contains users (principals) and passwords in one database table and groups (principals) in another. Giving a user store authentication capabilities (as an authentication provider or identity asserter) involves configuration steps not associated with the unified user profile configuration process. (See Developing Security Providers for WebLogic Server.) Unified user profile configuration is not dependent on the authentication provider configuration and vice versa. Once the RDBMS authentication provider is connected to WebLogic Server, WebLogic Server (and WebLogic Portal) can see those users and groups. Those users can log in to your portal applications, and you can include those users and groups in your rules for personalization, delegated administration, and visitor entitlements. Also, WebLogic Portal's ProfileWrapper maps the principals to properties kept in the Portal schema, thereby establishing the user profile.
2	***Unified User Profile*** − The same external table that contains users and passwords also contains additional properties (email and phone) for each user. These additional properties are not part of authentication; but they are

	part of each user's profile. If you want to access these properties in your portal applications (with the WebLogic Portal JSP tags, controls, or API), you must create a unified user profile for the RDBMS user store. When you create the unified user profile, the ProfileWrapper includes the external properties in the user profile. The unified user profile consists of a stateless session bean and associated classes that you create. If you want to surface any of these properties in the WebLogic Administration Portal to be used in defining rules for personalization, delegated administration, or visitor entitlements, create a user profile property set for the external user store in addition to implementing your unified user profile session bean. The property set provides metadata about your external properties so that WebLogic Workshop and the WebLogic Administration Portal know how to display them. Properties from an external data store are typically read only in the WebLogic Administration Portal.
3	WebLogic Portal lets you create user/group properties and set default values for those properties. Any user or group in WebLogic Server, whether created in the default LDAP store or brought in through a connection to an external user store, is automatically assigned those default property values; and you can change the default values for each user or group, programmatically or in the WebLogic Administration Portal. This does not involve unified user profiles, because the properties to be retrieved are local, not stored in an external data store. In the illustration, after the authentication provider or identity asserter provides the principals, the ProfileWrapper combines the principals with the external properties of email and phone (retrieved by the unified user profile) and the default WebLogic Portal properties of address and postal code, all of which make up the full user profile.

What a Unified User Profile is Not

A user profile is not a security realm, and it does not provide authentication. It is not even the external user store itself. It is the connection (stateless session bean with associated classes) that lets you read properties in the external user store.

When Should You Create a Unified User Profile?

Create a unified user profile for an external data store if you want to do any of the following:

- Use WebLogic Portal's JSP tags, controls, or API to retrieve property values from that external store.
- Surface external properties in the WebLogic Administration Portal for use in defining rules for personalization, delegated administration, or visitor entitlements. Users and groups are not considered properties.

When Don't You Need a Unified User Profile?

You do not need to create a unified user profile for an external data store if you only want to:

- Provide authentication for users in the external user store.
- Define rules for personalization, delegated administration, or visitor entitlements based only on users or groups in an external user store, not on user properties.
- Define rules for personalization, delegated administration, or visitor entitlements based on the WebLogic Portal user profile properties you create in WebLogic Workshop, which are kept in the Portal schema.

Setting up a Unified User Profile

See Setting up Unified User Profiles.

Related Topics

Using Multiple Authentication Providers in Portal Development (external user stores)

Setting up Unified User Profiles

This topic provides guidelines and instructions on creating a unified user profile to access user/group properties from an external user store. (See Unified User Profiles Overview for overview information.)

Best Practices: When possible, use WebLogic Portal's user profile functionality (default UserProfileManager) to assign properties to users and groups. Given the choice between creating and storing additional properties in an external user store (which requires write access to that external store, which must be implemented) and creating and storing them in WebLogic Portal, doing so in WebLogic Portal can greatly improve performance on accessing property values. If you are storing users and groups in an external store, the ideal configuration is storing only users, groups, and passwords in the external store and creating and setting additional properties in WebLogic Portal. With that configuration, performance is optimal and you do not have to create a unified user profile.

To create a UUP to retrieve user data from external sources, complete the following tasks:

Create an EntityPropertyManager EJB to Represent External Data

Deploy a ProfileManager That Can Use the New EntityPropertyManager

If you have an LDAP server for which you want to create a unified user profile, WebLogic Portal provides a default unified user profile you can modify. See Retrieving User Profile Data from LDAP.

Create an EntityPropertyManager EJB to Represent External Data

To incorporate data from an external source, you must first create a stateless session bean that implements the methods of the com.bea.p13n.property.EntityPropertyManager remote interface. EntityPropertyManager is the remote interface for a session bean that handles the persistence of property data and the creation and deletion of profile records. By default, EntityPropertyManager provides read−only access to external properties.

In addition, the stateless session bean should include a home interface and an implementation class. For example:

```
MyEntityPropertyManager
extends com.bea.p13n.property.EntityPropertyManager

MyEntityPropertyManagerHome
extends javax.ejb.EJBHome
```

Your implementation class can extend the EntityPropertyManagerImpl class. However the only requirement is that your implementation class is a valid implementation of the MyEntityPropertyManager remote interface. For example:

```
MyEntityPropertyManagerImpl extends
com.bea.p13n.property.internal.EntityPropertyManagerImpl
```

or

```
MyEntityPropertyManagerImpl extends
javax.ejb.SessionBean
```

Recommended EJB Guidelines

We recommend the following guidelines for your new EJB:

- Your custom EntityPropertyManager is not a default EntityPropertyManager. A default EntityPropertyManager is used to get/set/remove properties in the Portal schema. Your custom EntityPropertyManager does not have to support the following methods. It can throw java.lang.UsupportedOperationException instead:
    - getDynamicProperties()
    - getEntityNames()
    - getHomeName()
    - getPropertyLocator()
    - getUniqueId()
- If you want to be able to use the portal framework and tools to create and remove users in your external data store, you must support the createUniqueId() and removeEntity() methods. However, your custom EntityPropertyManager is not the default EntityPropertyManager so your createUniqueId() method does not have to return a unique number. It must create the user entity in your external data store and then it can return any number, such as −1.
- The following recommendations apply to the EntityPropertyManager() methods that you must support:
    - getProperty() − Use caching. You should support the getProperties() method to retrieve all properties for a user at once, caching them at the same time. Your getProperty() method should use getProperties().
    - setProperty() − Use caching.
    - removeProperties(), removeProperty() − After these methods are called, a call to getProperty() should return null for the property. Remove properties from the cache, too.
- Your implementations of the getProperty(), setProperty(), removeProperty(), and removeProperties() methods must include any logic necessary to connect to the external system.
- If you want to cache property data, the methods must be able to cache profile data appropriately for that system. (See the com.bea.p13n.cache package in the WebLogic Portal Javadoc.)
- If the external system contains read−only data, any methods that modify profile data must throw a java.lang.UnsupportedOperationException. Additionally, if the external data source contains users that are created and deleted by something other than your WebLogic Portal createUniqueId() and removeEntity() methods can simply throw an UnsupportedOperationException.
- To avoid class loader dependency issues, make sure that your EJB resides in its own package.
- For ease of maintenance, place the compiled classes of your custom EntityPropertyManager bean in your own JAR file (instead of modifying an existing WebLogic Portal JAR file).

Before you deploy your JAR file, follow the steps in the next section.

Deploy a ProfileManager That Can Use the New EntityPropertyManager

A "user type" is a mapping of a ProfileType name to a particular ProfileManager. This mapping is done in the UserManager EJB deployment descriptor.

To access the data in your new EntityPropertyManager EJB, you must do ***one*** of the following:

- Modifying the Existing ProfileManager Deployment Configuration − In most cases you will be able to use the default deployment of ProfileManager, the UserProfileManager. You will modify the UserProfileManager's deployment descriptor to map a property set and/or properties to your custom EntityPropertyManager. If you support the createUniqueId() and removeEntity() methods in your

custom EntityPropertyManager, you can use WebLogic Administration Portal to create a user of type "User" with a profile that can get/set properties using your custom EntityPropertyManager.
- Configuring and Deploying a New ProfileManager − In some cases you may want to deploy a newly configured ProfileManager that will be used instead of the UserProfileManager. This new ProfileManager is mapped to a ProfileType in the deployment descriptor for the UserManager. If you support the createUniqueId() and removeEntity() methods in your custom EntityPropertyManager, you can use the WebLogic Administration Portal (or API) to create a user of type "MyUser" (or anything else you name it) that can get/set properties using the customized deployment of the ProfileManager that is, in turn, configured to use your custom EntityPropertyManager.

ProfileManager is a stateless session bean that manages access to the profile values that the EntityPropertyManager EJB retrieves. It relies on a set of mapping statements in its deployment descriptor to find data. For example, the ProfileManager receives a request for the value of the "DateOfBirth" property, which is located in the "PersonalData" property set. ProfileManager uses the mapping statements in its deployment descriptor to determine which EntityPropertyManager EJB contains the data.

Modifying the Existing ProfileManager Deployment Configuration

If you use the existing UserProfileManager deployment to manage your user profiles, perform the following steps to modify the deployment configuration.

Under most circumstances, this is the method you should use to deploy your UUP. An example of this method is the deployment of the custom EntityPropertyManager for LDAP property retrieval, the LdapPropertyManager. The classes for the LdapPropertyManager are packaged in p13n_ejb.jar. The deployment descriptor for the UserProfileManager EJB is configured to map the "ldap" property set to the LdapPropertyManager. The UserProfileManager is deployed in p13n_ejb.jar.

1. Back up the p13n_ejb.jar file in your enterprise application root directory.
2. From p13n_ejb.jar, extract META−INF/ejb−jar.xml and open it for editing.
3. In ejb−jar.xml, find the <env−entry> element, as shown in the following example:
   ```
   <!-- map all properties in property set ldap to ldap server -->
   <env-entry>
       <env-entry-name>PropertyMapping/ldap</env-entry-name>
       <env-entry-type>java.lang.String</env-entry-type>
       <env-entry-value>LdapPropertyManager</env-entry-value>
   </env-entry>
   ```

   and add an <env−entry> element after this to map a property set to your custom EntityPropertyManager, a shown in the following example:
   ```
   <!-- map all properties in UUPExample property set to MyEntityPropertyManager -->
   <env-entry>
       <env-entry-name>PropertyMapping/UUPExample</env-entry-name>
       <env-entry-type>java.lang.String</env-entry-type>
       <env-entry-value>MyEntityPropertyManager</env-entry-value>
   </env-entry>
   ```
4. In ejb−jar.xml, find the <ejb−ref> element shown in the following example:
   ```
   <!-- an ldap property manager -->
   <ejb-ref>
       <ejb-ref-name>ejb/LdapPropertyManager</ejb-ref-name>
       <ejb-ref-type>Session</ejb-ref-type>
       <home>com.bea.p13n.property.EntityPropertyManagerHome</home>
       <remote>com.bea.p13n.property.EntityPropertyManager</remote>
   </ejb-ref>
   ```

and add an <ejb−ref> element after this to map a reference to an EJB that matches the name from the previous step with ejb/ prepended as shown in the following example:

```
<!-- an example property manager -->
<ejb-ref>
    <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>examples.usermgmt.MyEntityPropertyManagerHome</home>
    <remote>examples.usermgmt.MyEntityPropertyManager</remote>
</ejb-ref>
```

The home and remote class names match the classes from your EJB JAR file for your custom EntityPropertyManager.

5. If your EntityPropertyManager implementation handles creating and removing profile records, you must also add Creator and Remover entries. For example:

```
<env-entry>
    <env-entry-name>Creator/Creator1</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>

<env-entry>
    <env-entry-name>Remover/Remover1</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
```

   This instructs the UserProfileManager to call your custom EntityPropertyManager when creating or deleting user profile records. The names "Creator1" and "Remover1" are arbitrary. All Creators and Removers will be iterated through when the UserProfileManager creates or removes a user profile. The value for the Creator and Remover matches the ejb−ref−name for your custom EntityPropertyManager without the ejb/ prefix.
6. From p13n_ejb.jar, extract META−INF/weblogic−ejb−jar.xml and open it for editing.
7. In weblogic−ejb−jar.xml, find the elements shown in the following example:

```
<weblogic-enterprise-bean>
    <ejb-name>UserProfileManager</ejb-name>
    <reference-descriptor>
        <ejb-reference-description>
            <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
            <jndi-name>${APPNAME}.BEA_personalization. EntityPropertyManager</jndi-name>
        </ejb-reference-description>
```

   and add an ejb−reference−description to map the ejb−ref for your custom EntityPropertyManager to the JNDI name. This JNDI name must match the name you assigned in weblogic−ejb−jar.xml in the JAR file for your customer EntityPropertyManager. It should look like the following example:

```
<weblogic-enterprise-bean>
    <ejb-name>UserProfileManager</ejb-name>
    <reference-descriptor>
        <ejb-reference-description>
            <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
            <jndi-name>${APPNAME}.BEA_personalization. EntityPropertyManager</jndi-name>
        </ejb-reference-description>
        <ejb-reference-description>
            <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
            <jndi-name>${APPNAME}.BEA_personalization. MyEntityPropertyManager</jndi-nam
        </ejb-reference-description>
```

Note the ${APPNAME} string substitution variable. The WebLogic EJB container automatically substitutes the enterprise application name to scope the JNDI name to the application.

8. Update p13n_ejb.jar for your new deployment descriptors. You can use the jar uf command to update the modified META−INF/ deployment descriptors.
9. Edit your application's META−INF/application.xml to add an entry for your custom EntityPropertyManager EJB module as shown in the following example:
```
<module>
    <ejb>UUPExample.jar</ejb>
</module>
```
10. If you are using an application−wide cache, you can manage it from the WebLogic Administration Console if you add a <Cache> tag for your cache to the META−INF/application−config.xml deployment descriptor for your enterprise application like this:
```
<Cache Name="UUPExampleCache" TimeToLive="60000"/>
```
11. Verify the modified p13n_ejb.jar and your custom EntityPropertyManager EJB JAR archive are in the root directory of your enterprise application and start WebLogic Server.
12. Use the WebLogic Server Administration Console to verify your EJB module is deployed to the enterprise application and then use the console to add your server as a target for the EJB module. You need to select a target to have your domain's config.xml file updated to deploy your EJB module to the server.
13. Use the WebLogic Workshop Property Set Designer to create a User Profile (property set) that matches the name of the property set that you mapped to your custom EntityPropertyManager in ejb−jar.xml for the UserProfileManager (in p13n_ejb.jar). You could also map specific property names in a property set to your custom EntityPropertyManager, which would allow you to surface the properties and their values in the WebLogic Administration Portal for use in creating rules for personalization, delegated administration, and visitor entitlements.

Your new Unified User Profile type is ready to use. You can use the WebLogic Administration Portal to create a user, and it will use your UUP implementation when the "UUPExample" property set is being modified. When you call createUser("bob", "password") or createUser("bob", "password", null) on the UserManager, several things will happen:

- A user named "bob" is created in the security realm.
- A WebLogic Portal Server profile record is created for "bob" in the user store.
- If you set up the Creator mapping, the UserManager will call the default ProfileManager deployment (UserProfileManager) which will call your custom EntityPropertyManager to create a record for Bob in your data source.
- Retrieving Bob's profile will use the default ProfileManager deployment (UserProfileManager), and when you request a property belonging to the "UUPExample" property set, the request will be routed to your custom EntityPropertyManager implementation.

Configuring and Deploying a New ProfileManager

If you are going to deploy a newly configured ProfileManager instead of using the default ProfileManager (UserProfileManager) to manage your user profiles, perform the following steps to modify the deployment configuration. In most cases, you will not have to use this method of deployment. Use this method only if you need to support multiple types of users that require different ProfileManager deployments deployments that allow a property set to be mapped to different custom EntityPropertyManagers based on ProfileType.

An example of this method is the deployment of the custom CustomerProfileManager in customer.jar. The CustomerProfileManager is configured to use the custom EntityPropertyManager (CustomerPropertyManager) for properties in the "CustomerProperties" property set. The UserManager EJB in p13n_ejb.jar is configured to map the "WLCS_Customer" ProfileType to the custom deployment of the ProfileManager, CustomerProfileManager.

To configure and deploy a new ProfileManager, use this procedure.

1. Back up the p13n_ejb.jar file in your enterprise application root directory.
2. From p13n_ejb.jar, extract META−INF/ejb−jar.xml, and open it for editing.
3. In ejb−jar.xml, copy the entire <session> tag for the UserProfileManager, and configure it to use your custom implementation class for your new deployment of ProfileManager.
   In addition, you could extend the UserProfileManager home and remote interfaces with your own interfaces if you want to repackage them to correspond to your packaging (for example., examples.usermgmt.MyProfileManagerHome, examples.usermgmt.MyProfileManager).
   However, it is sufficient to replace the bean implementation class:
   You must create an <env−entry> element to map a property set to your custom EntityPropertyManager. You must also create a <ejb−ref> element to map a reference to an EJB that matches the name from the PropertyMapping with ejb/ prepended. The home and remote class names for your custom EntityPropertyManager match the classes from your EJB JAR file for your custom EntityPropertyManager.

   Also, if your EntityPropertyManager implementation handles creating and removing profile records, you must also add Creator and Remover entries. This instructs your new ProfileManager to call your custom EntityPropertyManager when creating or deleting user profile records.

   Note: The name suffixes for the Creator and Remover, "Creator1" and "Remover1", are arbitrary. All Creators and Removers will be iterated through when your ProfileManager creates or removes a user profile. The value for the Creator and Remover matches the <ejb−ref−name> for your custom EntityPropertyManager without the ejb/ prefix.
4. In ejb−jar.xml, you must add an <ejb−ref> to the UserManager EJB section to map your ProfileType to your new deployment of the ProfileManager, as shown in the following example:
   ```
   <ejb-ref>
       <ejb-ref-name>ejb/ProfileType/UUPExampleUser</ejb-ref-name>
       <ejb-ref-type>Session</ejb-ref-type>
       <home>com.bea.p13n.usermgmt.profile.ProfileManagerHome</home>
       <remote>com.bea.p13n.usermgmt.profile.ProfileManager</remote>
   </ejb-ref>
   ```

   The <ejb−ref−name> must start with ejb/ProfileType/ and must end with the name that you want to use as the profile type as an argument in the createUser() method of UserManager.
5. From p13n_ejb.jar, extract META−INF/weblogic−ejb−jar.xml and open it for editing.
6. In weblogic−ejb−jar.xml, copy the <weblogic−enterprise−bean> tag, shown in the following example, for the UserProfileManager and configure it for your new ProfileManager deployment:
   ```
   <weblogic-enterprise-bean>
       <ejb-name>MyProfileManager</ejb-name>
       <reference-descriptor>
           <ejb-reference-description>
               <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
               <jndi-name>${APPNAME}.BEA_personalization. EntityPropertyManager</jndi-name>
           </ejb-reference-description>
           <ejb-reference-description>
               <ejb-ref-name>ejb/PropertySetManager</ejb-ref-name>
               <jndi-name>${APPNAME}.BEA_personalization. PropertySetManager</jndi-name>
           </ejb-reference-description>
           <ejb-reference-description>
               <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
   ```

```
                <jndi-name>${APPNAME}.BEA_personalization. MyEnitityPropertyManager</jndi-na
            </ejb-reference-description>
        </reference-descriptor>
        <jndi-name>${APPNAME}.BEA_personalization. MyProfileManager</jndi-name>
    </weblogic-enterprise-bean>
```

You must create an <ejb−reference−description> to map the <ejb−ref> for your custom EntityPropertyManager to the JNDI name. This JNDI name must match the name you assigned in weblogic−ejb−jar.xml in the JAR file for your custom EntityPropertyManager.
Note the ${APPNAME} string substitution variable. The WebLogic Server EJB container automatically substitutes the enterprise application name to scope the JNDI name to the application.

7. In weblogic−ejb−jar.xml, copy the <transaction−isolation> tag for the UserProfileManager, shown in the following example, and configure it for your new ProfileManager deployment:
```
<transaction-isolation>
    <isolation-level>TRANSACTION_READ_COMMITTED</isolation-level>
    <method>
        <ejb-name>MyProfileManager</ejb-name>
        <method-name>*</method-name>
    </method>
</transaction-isolation>
```
8. Create a temporary p13n_ejb.jar for your new deployment descriptors and your new ProfileManager bean implementation class. This temporary EJB JAR archive should not have any container classes in it. Run ejbc to generate new container classes.
9. Edit your application's META−INF/application.xml to add an entry for your custom EntityPropertyManager EJB module, as shown in the following example:
```
<module>
    <ejb>UUPExample.jar</ejb>
</module>
```
10. If you are using an application−wide cache, you can manage it from the WebLogic Server Administration Console if you add a <Cache> tag for your cache to the META−INF/application−config.xml deployment descriptor for your enterprise application as shown in the following example:
```
<Cache Name="UUPExampleCache" TimeToLive="60000"/>
```

    Verify the modified p13n_ejb.jar and your custom EntityPropertyManager EJB JAR archive are in the root directory of your enterprise application and start your server.
11. Use the WebLogic Server Administration Console to verify your EJB module is deployed to the enterprise application and add your server as a target for the EJB module. You must select a target to have your domain's config.xml file updated to deploy your EJB module to the server.
12. Use the WebLogic Workshop Property Set Designer to create a User Profile (property set) that matches the name of the property set that you mapped to your custom EntityPropertyManager in ejb−jar.xml for the UserProfileManager (in p13n_ejb.jar). You could also map specific property names in a property set to your custom EntityPropertyManager, which would allow you to surface the properties and their values in the WebLogic Administration Portal for use in creating rules for personalization, delegated administration, and visitor entitlements.

Your new Unified User Profile type is ready to use. You can use the WebLogic Administration Portal to create a user, and it will use your UUP implementation when the "UUPExample" property set is being modified. That is because you mapped the ProfileType using an <ejb−ref> in your UserManager deployment descriptor, ejb/ProfileType/UUPExampleUser.

Now, when you call createUser("bob", "password", "UUPExampleUser") on the UserManager, several things will happen:

- A user named "bob" is created in the security realm.
- A WebLogic Portal Server profile record is created for "bob" in the WebLogic Portal RDBMS repository.
- If you set up the Creator mapping, the UserManager will call your new ProfileManager deployment, which will call your custom EntityPropertyManager to create a record for Bob in your data source.
- Retrieving Bob's profile will use your new ProfileManager deployment, and when you request a property belonging to the "UUPExample" property set, the request will be routed to your custom EntityPropertyManager implementation.

Retrieving User Profile Data from LDAP

WebLogic Portal provides a default unified user profile for retrieving properties from an LDAP server. Use this procedure to implement the LDAP unified user profile for retrieving properties from your LDAP server.

The LdapRealm security realm and the LdapPropertyManager unified user profile (UUP) for retrieving user properties from LDAP are independent of each other. They do not share configuration information and there is no requirement to use either one in conjunction with the other. A security realm has nothing to do with a user profile. A security realm provides user/password data, user/group associations, and group/group associations. A user profile provides user and group properties. A password is not a property.

In order to successfully retrieve the user profile from the LDAP server, ensure that you've done the following:

1. If you have already deployed the application on a WebLogic Portal instance, stop the server.
2. Extract p13n_ejb.jar from your application root to a temporary directory.
3. In the temporary directory, open META−INF/ejb−jar.xml, which contains a commented block called "Ldap Property Manager." Uncomment and reconfigure this section using the following steps:
    a. Remove the closing comment mark (−−>) from the end of the "Ldap Property Manager" block, just before the "Property Set Web Service EJB" block, and add it to the end of the first paragraph of the Ldap Property Manager block, like this:

    ```
    <!−− Ldap Property Manager
         To use this, uncomment it here as well as in weblogic−ejb−jar.xml.
         Configure the LDAP connection and settings using the env−entry values
         Do not forget to uncomment the ejb−link and method−permission tags for
         An easy way to ensure you don't miss anything is to search for "ldap"
         weblogic−ejb−jar.xml. Search from the beginning to the end of the file
    −−>
    ```
    b. In the "Ldap Property Manager" block, look for the following default settings and replace them with your own:

ldap://server.company.com:389	Change this to the value of your LDAP server URL.
uid=admin, ou=Administrators, ou=TopologyManagement, o=NetscapeRoot	Change this to the value of your LDAP server's principal.
<env−entry−value>weblogic</env−entry−value>	Change "weblogic" to your LDAP server's principalCredential.

ou=People,o=company.com	Change this to your LDAP server's UserDN.
ou=Groups,o=company.com	Change this to your LDAP server's GroupDN.
<env−entry−value>uid</env−entry−value>	Change "uid" to your LDAP server's usernameAttribute setting.
<env−entry−value>cn</env−entry−value>	Change "cn" to your LDAP server's groupnameAttribute setting.

c. In the "User Profile Manager" and "Group Profile Manager" sections, find the following lines:

```
<!−− <ejb−link>LdapPropertyManager</ejb−link> −−>
<ejb−link>EntityPropertyManager</ejb−link>
```

Uncomment the LdapPropertyManager line and delete the EntityPropertyManager line in both sections.

d. In the <method−permission> and <container−transaction> sections, find and uncomment the following:

```
<!−−
<method>
    <ejb−name>LdapPropertyManager</ejb−name>
    <method−name>*</method−name>
</method>
−−>
```

e. Check to see that you have uncommented all Ldap configurations by doing a search for "Ldap" in the file.

f. Save and close the file.

4. In the temporary directory, open META−INF/weblogic−ejb−jar.xml and perform the following modifications:

a. Uncomment the "LdapPropertyManager" block:

```
LdapPropertyManager
<weblogic−enterprise−bean>
    <ejb−name>LdapPropertyManager</ejb−name>
    <enable−call−by−reference>True</enable−call−by−reference>
    <jndi−name>${APPNAME}.BEA_personalization.LdapPropertyManager</jndi−nam
</weblogic−enterprise−bean>
```

b. In the "Security configuration" section of the file, uncomment the LdapPropertyManager method:

```
<method>
    <ejb−name>LdapPropertyManager</ejb−name>
    <method−name>*</method−name>
</method>
```

   c. Check to see that you have uncommented all Ldap configurations by doing a search for "Ldap" in the file.
   d. Save and close the file.
5. Replace the original p13n_ejb.jar with the modified version.

   a. Rename the original p13n_ejb.jar to use it as a backup. For example, rename it to p13n_ejb.jar.backup.
   b. JAR the temporary version of p13n_ejb.jar to which you made changes. Name it p13n_ejb.jar.
   c. Copy the new JAR to your application's root directory.
6. Start the server and re−deploy the application.
7. The properties from your LDAP server are now accessible through the WebLogic Portal API, JSP tags, and controls.

   If you want to surface the properties from your LDAP server in the WebLogic Administration Portal (for use in defining rules for personalization, delegated administration, and visitor entitlements), create a user profile property set called ldap.usr, and create properties in the property set that exactly match the names of the LDAP properties you want to surface.

Enabling SUBTREE_SCOPE Searches for Users and Groups

The LdapPropertyManager EJB in p13n_ejb.jar allows for the inspection of the LDAP schema to determine multi−valued versus single−value LDAP attributes, to allow for multiple userDN/groupDN, and to allow for SUBTREE_SCOPE searches for users and groups in the LDAP server. Following are more detailed explanations:

The determination of multi−value versus single−value LDAP attributes allows a developer to configure the ejb−jar.xml deployment descriptor for the LdapPropertyManager EJB to specify that the LDAP schema be used to determine if a property is single− or multi−value.

To enable SUBTREE−SCOPE for users and groups:

1. Stop the server.
2. Extract p13n_ejb.jar from your application root directory to a temporary directory and edit the temporary META−INF/ejb−jar.xml by setting the following env−entries.

```
<!−− Flag to specify if LDAP attributes will be determined to be single value
or multi−value via the schema obtained from the attribute. If false,
then the attribute is stored as multi−valued (a Collection) only if it has
more than one value. Leave false unless you intend to use multi−valued LDAP
attributes that may have only one value. Using true adds overhead to check
the LDAP schema. Also, if you use true beware that most LDAP attributes are
multi−value. For example, iPlanet Directory Server 5.x uses multi−value for
givenName, which you may not expect unless you are familiar with LDAP schemas.
This flag will apply to property searches for all userDNs and all groupDNs. −−>

<env−entry>
    <env−entry−name>config/detectSingleValueFromSchema</env−entry−name>
    <env−entry−type>java.lang.Boolean</env−entry−type>
    <env−entry−value>true</env−entry−value>
</env−entry>

<!−− Value of the name of the attribute in the LDAP schema that is used
to determine single value or multi−value (RFC2252 uses SINGLE−VALUE).
This attribute in the schema should be true for single value and false
```

```
or absent from the schema otherwise. The value only matters if
config/detectSingleValueFromSchema is true. -->

<env-entry>
    <env-entry-name>config/singleValueSchemaAttribute</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>SINGLE-VALUE</env-entry-value>
</env-entry>
```

It is not recommended that true be used for config/detectSingleValueFromSchema unless you are going to write rules that use multi−valued LDAP attributes that have a single value. Using config/detectSingleValueFromSchema = true adds the overhead of checking the LDAP schema for each attribute instead of the default behavior (config/detectSingleValueFromSchema = false), which only stores an attribute as multi−valued (in a Collection) if it has more than one value.

This feature also implements changes that allow you to use SUBTREE_SCOPE searches for users and groups. It also allows multiple base userDN and groupDN to be specified. The multiple base DN can be used with SUBTREE_SCOPE searches enabled or disabled.

A SUBTREE_SCOPE search begins at a base userDN (or groupDN) and works down the branches of that base DN until the first user (or group) is found that matches the username (or group name).

To enable SUBTREE_SCOPE searches you must set the Boolean config/objectPropertySubtreeScope env−entry in the ejb−jar.xml for p13n_ejb.jar.jar to true and then you must set the config/userDN and config/groupDN env−entry values to be equal to the base DNs from which you want your SUBTREE_SCOPE searches to begin.

For example, if you have users in ou=PeopleA,ou=People,dc=mycompany,dc=com and in ou=PeopleB,ou=People,dc=mycompany,dc=com then you could set config/userDN to ou=People,dc=mycompany,dc=com and properties for these users would be retrieved from your LDAP server because the user search would start at the "People" ou and work its way down the branches (ou="PeopleA" and ou="PeopleB").

You should not create duplicate users in branches below your base userDN (or duplicate groups below your base groupDN) in your LDAP server. For example, your LDAP server will allow you to create a user with the uid="userA" under both your PeopleA and your PeopleB branches. The LdapPropertyManager in p13n_ejb.jar.jar will return property values for the first userA that it finds.

It is recommended that you do not enable this change (by setting config/objectPropertySubtreeScope to true) unless you need the flexibility offered by SUBTREE_SCOPE searches.

An alternative to SUBTREE_SCOPE searches (with or without multiple base DNs) would be to configure multiple base DNs and leave config/objectPropertySubtreeScope set to false. Each base DN would have to be the DN that contains the users (or groups) because searches would not go any lower than the base DN branches. The search would cycle from one base DN to the next until the first matching user (or group) is found.
The new ejb−jar.xml deployment descriptor is fully commented to explain how to set multiple DNs, multiple usernameAttributes (or groupnameAttributes), and how to set the objectPropertySubtreeScope flag.

3. Save and close the file.
4. Replace the original p13n_ejb.jar with the modified version:

   a. Rename the original p13n_ejb.jar to use it as a backup. For example, rename it to p13n_ejb.jar.backup.
   b. JAR the temporary version of p13n_ejb.jar to which you made changes. Name it p13n_ejb.jar.
   c. Copy the new JAR to your application's root directory.
5. Start the server and re−deploy the application.

Related Topics

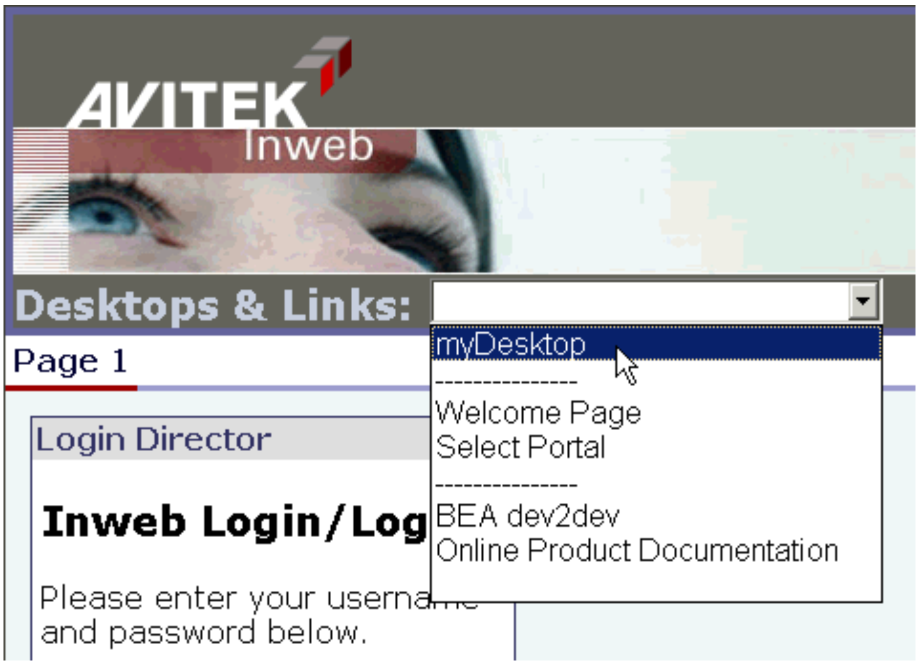
Using Multiple Authentication Providers in Portal Development

Enabling Desktop Selection

Oftentimes users are entitled to view multiple desktops in your portals. This topic shows you how to let users select from a list of the specific desktops to which they are entitled.

The desktop selection feature is a JSP used by the shell that provides a drop−down list of desktops and links to other resources. Because the desktop selector lets users switch between multiple desktops, it must run in streaming mode where multiple desktops exist. When viewing the feature in single file mode (development), only one desktop is ever available at a time.

The following figure shows the desktop selector in action.



To add the desktop selector to your desktops

The following procedure for adding Visitor Tools assumes you are adding them to a custom portal application (not the portalApp sample). If your application an/or project is not portal−enabled, install portal in both. See Creating a Portal Application and Portal Web Project.

1. Set up some form of authentication for your portal desktop. Authentication allows visitor entitlements to take effect. See Login Portlet, Login Director, or Implementing Authentication for information on adding authentication to your desktops.
2. In WebLogic Workshop create a new portal file.
3. Import the following files from Sample Portal into your application:

Import or copy this	***to this directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/header/header.jsp`	<PORTAL_APP>/<project>/portlets/header/
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/images/`	<PORTAL_APP>/<project>/images/

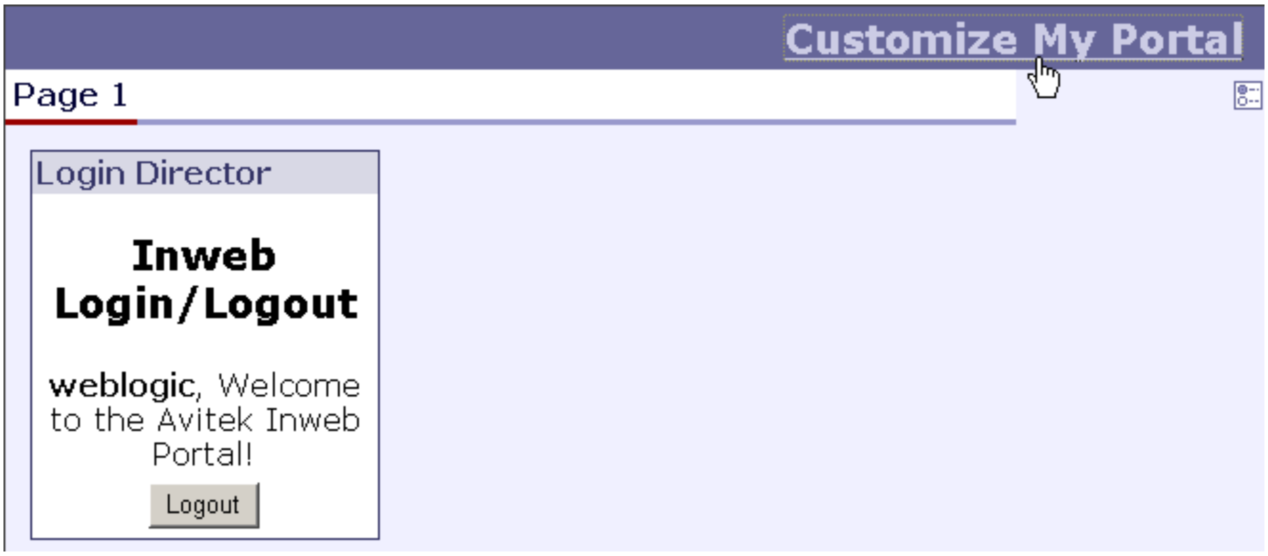
4. Open <PORTAL_APP>/<project>/portlets/header/header.jsp in WebLogic Workshop and replace the string sampleportal with the name of your project.
5. Create a shell and make <PORTAL_APP>/<project>/portlets/header/header.jsp the header content.
6. In a .portal file open in the Portal Designer, select the new shell for the desktop.
7. Save the portal file.

When portal administrators create desktops in the WebLogic Administration Portal and select that shell for the desktop, the desktop selector appears in the rendered desktops.

Adding Visitor Tools to Portals

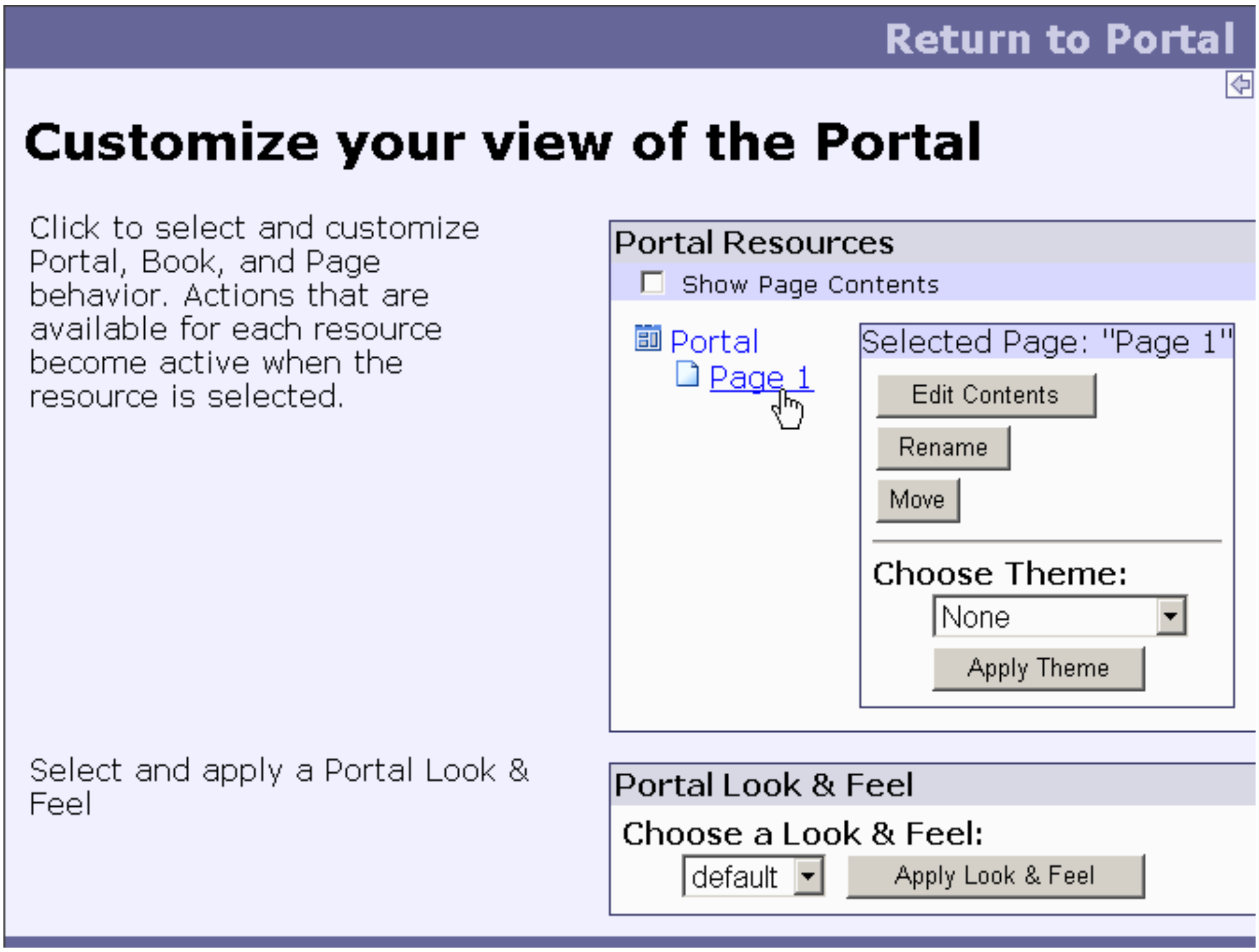
You can add functionality to your portal desktops that lets visitors modify their desktops, books, and pages. In order to use these Visitor Tools, visitors must be logged in to a desktop that is running in streaming mode.

Visitors access the visitor tools by clicking a text link or an icon in the desktop menu bar, as shown in the following illustration.



In this example, visitors can click on either the ***Customize My Portal*** link or the icon below it to access the Visitor Tools. The Customize My Portal link is supplied by the JSP used in the shell (described later in this topic), and the Edit icon is inserted by the menu skeleton JSP. Notice that the visitor must be logged in to access the Visitor Tools.

The following figure shows the Visitor Tools.


Return to Portal

Customize your view of the Portal

Click to select and customize Portal, Book, and Page behavior. Actions that are available for each resource become active when the resource is selected.

Select and apply a Portal Look & Feel

To add the Visitor Tools to Your Portals

The following procedure for adding Visitor Tools assumes you are adding them to a custom portal application (not the portalApp sample). If your application an/or project is not portal−enabled, install portal in both. See Creating a Portal Application and Portal Web Project.

1. Set up some form of authentication for your portal desktop. See Login Portlet, Login Director, or Implementing Authentication for information on adding authentication to your desktops.
2. In WebLogic Workshop create a new portal file.
3. In the Portal Designer, select the Main Page Book.
4. In the Property Editor window, set either of the following combinations of property values:

   Navigation: Single Level Menu or Multi Level Menu
   Editable: Edit in Menu

   or

   Navigation: No Navigation
   Editable: Edit in Titlebar
5. In the Mode Properties that appear, click the ellipsis icon [...] in the ***Content URI*** field, select <project>/visitorTools/visitorTools.portion, and click ***Open***.
6. Set the ***Visible*** property to false.

7. In the Portal Designer, select the Desktop.
8. In the Property Editor window, set the ***Shell*** to "Visitor Tools Shell."

   Now you must create a streaming desktop using the .portal file as a template to use the Visitor Tools.
9. If the server is not running, start it. Choose ***Tools*** −−> ***WebLogic Server*** −−> ***Start WebLogic Server***.
10. When the server is running, choose ***Portal*** −−> ***Portal Administration*** to start the WebLogic Administration Portal.
11. Log in to the WebLogic Administration Portal (the default username and password is ***weblogic***/***weblogic***).
12. Create a new desktop using your .portal file as a template. See Create a New Portal and Create a Desktop in the WebLogic Administration Portal online help posted on e−docs.
13. Select the new desktop in the Portal Resources tree, and go to the Desktop Properties page. At the bottom of the page, click ***View Desktop***.
14. When the desktop appears, log in and access the Visitor Tools.

You will notice that you can access the Visitor Tools by clicking the Customize My Portal link or the Edit icon. You do not have to use both ways to access the Visitor Tools.

- To use the Customize My Portal link only, use the Visitor Tools Shell and set the Main Page Book's ***Editable*** property to "Not Editable."
- To use the Edit icon, leave the ***Editable*** and ***Content URI*** property values in place and choose a shell other than Visitor Tools Shell.

The main page book in your .portal file can be used as the main page book when creating a desktop in the WebLogic Administration Portal to enable Visitor Tools. This will provide the desktop with Visitor Tools.

Note: You can also use the default New Blank Desktop template in the WebLogic Administration Portal to create a desktop that has Visitor Tools enabled.

Related Topics

Creating Shells

Creating URLs to Portal Resources

WebLogic Portal provides a convenient, extensible mechanism for creating URLs to your portal resources in a portal Web project that can transfer from domain to domain without breaking, especially when server names and port numbers change. This URL−creation mechanism also lets you switch between secure and non−secure URLs (http and https).

The two pieces involved in creating portable URLs are:

- The <render:*Url> JSP tags in the Portal Rendering JSP tag library.
- A portal Web project's WEB−INF/url−template−config.xml file.

The url−template−config.xml file contains multiple URL "templates," each with a unique name. Those template URLs contain variables such as url:domain and url:port that are read in from the active server. The <render:*Url> JSP tags have a "template" attribute in which you can specify the name of a URL template in url−template−config.xml.

The following examples show how the JSP tags use the templates to create URLs.

url−template−config.xml	***<render:resourceUrl>***

The following is a sample URL template in url−template−config.xml.
 	
`<url-template name="secure-url">`
`    https://{url:domain}:{url:securePort}/{url:path}?{url:queryString}`
`</url-template>`
	The following is how the <ren
	template.
	
	`<% String reportpath = "`
	
	`<a href="<render:resourc`
	`View the Report`
	`</a>`

You can use any of the URL templates in url−template−config.xml provided by WebLogic Portal, and you can add as many templates as you want to the file.

The following variables are available for use in URL template building:

{url:domain} − Reads the name of the server from the current request.

{url:port} − Reads the listen port number of the server from the current request. (See Troubleshooting below.)

{url:securePort} − Reads the SSL port number of the server from the current request. (See Troubleshooting below.)

{url:path} − Reads the name of the Web application. The URLs to all resources in a Web application are relative to the Web application directory.

{url:queryString} − Reads a queryString variable for the URL.

Troubleshooting

If you are using a proxy server or switching back and forth between non−secure and secure ports, you may find that URLs do not resolve if you use the {url:port} or {url:securePort} variables. This is because the variables for those values are read from the request. For example, if a user in a non−secure URL (port number 80) clicks a secure https link that was created with a URL template that uses the {url:securePort} variable, the port number of the request (80) is used for the {url:securePort} variable, which would create a secure request (https) on an non−secure port. The same could happen if a user on a proxy server (port 80) clicks a link to a resource outside the proxy server (port 443).

In both of those cases, you need to hard code port numbers in the URL templates to get URLs to resolve correctly.

Web Services for Remote Portlets (WSRP)

The url−template−config.xml file automatically created in a portal Web project also contains URL templates and variables for WSRP portlets. These templates must remain in the file if you are going to be a WSRP producer.

Related Topics

Portal Rendering JSP Tags

Building Portlets

Portlets are the basic building blocks of portal Web applications, and enable the presentation behavior of a subset of an application to be managed as a single unit. A portlet exists as a set of associated files, mostly XML and JPSs. In the WebLogic Workshop IDE, portlets can be edited visually in Design View, and the JSPs can be edited in Design View and Source View.

You can think of portlets as the windows that surface your applications, information, and business processes. Portlets can communicate with each other, they can work with Java controls, and take part in Java Page Flows that determine a user's path through an application. You can have multiple portlets on a page. You can also have multiple instances of a single portlet.

Architecturally, a portlet is a collection of objects described by an XML file with the .portlet extension. The framework uses the elements described in that file to render the Portlet at runtime, applying any permissions and customization at specific points in the assembly of the HTML that is eventually generated. The Portlet itself can use a JSP, a Page Flow, or an optional backing file, and can be built to conform to the JSR 168 standard for portlet compatibility. Portlets can consume existing Web applications and content (ASP, JSP, HTML, XML, and so on).

The following topics guide you through the portlet creation process:

Using Portlets from the Portlet Library

To save time and get a head start on your development work, you can use the sample portlets in the Portlet Library.

Creating Portlets

If the portlets in the Portlet Library do not match your current needs, you can develop new portlets in a variety of ways, including through the use of the Portlet Wizard.

Customizing Portlets

After creating your portlets, you can customize them to suit the requirements of your portal application.

Related Topics

Developing Portal Applications

Developing Personalized Applications

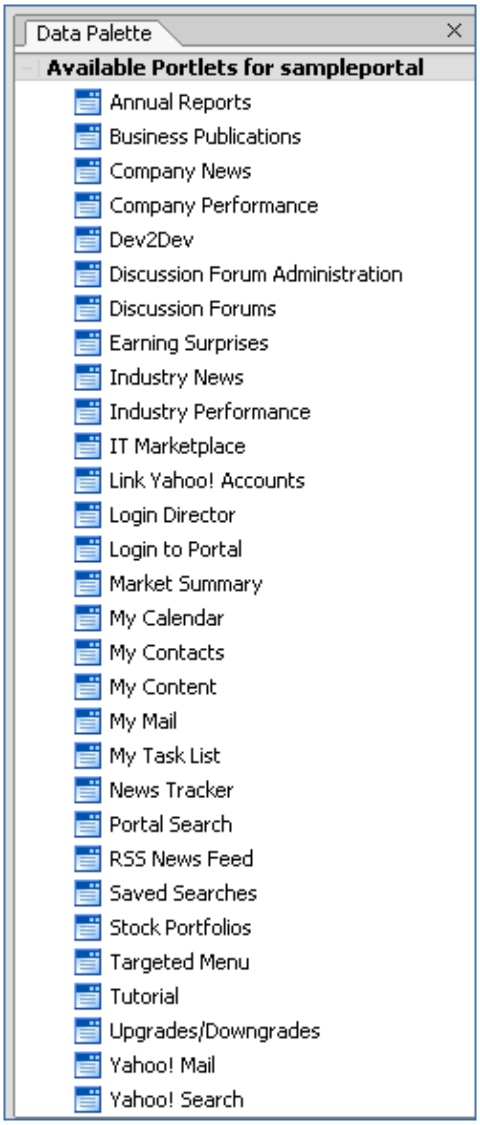
Portal Reference

Portal Samples

Portal Tutorials

Using Portlets from the Portlet Library

The Portlet Library contains a variety of pre−built portlets that you can bring into your portal application and modify. These portlets are examples of commonly used webapp functions, and should help you get a quick start on building the components for your portal.



Available Portlets for sampleportal

- Annual Reports
- Business Publications
- Company News
- Company Performance
- Dev2Dev
- Discussion Forum Administration
- Discussion Forums
- Earning Surprises
- Industry News
- Industry Performance
- IT Marketplace
- Link Yahoo! Accounts
- Login Director
- Login to Portal
- Market Summary
- My Calendar
- My Contacts
- My Content
- My Mail
- My Task List
- News Tracker
- Portal Search
- RSS News Feed
- Saved Searches
- Stock Portfolios
- Targeted Menu
- Tutorial
- Upgrades/Downgrades
- Yahoo! Mail
- Yahoo! Search

To examine the available portlets, select ***File > Open > Application*** from the top−level menu in the WebLogic Workshop IDE. Browse to the following folder, and select the portalApp.work file:

<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp

When the application is open in the IDE, navigate to the /sampleportal folder, and open the sample.portal file. In the Data Palette pane shown here, the IDE displays the portlets used in sampleportal, which comprise the Portlet Library.

In the IDE, you can drag and drop a portlet from the Portlet Library to a region, or placeholder, on your open portal page.

A number of these portlets are described in the Sample Portal topic. A table identifies the files that comprise the portlet, and the corresponding location in your portal application where you should install or copy the files. Typically this process involves the following:

- Copying the *.portlet file to your <PORTAL_APP>/<project>/portlets/includes folder.
- Copying one or more *.jsp file(s) to your <PORTAL_APP>/<project>/portlets folder.
- In some cases, copying a *.java backing file to your <PORTAL_APP>/<project>/ WEB−INF/src/<path>/... folder.
- In some cases, copying a JAR used by the portlet to your <PORTAL_APP>/<project>/WEB−INF/lib folder.

If you are working in another enterprise application on your system separate from the sample portalApp that contains the Portlet Library you can use the IDE ***File > Import Files...*** feature to copy a portlet's files into your web project.
For example in a web project of another enterprise application, you could add a /portlets folder, right−click on that folder name and select Import... from the menu. Then browse to the following directory:

<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets

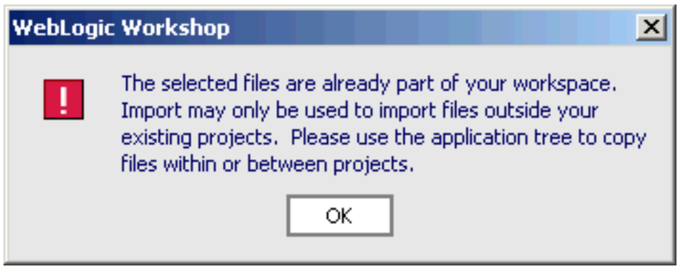
In the display of portlet folders, select the folder that contains the portlet you want to import into your application. Note, however, that the Import operation simply copies the files in the folder that you selected. You may need to also import or copy related files, such as a source *.java backing file used by the portlet.

Here is an example of the files needed for the Login Director portlet:

Import or copy this	***to this directory*** (create if
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/login/director.portlet`	<PORTAL_APP>/<projec
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/login/director.jsp`	<PORTAL_APP>/<projec
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB-INF/src/examples/login/DirectorBacking.java`	<PORTAL_APP>/<projec WEB−INF/src/examples/l
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB-INF/src/examples/login/DirectorUtil.java`	<PORTAL_APP>/<projec WEB−INF/src/examples/l

This type of information is presented for many of the portlets in the samples Help topics. Start in Sample Portal.

You cannot use the Import feature to copy in a portlet that already exists in the current enterprise application; WebLogic Workshop will display this message:



The selected files are already part of your workspace. Import may only be used to import files outside your existing projects. Please use the application tree to copy files within or between projects.

The easiest method is to use the file system to simply copy the files that comprise the portlet to your web project's folders. Again, in many cases there are more files than the *.portlet and *.jsp file(s) that comprise the portlet. For details, see the each topic listed in Sample Portal.

Related Topics

Sample Portal

Creating Portlets

Customizing Portlets

Login to Portal Portlet

The Login to Portal portlet illustrates login functionality for a portal desktop.

Concepts Demonstrated by this Sample

The JSP portlet uses a backing file to authenticate users.

Location of Sample Files

This sample is located in the
`<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work`
application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

1. Create a portal application. Make sure you create a Portal Application and add a Portal Web Project to it.
2. Import or copy the following directories and files into your portal application and portal Web project. You may need to create the appropriate directories in your application:

Import or copy this	***to this directory*** (creat
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/includes/Login.portlet`	<PORTAL_APP>/<pr
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/login.jsp`	<PORTAL_APP>/<pr
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/src/examples/login/LoginBacking.java`	<PORTAL_APP>/<pr WEB−INF/src/exampl
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/lib/yahoo_servlet.jar`	<PORTAL_APP>/<pr WEB−INF/lib/

3. Open your portal file and navigate the page where you want the portlet to appear.
4. In the ***Data Palette*** window, drag the ***Login to Portal*** portlet onto a placeholder on the page.
5. In the ***Property Editor*** window, set any relevant properties.
6. Save the portal file.
7. View your portal with the WebLogic Test Browser or with your default browser.
    - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
    - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

Related Topics

Creating a Portal File

Login Director Portlet

This login portlet directs the user to a "default" desktop the first desktop to which the user is entitled and returns the user to that desktop when the user logs out.

Concepts Demonstrated by this Sample

The JSP portlet uses a backing file to authenticate users and direct them to the default desktop.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

While you can run the portlet in the development environment, the full functionality of this portlet is visible only in streaming mode where multiple desktops and visitor entitlement rules exist. For more information, see Single File vs. Streamed Rendering.

To run the portlet in the development environment, see Viewing the Samples in Portal Samples.

To run the portlet in streaming mode, add the portlet to a page in sample.portal, then use the WebLogic Administration Portal to create a new desktop that uses sample.portal as a template. On the Desktop Properties page for the new desktop, click ***View Desktop*** to view the portlet.

How to Use the Sample in Your Portals

1. Create a portal application. Make sure you create a Portal Application and add a Portal Web Project to it.
2. Import or copy the following directories and files into your portal application and portal Web project. You may need to create the appropriate directories in your application:

Import or copy this	***to this directory*** (c
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/login/director.portlet`	<PORTAL_APP>
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/login/director.jsp`	<PORTAL_APP>
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB-INF/src/examples/login/DirectorBacking.java`	<PORTAL_APP> WEB−INF/src/exa
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB-INF/src/examples/login/DirectorUtil.java`	<PORTAL_APP> WEB−INF/src/exa
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB-INF/lib/yahoo_servlet.jar`	<PORTAL_APP> WEB−INF/lib/

3. Open your portal file and navigate the page where you want the portlet to appear.
4. In the ***Data Palette*** window, drag the ***Login Director*** portlet onto a placeholder on the page.
5. In the ***Property Editor*** window, set any relevant properties.

6. Save the portal file.
7. To view the portlet in a desktop, create a desktop in the WebLogic Administration Portal using the current .portal file as a template, and in the Desktop Properties page for the desktop, click ***View Desktop***.

Related Topics

Portal Samples

Login Portlet

Targeted Menu Portlet

This portlet demonstrates using navigation from a portlet to control a specific book.

Concepts Demonstrated by this Sample

The JSP portlet provides portlet−based navigation by acquiring a book's context, sub−books, and pages and constructing a tree−like list of links to the sub−books and pages.

This portlet uses functionality similar to the Left Navigation Shell. The difference between the two samples is scope. This portlet provides navigation for a single book, and the left navigation shell provides navigation for the entire desktop.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

To run the sample:

1. Add the portlet to a page in sample.portal.
2. In the Portal Designer, select the book for which you want to provide navigation. In the Property Editor window, note the value for the book's Definition Label property.
3. Open the portlet in the Portlet Designer, and select the ***TargetBook*** portlet preference.
4. In the Property Editor window, set the ***Preference Value*** property value to the book's Definition Label.
5. Save the portlet and portal files.
6. In the WebLogic Workshop menu, choose ***Portal*** `-->` ***Open Current Portal***.

How to Use the Sample in Your Portals

1. Create a portal application. Make sure you create a Portal Application and add a Portal Web Project to it.
2. Import or copy the following directories and files into your portal application and portal Web project. You may need to create the appropriate directories in your application:

Import or copy this	*to this direc*
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/navigation/targeted/targetedMenu.portlet`	<PORTAL_
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/navigation/targeted/targetedMenu.jsp`	<PORTAL_
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/src/examples/navigation/NavigationNode.java`	<PORTAL_ WEB−INF/

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/src/examples/navigation/NavigationUtil.java`	<PORTAL_ WEB−INF/

3. Open your portal file and navigate the page where you want the portlet to appear.
4. In the ***Data Palette*** window, drag the ***Targeted Menu*** portlet onto a placeholder on the page.
5. In the Portal Designer, select the book for which you want to provide navigation. In the Property Editor window, note the value for the book's Definition Label property.
6. Open the portlet in the Portlet Designer, and select the ***TargetBook*** portlet preference.
7. In the Property Editor window, set the ***Preference Value*** property value to the book's Definition Label.
8. Save the portlet and portal files.
9. In the WebLogic Workshop menu, choose ***Portal*** **-->** ***Open Current Portal***.

Related Topics

Left Navigation Shell

Portal Samples

dev2dev Portlet

The dev2dev portlet illustrates a portlet that uses static HTML links for accessing valuable BEA resources and information.

Concepts Demonstrated by this Sample

This HTML portlet illustrates that portlets can be created with only HTML tags. No knowledge of JSP development is necessary. Simply give an HTML file a .jsp extension and create a portlet with it.

The file does not need <HTML>, <HEAD>, <TITLE>, or <BODY> tags. Simply enclose your HTML content in opening and closing <div></div> tags.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

1. Create a portal application.
2. Import or copy the following directories and files into your portal application and portal Web project. You may need to create the appropriate directories in your application:

Import or copy this	***to this directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/includes/dev2dev.portlet`	<PORTAL_APP>/<project>/portlets/i
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/dev2dev/`	<PORTAL_APP>/<project>/portlets/

3. Open your portal file and navigate the page where you want the portlet to appear.
4. In the ***Data Palette*** window, drag the ***Dev2Dev*** portlet onto a placeholder on the page.
5. In the ***Property Editor*** window, set any relevant properties.
6. Save the portal file.
7. View your portal with the WebLogic Test Browser or with your default browser.

   - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
   - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

Related Topics

Creating a Portal File

RSS News Feed Portlet

The RSS News Feed Portlet retrieves news content and links based on visitor news feed preferences. This portlet provides an edit mode that lets you change the news feed and set portlet preferences.

Concepts Demonstrated by this Sample

This JSP portlet, which uses a backing file, illustrates a news aggregator portlet that connects to external news feeds and weblogs that provide Really Simple Syndication (RSS) content. This portlet provides an edit mode.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

1. Create a portal application.
2. Import or copy the following directories and files into your portal application and portal Web project. You may need to create the appropriate directories in your application:

Import or copy this	***to this directory*** (create if ne
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/rss/`	<PORTAL_APP>/<project>
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB-INF/src/examples/rss/`	<PORTAL_APP>/<project> WEB−INF/src/examples/

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/`lib/xmlx−tags.jar
 	
Add the following entry in the tag library section of <project>/WEB−INF/web.xml to register the tag library:
 	
`<taglib>`
`    <taglib-uri>xmlx.tld</taglib-uri>`
`    <taglib-location>/WEB-INF/lib/xmlx-tags.jar</taglib-location>`
`</taglib>`
	<PORTAL_APP>/<project> WEB−INF/lib/

3. Open your portal file and navigate the page where you want the portlet to appear.
4. In the ***Data Palette*** window, drag the ***RSS News Feed*** portlet onto the portal page.
5. In the ***Property Editor*** window, set any relevant properties.
6. Save the portal file.
7. View your portal with the WebLogic Test Browser or with your default browser.
    - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).

- ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

To Change the News Feed

1. In the ***Portal Designer***, double−click the portlet to open it.
2. With the portlet file open, click the arrow icon on the Portlet Preferences bar on the portlet footer to expand the preferences.
3. Select the ***contentURL*** preference.
4. In the ***Property Editor*** window, enter an absolute HTTP path to a valid news feed (.rdf file) in the ***Preference Value*** field. For example: http://www.theserverside.com/rss/theserverside−1.0.rdf.

You can also let users change news feeds for a portlet with the portlet's edit mode.

The portlet's ***Edit*** mode uses the rss.properties file to let you select from a list of valid RSS feeds. Modify rss.properties to remove or add feeds. The ***rss*** portlet shows one feed at a time. To show multiple feeds, add more ***rss*** portlets and set a different ***contentURL*** for each. Set the ***Edit URI*** field to /portlets/rss/rssedit.jsp to allow users to change the portlet feed using the rss.properties list. Save the portal file.
Related Topics

Creating a Portal File

Portal Samples

<pref:getPreference> Tag

<pref:ifModifiable> Tag

<pref:else> Tag

Portal Search Portlet

The Portal Search portlet lets you perform searches in your enterprise databases. This portlet also provides an edit mode to let you set search preferences, including selecting available databases to search.

Concepts Demonstrated by this Sample

This is a JSP portlet that provides edit mode.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

1. Create a portal application.
2. Import or copy the following directories and files into your portal application and portal Web project. You may need to create the appropriate directories in your application:

Import or copy this	***to this directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/includes/search.portlet`	<PORTAL_APP>/<project>/portlets/inc
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/search/`	<PORTAL_APP>/<project>/portlets/
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/lib/autonomyClient1.5.0.jar and autonomySupport.jar`	<PORTAL_APP>/<project>/WEB−INF

3. Open your portal file and navigate the page where you want the portlet to appear.
4. In the ***Data Palette*** window, drag the ***Portal Search*** portlet onto the portal page.
5. In the ***Property Editor*** window, set any relevant properties.
6. Save the portal file.
7. View your portal with the WebLogic Test Browser or with your default browser.
    - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
    - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

Related Topics

Creating a Portal File

My Mail Portlet

The My Mail portlet lets you configure an e−mail account that uses a POP3 or IMAP standard e−mail protocol. Once your e−mail account is configured, you can send, receive, store, and delete e−mail.

Concepts Demonstrated by this Sample

This Java Page Flow portlet illustrates group collaboration functionality within a portlet. The portlet provides edit mode.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

When this portlet is used in a domain (for example, in the portalApp in the Sample Portal Domain), the EJBs it uses are registered with JNDI names that can be used only once in the domain. That means you can use the following collaboration portlets in only one portal application in a domain: My Mail, My Task List, My Calendar, My Contacts, Discussion Forums, and Discussion Forum Administration. Within that portal application you can create multiple portal Web projects that can each contain multiple portals that reuse these portlets.

1. Create a portal application in a domain that has not used the collaboration portlets.
2. Make sure your portal application is open and the server is running (***Tools−−>WebLogic Server−−>Start WebLogic Server***).
3. Import/add the following directories and files into your portal application and portal Web project using WebLogic Workshop. (Right−click−−>***Import*** or ***Add Module*** or ***Add Library*** on the target directory). You may need to create the appropriate directories in your application.

   Be sure to add the harmony_portlets.jar library first, as shown in the following table.

Import this	***into this WebLogic Worksho*** ***directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/APP−INF/lib/harmony_portlets.jar`	<PORTAL_APP>/Libraries/

`<WEBLOGIC_HOME>/samples/portal/portalApp/`
`security_ejb.jar`
`uniqueid_ejb.jar`
	<PORTAL_APP>/Modules/

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/includes/collaboration/native_mail.portlet`	<PORTAL_APP>/<project>/ portlets/includes/collaboration

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/collaboration/nativedb/mail/`	<PORTAL_APP>/<project>/p `collaboration/native`

Note: If you add the non−JAR resources to directories other than those shown, you must open the portlet file in WebLogic Workshop and edit the Content URI for both the portlet's main content and Edit page content; you must modify the package path and <view−properties> paths in the Java Page Flow files; and you must modify the import statement to the ContentController and modify any other relevant paths in the JSPs.

4. Add the collaboration tables to your database. If you have already performed this for another collaboration portlet, skip this step.

   Note: If you ran the create_* database script to set up your database, the collaboration tables already exist. Do not run create_* if, for example, you are using the default PointBase database and have already added records to the database. Follow these instructions instead.

   To add the collaboration tables to an existing database, run the following database script:

   <BEA_HOME>/<WEBLOGIC_HOME>/portal/db/<DB_TYPE>/<DB_VERSION>/collaboration_create_tab

   for example

   bea/weblogic81/portal/db/pointbase/44/collaboration_create_tables.sql
5. To run this script for PointBase:
   a. Start the PointBase Console. In a command window, run

      <DOMAIN>/startPointBaseConsole.cmd(.sh)
   b. Log into the console. The default login is weblogic/weblogic.
   c. Choose ***File−−>Open***.
   d. Open the collaboration_create_tables.sql script. The script opens in the Enter SQL Commands window.
   e. Click the ***Execute All*** button. The collaboration tables are created.
   f. Close the PointBase Console.
6. Add entries to <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.

   a. Open <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   b. Open <BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/web.xml
   c. From the sampleportal web.xml file, copy the following sections into your project web.xml file in the appropriate locations:

```
<!−− Compoze Collaboration Mail Attachment Servlet −−>
<servlet>
                <servlet−name>CompozeNativeMailFileAttachmentServlet</servl
                <servlet−class>com.compoze.mail.FileAttachmentServlet</serv
</servlet>
.
.
.
<!−− Compoze Collaboration Mail Attachment Servlet Mapping −−>
<servlet−mapping>
                <servlet−name>CompozeNativeMailFileAttachmentServlet</servl
                <url−pattern>*.compozenativemailfileattachmentservlet</url−
</servlet−mapping>
```

```
.
.
.
<ejb-ref>
    <description>Unique ID Generator</description>
    <ejb-ref-name>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</ejb-ref-
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</home>
    <remote>com.compoze.ejb.uniqueid.IUniqueIDGenerator</remote>
    <ejb-link>UniqueIDGenerator</ejb-link>
</ejb-ref>
<ejb-ref>
    <description>Access Control Manager</description>
    <ejb-ref-name>com.compoze.security.acl.IAccessControllerHome</ejb-ref-n
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.security.acl.IAccessControllerHome</home>
    <remote>com.compoze.security.acl.IAccessController</remote>
    <ejb-link>AccessController</ejb-link>
</ejb-ref>
```

   d. Save and close your project web.xml file.

9. Perform the following steps only if you are using a database other than PointBase.
   a. Stop the server. Choose ***Tools−−>WebLogic Server−−>Stop WebLogic Server***.
   b. Modify your domain's setDomainEnv.cmd(.sh) to use the correct database.

      In the entry set HARMONY_PORTLETS_PROPERTIES=−Dejbruntime.database=pointbase44, use the commented area above this entry to replace the pointbase44 entry with the name of your database driver. Possible values are listed above that entry.
   c. Save setDomainEnv.cmd(.sh).
   d. Restart the server.

10. Add the Login to Portal Portlet to your portal Web project. Users must log in to use the collaboration portlets.
11. Open your portal file and navigate the page where you want the portlet to appear.
12. In the ***Data Palette*** window, drag the ***My Mail*** portlet onto the portal page.
13. In the ***Property Editor*** window, set any relevant properties.
14. Save the portal file.
15. View your portal with the WebLogic Test Browser or with your default browser.
    - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
    - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

For instructions on using the portlet's features, see *Compoze Portlets for BEA WebLogic Portal User's Guide* at http://e−docs.bea.com/wlp/docs81/pdf/compoze_portlets_users_guide.pdf.

Related Topics

Creating a Portal File

Portal Samples

Getting Started with Page Flows

My Task List Portlet

The My Task List portlet lets you create a To Do list, set dates, status, priorities, completion percentages, and other details on tasks. This portlet also provides an edit mode to let you customize your task views.

Concepts Demonstrated by this Sample

This Java Page Flow portlet illustrates group collaboration functionality within a portlet. The portlet provides edit mode.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

When this portlet is used in a domain (for example, in the portalApp in the Sample Portal Domain), the EJBs it uses are registered with JNDI names that can be used only once in the domain. That means you can use the following collaboration portlets in only one portal application in a domain: My Mail, My Task List, My Calendar, My Contacts, Discussion Forums, and Discussion Forum Administration. Within that portal application you can create multiple portal Web projects that can each contain multiple portals that reuse these portlets.

1. Create a portal application in a domain that has not used the collaboration portlets.
2. Make sure your portal application is open and the server is running (***Tools−−>WebLogic Server−−>Start WebLogic Server***).
3. Import/add the following directories and files into your portal application and portal Web project using WebLogic Workshop. (Right−click−−>***Import*** or ***Add Module*** or ***Add Library*** on the target directory). You may need to create the appropriate directories in your application.

   Be sure to add the harmony_portlets.jar library first, as shown in the following table.

Import this	***into this WebLogic Workshop directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/APP−INF/lib/harmony_portlets.jar`	<PORTAL_APP>/Libraries/
`<WEBLOGIC_HOME>/samples/portal/portalApp/` `todo_ejb.jar` `security_ejb.jar` `uniqueid_ejb.jar`	<PORTAL_APP>/Modules/
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/includes/collaboration/`	<PORTAL_APP>/<project>/portlets/includes/collaboration

native_task.portlet	
<WEBLOGIC_HOME>/samples/portal/portalApp/ sampleportal/portlets/collaboration/nativedb/ todo/	<PORTAL_APP>/<project>/p collaboration/native

Note: If you add the non−JAR resources to directories other than those shown, you must open the portlet file in WebLogic Workshop and edit the Content URI for both the portlet's main content and Edit page content; you must modify the package path and <view−properties> paths in the Java Page Flow files; and you must modify the import statement to the ContentController and modify any other relevant paths in the JSPs.

4. Add the collaboration tables to your database. If you have already performed this for another collaboration portlet, skip this step.

   Note: If you ran the create_* database script to set up your database, the collaboration tables already exist. Do not run create_* if, for example, you are using the default PointBase database and have already added records to the database. Follow these instructions instead.

   To add the collaboration tables to an existing database, run the following database script:

   <BEA_HOME>/<WEBLOGIC_HOME>/portal/db/<DB_TYPE>/<DB_VERSION>/collaboration_create_tabl

   for example

   bea/weblogic81/portal/db/pointbase/44/collaboration_create_tables.sql
5. To run this script for PointBase:
   a. Start the PointBase Console. In a command window, run

      <DOMAIN>/startPointBaseConsole.cmd(.sh)
   b. Log into the console. The default login is weblogic/weblogic.
   c. Choose ***File−−>Open***.
   d. Open the collaboration_create_tables.sql script. The script opens in the Enter SQL Commands window.
   e. Click the ***Execute All*** button. The collaboration tables are created.
   f. Close the PointBase Console.
6. Add entries to <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   a. Open <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   b. Open <BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/web.xml
   c. From the sampleportal web.xml file, copy the following sections into your project web.xml file in the appropriate locations:

```
<ejb−ref>
    <description>Unique ID Generator</description>
    <ejb−ref−name>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</ejb−ref−n
    <ejb−ref−type>Session</ejb−ref−type>
    <home>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</home>
    <remote>com.compoze.ejb.uniqueid.IUniqueIDGenerator</remote>
    <ejb−link>UniqueIDGenerator</ejb−link>
</ejb−ref>
<ejb−ref>
    <description>Access Control Manager</description>
    <ejb−ref−name>com.compoze.security.acl.IAccessControllerHome</ejb−ref−na
    <ejb−ref−type>Session</ejb−ref−type>
    <home>com.compoze.security.acl.IAccessControllerHome</home>
```

```
        <remote>com.compoze.security.acl.IAccessController</remote>
        <ejb-link>AccessController</ejb-link>
    </ejb-ref>
    <ejb-ref>
        <description>To Do Manager</description>
        <ejb-ref-name>com.compoze.todo.ejb.IToDoManagerHome</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.compoze.todo.ejb.IToDoManagerHome</home>
        <remote>com.compoze.todo.ejb.IToDoManager</remote>
        <ejb-link>ToDoManager</ejb-link>
    </ejb-ref>
```

   d. Save and close your project web.xml file.
7. Perform the following steps only if you are using a database other than PointBase.
   a. Stop the server. Choose ***Tools−−>WebLogic Server−−>Stop WebLogic Server***.
   b. Modify your domain's setDomainEnv.cmd(.sh) to use the correct database.

      In the entry set HARMONY_PORTLETS_PROPERTIES=−Dejbruntime.database=pointbase44, use the commented area above this entry to replace the pointbase44 entry with the name of your database driver. Possible values are listed above that entry.
   c. Save setDomainEnv.cmd(.sh).
   d. Restart the server.
8. Add the Login to Portal Portlet to your portal Web project. Users must log in to use the collaboration portlets.
9. Open your portal file and navigate the page where you want the portlet to appear.
10. In the ***Data Palette*** window, drag the ***My Task List*** portlet onto the portal page.
11. In the ***Property Editor*** window, set any relevant properties.
12. Save the portal file.
13. View your portal with the WebLogic Test Browser or with your default browser.
    - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
    - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

For instructions on using the portlet's features, see *Compoze Portlets for BEA WebLogic Portal User's Guide* at http://e−docs.bea.com/wlp/docs81/pdf/compoze_portlets_users_guide.pdf.

Related Topics

Creating a Portal File

Portal Samples

Getting Started with Page Flows

My Calendar Portlet

The My Calendar portlet provides a full−featured calendar system to let you manage and configure appointments and reminders. The portlet also provides an edit mode to let you set calendar preferences and options.

Concepts Demonstrated by this Sample

This Java Page Flow portlet illustrates group collaboration functionality within a portlet. The portlet provides edit mode.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

When this portlet is used in a domain (for example, in the portalApp in the Sample Portal Domain), the EJBs it uses are registered with JNDI names that can be used only once in the domain. That means you can use the following collaboration portlets in only one portal application in a domain: My Mail, My Task List, My Calendar, My Contacts, Discussion Forums, and Discussion Forum Administration. Within that portal application you can create multiple portal Web projects that can each contain multiple portals that reuse these portlets.

1. Create a portal application in a domain that has not used the collaboration portlets.
2. Make sure your portal application is open and the server is running (***Tools−−>WebLogic Server−−>Start WebLogic Server***).
3. Import/add the following directories and files into your portal application and portal Web project using WebLogic Workshop. (Right−click−−>***Import*** or ***Add Module*** or ***Add Library*** on the target directory). You may need to create the appropriate directories in your application.

   Be sure to add the harmony_portlets.jar library first, as shown in the following table.

Import this	***into this WebLogic Workshop directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/APP−INF/lib/harmony_portlets.jar`	<PORTAL_APP>/Libraries/

`<WEBLOGIC_HOME>/samples/portal/portalApp/calendar_ejb.jar`
`security_ejb.jar`
`uniqueid_ejb.jar`
	<PORTAL_APP>/Modules/

`<WEBLOGIC_HOME>/samples/portal/portalApp/`	<PORTAL_APP>/<project>/

`sampleportal/portlets/includes/collaboration/native_calendar.portlet`	portlets/includes/collaboration
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/collaboration/nativedb/calendar/`	<PORTAL_APP>/<project>/p `collaboration/native`

Note: If you add the non−JAR resources to directories other than those shown, you must open the portlet file in WebLogic Workshop and edit the Content URI for both the portlet's main content and Edit page content; you must modify the package path and <view−properties> paths in the Java Page Flow files; and you must modify the import statement to the ContentController and modify any other relevant paths in the JSPs.

6. Add the collaboration tables to your database. If you have already performed this for another collaboration portlet, skip this step.

   Note: If you ran the create_* database script to set up your database, the collaboration tables already exist. Do not run create_* if, for example, you are using the default PointBase database and have already added records to the database. Follow these instructions instead.

   To add the collaboration tables to an existing database, run the following database script:

   <BEA_HOME>/<WEBLOGIC_HOME>/portal/db/<DB_TYPE>/<DB_VERSION>/collaboration_create_tab

   for example

   bea/weblogic81/portal/db/pointbase/44/collaboration_create_tables.sql

   To run this script for PointBase:

   a. Start the PointBase Console. In a command window, run

      <DOMAIN>/startPointBaseConsole.cmd(.sh)
   b. Log into the console. The default login is weblogic/weblogic.
   c. Choose ***File−−>Open***.
   d. Open the collaboration_create_tables.sql script. The script opens in the Enter SQL Commands window.
   e. Click the ***Execute All*** button. The collaboration tables are created.
   f. Close the PointBase Console.

7. Add entries to <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   a. Open <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   b. Open <BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/web.xml
   c. From the sampleportal web.xml file, copy the following sections into your project web.xml file in the appropriate locations:

```
<!--  Compoze Collaboration V Calendar Servlet -->
<servlet>
    <servlet-name>CompozeNativeCalendarVCalendarServlet</servlet-name>
    <servlet-class>com.compoze.calendar.AppointmentVCalendarServlet</servle
</servlet>
```

```
.
.
.
<!--  Compoze Collaboration V Calendar Servlet Mapping -->
<servlet-mapping>
    <servlet-name>CompozeNativeCalendarVCalendarServlet</servlet-name>
    <url-pattern>*.compozevcalendarservlet</url-pattern>
</servlet-mapping>
.
.
.
<ejb-ref>
    <description>Unique ID Generator</description>
    <ejb-ref-name>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</ejb-ref-
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</home>
    <remote>com.compoze.ejb.uniqueid.IUniqueIDGenerator</remote>
    <ejb-link>UniqueIDGenerator</ejb-link>
</ejb-ref>
<ejb-ref>
    <description>Access Control Manager</description>
    <ejb-ref-name>com.compoze.security.acl.IAccessControllerHome</ejb-ref-n
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.security.acl.IAccessControllerHome</home>
    <remote>com.compoze.security.acl.IAccessController</remote>
    <ejb-link>AccessController</ejb-link>
</ejb-ref>
<ejb-ref>
    <description>Calendar Manager</description>
    <ejb-ref-name>com.compoze.calendar.ejb.ICalendarManagerHome</ejb-ref-na
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.calendar.ejb.ICalendarManagerHome</home>
    <remote>com.compoze.calendar.ejb.ICalendarManager</remote>
    <ejb-link>CalendarManager</ejb-link>
</ejb-ref>
```

d. Save and close your project web.xml file.

8. Perform the following steps only if you are using a database other than PointBase.
    a. Stop the server. Choose ***Tools−−>WebLogic Server−−>Stop WebLogic Server***.
    b. Modify your domain's setDomainEnv.cmd(.sh) to use the correct database.

       In the entry set HARMONY_PORTLETS_PROPERTIES=−Dejbruntime.database=pointbase44, use the commented area above this entry to replace the pointbase44 entry with the name of your database driver. Possible values are listed above that entry.
    c. Save setDomainEnv.cmd(.sh).
    d. Restart the server.

9. Add the Login to Portal Portlet to your portal Web project. Users must log in to use the collaboration portlets.
10. Open your portal file and navigate the page where you want the portlet to appear.
11. In the ***Data Palette*** window, drag the ***My Calendar*** portlet onto the portal page.
12. In the ***Property Editor*** window, set any relevant properties.
13. Save the portal file.
14. View your portal with the WebLogic Test Browser or with your default browser.
    ♦ ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).

- ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

For instructions on using the portlet's features, see *Compoze Portlets for BEA WebLogic Portal User's Guide* at http://e−docs.bea.com/wlp/docs81/pdf/compoze_portlets_users_guide.pdf.

Related Topics

Creating a Portal File

Portal Samples

Getting Started with Page Flows

My Contacts Portlet

The My Contacts portlet provides full−featured address book management. The portlet also provides an edit mode to let you set contact management preferences.

Concepts Demonstrated by this Sample

This Java Page Flow portlet illustrates group collaboration functionality within a portlet. The portlet provides edit mode.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

When this portlet is used in a domain (for example, in the portalApp in the Sample Portal Domain), the EJBs it uses are registered with JNDI names that can be used only once in the domain. That means you can use the following collaboration portlets in only one portal application in a domain: My Mail, My Task List, My Calendar, My Contacts, Discussion Forums, and Discussion Forum Administration. Within that portal application you can create multiple portal Web projects that can each contain multiple portals that reuse these portlets.

1. Create a portal application in a domain that has not used the collaboration portlets.
2. Make sure your portal application is open and the server is running (***Tools−−>WebLogic Server−−>Start WebLogic Server***).
3. Import/add the following directories and files into your portal application and portal Web project using WebLogic Workshop. (Right−click−−>***Import*** or ***Add Module*** or ***Add Library*** on the target directory). You may need to create the appropriate directories in your application.

   Be sure to add the harmony_portlets.jar library first, as shown in the following table.

Import this	***into this WebLogic Workshop directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/APP−INF/lib/harmony_portlets.jar`	<PORTAL_APP>/Libraries/

`<WEBLOGIC_HOME>/samples/portal/portalApp/`
`contact_ejb.jar`
`security_ejb.jar`
`uniqueid_ejb.jar`
	<PORTAL_APP>/Modules/

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/includes/collaboration/`	<PORTAL_APP>/<project>/portlets/includes/collaboration

native_contact.portlet	
<WEBLOGIC_HOME>/samples/portal/portalApp/ sampleportal/portlets/collaboration/nativedb/ contact/	<PORTAL_APP>/<project>/p collaboration/native

Note: If you add the non−JAR resources to directories other than those shown, you must open the portlet file in WebLogic Workshop and edit the Content URI for both the portlet's main content and Edit page content; you must modify the package path and <view−properties> paths in the Java Page Flow files; and you must modify the import statement to the ContentController and modify any other relevant paths in the JSPs.

4. Add the collaboration tables to your database. If you have already performed this for another collaboration portlet, skip this step.

   Note: If you ran the create_* database script to set up your database, the collaboration tables already exist. Do not run create_* if, for example, you are using the default PointBase database and have already added records to the database. Follow these instructions instead.

   To add the collaboration tables to an existing database, run the following database script:

   <BEA_HOME>/<WEBLOGIC_HOME>/portal/db/<DB_TYPE>/<DB_VERSION>/collaboration_create_tab

   for example

   bea/weblogic81/portal/db/pointbase/44/collaboration_create_tables.sql
5. To run this script for PointBase:
   a. Start the PointBase Console. In a command window, run

      <DOMAIN>/startPointBaseConsole.cmd(.sh)
   b. Log into the console. The default login is weblogic/weblogic.
   c. Choose ***File−−>Open***.
   d. Open the collaboration_create_tables.sql script. The script opens in the Enter SQL Commands window.
   e. Click the ***Execute All*** button. The collaboration tables are created.
   f. Close the PointBase Console.
6. Add entries to <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   a. Open <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   b. Open <BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/web.xml
   c. From the sampleportal web.xml file, copy the following sections into your project web.xml file in the appropriate locations:

```
<ejb-ref>
    <description>Unique ID Generator</description>
    <ejb-ref-name>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</ejb-ref-
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</home>
    <remote>com.compoze.ejb.uniqueid.IUniqueIDGenerator</remote>
    <ejb-link>UniqueIDGenerator</ejb-link>
</ejb-ref>
<ejb-ref>
    <description>Access Control Manager</description>
    <ejb-ref-name>com.compoze.security.acl.IAccessControllerHome</ejb-ref-n
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.security.acl.IAccessControllerHome</home>
```

```
        <remote>com.compoze.security.acl.IAccessController</remote>
        <ejb-link>AccessController</ejb-link>
    </ejb-ref>
    <ejb-ref>
        <description>Contact Manager</description>
        <ejb-ref-name>com.compoze.contact.ejb.IContactManagerHome</ejb-ref-name
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.compoze.contact.ejb.IContactManagerHome</home>
        <remote>com.compoze.contact.ejb.IContactManager</remote>
        <ejb-link>ContactManager</ejb-link>
    </ejb-ref>
```

   d. Save and close your project web.xml file.

9. Perform the following steps only if you are using a database other than PointBase.
   a. Stop the server. Choose ***Tools−−>WebLogic Server−−>Stop WebLogic Server***.
   b. Modify your domain's setDomainEnv.cmd(.sh) to use the correct database.

      In the entry set HARMONY_PORTLETS_PROPERTIES=−Dejbruntime.database=pointbase44, use the commented area above this entry to replace the pointbase44 entry with the name of your database driver. Possible values are listed above that entry.
   c. Save setDomainEnv.cmd(.sh).
   d. Restart the server.

10. Add the Login to Portal Portlet to your portal Web project. Users must log in to use the collaboration portlets.
11. Open your portal file and navigate the page where you want the portlet to appear.
12. In the ***Data Palette*** window, drag the ***My Contacts*** portlet onto the portal page.
13. In the ***Property Editor*** window, set any relevant properties.
14. Save the portal file.
15. View your portal with the WebLogic Test Browser or with your default browser.
    - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
    - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

For instructions on using the portlet's features, see *Compoze Portlets for BEA WebLogic Portal User's Guide* at http://e−docs.bea.com/wlp/docs81/pdf/compoze_portlets_users_guide.pdf.

Related Topics

Creating a Portal File

Portal Samples

Getting Started with Page Flows

Discussion Forums Portlet

The Discussion Forums portlet lets you participate in threaded forum conversations. To administer discussion forums in this portlet, use the Discussion Forum Administration portlet.

Concepts Demonstrated by this Sample

This Java Page Flow portlet illustrates group collaboration functionality within a portlet. The portlet provides edit mode.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

When this portlet is used in a domain (for example, in the portalApp in the Sample Portal Domain), the EJBs it uses are registered with JNDI names that can be used only once in the domain. That means you can use the following collaboration portlets in only one portal application in a domain: My Mail, My Task List, My Calendar, My Contacts, Discussion Forums, and Discussion Forum Administration. Within that portal application you can create multiple portal Web projects that can each contain multiple portals that reuse these portlets.

1. Create a portal application in a domain that has not used the collaboration portlets.
2. Make sure your portal application is open and the server is running (***Tools−−>WebLogic Server−−>Start WebLogic Server***).
3. Import/add the following directories and files into your portal application and portal Web project using WebLogic Workshop. (Right−click−−>***Import*** or ***Add Module*** or ***Add Library*** on the target directory). You may need to create the appropriate directories in your application.

   Be sure to add the harmony_portlets.jar library first, as shown in the following table.

Import this	***into this WebLogic Workshop directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/APP−INF/lib/harmony_portlets.jar`	<PORTAL_APP>/Libraries/

`<WEBLOGIC_HOME>/samples/portal/portalApp/`
`discussion_ejb.jar`
`security_ejb.jar`
`uniqueid_ejb.jar`
	<PORTAL_APP>/Modules/

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/includes/collaboration/`	<PORTAL_APP>/<project>/portlets/includes/collaboration

native_discussion.portlet	
<WEBLOGIC_HOME>/samples/portal/portalApp/ sampleportal/portlets/collaboration/nativedb/ discussion/	<PORTAL_APP>/<project>/p collaboration/native

Note: If you add the non−JAR resources to directories other than those shown, you must open the portlet file in WebLogic Workshop and edit the Content URI for both the portlet's main content and Edit page content; you must modify the package path and <view−properties> paths in the Java Page Flow files; and you must modify the import statement to the ContentController and modify any other relevant paths in the JSPs.

6. Add the collaboration tables to your database. If you have already performed this for another collaboration portlet, skip this step.

   Note: If you ran the create_* database script to set up your database, the collaboration tables already exist. Do not run create_* if, for example, you are using the default PointBase database and have already added records to the database. Follow these instructions instead.

   To add the collaboration tables to an existing database, run the following database script:

   <BEA_HOME>/<WEBLOGIC_HOME>/portal/db/<DB_TYPE>/<DB_VERSION>/collaboration_create_tab

   for example

   bea/weblogic81/portal/db/pointbase/44/collaboration_create_tables.sql

   To run this script for PointBase:

   a. Start the PointBase Console. In a command window, run

      <DOMAIN>/startPointBaseConsole.cmd(.sh)
   b. Log into the console. The default login is weblogic/weblogic.
   c. Choose ***File−−>Open***.
   d. Open the collaboration_create_tables.sql script. The script opens in the Enter SQL Commands window.
   e. Click the ***Execute All*** button. The collaboration tables are created.
   f. Close the PointBase Console.

7. Add entries to <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   a. Open <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   b. Open <BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/web.xml
   c. From the sampleportal web.xml file, copy the following sections into your project web.xml file in the appropriate locations:

```
<!−− Compoze Collaboration Discussion Thread Attachment Servlet −−>
<servlet>
    <servlet−name>CompozeDiscussionMessageFileAttachmentServlet</servlet−na
    <servlet−class>com.compoze.discussion.MessageFileAttachmentServlet</ser
</servlet>
```

```
<!-- Compoze Collaboration Discussion Thread Topic Attachment Servlet -->
<servlet>
    <servlet-name>CompozeDiscussionTopicFileAttachmentServlet</servlet-name
    <servlet-class>com.compoze.discussion.TopicFileAttachmentServlet</servl
</servlet>
.
.
.
<!-- Compoze Collaboration Discussion Thread Attachment Servlet Mapping -->
<servlet-mapping>
    <servlet-name>CompozeDiscussionMessageFileAttachmentServlet</servlet-na
    <url-pattern>*.compozediscussionmessagefileattachmentservlet</url-patte
</servlet-mapping>

<!-- Compoze Collaboration Discussion Thread Topic Attachment Servlet Mappi
<servlet-mapping>
    <servlet-name>CompozeDiscussionTopicFileAttachmentServlet</servlet-name
    <url-pattern>*.compozediscussiontopicfileattachmentservlet</url-pattern
</servlet-mapping>
.
.
.
        <resource-ref>
                <res-ref-name>ebusinessDataSource</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
        </resource-ref>
.
.
.
<ejb-ref>
    <description>Unique ID Generator</description>
           <ejb-ref-name>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</e
           <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</home>
    <remote>com.compoze.ejb.uniqueid.IUniqueIDGenerator</remote>
    <ejb-link>UniqueIDGenerator</ejb-link>
</ejb-ref>
<ejb-ref>
    <description>Discussion Forum Manager</description>
    <ejb-ref-name>com.compoze.discussion.ejb.IDiscussionManagerHome</ejb-re
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.discussion.ejb.IDiscussionManagerHome</home>
    <remote>com.compoze.discussion.ejb.IDiscussionManager</remote>
    <ejb-link>DiscussionManager</ejb-link>
</ejb-ref>
<ejb-ref>
    <description>Access Control Manager</description>
    <ejb-ref-name>com.compoze.security.acl.IAccessControllerHome</ejb-ref-n
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.security.acl.IAccessControllerHome</home>
    <remote>com.compoze.security.acl.IAccessController</remote>
    <ejb-link>AccessController</ejb-link>
</ejb-ref>
```

d. Save and close your project web.xml file.

8. Add entries to <PORTAL_APP>/<PROJECT>/WEB−INF/weblogic.xml.
   a. Open <PORTAL_APP>/<PROJECT>/WEB−INF/weblogic.xml.
   b. Open <BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/weblogic

c. From the sampleportal weblogic.xml file, copy the following sections into your project weblogic.xml file inside the <reference−descriptor> element:

```
<resource-description>
    <res-ref-name>ebusinessDataSource</res-ref-name>
    <jndi-name>weblogic.jdbc.jts.ebusinessPool</jndi-name>
</resource-description>
```

d. Save and close your project weblogic.xml file.

9. Perform the following steps only if you are using a database other than PointBase.
   a. Stop the server. Choose ***Tools−−>WebLogic Server−−>Stop WebLogic Server***.
   b. Modify your domain's setDomainEnv.cmd(.sh) to use the correct database.

      In the entry set HARMONY_PORTLETS_PROPERTIES=−Dejbruntime.database=pointbase44, use the commented area above this entry to replace the pointbase44 entry with the name of your database driver. Possible values are listed above that entry.
   c. Save setDomainEnv.cmd(.sh).
   d. Restart the server.

9. Add the Login to Portal Portlet to your portal Web project. Users must log in to use the collaboration portlets.
10. Open your portal file and navigate the page where you want the portlet to appear.
11. In the ***Data Palette*** window, drag the ***Discussion Forums*** portlet onto the portal page.
12. In the ***Property Editor*** window, set any relevant properties.
13. Save the portal file.
14. View your portal with the WebLogic Test Browser or with your default browser.
    - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
    - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

For instructions on using the portlet's features, see *Compoze Portlets for BEA WebLogic Portal User's Guide* at http://e−docs.bea.com/wlp/docs81/pdf/compoze_portlets_users_guide.pdf.

Related Topics

Creating a Portal File

Portal Samples

Getting Started with Page Flows

Discussion Forum Administration Portlet

The Discussion Forum Administration portlet lets you administer threaded discussion forums in the Discussion Forums portlet.

Concepts Demonstrated by this Sample

This portlet surfaces a Java Page Flow and provides edit mode.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

When this portlet is used in a domain (for example, in the portalApp in the Sample Portal Domain), the EJBs it uses are registered with JNDI names that can be used only once in the domain. That means you can use the following collaboration portlets in only one portal application in a domain: My Mail, My Task List, My Calendar, My Contacts, Discussion Forums, and Discussion Forum Administration. Within that portal application you can create multiple portal Web projects that can each contain multiple portals that reuse these portlets.

1. Create a portal application in a domain that has not used the collaboration portlets.
2. Make sure your portal application is open and the server is running (***Tools−−>WebLogic Server−−>Start WebLogic Server***).
3. Import/add the following directories and files into your portal application and portal Web project using WebLogic Workshop. (Right−click−−>***Import*** or ***Add Module*** or ***Add Library*** on the target directory). You may need to create the appropriate directories in your application.

   Be sure to add the harmony_portlets.jar library first, as shown in the following table.

Import this	***into this WebLogic Workshop directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/APP−INF/lib/harmony_portlets.jar`	<PORTAL_APP>/Libraries/
`<WEBLOGIC_HOME>/samples/portal/portalApp/` `discussion_ejb.jar` `security_ejb.jar` `uniqueid_ejb.jar`	<PORTAL_APP>/Modules/
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/includes/collaboration/native_discussion_admin.portlet`	<PORTAL_APP>/<project>/ portlets/includes/collaboration

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/collaboration/nativedb/discussion/`	<PORTAL_APP>/<project>/p `collaboration/native`

Note: If you add the non−JAR resources to directories other than those shown, you must open the portlet file in WebLogic Workshop and edit the Content URI for both the portlet's main content and Edit page content; you must modify the package path and <view−properties> paths in the Java Page Flow files; and you must modify the import statement to the ContentController and modify any other relevant paths in the JSPs.

4. Add the collaboration tables to your database. If you have already performed this for another collaboration portlet, skip this step.

   Note: If you ran the create_* database script to set up your database, the collaboration tables already exist. Do not run create_* if, for example, you are using the default PointBase database and have already added records to the database. Follow these instructions instead.

   To add the collaboration tables to an existing database, run the following database script:

   <BEA_HOME>/<WEBLOGIC_HOME>/portal/db/<DB_TYPE>/<DB_VERSION>/collaboration_create_tabl

   for example

   bea/weblogic81/portal/db/pointbase/44/collaboration_create_tables.sql

   To run this script for PointBase:

   a. Start the PointBase Console. In a command window, run

      <DOMAIN>/startPointBaseConsole.cmd(.sh)
   b. Log into the console. The default login is weblogic/weblogic.
   c. Choose ***File−−>Open***.
   d. Open the collaboration_create_tables.sql script. The script opens in the Enter SQL Commands window.
   e. Click the ***Execute All*** button. The collaboration tables are created.
   f. Close the PointBase Console.

7. Add entries to <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   a. Open <PORTAL_APP>/<PROJECT>/WEB−INF/web.xml.
   b. Open <BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/web.xml
   c. From the sampleportal web.xml file, copy the following sections into your project web.xml file in the appropriate locations:

```
<!−− Compoze Collaboration Discussion Thread Attachment Servlet −−>
<servlet>
    <servlet-name>CompozeDiscussionMessageFileAttachmentServlet</servlet-na
    <servlet-class>com.compoze.discussion.MessageFileAttachmentServlet</ser
</servlet>

<!−− Compoze Collaboration Discussion Thread Topic Attachment Servlet −−>
<servlet>
    <servlet-name>CompozeDiscussionTopicFileAttachmentServlet</servlet-name
    <servlet-class>com.compoze.discussion.TopicFileAttachmentServlet</servl
</servlet>
.
.
.
<!-- Compoze Collaboration Discussion Thread Attachment Servlet Mapping -->
<servlet-mapping>
    <servlet-name>CompozeDiscussionMessageFileAttachmentServlet</servlet-na
    <url-pattern>*.compozediscussionmessagefileattachmentservlet</url-patte
</servlet-mapping>

<!-- Compoze Collaboration Discussion Thread Topic Attachment Servlet Mappi
<servlet-mapping>
    <servlet-name>CompozeDiscussionTopicFileAttachmentServlet</servlet-name
    <url-pattern>*.compozediscussiontopicfileattachmentservlet</url-pattern
</servlet-mapping>
.
.
.
        <resource-ref>
                <res-ref-name>ebusinessDataSource</res-ref-name>
                <res-type>javax.sql.DataSource</res-type>
                <res-auth>Container</res-auth>
        </resource-ref>
.
.
.
<ejb-ref>
    <description>Unique ID Generator</description>
    <ejb-ref-name>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</ejb-ref-
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.ejb.uniqueid.IUniqueIDGeneratorHome</home>
    <remote>com.compoze.ejb.uniqueid.IUniqueIDGenerator</remote>
    <ejb-link>UniqueIDGenerator</ejb-link>
</ejb-ref>
<ejb-ref>
    <description>Discussion Forum Manager</description>
    <ejb-ref-name>com.compoze.discussion.ejb.IDiscussionManagerHome</ejb-re
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.discussion.ejb.IDiscussionManagerHome</home>
    <remote>com.compoze.discussion.ejb.IDiscussionManager</remote>
    <ejb-link>DiscussionManager</ejb-link>
</ejb-ref>
<ejb-ref>
    <description>Access Control Manager</description>
    <ejb-ref-name>com.compoze.security.acl.IAccessControllerHome</ejb-ref-n
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.compoze.security.acl.IAccessControllerHome</home>
    <remote>com.compoze.security.acl.IAccessController</remote>
    <ejb-link>AccessController</ejb-link>
</ejb-ref>
```

   d. Save and close your project web.xml file.

8. Add entries to <PORTAL_APP>/<PROJECT>/WEB−INF/weblogic.xml.
   a. Open <PORTAL_APP>/<PROJECT>/WEB−INF/weblogic.xml.
   b. Open <BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB−INF/weblogic
   c. From the sampleportal weblogic.xml file, copy the following sections into your project weblogic.xml file inside the <reference−descriptor> element:

```
<resource-description>
    <res-ref-name>ebusinessDataSource</res-ref-name>
    <jndi-name>weblogic.jdbc.jts.ebusinessPool</jndi-name>
</resource-description>
```

   d. Save and close your project weblogic.xml file.

9. Perform the following steps only if you are using a database other than PointBase.
   a. Stop the server. Choose ***Tools-->WebLogic Server-->Stop WebLogic Server***.
   b. Modify your domain's setDomainEnv.cmd(.sh) to use the correct database.

      In the entry set HARMONY_PORTLETS_PROPERTIES=-Dejbruntime.database=pointbase44, use the commented area above this entry to replace the pointbase44 entry with the name of your database driver. Possible values are listed above that entry.
   c. Save setDomainEnv.cmd(.sh).
   d. Restart the server.
10. Add the Login to Portal Portlet to your portal Web project. Users must log in to use the collaboration portlets.
11. Open your portal file and navigate the page where you want the portlet to appear.
12. In the ***Data Palette*** window, drag the ***Discussion Forum Administration*** portlet onto the portal page.
13. In the ***Property Editor*** window, set any relevant properties.
14. Save the portal file.
15. View your portal with the WebLogic Test Browser or with your default browser.
    - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
    - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal-->Open Current Portal***.

For instructions on using the portlet's features, see *Compoze Portlets for BEA WebLogic Portal User's Guide* at http://e−docs.bea.com/wlp/docs81/pdf/compoze_portlets_users_guide.pdf.

Related Topics

Creating a Portal File

Portal Samples

Getting Started with Page Flows

My Content Portlet

The My Content portlet lets you completely manage your content in the BEA Virtual Content Repository without having to use the WebLogic Administration Portal. With My Content portlet users can create, update, and delete content directories and nodes, and they can browse content hierarchies and and search for content.

My Content portlet supports delegated administration, letting users view and manage only the content nodes delegated to them.

My Content Portlet does not support BEA Library Services−Enabled repositories. To access a library services−enabled repository via a portlet, use the Content Management Portlet.

Concepts Demonstrated by this Sample

This Java Page Flow portlet supports full create, read, update, and delete (CRUD) capabilities and provides security through delegated administration.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

For detailed instructions for Setting Up the My Content Portlet, see Setting Up My Content Portlet.

Using My Content Portlet

The following procedures show you how to use My Content portlet:

Creating and Modifying Content with My Content Portlet

Searching with My Content Portlet

Related Topics

Portal Samples

For information on setting up content management in the BEA Virtual Content Repository and setting up delegated administration, see the WebLogic Administration Portal documentation.

Content Management Portlet

The Content Management portlet lets you manage your content in the BEA Virtual Content Repository without having to use the WebLogic Administration Portal. With Content Management portlet users can create, update, and delete content directories and nodes.

The Content Management portlet does not allow you to view and modify content types, modify repositories, or set Delegated Administration policies. To set up types, configure repositories or set Delegated Administration policies, you must use the WebLogic Administration portal.

Concepts Demonstrated by this Sample

This portlet supports full create, read, update, and delete (CRUD) capabilities for content, BEA Library Services and provides security through previously set delegated administration.

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

For detailed instructions for Setting Up the My Content Portlet, see Setting Up the Content Management Portlet.

Related Topics

Portal Samples

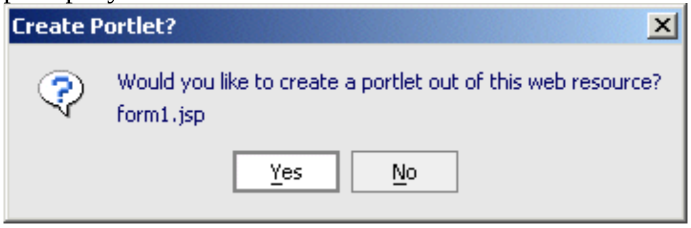
For information on setting up content management in the BEA Virtual Content Repository, library services and setting up delegated administration, see the WebLogic Administration Portal documentation.

Creating Portlets

If the pre−built portlets in the Portlet Library do not match your current needs, you can create new portlets in a variety of ways, including through the use of the Portlet Wizard.

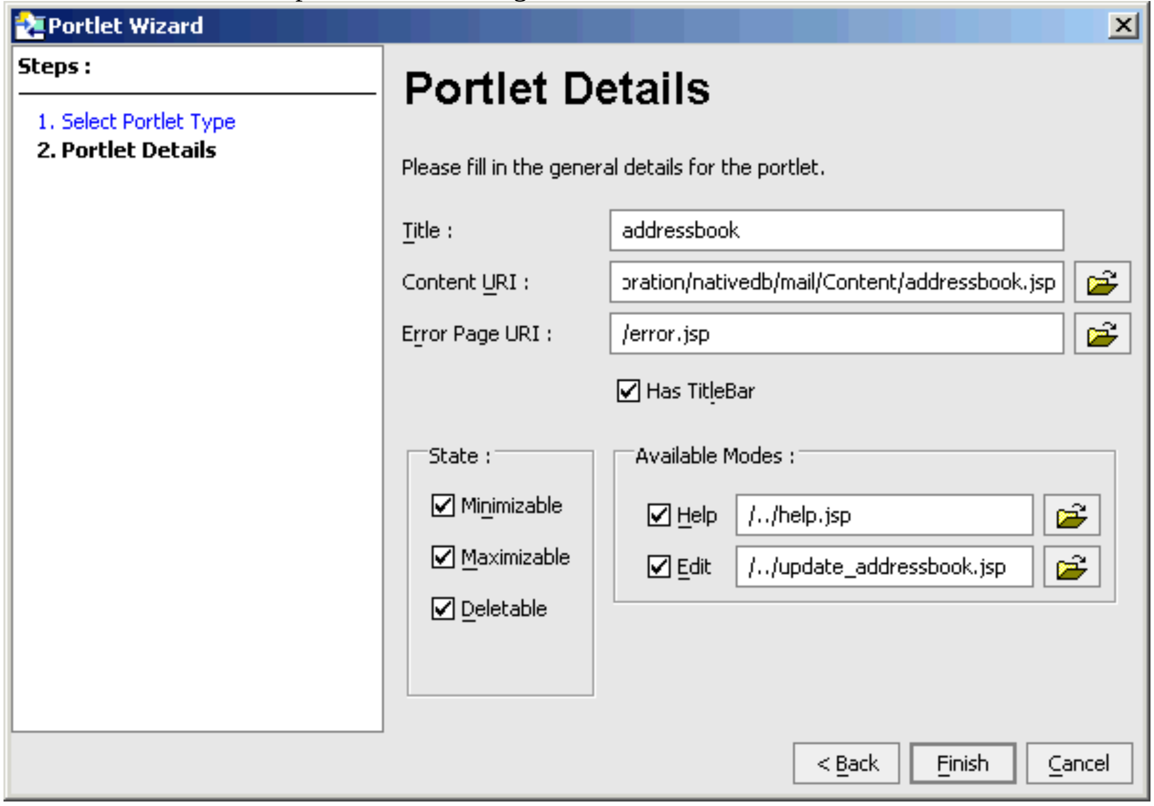
A new portlet can be associated with an existing resource, or created first and associated with a resource later. The Portlet Wizard is invoked in the WebLogic Workshop IDE anytime you perform one of these operations:

- Select ***File*** > ***New*** > ***Portlet*** from the IDE's top−level menu. Or right−mouse click on a folder in your web application, and select ***New*** > ***Portlet***. After naming the portlet and choosing the Create button, the Portlet Wizard is invoked and you can select the portlet type.
- Drag and drop a resource such as a JSP from the Application pane onto a placeholder area of an open portal. (That is, a <portal−name>.portal file is open in the Portal Designer.) WebLogic Workshop prompts you:


Create Portlet?

Would you like to create a portlet out of this web resource?
form1.jsp

  If you select Yes, the Portlet Wizard is invoked in the same way shown for the next case.
- Right−mouse click on an existing resource such as a JSP page, a page flow, a portal placeholder, or a portal content selector; then select ***Generate Portlet...*** from the menu. The Portlet Wizard displays a Details screen. For example, here we had right−mouse clicked on a JSP file.


Portlet Wizard

Steps :
1. Select Portlet Type
2. Portlet Details

Portlet Details

Please fill in the general details for the portlet.

Title : addressbook
Content URI : ɔration/nativedb/mail/Content/addressbook.jsp
Error Page URI : /error.jsp
☑ Has TitleBar

State :
☑ Minimizable
☑ Maximizable
☑ Deletable

Available Modes :
☑ Help /../help.jsp
☑ Edit /../update_addressbook.jsp

Types of Portlets Created by the Portlet Wizard

The Portlet Wizard can create several types of portlets. The portlet type is set on the wizard's first screen; when generating a portlet for an existing resource, the type may have been already detected.

Type	***Description***
JSP/HTML Portlet	Creates a portlet that points to a JSP or HTML file for its content. These types of portlets can be simple to implement and deploy, and provide basic functionality without a lot of complexity. However, business logic and presentation layer can get combined in the JSPs; as the application grows, this often leads to escalating maintenance costs while trying to update the webapp and share code. This type of portlet is not well suited for advanced portlet navigation.
Java Portlet	Creates a JSR 168 compliant portlet. This creates a Java file. Accommodates portability for portlets across platforms. Does not require the use of portal server specific JSP tags. The behavior is similar to a Servlet (although there are differences). For related information, see "Developing JSR 168 Portlets with WebLogic Portal 8.1" on the BEA dev2dev site. This type of portlet is intended for software companies and other enterprises that are concerned with portability across multiple portlet containers. Current disadvantages are that this type of portlet does not leverage BEA advanced portlet features, and this type requires a deeper understanding of the J2EE programming model.
Java Page Flow Portlet	Creates a portlet that uses Java Page Flows to retrieve its content. Allow you to separate the user interface code from navigation control and other business logic. Provides the ability to model both simple and advanced portlet navigation. Allow you to leverage other resources such as Java Controls and Web Services. Provides a visual IDE environment to build rich applications based on Struts. The advanced page flow features are not necessary for static or simple, one−view portlets.
Struts Portlet	Creates a Struts−based portlet.
Remote Portlet	Creates a WSRP−compliant remote, or "proxy," portlet. These portlets present content collected from WSRP−compliant producers, allowing you to leverage external sources for portlet content, rather than have to create this content or its framework yourself.

The following topics describe the options you have while generating these types of portlets.

* Building JSP/HTML Portlets
* Building Java Portlets
* Building Java Page Flow Portlets
* Building Struts Portlets
* Building a Remote Portlet
* In addition, you can create portlets that use web services. See Creating a Web Services Portlet.

Related Topics

Implementing WSRP−compliant Portlets

For more detailed infomation on how WSRP works with WebLogic Portal, please refer to ***Using WSRP with WebLogic Portal***.

WSRP Web Service for Remote Portlets is a web services standard that allows you to plug−n−play visual, user−facing web services with portals or other intermediary web applications. It allows you to create portlets that can either provide content to other portlets or consume content from other sources, even those far removed from your enterprise.

Producers and Consumers

WSRP−compliant portlets are either hosted on a producer ("producers") or created on a consumer ("consumers").

Producers

Producers host portlets and provide such services as self−description, mark up, registration, and portlet management. Producers can optionally manage the registration of consumers and require them to pre−register prior to interacting with portlets. A registration establishes a relationship between Consumers and Producers.

Producers are further classified as either complex or simple.

- A *complex* producer requires registration, does support URL rewriting in the consumer, and does support a management interface. By default, all portlets created with WebLogic Workshop 8.1 SP3 are complex producers. You can convert a complex producer to a simple producer to make its pageflows and Struts applications available as "portlets" to remote portals.
- A *simple* producer is a non−portal web application that contains Java page flows and Struts applications. It does not depend upon any portal features (for example, customization). It doesn't require registration, doesn't support URL rewriting in the consumer, and does not support a management interface. A simple producer is often advantageous because it easier to manage and you don't need to have the complete portal installed to run it. You will most commonly see simple producers used as departmental applications.

Since all WLW−created portlets are, by default, hosted on producers, in this documentation, they will be referred to simply as portlets.

Consumers

Consumers aggregate remote portlets and presentt hose portlets to end users. Consumers route requests from users to the appropriate producer, which, in turn processes the request and sends results back to the consumer. The Consumer aggregates the results coming from various producers and send the final result back to the user. In their role as proxies for producers, consumers proxy for end users, consumers provide the necessary separation of the large amount of traffic flowing between them and the producers. They also ensure that all interactions are kept private to that specific user during the session.

In this documentation, portlets created on a consumer are referred to as "remote portlets."

Creating and Maintaining Remote Portlets

The following topics will show you how to use WebLogic Workshop to build and maintain WSRP−compliant remote portlets and how to modify portlets hosted on WSRP−compliant producers:

Building a Remote Portlet

This topic walks you through the steps necessary to create a remote portlet. It also shows you how to add the portlet to a portal and then view that portal on your desktop.

Modifying a Remote Portlet

This topic describes how to add states and modes to a remote portlet,

Customizing a Remote Portlet

This topic tells you where to find complete information about and procedures for changing the appearance of a remote portlet.

Disabling A Producer

This topic describes how to modify the producer configuration file so that producer application cannot be consumed by a remote portlet.

Related Topics

Creating a Portal Application and Portal Web Project

Building Portlets

Customizing Portlet

Building a Remote Portlet

Remote, or "proxy," portlets present content collected from WSRP−compliant producers. Remote portlets route requests from users to the appropriate Producers which, in turn process the request and send results back to the consumer. Consumers then aggregate the results coming from various producers and send the final result back to the user, who can view and use those results in remote portlets. Consumers have the ability to keep traffic separated and maintain all interactions private to that specific user during the interaction.

This procedure describes how to:

- Create a remote portlet for a producer.
- Add the portlet to a portal.
- View the portal and the new remote portlet.

Before You Begin

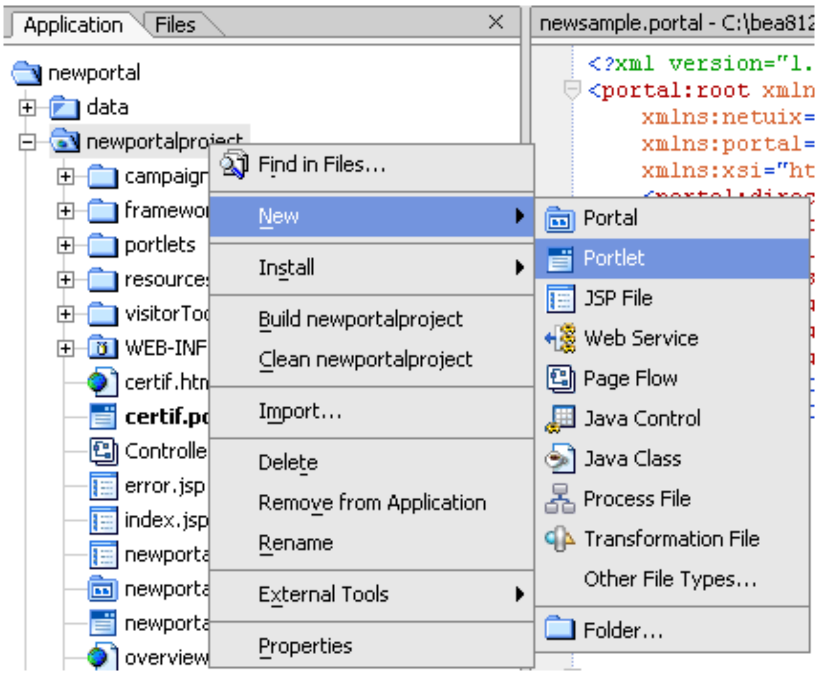
Before you begin this procedure, you must have already created a domain, a portal application, and a portal project. If you haven't, please do so by using the procedures outlined in:

- Creating and Configuring Domains Using the Configuration Wizard (link)
- Creating a Portal Application and Portal Web Project

Creating the Portlet

To create a remote portlet, use this procedure:

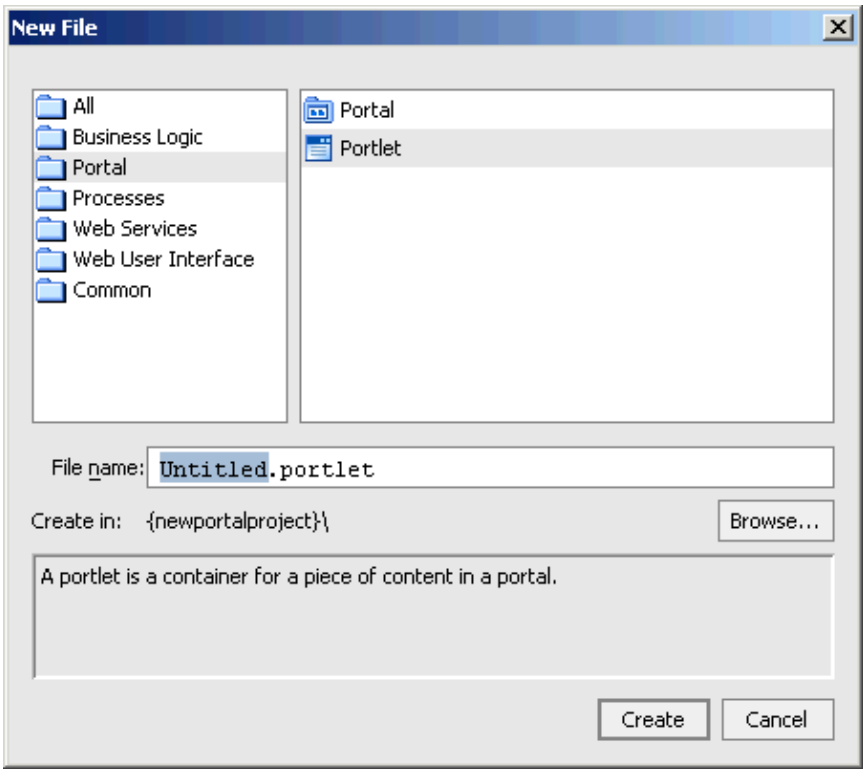
1. With WebLogic Workshop running, right−click the application that hosts the portal for which you are creating the portlet and right−click the portal application (alternately, select that application and select ***File***>***New***>***Portlet***).
2. Select ***New*** > ***Portlet***.



```
<?xml version="1.
<portal:root xmln
    xmlns:netuix=
    xmlns:portal=
    xmlns:xsi="ht
```

(Alternately, open the ***File*** menu and select ***New*** > ***Portlet***)

The New File window appears.

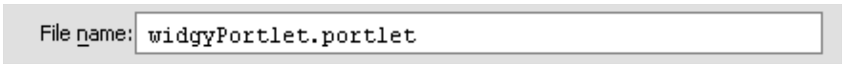


File name: Untitled.portlet

Create in: {newportalproject}\

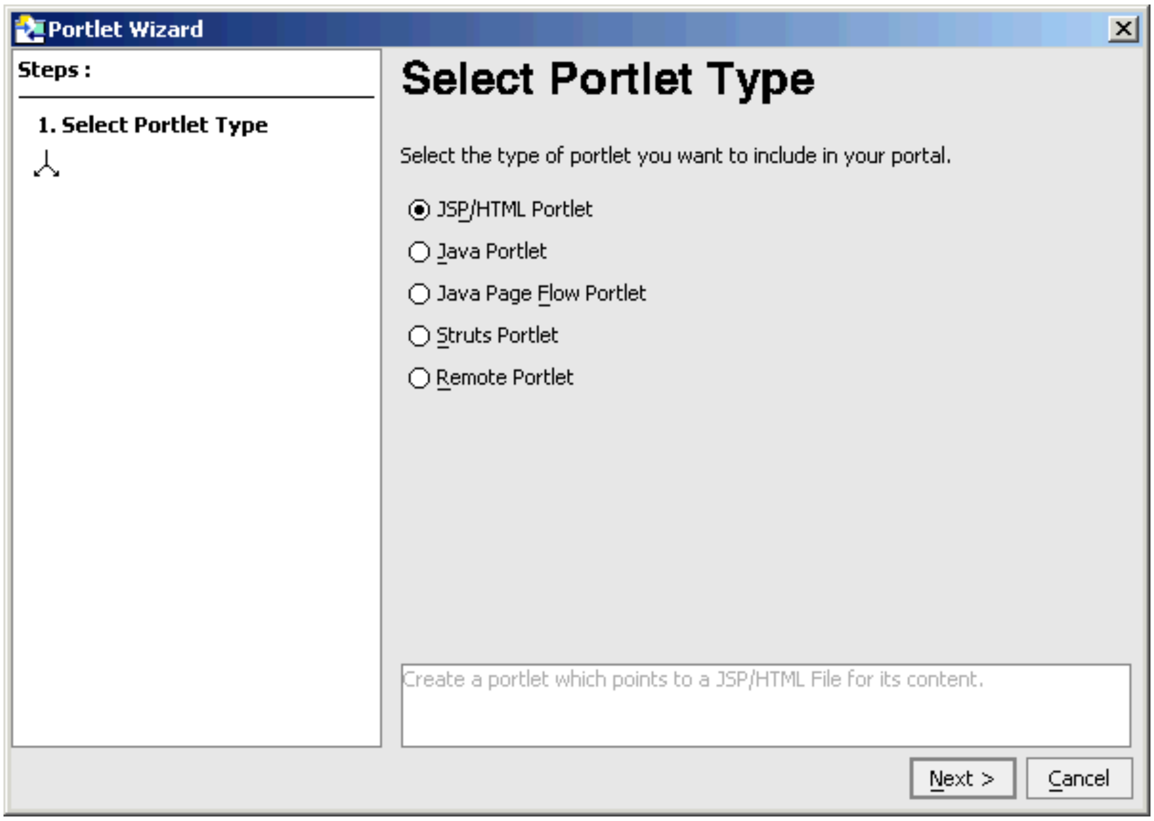
A portlet is a container for a piece of content in a portal.

2. On New File, do the following:
   A. In ***File name***, replace "Untitled" with the name of the portlet (do not change the file extension, `.portlet`).


File name: widgyPortlet.portlet

B. If the portal project displayed in ***Create in***: is not the project inwhich you want to create the portlet, click ***Browse*** to display the Select dialog box and select the desired project (optional)
C. Click ***Create.***

The Portlet Wizard appears.


Portlet Wizard

Steps :

1. Select Portlet Type

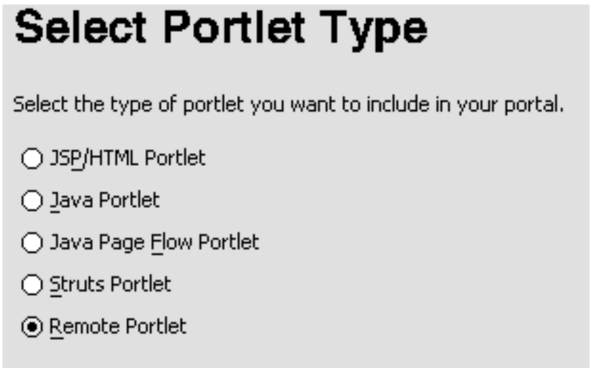
Select Portlet Type

Select the type of portlet you want to include in your portal.

[✓ JSP/HTML Portlet]
Java Portlet
Java Page Flow Portlet
Struts Portlet
Remote Portlet

Create a portlet which points to a JSP/HTML File for its content.

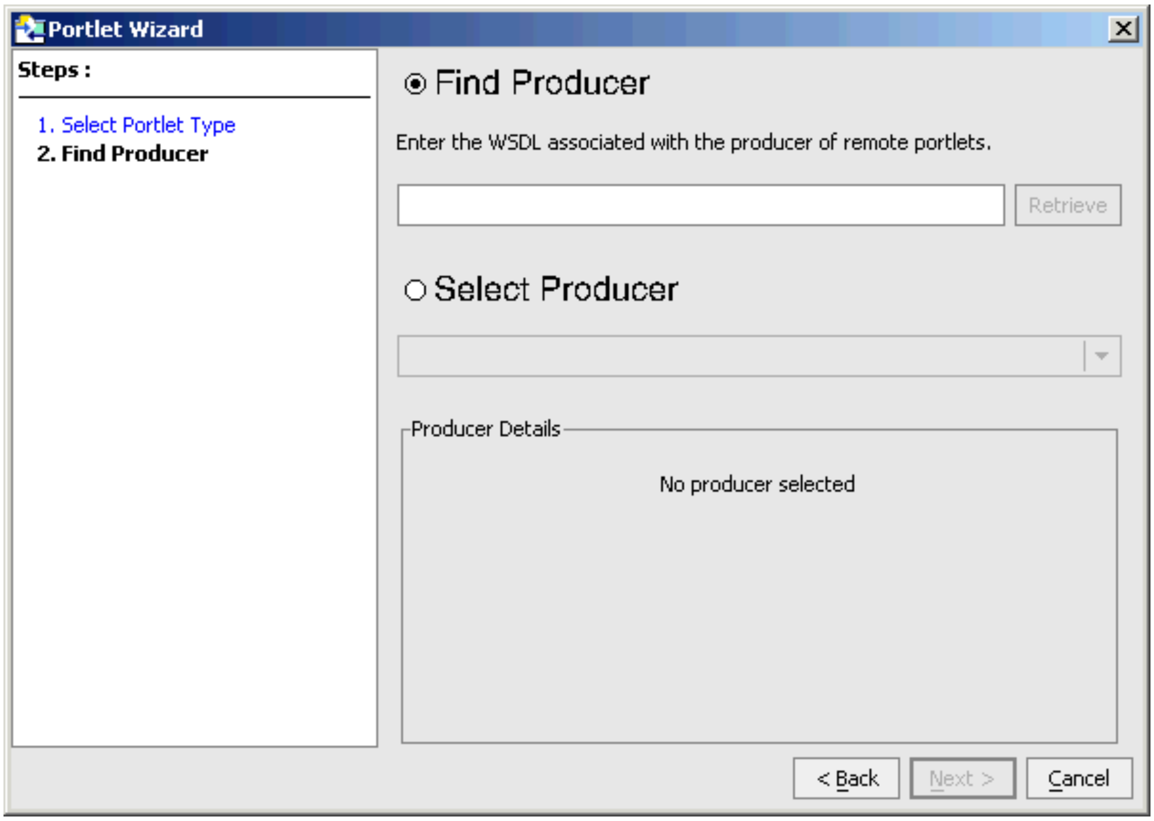
3. Under Select Portlet Type, select ***Remote Portlet*** and click ***Next***.


Select Portlet Type

Select the type of portlet you want to include in your portal.

JSP/HTML Portlet
Java Portlet
Java Page Flow Portlet
Struts Portlet
[✓ Remote Portlet]

The Portlet Wizard's Find Producer window appears:



● Find Producer

Enter the WSDL associated with the producer of remote portlets.

○ Select Producer

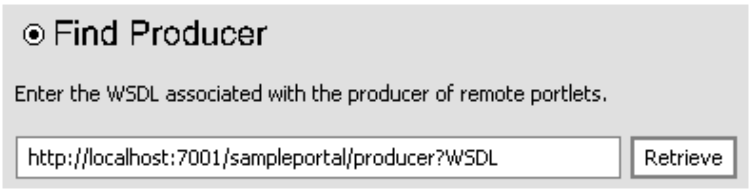
Producer Details

No producer selected

On this window, you can either specify the WSDL for the producer you want to use or select one from the drop−down list.

4. Do one of the following:

• Type the WSDL of the producer of the remote portlet you want to use and click ***Retrieve***; for example:



● Find Producer

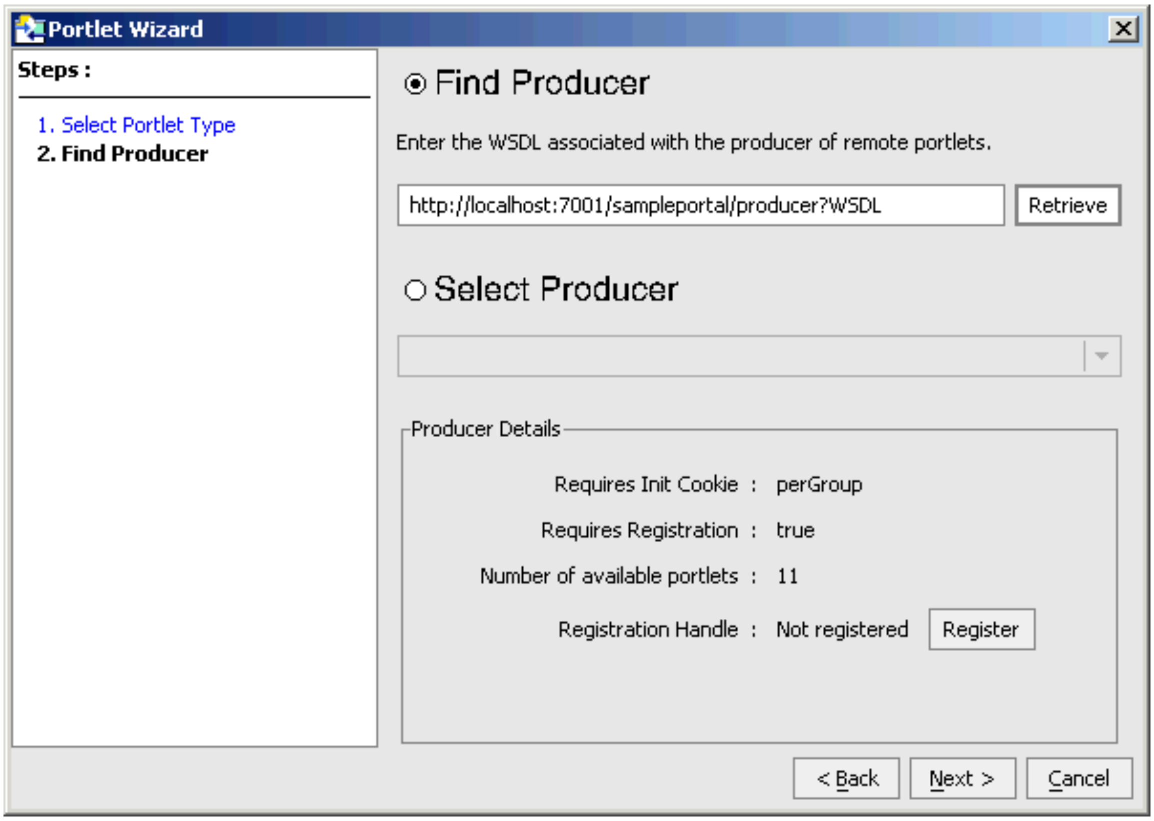
Enter the WSDL associated with the producer of remote portlets.

http://localhost:7001/sampleportal/producer?WSDL

OR

• Click ***Select Producer***, open the drop−down list of producers handles, and select a producer from that list.

The Find Producer window refreshes, this time displaying details on the producer you selected:



● Find Producer

Enter the WSDL associated with the producer of remote portlets.

http://localhost:7001/sampleportal/producer?WSDL [Retrieve]

○ Select Producer

Producer Details

Requires Init Cookie :	perGroup
Requires Registration :	true
Number of available portlets :	11
Registration Handle :	Not registered [Register]

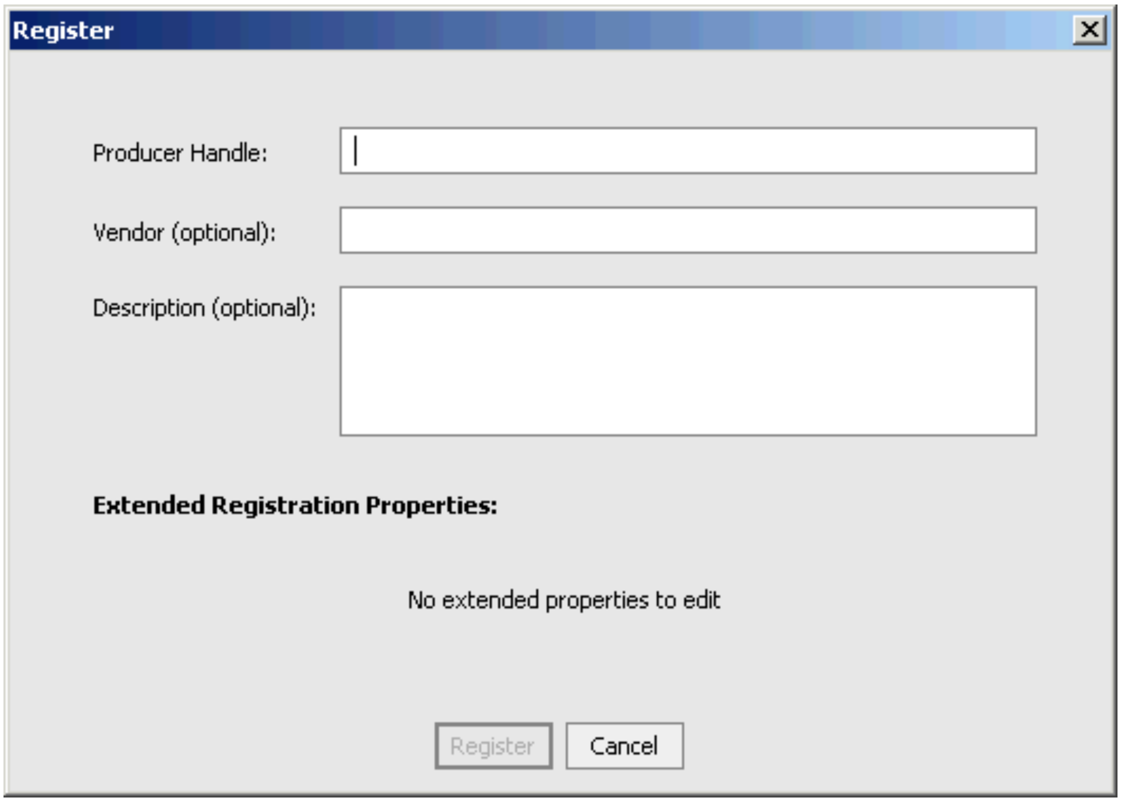
NOTE: In this example, the server is running, thus the ability to use localhost:7001 for the producer. Your needs might differ.

5. Click ***Register*** and do the following:

NOTE: If registration is not required (***Requires Registration: False***) proceed to step 4.

    1. Click Register.

       The Register window appears:

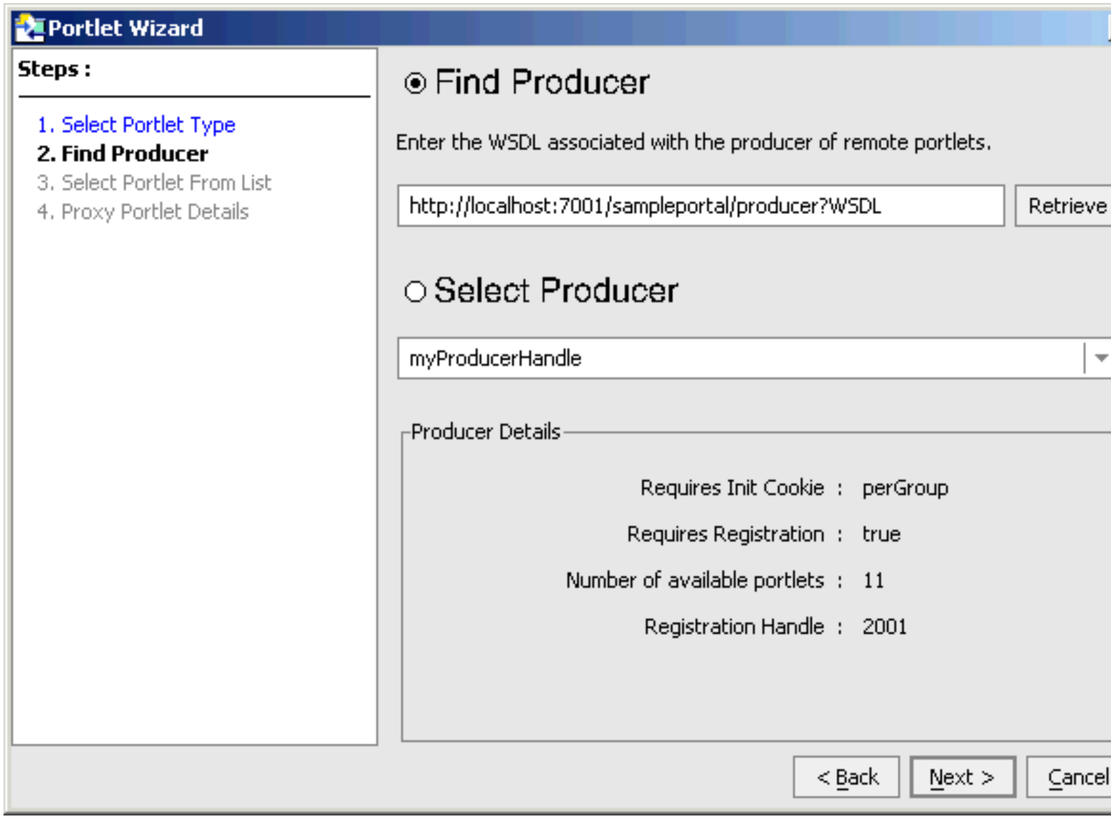


b. In Producer Handle, type a name with which you want to identify the producer (on subsequent uses, this name will appear in the ***Select Producer*** drop−down list, making it easy to use them again)**.** This value must be unique for each producer added to a remote project.

- Entering the Producer Handle will activate the ***Registration*** button.
- Optionally, you can enter the name of the ***Vendor*** and a ***Description*** of the resource.
- Depending upon the producer you selected, you might also see a list of Extended Registration Properties. These properties are set by the producer and some might require you to provide an accepted value before registration can be completed. Contact the producer for details if you are required to provide any extended properties.

3. Click ***Register***

   The Find Producer window reappears with the producer handle displayed in the ***Select Producer*** field.


Portlet Wizard

Steps :

1. Select Portlet Type
2. Find Producer
3. Select Portlet From List
4. Proxy Portlet Details

● Find Producer

Enter the WSDL associated with the producer of remote portlets.

http://localhost:7001/sampleportal/producer?WSDL

○ Select Producer

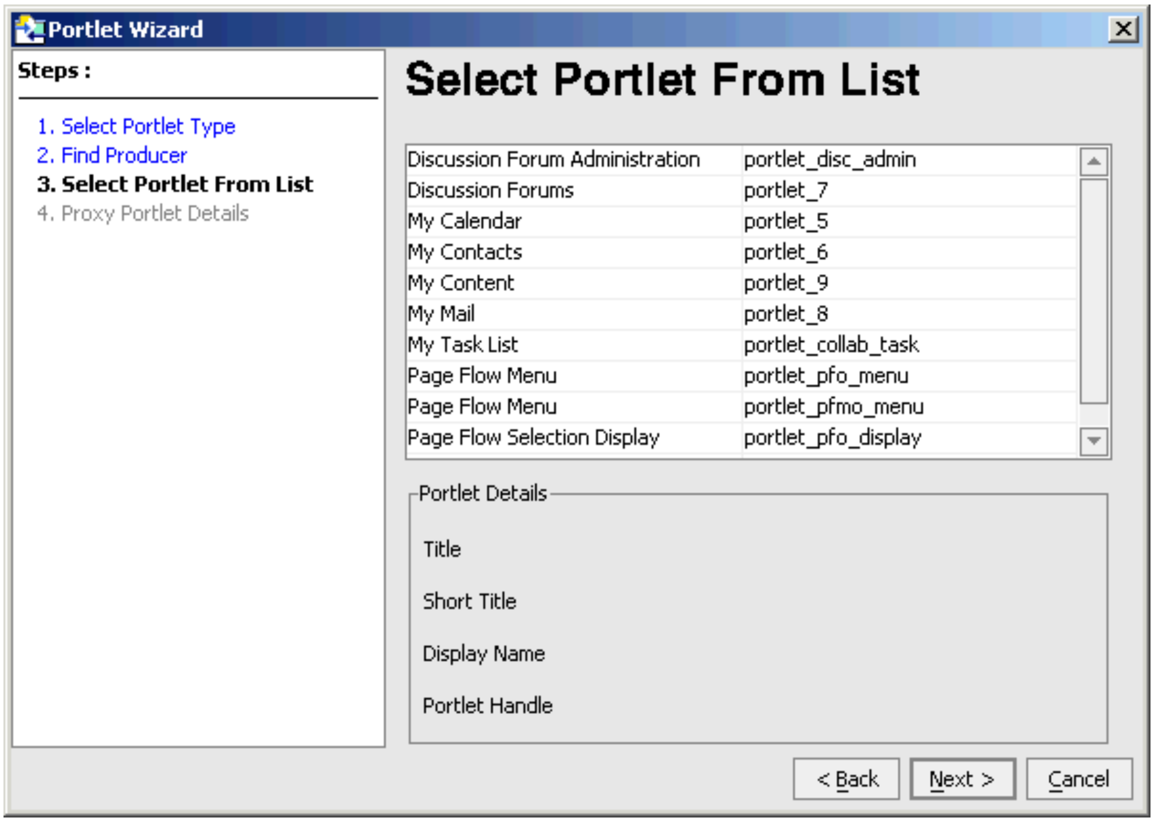
myProducerHandle

Producer Details

Requires Init Cookie :	perGroup
Requires Registration :	true
Number of available portlets :	11
Registration Handle :	2001

6. Click Next

The Select Portlet from List window appears:


Portlet Wizard

Steps :

1. Select Portlet Type
2. Find Producer
3. Select Portlet From List
4. Proxy Portlet Details

Select Portlet From List

Discussion Forum Administration	portlet_disc_admin
Discussion Forums	portlet_7
My Calendar	portlet_5
My Contacts	portlet_6
My Content	portlet_9
My Mail	portlet_8
My Task List	portlet_collab_task
Page Flow Menu	portlet_pfo_menu
Page Flow Menu	portlet_pfmo_menu
Page Flow Selection Display	portlet_pfo_display

Portlet Details

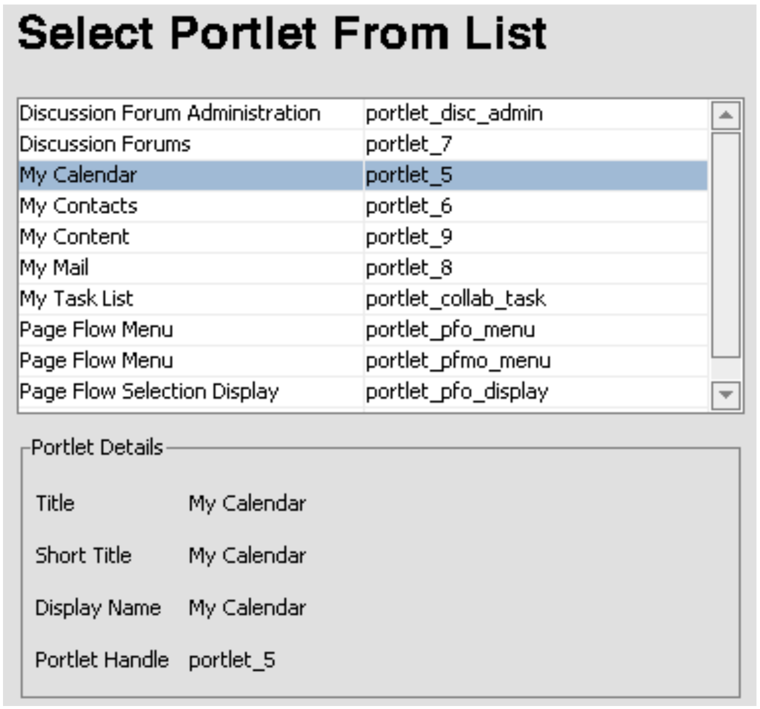
Title

Short Title

Display Name

Portlet Handle

7. Select the portlet you want to use. Note that when selected, details about that portlet appear in the Portlet Details panel:



Select Portlet From List

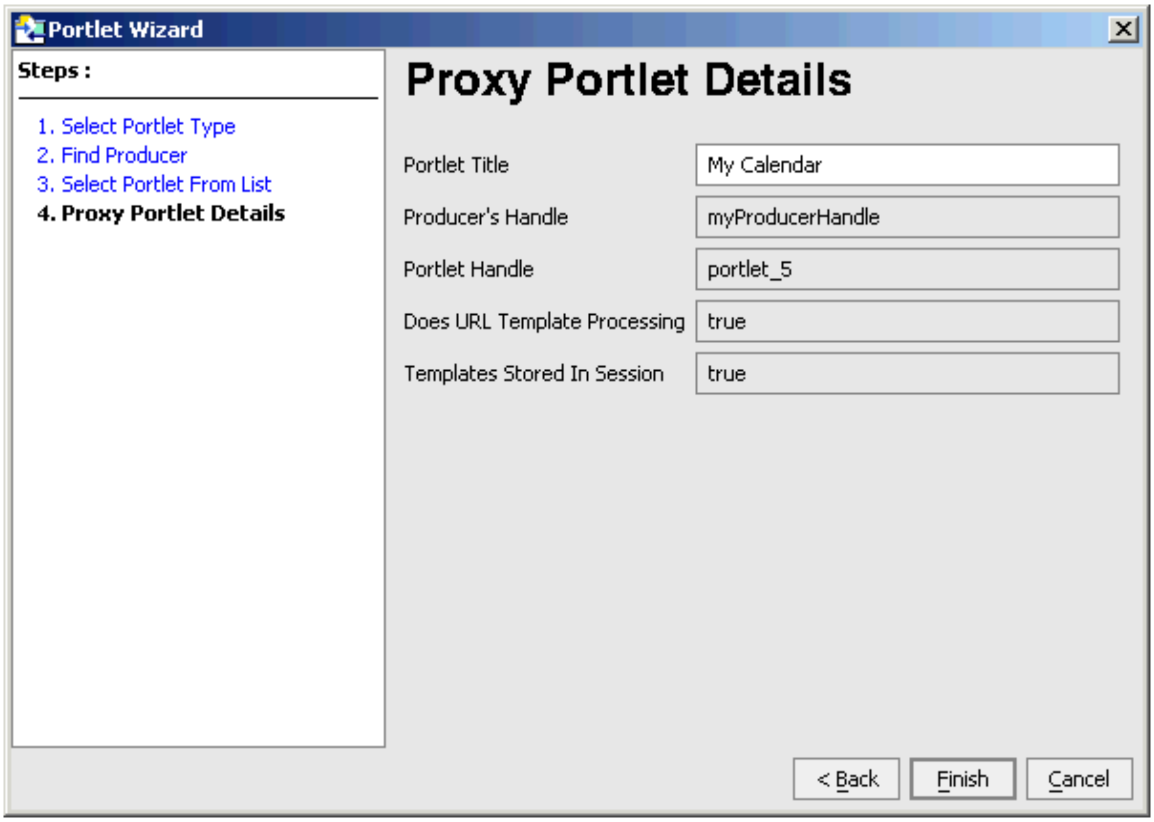
Discussion Forum Administration	portlet_disc_admin
Discussion Forums	portlet_7
My Calendar	portlet_5
My Contacts	portlet_6
My Content	portlet_9
My Mail	portlet_8
My Task List	portlet_collab_task
Page Flow Menu	portlet_pfo_menu
Page Flow Menu	portlet_pfmo_menu
Page Flow Selection Display	portlet_pfo_display

Portlet Details

Title	My Calendar
Short Title	My Calendar
Display Name	My Calendar
Portlet Handle	portlet_5

8. Click ***Next***.

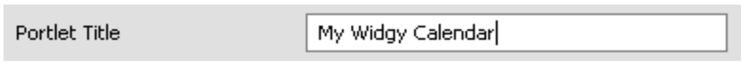
The Proxy Portlet Details window appears:



Proxy Portlet Details

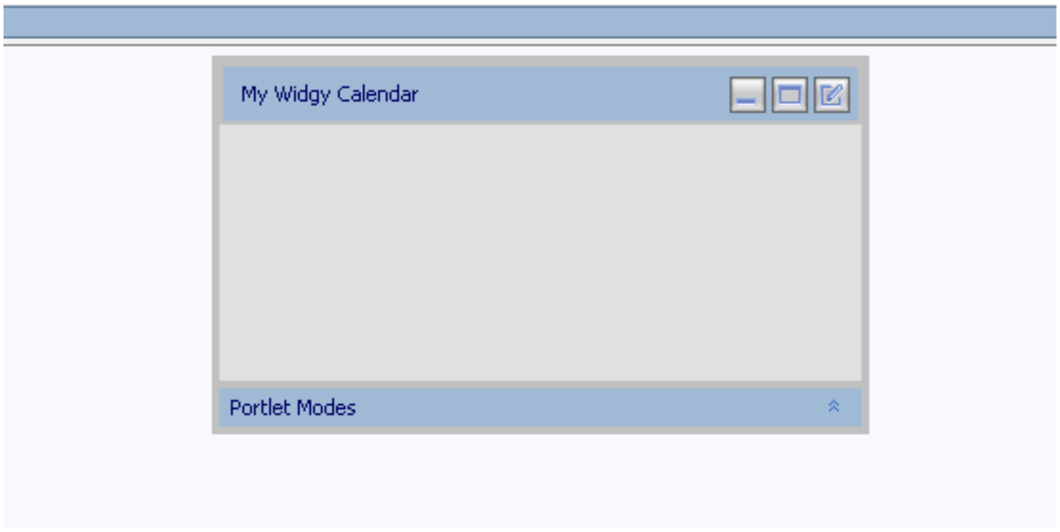
Portlet Title	My Calendar
Producer's Handle	myProducerHandle
Portlet Handle	portlet_5
Does URL Template Processing	true
Templates Stored In Session	true

9. The window displays pertinent information about the portlet. If you want, you can change the name in ***Portlet Title*** to something more meaninful to your portal application; for example:



Portlet Title My Widgy Calendar

The name will appear in the title bar of the portlet when it is rendered in the parent portal.
10. Click Finish.
11. The portlet will be created and appear as a placeholder in the IDE:



Note the new portlet name in the title bar.

Adding the Portlet to a Portal

To add the remote portlet to a portal, do the following (before you begin, ensure that WebLogic Workshop is in the xxxxxxx view and the data palatte is displayed):

1. In WebLogic Workshop, open the portal to which you want to add the portlet.
2. Select the remote portlet from the list of portlets in the data palatte and drag it onto the portal workspace.
3. Save the portal.

For more information on adding a portlet to a portal, please refer to Adding a Portlet to a Portal.

Viewing the Portlet

To test the new remote portlet, do the following:

1. Add the remote portlet to a portal by dragging it into the portal workspace.
2. Start WebLogic Server.
3. In Workshop, open the ***Portal*** menu and select ***Open Current Portal***.

   The portal will render in your browser and show the new remote portlet.

Modifying a Remote Portlet

You can modify the modes and states available in a remote portlet to the extent that the states and modes are editable in the producer being consumed.

Before You Begin

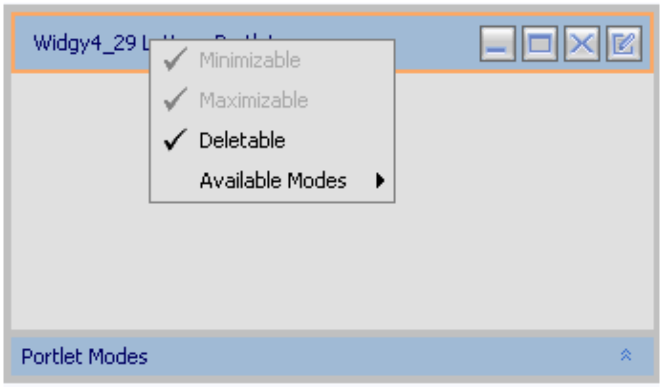
Before beginning these procedures, you should review the procedures outlined in Setting Portlet Modes and States.

Modifying Portlet States

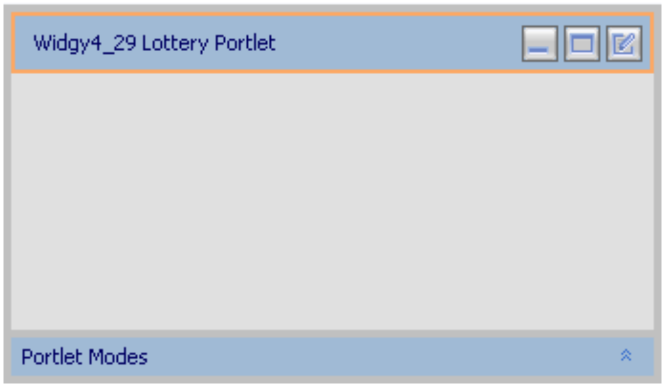
As with all portlets built with WeLogic Workshop, remote portlets can exist in one of three states: minimizable, maximizable, and deletable. You can select which of these states you want to include with the portlet by doing the following:

1. Right−click the portlet title bar.

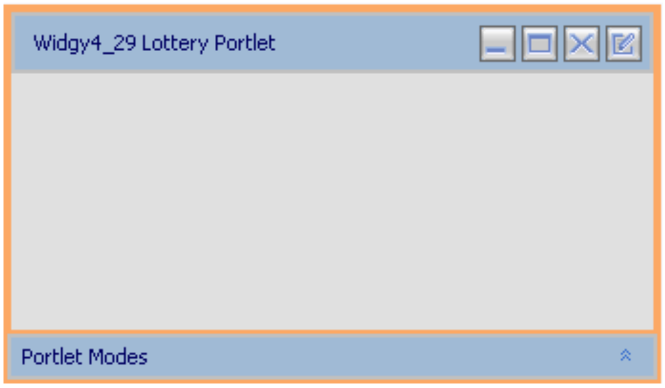
   A context menu showing applicable states appears.

   Menu options that are greyed−out, such as Minimizable and Maximizable in the preceding example, cannot be changed. They have been applied as uneditable in the producer.
2. Select the state you want to change. Selecting a state adds it to the portlet, while deselecting the state removes it from the portlet; for example, in the above image, all three states are selected. If you were to deselect ***Deletable***, the deletion button ☒ on the portlet would disappear:

If you were to then open the context menu and select ***Deletable***, the button would reappear:



3. Once you've made the necessary changes, save the portlet.

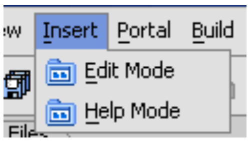
Modifying Portlet Modes

Remote Portlets provide the following modes:

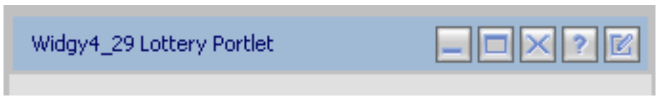
- Edit − Lets you specify a custom file that lets users modify the portlet's content when they click the Edit button.
- Help − Lets you specify a custom file that shows users help content for the portlet when they click the Help button.
- Float − Lets you display the portlet in a popup window when users click the Float button.

Making Modes Available

To make the Help and/or Edit mode available, open the ***Insert*** menu and select ***Edit Mode*** or ***Help Mode***, as necessary:



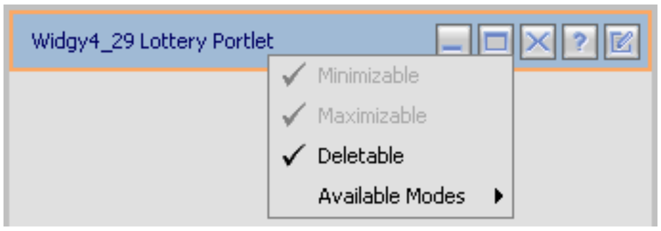
Buttons for the selected modes will appear in the title bar.



Adding and Deleting Modes

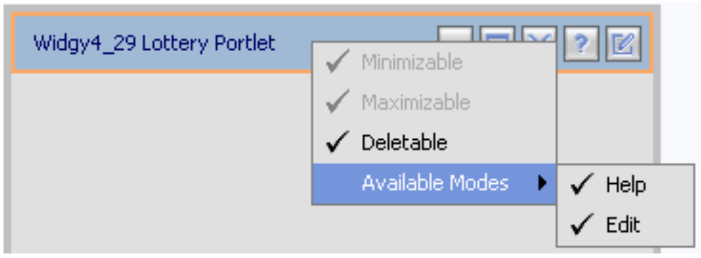
Using the ***Insert*** menu makes the modes available for use. You can remove them as necessary or, if they've been removed, add them by using the title bar context menu, as described in the following procedure.

1. Display the portlet as described in step 1 of the preceding procedure.
2. Right−click the title bar to display the states and modes context menu.

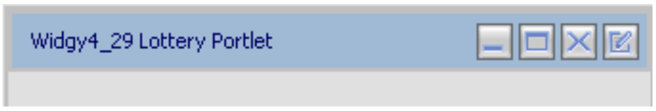


3. Select Available Modes.

   A submenu listing the available modes for this portlet, which were determined by the producer.



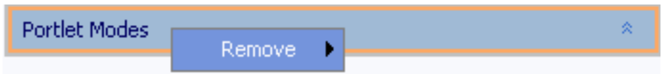
4. Select the mode you want to change; for example, if Help is selected and you deselect it, the Help button ? will disappear from the title bar:



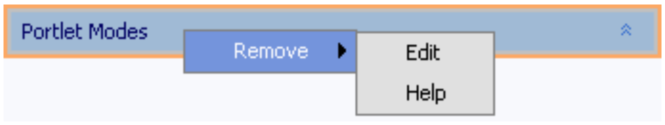
   Selecting a mode makes it available for use and will cause a representative button to appear on the title bar.
5. Once you've made the necessary changes, save the portlet.

An alternate method for removing the Edit or Help mode while still keeping it available is to do the following:

1. Right−click the Portlet Modes bar to display the ***Remove*** menu:



2. Highlight Remove to open the Modes submenu:



3. Select the mode you want to remove (available modes that aren't in use will appear greyed−out).

   The button for the selected mode will disappear from the title bar; for example, if you select Help, the Help button ? will no longer appear on the title bar.

Customizing a Remote Portlet

You can customize the look−and−feel of a remote portlet by adding a new CSS file or by applying a different theme. Since producers do not enforce a remote portlet's look−and−feel, this is an effective way to create uniform appearance in a portal comprised of remote portlets from multiple, disparate producers.

Adding a CSS

For information on adding a new CSS file to a consumer, please refer to Creating Look & Feel.

Adding a Theme

You can apply a theme to a remote portlet by using the Administration Portal. For more information, please refer to Assign a Theme to a Portal Element.

Disabling A Producer

By default, all portlets created using WebLogic Workshop 8.1 SP3 are available by way of WSRP for remote portals to consume; that is, they comply with the WSRP standard for producer and thus can be accessed and their content used by remote portlets. In most circumstances, you will leave the portlet's producer status in place; however, if you want to keep the content of the portlet private, you can disable this status by making small changes to the `WEB-INF/web.xml` file.

1. Open the producer project's `WEB-INF/web.xml` file.
2. Locate the `<servlet>` element and remove:

```
<servlet>
   <servlet-name>com.bea.wsrp.producer.WsrpServer</servlet-name>
   <servlet-class>com.bea.wsrp.producer.WsrpServer</servlet-class>
   <load-on-startup>2</load-on-startup>
</servlet>
```

3. Locate the `<servlet-mapping>` element and remove:

```
<servlet-mapping>
      <servlet-name>com.bea.wsrp.producer.WsrpServer</servlet-name>
      <url-pattern>/producer/*</url-pattern>
   </servlet-mapping>
```

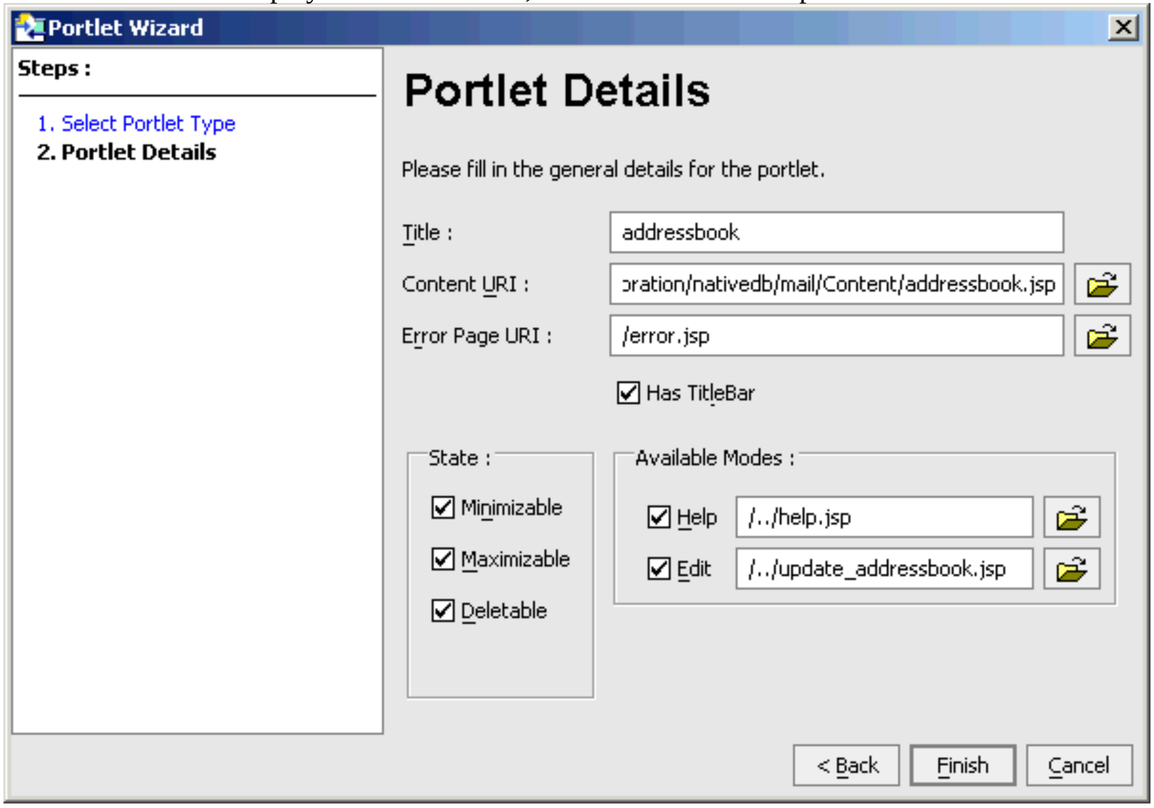
4. Redeploy the project.

Building JSP/HTML Portlets

You can use the Portlet Wizard to build a portlet that points to a JSP or HTML file for its content. These types of portlets can be simple to implement and deploy, and provide basic functionality without a lot of complexity. However, business logic and presentation layer can get combined in the JSPs; as the application grows, this often leads to escalating maintenance costs while trying to update the webapp and share code. This type of portlet is not well suited for advanced portlet navigation.

There are several ways to invoke the Portlet Wizard, as explained in the topic Creating Portlets. One way is to right−mouse click on your JSP file and select ***Generate Portlet...*** from the menu.

The Portlet Wizard displays a Details screen, as shown in this example:



Portlet Details

Please fill in the general details for the portlet.

Title : addressbook

Content URI : ɔration/nativedb/mail/Content/addressbook.jsp

Error Page URI : /error.jsp

☑ Has TitleBar

State :

☑ Minimizable

☑ Maximizable

☑ Deletable

Available Modes :

☑ Help /../help.jsp

☑ Edit /../update_addressbook.jsp

On this wizard dialog, the values for the Title and the Content URI (location of the JSP) are probably already filled in for you. You can specify additional options, such as whether the portlet should have Help and Edit icons. If you want those features on your portlet, specify the path to the JSP page that will provide the Help and Edit functions.

When you are ready, click the Finish button. A <portlet−name>.portlet file will be created for you, by default in the same directory as the content file.

Related Topics

Creating Portlets

Building Java Portlets

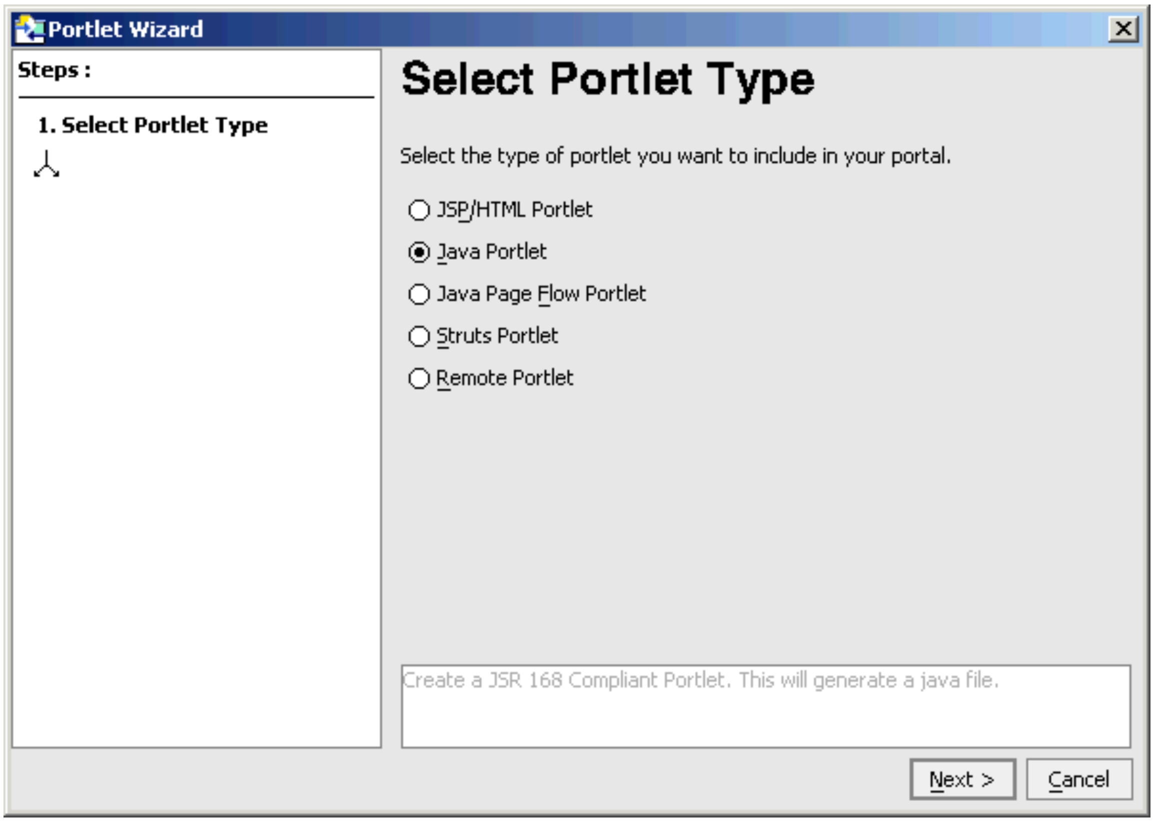
JSR 168 (Java Portlet) is a Java specification that aims at establishing portability between portlets and portals. One of the main goals of the specification is to define a set of standard Java APIs for portal and portlet vendors. These APIs will cover areas such as presentation, aggregation, security, and portlet lifecycle.

Java Portlets are intended for software companies and other enterprises that are concerned with portability across multiple portlet containers. For information about the emerging JSR 168 work, see the article Developing JSR 168 Portlets with WebLogic Portal 8.1 on the BEA dev2dev site.

In the WebLogic Workshop IDE, you can use the Portlet Wizard to create a new Java Portlet. For example, in the /portalApp/sampleportal web project, there is a /portlets folder. In the IDE Application pane, right−mouse click on the portlets folder and select ***New*** > ***Folder...***

Give your new folder a name; for example, aJavaPortlet. Then right click on the aJavaPortlet folder and select ***New*** > ***Portlet...***

This will invoke the Portlet Wizard. Name your portlet; for example, helloWorld.portlet. Then click the Create button. The Portlet Wizard displays its first screen, on which we have already selected the Java Portlet type from the available options:



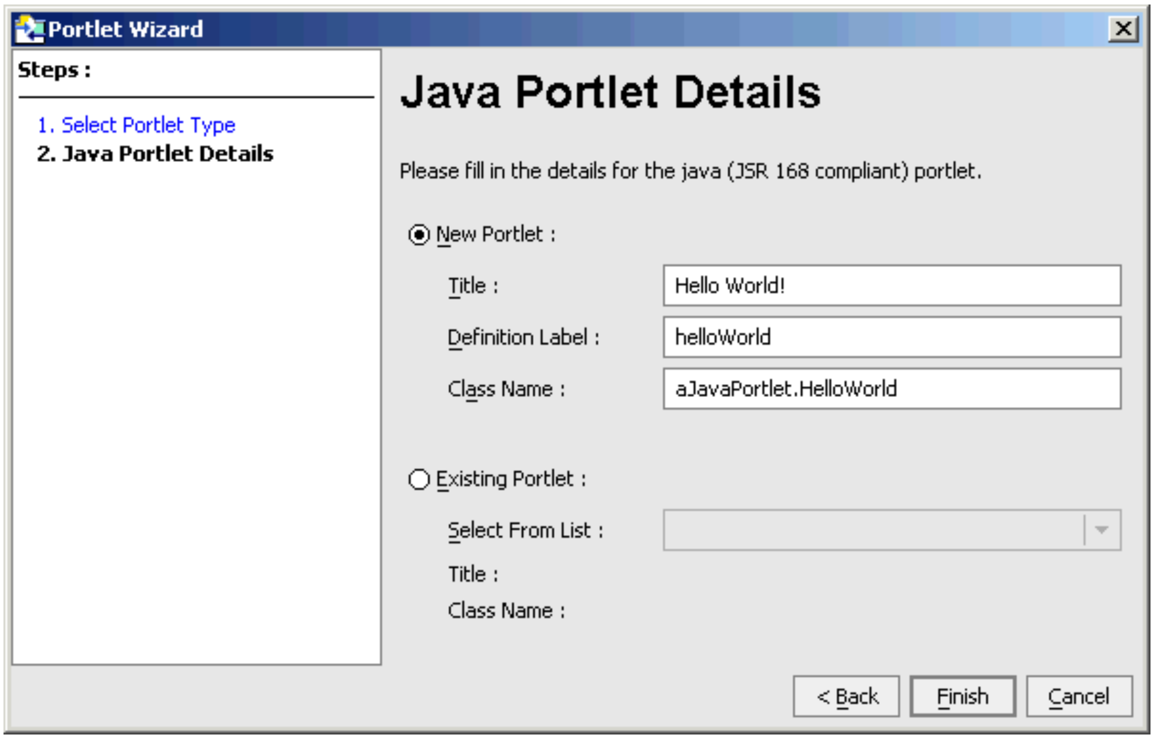
Select Portlet Type

Select the type of portlet you want to include in your portal.

- ○ JSP/HTML Portlet
- [✓ Java Portlet]
- ○ Java Page Flow Portlet
- ○ Struts Portlet
- ○ Remote Portlet

Create a JSR 168 Compliant Portlet. This will generate a java file.

Click the Next button. The Portlet Wizard displays this screen, on which we have already filled in three values:



Steps :

1. Select Portlet Type
2. **Java Portlet Details**

Java Portlet Details

Please fill in the details for the java (JSR 168 compliant) portlet.

[● New Portlet :]

Title :	Hello World!
Definition Label :	helloWorld
Class Name :	aJavaPortlet.HelloWorld

○ Existing Portlet :

Select From List :	
Title :	
Class Name :	

Before we describe the properties on the Wizard dialog, note that there is a separate deployment descriptor for Java Portlets. The file is /WEB−INF/portlet.xml. In addition, there is a Portal−specific deployment descriptor, /WEB−INF/weblogic−portlet.xml, to inject some additional features.

Here is an example of how entries may look in portlet.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app version="1.0"
   xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.co

    <portlet>
        <description>Description goes here</description>
        <portlet-name>helloWorld</portlet-name>
        <portlet-class>aJavaPortlet.HelloWorld</portlet-class>
        <portlet-info><title>Hello World!</title></portlet-info>
    </portlet>
</portlet-app>
```

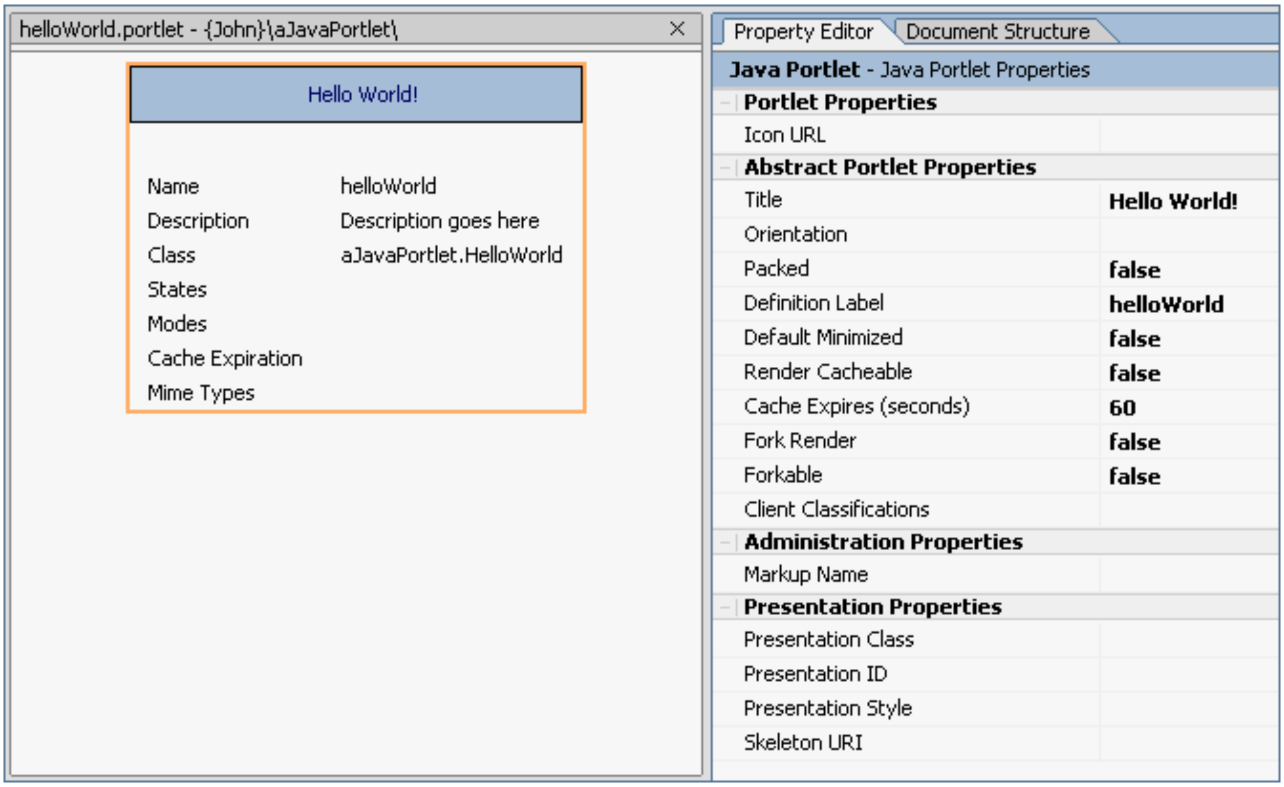
For the properties on the Portlet Wizard dialog:

- Title: This is default title of the portlet, which maps to the <title> element in portlet.xml.
- Definition Label: This is similar to a definition label for any portlet. However this also maps to the name of the portlet in the deployment descriptor; in the simple example above, the <portlet−name> element.
- Class Name: This maps to the <portlet−class> element. After the Wizard runs, in this example a HelloWorld.java file will be created in /WEB−INF/src/aJavaPortlet.

  Note: Another option for developers is to generate a Java Portlet's *.portlet file based on pre−existing classes.

Based on these values, the Wizard creates a .portlet file, and adds an entry to /WEB−INF/portlet.xml. All these fields are required to create a Java portlet.

Enter a title, a definition label, and valid class name for your Java Portlet. Then click the Finish button. WebLogic Workshop displays the newly created portlet and its current properties, as shown here:



Hello World!

Name	helloWorld
Description	Description goes here
Class	aJavaPortlet.HelloWorld
States	
Modes	
Cache Expiration	
Mime Types	

Java Portlet - Java Portlet Properties

Portlet Properties	
Icon URL	
Abstract Portlet Properties	
Title	**Hello World!**
Orientation	
Packed	**false**
Definition Label	**helloWorld**
Default Minimized	**false**
Render Cacheable	**false**
Cache Expires (seconds)	**60**
Fork Render	**false**
Forkable	**false**
Client Classifications	
Administration Properties	
Markup Name	
Presentation Properties	
Presentation Class	
Presentation ID	
Presentation Style	
Skeleton URI	

You can then modify the Java Portlet by changing the properties on the Property Editor pane, and by editing the generated Java class. In this simple example, the initial HelloWorld.java file contains:

```
package aJavaPortlet;

import java.io.IOException;
import javax.portlet.PortletException;
import javax.portlet.GenericPortlet;
import javax.portlet.RenderResponse;
import javax.portlet.RenderRequest;

/**
 * <p>A simple hello world portlet.</p>
 */
 public class HelloWorld extends GenericPortlet

{

     public void doView(RenderRequest request, RenderResponse response) throws PortletException
     {
          response.setContentType("text/html");
          response.getWriter().write("<p>Hello World</p>");
     }

}
```

Related Topics

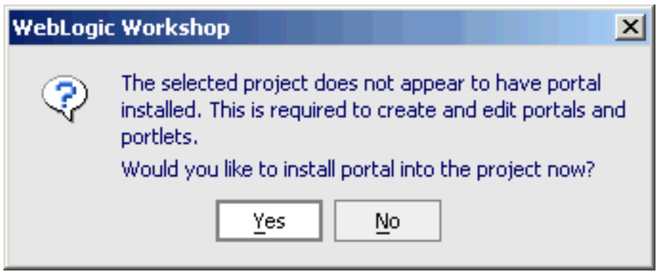
Creating Portlets

Building Java Page Flow Portlets

You can use the Portlet Wizard to built a portlet that uses Java Page Flows to retrieve its content. Java Page Flows allow you to separate the user interface code from navigation control and other business logic. Page Flows provide the ability to model both simple and advanced portlet navigation. They allow you to leverage other resources such as Java Controls and Web Services. Page Flows in WebLogic Workshop also provide a visual IDE environment to build rich applications based on Struts. The advanced page flow features are not necessary for static or simple, one−view portlets.

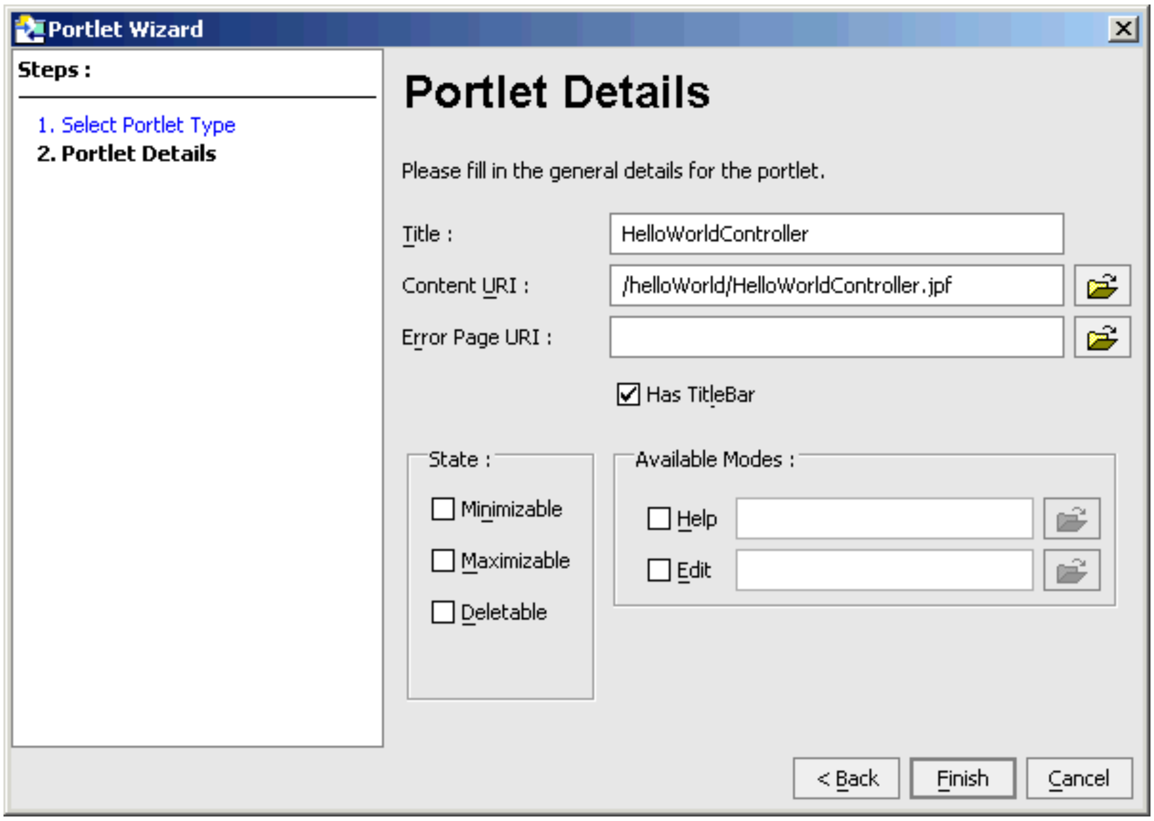
To invoke the Portlet Wizard, navigate in the IDE Application pane to the folder that contains the page flow. Open the folder, select the <page−flow>Controller.jpf class file, then right−mouse click and select ***Generate Portlet...*** from the menu. (This feature is available if you are running WebLogic Workshop Enterprise Edition, which includes the Portal functionality.)

If the page flow is in a web project that does not yet have the Portal libraries installed, WebLogic Workshop prompts you with the following message:



The selected project does not appear to have portal installed. This is required to create and edit portals and portlets.
Would you like to install portal into the project now?

If you receive this prompt, click the Yes button. When the portal libraries are installed, the Portlet Wizard displays this dialog:



Portlet Details

Please fill in the general details for the portlet.

Title : HelloWorldController

Content URI : /helloWorld/HelloWorldController.jpf

Error Page URI :

[✓] Has TitleBar

State :
- [ ] Minimizable
- [ ] Maximizable
- [ ] Deletable

Available Modes :
- [ ] Help
- [ ] Edit

On this wizard dialog, the values for the Title and the Content URI (location of the page flow JPF class) are probably already filled in for you. You can specify additional options, such as whether the portlet should have Help and Edit icons. If you want those features on your portlet, specify the path to the JSP page that will provide the Help and Edit functions.

When you are ready, click the Finish button. A <portlet−name>.portlet file will be created for you, by default in the same directory as the page flow.

Related Topics

Guide to Building Page Flows

Creating Portlets

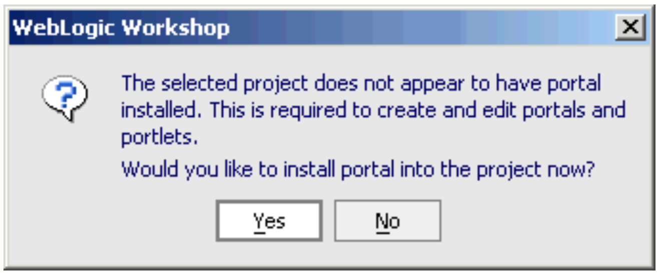
Building Struts Portlets

You can use the Portlet Wizard to generate a portlet based on a Struts Module.

Note: Before you can create a Struts portlet, you must first integrate your existing Struts application into your portal application. See Integrating Struts Applications.

In the WebLogic Workshop IDE, open the Portal application that contains the Struts module. Then create or navigate to the folder that will contain the <portlet−name>.portlet file you are about to generate.

From the IDE top−level menu, select ***File > New > Portlet.*** If the project does not yet have the Portal libraries installed, WebLogic Workshop prompts you with the following message:



The selected project does not appear to have portal installed. This is required to create and edit portals and portlets.

Would you like to install portal into the project now?

If you receive this prompt, click the Yes button.

When the portal libraries are installed, the Portlet Wizard prompts you with a series of screens. First, enter a name for the portlet, then click the Create button. The Portlet Wizard displays its Select Portlet Type screen. Select the Struts Portlet option, and click Next.

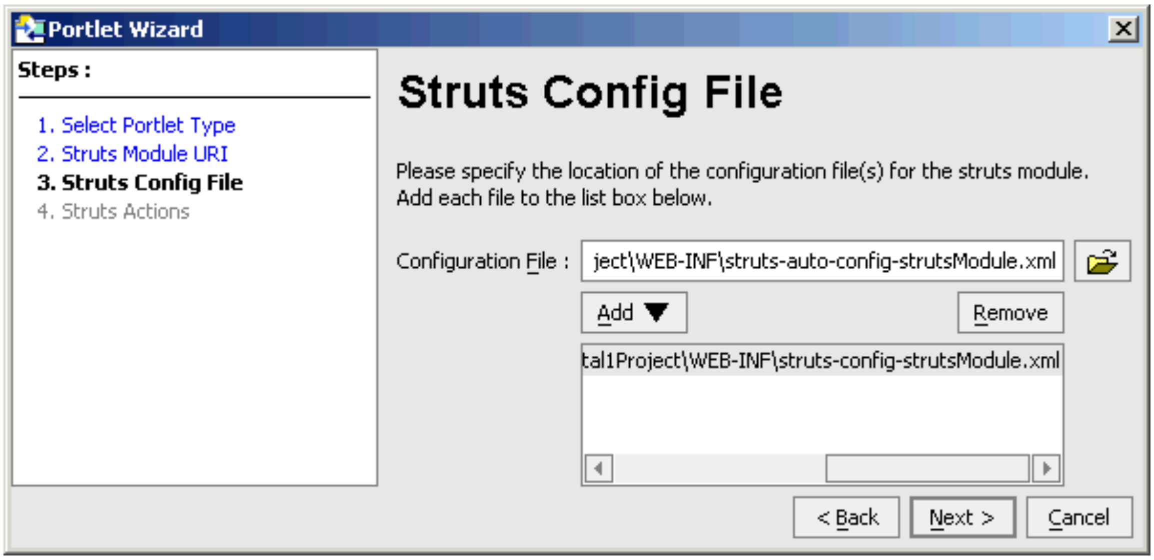
On the wizard's Struts Module URI screen, specify the folder that will contain the <portlet−name>.portlet file.

On the wizard's Struts Config File screen, identify or browse to the Struts module's XML configuration file. In this example, we had already copied into our Portal application several files that comprise a Struts module and related files. These files are part of a Struts Interop feature sample that is described in the topic Interoperating with Struts and Page Flows.

The following files that are under the <WEBLOGIC_HOME>/samples/workshop/SamplesApp/WebApp/ folder were copied into our Portal application:

/strutsModule/Jsp2.jsp
/WEB−INF/struts−config−strutsModule.xml
/WEB−INF/src/strutsModule/Struts*.java

We specified the struts−config−strutsModule.xml file on the following Portlet Wizard screen, and then clicked the Next button:



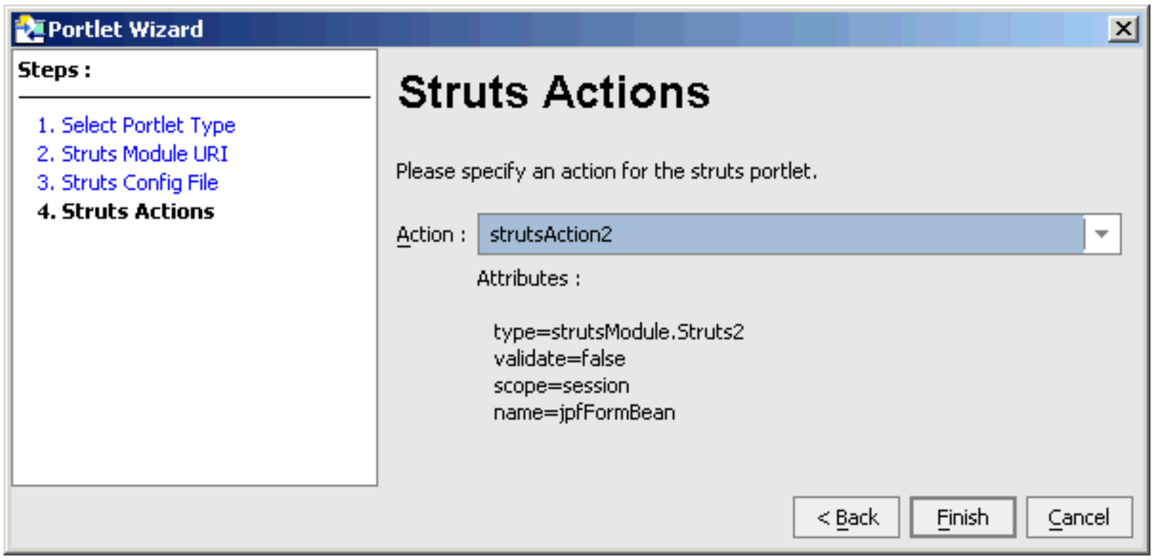
Struts Config File

Please specify the location of the configuration file(s) for the struts module.
Add each file to the list box below.

Configuration File : ject\WEB-INF\struts-auto-config-strutsModule.xml

tal1Project\WEB-INF\struts-config-strutsModule.xml

On the final screen, Struts Actions, we specified an action for the Struts Portlet:



Struts Actions

Please specify an action for the struts portlet.

Action : strutsAction2

Attributes :

type=strutsModule.Struts2
validate=false
scope=session
name=jpfFormBean

After clicking the Finish button, the wizard created our <portlet−name>.portlet file in the directory we specified in the Struts Module URI screen.

The Struts portlet can now be brought into the portal application.

Related Topics

Creating Portlets

Customizing Portlets

Adding a Portlet to a Portal

Creating a Web Service Portlet

To create a portlet that calls a Web Service, follow the steps in these topics:

1. Creating a Java Control from a Web Service.
2. Calling the Java control from a page flow, as explained in Tutorial: Page Flow.
3. Creating a portlet from the Java Page Flow.

Related Topics

Introduction to Web Services

How Do I Create a New Web Service with WebLogic Workshop?

How Do I: Create a Personalized Portlet?

Since portlets simply surface JSPs and Java Page Flows in a portlet window, any interaction management functionality you develop (Content Selectors, Placeholders, Campaigns, or personalized content provided inline in a JSP) can be easily surfaced in a portlet.

To Create a Personalized Portlet

1. Use the WebLogic Workshop Portal Extensions to develop interaction management functionality.
2. Using the WebLogic Workshop Portal Extensions Portal Designer, create portlets with your interaction management JSP.

After you create a portlet, you can use the Portal Designer to drag the portlet from the ***Data Palette*** window onto a page in your portal.

Related Topics

Creating a JSP Portlet

Building Portlets

Developing Personalized Applications

Adding a Portlet to a Portal

You can drag and drop portlets onto pages in the WebLogic Workshop Platform Edition Portal Designer.

1. In WebLogic Workshop Platform Edition, Create portlets or import sample portlets into your portal Web project.

   For instructions on importing sample portlets, see Portlet Samples.
2. Open the portal file and navigate to the page on which you want to put the portlet.
3. In the Data Palette window, drag the portlet you want into a placeholder on the page.
4. Select the portlet and use the Property Editor window to set the portlet properties.
5. Save the portal file.

The vertical or horizontal placement of portlets in a placeholder is determined by the selected layout for the page.

When you add a portlet to a page in the Portal Designer, a reference to that portlet is added to the .portal file. The .portal file is a template that can be used to create desktops in the WebLogic Administration Portal. When a portal administrator creates a desktop based on that .portal template, the portlet is added to the portal resource library where it can be added to pages in streaming desktops.

For details in adding a portlet to a portal desktop in the WebLogic Administration Portal, see Add a Portlet to a Page in the WebLogic Administration Portal Online Help on e−docs.

Removing and Deleting Portlets

- To remove a portlet from a portal (without deleting the portlet from your portal Web project), right−click the portlet in the Portal Designer and choose ***Remove***.
- To delete a portlet from your portal Web project, right−click the portlet in the Application window and choose ***Delete***.

Samples

The Portal Samples contain sample portlets that you can reuse in your own portals.

Related Topics

Creating Layouts

Building Portlets

Customizing Portlets

This section outlines the ways in which Portlets can be customized.

Setting Portlet Modes and States: You can set and modify the Edit, Help, and Float modes for a portlet; and the Minimizable, Maximizable, and Deletable states for a portlet.

Setting Portlet Height and Scrolling: You can control the height of portlets and determine whether their contents scroll.

Establishing Inter−Portlet Communication: Inter−portlet communication can be achieved with, or without, page flows and/or backing files.

Related Topics

Page Flow Portlets: Page Flows can be added to Portlets to add rich navigation and interaction.

Portal Controls: Personalization and Tracking are easy to add to a Portlet using the Portal Controls and the Portal EJB Controls included with WebLogic Portal Extensions.

Building Portlets: You can use pre−built portlets from the Portal Library, or create new portlets with the Portlet Wizard.

Using Portal JSP Tags: Portal tags provide easy access to rich personalization and tracking functionality, and require almost no coding. JSP tags can also be used within the JSPs that make up your portlets.

Setting Portlet Modes and States

You can add buttons to portlet titlebars that provide the following portlet modes and states:

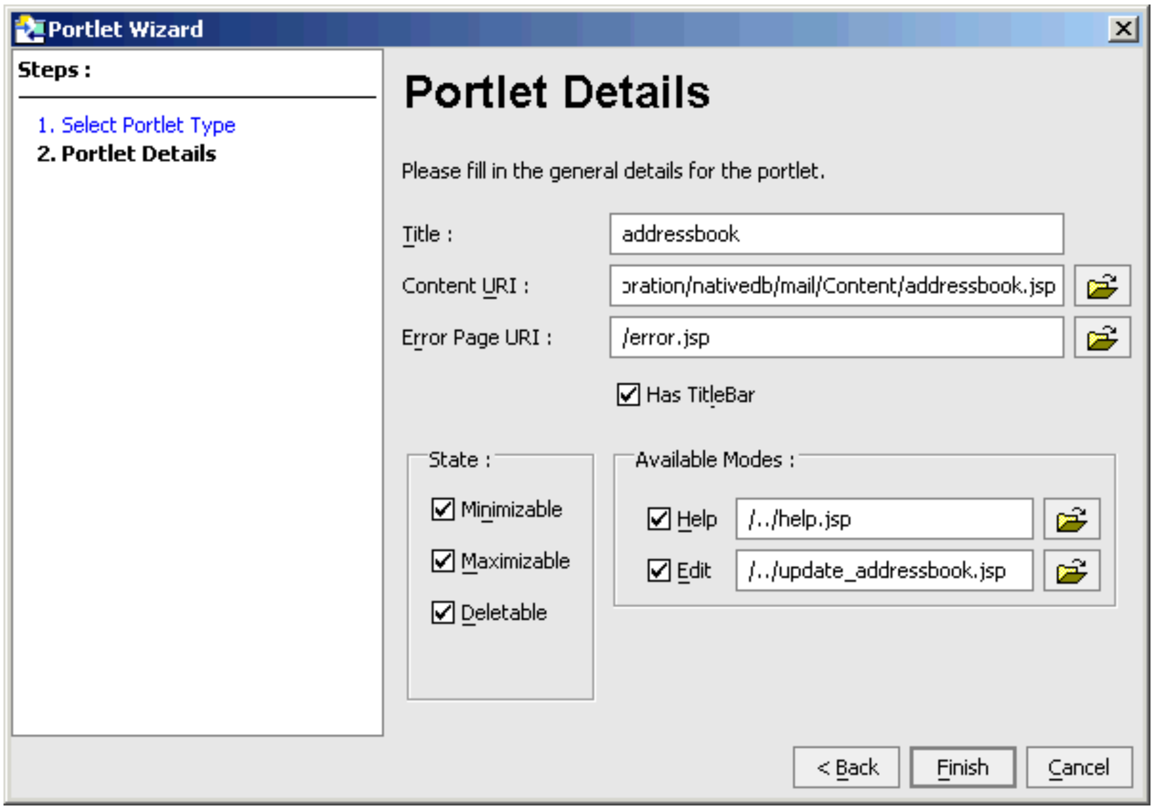
Modes

- Edit − Lets you specify a custom file that lets users modify the portlet's content when they click the Edit button.
- Help − Lets you specify a custom file that shows users help content for the portlet when they click the Help button.
- Float − Lets you display the portlet in a popup window when users click the Float button.

States

- Minimize − Collapses the portlet, leaving only the titlebar, when users click the Minimize button.
- Maximize − Makes the portlet take up the entire desktop area (not including the desktop header and footer) when users click the Maximize button.
- Delete − Removes the portlet from the users' desktops when they click the Delete button.

When you define a portlet with the Portlet Wizard, one of the options is to enable or disable the titlebar. For example:



Portlet Details

Please fill in the general details for the portlet.

Title : addressbook

Content URI : oration/nativedb/mail/Content/addressbook.jsp

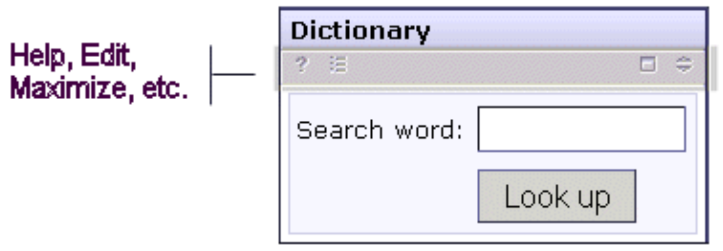
Error Page URI : /error.jsp

☑ Has TitleBar

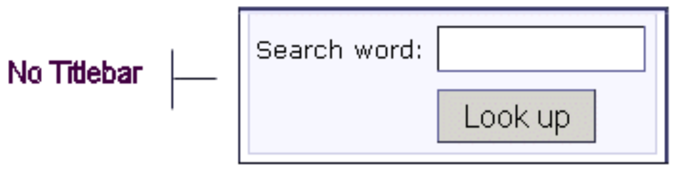
State :
☑ Minimizable
☑ Maximizable
☑ Deletable

Available Modes :
☑ Help /../help.jsp
☑ Edit /../update_addressbook.jsp

Here is an example of the Dictionary Portlet with a titlebar:



The Dictionary Portlet without a titlebar:



The Titlebar can be edited in the Portlet Wizard at creation time, or in the Portlet Designer by taking the following steps:

- From WebLogic Workshop, open the Portlet Designer by double−clicking on the *.portlet file.
- From the Properties Editor, change the values for the Titlebar attribute.
- Select ***File*** > ***Save*** to preserve your changes.

In the WebLogic Workshop IDE, you can add or modify a portlet's modes and states. The possible values are as follows:

Modes: Edit, Help, Float

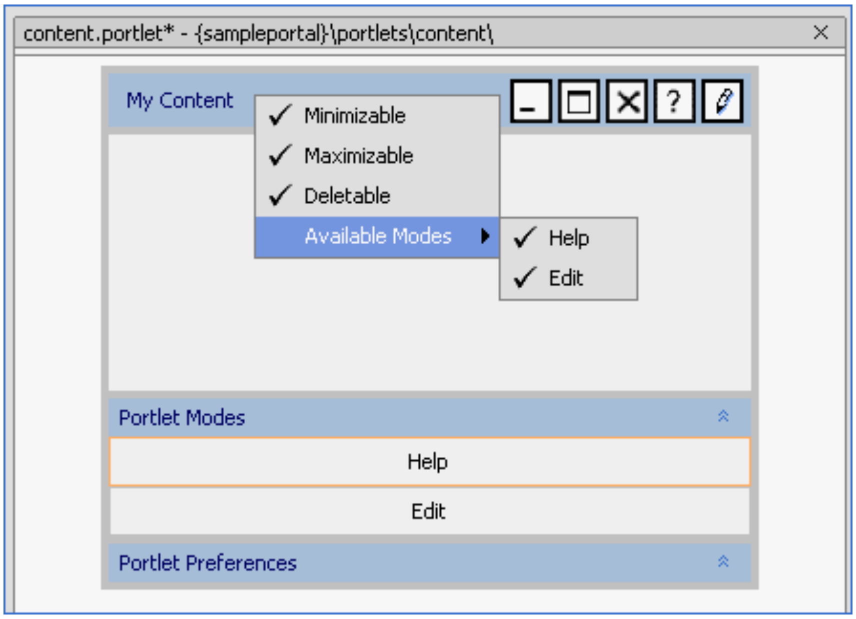
States: Minimizable, Maximizable, Deletable

About the Icons Used in Portlet Titlebars

The state and mode icons used in portlet titlebars are stored in a skin or theme /images directory. The portal framework reads the portal Web project's WEB−INF/netuix−config.xml file to determine which of these graphics to use for the portlet's different states and modes (minimize, maximize, help, edit).

Setting Portlet Modes

When you create a portlet mode, a set of Mode Properties appears in the Property Editor. To create a portlet mode, you can click ***Insert*** > ***Edit Mode*** and ***Insert*** > ***Help Mode*** from the IDE's top−level menu. Or you can right−mouse click on the portlet's titlebar, and then enable or disable the modes on the menu. For example:



The default icon for the Edit mode is the pencil, and the default icon for Help is the question mark.

By selecting the mode in the Portlet Modes rectangle as shown above, you can view the mode's details in the Property Editor pane. In those property sheets, you will specify the URI for your Help JSP that gets called when a user clicks the Help icon, and the Edit JSP that can be used to modify characteristics about the portlet.

For descriptions of these properties, see Mode Properties in the Portlet Properties topic.

Making Portlets Floatable

Float is a standard mode supported by WebLogic Portal and can be added to any portlet by inserting the float statement in the titlebar element of the .portlet file. For example:

```
<netuix:titlebar>
    <netuix:float/>
</netuix:titlebar>
```

Setting Portlet States

You can determine whether a portlet is to be minimizable, maximizable, or deletable. When using the portlet wizard, these settings are available on the Portlet Details screen. These settings can also be edited in the Portlet Designer by taking the following steps:

1. From WebLogic Workshop, open the Portlet Designer by double−clicking on the portlet.
2. From the Properties Editor, change the values for Mode and State attributes.
3. Select ***File*** > ***Save*** to preserve your changes.

Related Topics

Customizing Portlets

Setting Portlet Height and Scrolling

You can control the height of portlets and determine whether or not their contents scroll.

Portlet height and scrolling is controlled by the following CSS style attributes:

- {overflow−y:auto} − Enables vertical (y−axis) scrolling
- {overflow−x:auto} − Enables horizontal (x−axis) scrolling
- {overflow:auto} − Enables vertical and horizontal scrolling
- {height:200px} (where 200px is any valid HTML setting)

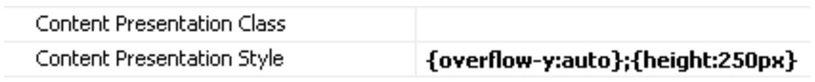
You can set these attributes on a portlet that is open in the Portlet Designer.

To set these properties:

1. Open a portlet in the the Portlet Designer.
2. In the Document Structure window, select ***Window Portlet***.
3. In the Property Editor window, under Portlet Properties, set one of the following properties:

- ***Content Presentation Style*** − Enter any of the previously listed attributes for this property. You can use overflow and height. Separate the values with a semicolon.
- ***Content Presentation Class*** − Enter the name of a style sheet class that contains the height or scrolling attributes you want to use.

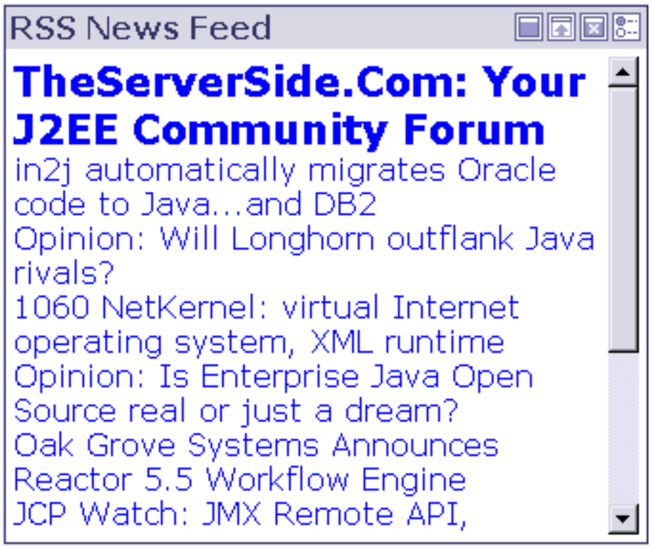
For example:



Content Presentation Class	
Content Presentation Style	**{overflow-y:auto};{height:250px}**

Where ***Content Presentation Style*** = ***{overflow−y:auto};{height:250px}***

The result looks like the following figure:


RSS News Feed
TheServerSide.Com: Your J2EE Community Forum
in2j automatically migrates Oracle code to Java...and DB2
Opinion: Will Longhorn outflank Java rivals?
1060 NetKernel: virtual Internet operating system, XML runtime
Opinion: Is Enterprise Java Open Source real or just a dream?
Oak Grove Systems Announces Reactor 5.5 Workflow Engine
JCP Watch: JMX Remote API,

Here is an example of using the Content Presentation Class property instead:

Content Presentation Class = ***portlet−scroll***

In this case, you would have to have the following style class defined in a CSS file:

```
.portlet−scroll
{
    overflow−y:auto;
    height:250px;
}
```

Making All Portlets Scroll

To provide portlet height or scrolling automatically, you can also modify window.jsp in each skeleton to incorporate a CSS style or class. For example, in the default skeleton's window.jsp, do one of the following:

- Replace the string bea−portal−window−content with the name of the scrolling portlet style class you created, such as portlet−scroll. (Be sure the CSS file containing the scrolling class is registered in the skin's skin.properties or skin_custom.properties files.)
- In the last line of the JSP, change

  style value="<%= window.getContentPresentationStyle() %>" />

  to

  style value="<%= window.getContentPresentationStyle() %>"
  defaultValue="{overflow−y:auto};{height:250px}" />

You could also simply modify the skin CSS style class. For example, in the skin's window.css file, define the bea−portal−window−content class like this:

```
.bea−portal−window−content
{

    margin: 4px;

    padding: 0px;

    scrollbar−base−color: #d8d8e5;
    overflow−y: auto;
    height: 250px;
}
```

Related Topics

Creating Skins and Skin Themes

Creating Skeleton and Skeleton Themes

How Do I: Establish Inter−Portlet Communication?

Inter−portlet communication can be achieved using backing files, but the page flow is the recommended mechanism. This topic explains how to use multiple page flows that react to browser events such as forms. To create page flow portlets that communicate with each other, take the following steps:

To Create Portlets that Share Messages

1. Inside the portal Web application, create a ***portlets*** directory.
2. Within this portlets directory, create two new page flows. (A separate directory will be automatically created for each one).
3. Create a portlet for each page flow and place these portlets on a portal.
4. In the Portal, click on the second portlet and set the listenTo attribute to the instanceLabel of the first portlet. (Open the Portal in Design View, click once on the second portlet and find the properties editor.)

   NOTE: The listenTo attribute is associated with the instanceLabel of the other portlet. You can change the definitionLabel without affecting the listenTo behavior.
5. In the .jpf file for the second portlet you can do one of two things.

   The first option is to use the same action method signature as in the first page flow. For example, this action definition is from the page flow controller for portlet 2:

   ```
    /**
     * @jpf:action
     * @jpf:forward name="listening" path="listening.jsp"
     */
     public Forward passString1(portlets.j1.j1Controller.Form form)
     {
         thePassedText = form.getText();
         return new Forward( "listening" );
     }
   ```

   Or you can add a handler for ActionNotFoundException handler. For example, in the page flow controller for portlet 2, make sure the @jpf:catch annotation is defined at the class level:

   ```
   /**
    * @jpf:controller
    * @jpf:catch type="ActionNotFoundException" method="doNothing"
    */
   ```

   And in the same page flow controller, that an action method such as the following is defined:

   ```
    /**
     * @jpf:exception−handler
     * @jpf:forward name="current" return−to="currentPage"
     */
     protected Forward doNothing( ActionNotFoundException e, String actionName, String messa
     {
         return new Forward( "current" );
     }
   ```
6. As you edit the page flow, you can verify the navigation by opening a viewer that will preview the pages without the portal.

7. Place the two portlets inside placeholders within your portal.
8. When the navigation and interaction works within the page flow, preview the portlet by navigating to ***YourWebappp/YourPortalName.portal***.

Related Topics

Tutorial: Using Page Flows Inside Portlets

Portal Key Concepts and Architecture

Developing Portal Applications

Handling Exceptions in Page Flows