



# BEA WebLogic Workshop™ Help

Version 8.1 SP2  
November 2003

# Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

## Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software–Restricted Rights Clause at FAR 52.227–19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227–7013, subparagraph (d) of the Commercial Computer Software—Licensing clause at NASA FAR supplement 16–52.227–86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

## Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E–Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

# Table of Contents

<b>Developing Portal Applications.....</b>	<b>1</b>
<b>Updating Portal Libraries with New Service Packs.....</b>	<b>2</b>
<b>Integrating Existing Applications into Portals.....</b>	<b>3</b>
<b>Integrating Web Applications.....</b>	<b>5</b>
<b>Integrating Java Page Flow Applications.....</b>	<b>6</b>
<b>Integrating Struts Applications.....</b>	<b>8</b>
<b>Overview of Content Management.....</b>	<b>11</b>
<b>Setting up Unified User Profiles.....</b>	<b>15</b>
<b>Adding WebLogic Portal Functionality to an Application.....</b>	<b>25</b>
<b>Enabling Desktop Selection.....</b>	<b>27</b>
<b>Adding Visitor Tools to Portals.....</b>	<b>29</b>
<b>Developing a New Portal Application.....</b>	<b>32</b>
<b>Creating a Portal Application and Portal Web Project.....</b>	<b>33</b>
<b>Building Different Types of Applications.....</b>	<b>36</b>
<b>Developing Web Applications.....</b>	<b>37</b>
<b>Building a Java Page Flow Application.....</b>	<b>38</b>
<b>Adding Portal Controls to Java Page Flows.....</b>	<b>39</b>
<b>Using Portal Controls.....</b>	<b>41</b>
<b>Portal Controls and Page Flows.....</b>	<b>43</b>
<b>Portal Controls, Properties, and Annotations.....</b>	<b>44</b>
<b>Portal Control Declaration.....</b>	<b>45</b>
<b>Portal Control Security.....</b>	<b>46</b>
<b>Click Content Event Control.....</b>	<b>47</b>

# Table of Contents

Create User Control.....	49
Display Content Event Control.....	51
Generic Event Control.....	52
Generic Tracking Control.....	54
Rule Event Control.....	55
Session Login Event Control.....	57
User Login Control.....	59
User Profile Control.....	61
User Registration Event Control.....	62
.....	63
User Information Query Control.....	64
Creating a New EJB Control.....	65
Entity Property Manager Control.....	66
Event Service Control.....	67
Group Manager Control.....	68
Group Profile Manager Control.....	69
Property Set Manager Control.....	70
Realm Configuration Control.....	71
User Manager Control.....	72
User Profile Control.....	73
Building a Struts Application.....	74
Building a Commerce Application.....	75
Adding Commerce Services to an Application.....	76
Enabling Catalog Management.....	78

# Table of Contents

<b>Creating Catalog Structure Properties.....</b>	<b>80</b>
<b>Creating Discounts.....</b>	<b>82</b>
<b>Creating Portals for Mobile Devices.....</b>	<b>84</b>
<b>Developing Personalized Applications.....</b>	<b>89</b>
<b>Using Portal JSP Tags.....</b>	<b>90</b>
<b>Overview of Content Management.....</b>	<b>93</b>
<b>Setting up Unified User Profiles.....</b>	<b>97</b>
<b>Enabling Desktop Selection.....</b>	<b>107</b>
<b>Adding Visitor Tools to Portals.....</b>	<b>109</b>
<b>Building Portlets.....</b>	<b>112</b>
<b>Using Portlets from the Portlet Library.....</b>	<b>113</b>
<b>Creating Portlets.....</b>	<b>115</b>
<b>Building JSP/HTML Portlets.....</b>	<b>117</b>
<b>Building Java Portlets.....</b>	<b>118</b>
<b>Building Java Page Flow Portlets.....</b>	<b>122</b>
<b>Building Struts Portlets.....</b>	<b>124</b>
<b>Creating a Web Service Portlet.....</b>	<b>126</b>
<b>How Do I: Create a Personalized Portlet?.....</b>	<b>127</b>
<b>Adding a Portlet to a Portal.....</b>	<b>128</b>
<b>Customizing Portlets.....</b>	<b>129</b>
<b>Setting Portlet Modes and States.....</b>	<b>130</b>
<b>Setting Portlet Height and Scrolling.....</b>	<b>133</b>
<b>How Do I: Establish Inter-Portlet Communication?.....</b>	<b>135</b>

# 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**.

If you choose not to update a Web project's libraries with the dialog box, you can update the Web project later by right-clicking the Web project directory in the Application window and choosing **Install --> Update Portal Libraries**.

6. 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**.

# 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 the [Guide to Building 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>
```

## Developing Portal Applications

</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

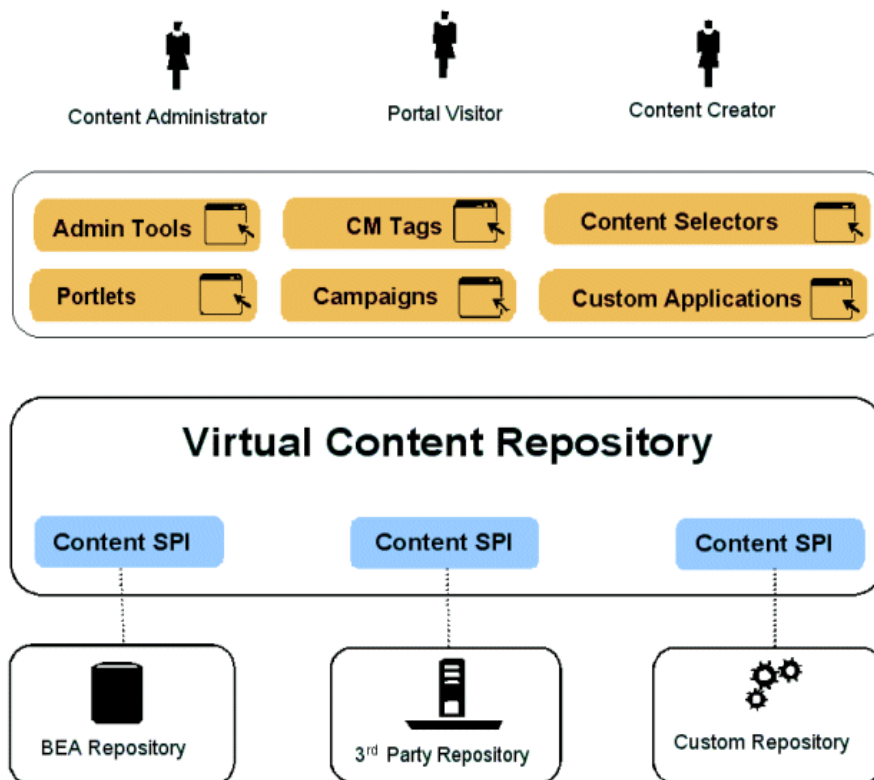
# Overview of Content Management

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 may 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](#)

[Designing Interaction Management](#)

[Creating Personalization Conditions](#)

[Personalizing Portal Applications](#)



# Setting up Unified User Profiles

A Unified User Profile provides the capability to leverage user data from external sources such as LDAP servers, legacy systems and databases. This allows for the use of a single profile to access user data from many different sources.

**Note:** If you are using WebLogic's default user store, unified user profile functionality is already configured.

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

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.

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.UnsupportedOperationException` 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 then 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 at <http://edocs.bea.com/wlp/docs81/javadoc/index.html>.)
- 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:

- 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. For more information, refer to Modifying the

Existing ProfileManager Deployment Configuration.

- 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. For more information, refer to Configuring and Deploying a New ProfileManager.

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 ldapprofile.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, as 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>
  </reference-descriptor>
</weblogic-enterprise-bean>
```

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-name>
    </ejb-reference-description>
  </reference-descriptor>
</weblogic-enterprise-bean>
```

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 `META-INF/application.xml` for your enterprise application 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`.

Your new Unified User Profile type is ready to use. You can use the WebLogic Administration Portal to create a user of type "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 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`.



## Developing Portal Applications

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>${APPNAME}.BEA_personalization.MyEntityPropertyManager</jndi-name>
    </ejb-reference-description>
  </reference-descriptor>
  <jndi-name>${APPNAME}.BEA_personalization.MyProfileManager</jndi-name>
```

## Developing Portal Applications

```
</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 `META-INF/application.xml` for your enterprise application 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`.

Your new Unified User Profile type is ready to use. You can use the WebLogic Administration Portal to create a user of type `"UUPEXampleUser,"` 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`.

**Note:** Tell your administrators that when they create a user in the WebLogic Administration Portal, they must select the new user type.

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

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. Deploy the ldaprofile.jar component within your application.

The LdapPropertyManager EJB in ldaprofile.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.

This is done by editing ejb-jar.xml to 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. -->

<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>
```

## Developing Portal Applications

```
<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 ldaprofile.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 ldaprofile.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. Start the server and deploy the application.
4. Start the WebLogic Server Administration Console for the active domain.

You can set a default profile type for each Web project by setting a context parameter in web.xml for DEFAULT\_USER\_PROFILE\_TYPE. For example:

```
<context-param>
```

## Developing Portal Applications

```
<param-name>DEFAULT_USER_PROFILE_TYPE</param-name>  
  <param-value>WLCS_Customer</param-value>  
</context-param>
```

To create a custom user profile property set, see [Creating User Profile Properties](#).

# 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 | My Yahoo! Enterprise Edition Portlets

WebLogic Portal provides many sample portlets that you can reuse in your portal applications, as well as support for incorporating My Yahoo! Enterprise Edition portlets.

## 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

Pre-built Java controls let portal developers quickly add Java code to portal applications for functionality such as user creation, authentication, and property set management.

## Developing Portal Applications

### 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

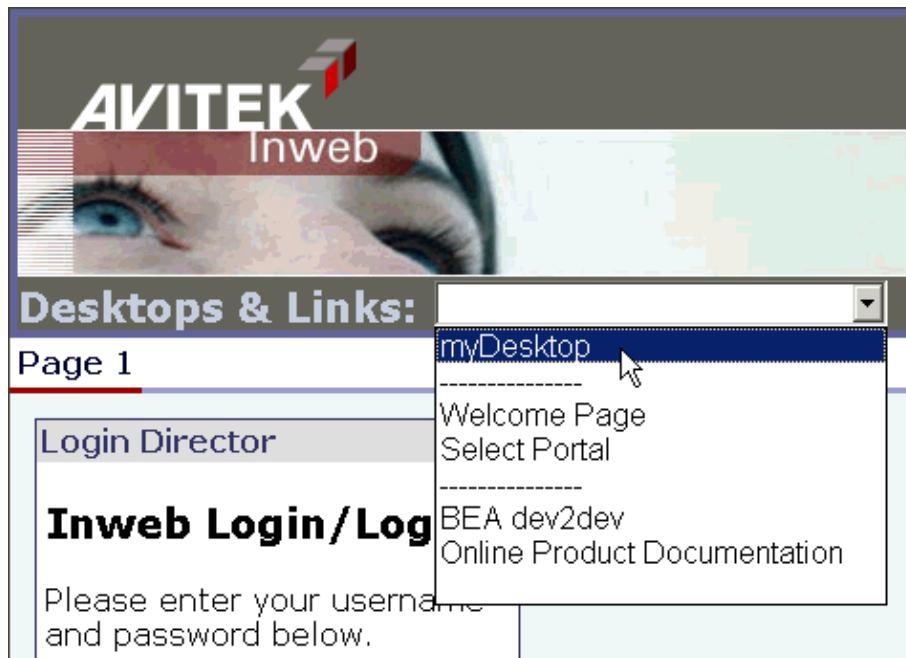
WebLogic Portal provides a full Java API for application development.

# 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:

<i>Import or copy this</i>	<i>to this directory</i> (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/



## Developing Portal Applications

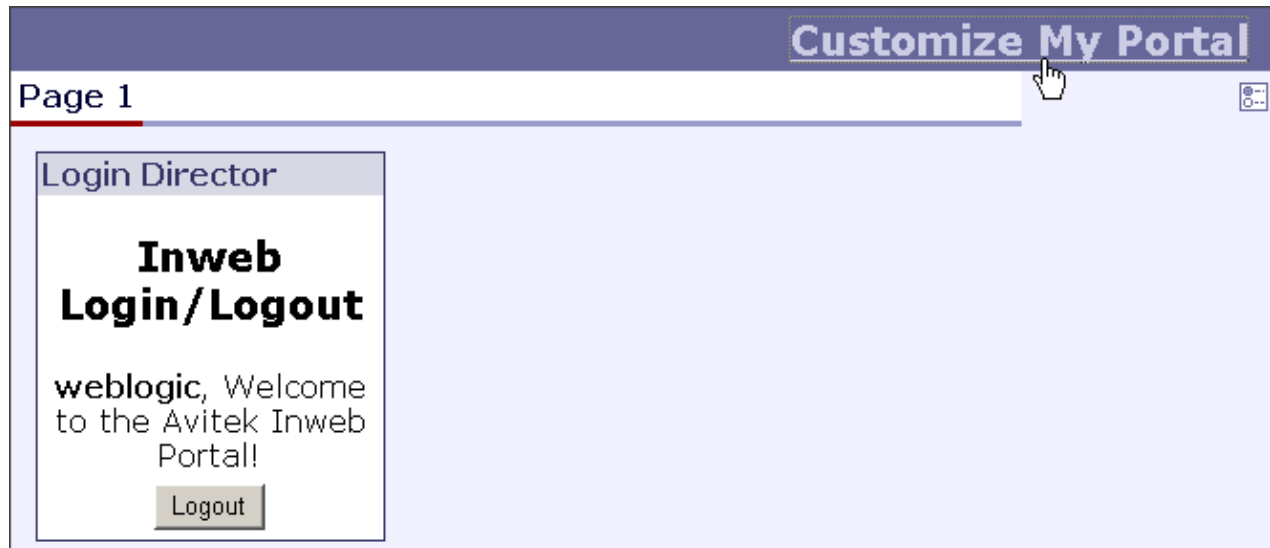
4. Open `<PORTAL_APP>/<project>/portlets/header/header.jsp` in WebLogic Workshop and replace the string `sampleportal` with the name of your project.
5. Create a shell and make `<PORTAL_APP>/<project>/portlets/header/header.jsp` the header content.
6. In a `.portal` file open in the Portal Designer,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

## Developing Portal Applications

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

# 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,

## Developing Portal Applications

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

## Developing Portal Applications

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 procedures 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 contain hundreds, if not thousands of pages. These pages should not only look great visually, but increasingly 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 changing 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

The development guide explains the key concepts involved in page flows, JavaServer Pages (JSPs), and WebLogic Workshop.

### Page Flow and JSP Reference

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

### Designing a Portal Interface

This topic discusses how portals can be used to give employees, partners, and customers a single point of access on your web site to applications and business information.

### Related Topics

#### Tutorial: Page Flow

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

#### 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

Portal Controls are used to build applications. They allow you to leverage portal functions more rapidly in application development.

Like the built-in controls included with WebLogic Workshop, Portal Controls enable you to insert well-implemented functionality into your portlets without doing lots of your own coding. Portal-specific controls provide reusable solutions to problems portal developers often face.

## Databinding: Making Development Easier

When a control is added to a Page Flow, the form editor makes the field types available to the form based on the action method with which the form is associated.

For detailed information, consult Using databinding in Page Flows.

## What Portal Controls Are Available?

Three types of controls are available to any instance of portal built from the portal template; Personalization Controls, Portal Event Controls and Portal EJB Controls:

### Personalization Controls

Personalization Controls are easy to insert into a Pageflow with minimal coding. WebLogic Portal Extensions ships with the following Personalization Controls:

- *Create User*
- *User Login*
- *User Profile*
- *User Information [Query]*

### Portal Event Controls

Portal Event Controls enable personalization and tracking functionality to be inserted into a Pageflow with minimal coding. WebLogic Portal Extensions ships with the following Portal Event Controls:

- *Click Content Event*
- *Display Content Event*
- *Generic Tracking*
- *Rule Event*
- *Session Login Event*
- *Generic Event*
- *User Registration Event*

### Portal EJB Controls

These portal controls are simply control representations of Personalization EJBs with Portal. Using an EJB Control instead of an EJB is a much easier way to implement Pageflows. WebLogic Portal Extensions ships with the following Portal EJB Controls:

- *Entity Property Manager*
- *Event Service*
- *Group Manager*
- *Group Profile Manager*
- *Property Set Manager*
- *Realm Configuration*
- *User Manager*
- *User Profile Manager*

Related Topics

Portal Controls and Page Flows

Using Portal Controls

WebLogic Controls

# Using Portal Controls

Three different types of controls are available for use in Portlets:

- Portal (Personalization) Controls
- Portal Event Controls
- Portal EJB Controls

For detailed instructions on using Portal Controls, consult the following: Tutorial: Creating a Login Portlet Using Portal Controls.

## When to Use a Portal Control

Use portal controls to expose tracking and personalization functions in multi-page portlets. For instance, to enable users to register, login and edit their properties, you would use the Portlet Wizard to create a Page Flow portlet, use the design view to insert a combination of the User Management controls with a form control, set a few properties and view the portlet immediately.

WebLogic Portal Extensions ships with the following Portal (Personalization) Controls:

- **Create User** – Exposes user creation, returns an instance of profileWrapper if successful, throws exception if user exists or creation fails for some other reason.
- **User Login** – This control supports authentication by simply associating this control with one of the User Interface controls. It allows a visitor to log in to a portal, indicates success and surfaces profile information.
- **User Profile** – This control exposes the user profile information to an action raised by a Page Flow. It is useful if you need to get all properties for a user, or only a subset of properties.
- **User Information [Query]** – This utility control returns the list of roles for a particular user, as well as the list of immediate parent groups.

## When to Use an Event Control

Portal Event Controls enable personalization and tracking functionality to be inserted into a Pageflow with minimal coding. The following Portal Event Controls are available:

- **Click Content Event** – Use this control to dispatch events invoked by clicking on portlet content.
- **Display Content Event** – Use this control to dispatch events involving content display.
- **Generic Tracking** – This control can expose the ability to configure, generate and dispatch a tracking event, (events that are generally persisted,) to the event service. Once an event object is created, you can set its attributes such as EventType and XML persistence, and then configure the event service to listen for your generic tracking event.
- **Rule Event** – This control dispatches a RuleEvent to the Portal Behavior Tracking System.
- **Session Login Event** – This control is used to dispatch a login event from within a Page Flow.
- **Generic Event** – This control is similar in appearance to the standard event control, except that it is configurable by the user.
- **User Registration Event** – This control dispatches a 'UserRegistrationEvent' to the Portal Behavior Tracking System.

## When to Use an Portal EJB Control

Portal EJB controls allow you to create quick, accurate and reusable access points to EJBs. Automatically create JSPs and databound forms speed the process of development dramatically.

These portal controls are simply control representations of Personalization EJBs with Portal. Using an EJB Control instead of an EJB is a much easier way to implement Pageflows. WebLogic Portal Extensions ships with the following Portal EJB Controls:

- **Entity Property Manager** – The remote interface for a session bean that manages persistence of ConfigurableEntity EJB's and their properties.
- **Event Service** – This control is placed in a Page Flow so that events can be passed to it. These events will be handled by registered listeners.
- **Group Manager** – This control is a remote interface for the GroupManager session bean, and requires special security considerations. Use this control to expose the ability to add a User to a Group, for example.
- **Group Profile Manager** – This control is a remote interface for the ProfileManager, a stateless session bean used to access profile values.
- **Property Set Manager** – PropertySetManager Control exposes a stateless session bean that provides access to property sets, through PropertySetRepositories.
- **Realm Configuration** – This public interface is for keeping personalization profile records in sync with the WLS realm.
- **User Manager** – A Remote Interface for the UserManager session bean.
- **User Profile Manager** – This control provides the business logic to retrieve and update user profile information.

Related Topics

Portal Controls and Page Flows

Using Portal Controls

# Portal Controls and Page Flows

Portal Controls are designed for use within Page Flows, where the Page Flow handles navigation logic and the Portal Control encapsulates tracking and personalization functionality.

Like Java Controls, Portal Controls make it easy to insert well-implemented tracking and personalization code inside portlets with a few simple actions. Portal Controls are used within Page Flows to expose resources and business logic in predefined ways.

## About Java Controls and Page Flows

Consult the following topics to familiarize yourself with how Java Controls are used with Page Flows.

[Getting Started with Java Controls](#)

[Guide to Building Page Flows](#)

[Working with Java Controls](#)

[Source Files for Java Controls](#)

[Using Controls in Business Processes](#)

[Using Portal Controls](#)

[Building Custom Java Controls](#)

[Using WebLogic Built-In Controls](#)



# Portal Controls, Properties, and Annotations

Some Controls have annotations, that is, properties that may be edited in WebLogic Workshop and applied to the Control at runtime. For example, an annotation may let the user describe the type of generic event to fire. The user types in the new event type using the Property Editor for the Generic Event Control.

## How Do Portal Controls Expose Properties?

The configuration XML file for Portal Controls has a well-defined schema that can declare:

- Default values for attributes
- Whether the attributes are required when the control is declared
- Where the attributes may be specified (eg, on method, on control declaration)

For example, the Click Content Event Control provides resourceType properties that specify whether the resource is one of the following types:

- GlobalRoleResource
- EnterpriseRoleResource
- WebappRoleResource
- HierarchyRoleResource
- LeafRoleResource

The documentation for each Portal Control indicates what properties – if any – are exposed. Also, when using WebLogic Workshop, inserting a control into a Pageflow will expose any properties in the Property Set Designer.

## Your Custom Controls

Properties can be defined by creating an annotation XML file that describes them. Then associate the file with the control source code through the JCS file's control-tags property. When using the control, setting its properties, the settings are saved as annotations in the code.

For step-by-step information on defining control properties, see [How Do I: Define Properties for a Java Control?](#)

## Related Topics

[Control Security](#)

[Using WebLogic Built-In Controls](#)

[How Do I: Establish Inter-portlet Communication?](#)

# 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're creating a control without using WebLogic Workshop, be sure to include this control declaration in this form.

## Related Topics

[Control Security](#)

[Using WebLogic Built-In Controls](#)

# Portal Control Security

Many Portal Controls are backed by EJBs; some of those EJBs have secured methods, meaning that any Control attempting to execute such a method would need to be in an authorized role.

For example, the User Profile Control is backed by the `getAllProfileName()` method of the `RealmConfiguration` EJB, which requires the caller to be in the role of "PortalSystemAdministrator" or "Admin".

For this reason, WebLogic Workshop does not support 'externally defined' roles. As a result, any security role definitions must be placed in `web.xml` and `weblogic.xml`.

## Related Topics

[Portal Control Declaration](#)

[Using WebLogic Built-In Controls](#)

[Using an EJB Control](#)

# Click Content Event Control

This control provides a simple way to dispatch events involving content display from within a Page Flow. After this control is added to a Page Flow, this dispatch action can be exposed in a portlet, and then a GUI element such as a Button can be used to invoke the dispatch action.

This control is used to handle the following two variables: Document Type and Document ID.

The Session and Request objects maybe obtained from a Page Flow as follows:

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

## Annotations

Each control can be configured with annotations, to parameterize the control as follows:

The configuration XML file has a well-defined schema that can declare:

Default values for attributes

Whether the attributes are required when the control is declared

Where the attributes may be specified (eg, on method, on control declaration)

The *resourceType* property specifies whether the resource is one of the following types:

- GlobalRoleResource
- EnterpriseRoleResource
- WebappRoleResource
- HierarchyRoleResource
- LeafRoleResource

## Security

The caller must be in the role of "PortalSystemAdministrator" or "Admin" to invoke method 'getAllGroupNames().

## Using the control

To place this control into a Page Flow, take the following steps:

1. Create a "basic" Page Flow, and click on the Flow View.
2. In the Data Palette on the **Controls** bar, click **Add** → **Portal Controls** → **Click Content Event Control**.
3. Enter a name for the new control, and click **Create**.
4. When the new control appears beneath the data palette, drag the dispatch() method into the Flow View for the new Page Flow.
5. In the Flow View, click on the **dispatch** action and then click **Edit Form Bean** in the Form Bean palette.
6. Use the Form Bean Editor to set the variable names in Source View. Save your work.

## Developing Portal Applications

7. In Flow View, double-click on Index.html and drag a button from the NetUI Palette into the Design View. The Button Design Wizard appears.
8. Type "Dispatch" in the Button Name field, set Type to Button, and Action to Dispatch. Click OK.

```
(this.getRequest(), form.docType(), form.docID());
```

```
public void setDocType(java.lang.String docType)
{
    this.docType = docType;
}
```

```
public java.lang.String getDocType()
{
    return this.docType;
}
```

```
public void setDocId(java.lang.String docId)
{
    this.docId = docId;
}
```

```
public java.lang.String getDocId()
{
    return this.docId;
}
```

### Related Topics

[Tutorial: Creating a Login Control Page Flow Using the Wizard](#)

[Overview: Working with Portal Control](#)

# Create User Control

This control is used by the portal interface components (such as the Form control) to create a user.

## Overview

Using the Form control to submit fields to this control from a Page Flow, you can create a new user from within a portlet. The results can be displayed using an instance of ProfileWrapper if user creation was successful, or by displaying an error message if it fails.

**NOTE:** This control requires the UserManager EJB to be deployed in the application. Because the p13n\_ejb.jar package contains this UserManagerEJB, it is always deployed as part of portal. To use it outside of the Portal, deploy the EJB to the application that has the WebApp which consumes the control.

## Annotations

The *resourceType* property specifies whether the resource is one of the following types:

- GlobalRoleResource
- EnterpriseRoleResource
- WebappRoleResource
- HierarchyRoleResource
- LeafRoleResource

## Security

The caller must be in the role of "PortalSystemAdministrator" or "Admin" to invoke method 'getAllGroupNames().

## Method Summary

com.bea.p13n.usermgmt.profile.ProfileWrapper	<code>createUser(java.lang.String username, java.lang.String password, javax.servlet.http.HttpServletRequest request)</code>  Create a new user and return an object representing the user's information.
--	---

## Method Detail

### createUser

```
public com.bea.p13n.usermgmt.profile.ProfileWrapper createUser(java.lang.String username,
```

java.lang.String password,  
javax.servlet.http.HttpServletRequest request)  
throws P13nControlException Create a new user and return an object representing the user's information.

**Parameters:**

username – The user's login name  
password – The user's password

**Returns:**

a ProfileWrapper representing the user's stored information.

**Throws:**

P13nControlException – if username or password is invalid, user already exists, or a remote exception occurs accessing the UserManager EJB.

# ***Display Content Event Control***

This control dispatches a 'DisplayContentEvent' to the Portal Behavior Tracking System.

The Session and Request objects maybe obtained from a Page Flow by: `HttpServletRequest request = this.getRequest();`

## **Related Topics**

[Tutorial: Creating a Login Control Page Flow Using the Wizard](#)

[Overview: Working with Portal Control](#)



# Generic Event Control

This control is similar in appearance to the standard event control, except that you can configure it using the Property Set editor. You may specify the event type, add event attribute, then create and dispatch the event.

The GenericEventControl has the following characteristics:

- It is not persisted
- It has a single editable property, the event type
- It may be used in JWS or Page Groups

## Overview

The eventType is set as a property on the control. Additional properties are the XML persistence information for this event. See the property editor descriptions for more information.

## Annotations

- **eventType** The type of event, eg "FooEvent", or "MyShoppingEvent".
- **xmlNamespace** XML namespace for the schema of the persisted event data, eg: "http://www.bea.com/servers/p13n/xsd/tracking/session-login/1.0.2".
- **xsdFile** File name of the XSD for the persisted event data schema, eg: "tracking-session-login-1\_0\_2.xsd".
- **schemaKeys** List of comma separated Strings that represent the keys of the schema, eg: "application, date, item"

## Method Summary

void	dispatch(javax.servlet.http.HttpServletRequest request) Dispatch a TrackingEvent.
------	--

## Method Detail

### dispatch

public void dispatch(javax.servlet.http.HttpServletRequest request)  
throws P13nControlException  
Dispatch a TrackingEvent. The type of event is specified on the control property.

#### Parameters:

request – The HttpServletRequest from the Page Group.

**Throws:**

P13nControlException – if request is null, or if errors encountered creating and dispatching event.

# Generic Tracking Control

This control is used to expose the configuration, generation and dispatch of behavior tracking events in a portlet.

## Overview

Control from which tracking events may be configured, generated, and dispatched to the event service. Once you have an Event object, you may set its attributes:

```
event.setAttribute(String theKey, Serializable theValue);
```

The eventType is set as a property on the control. Additional properties are the XML persistence information for this event. See the property editor descriptions for more information.

## Annotations

- **eventType** The type of event, eg "FooEvent", or "MyShoppingEvent".
- **xmlNamespace** XML namespace for the schema of the persisted event data, eg: "http://www.bea.com/servers/p13n/xsd/tracking/session-login/1.0.2".
- **xsdFile** File name of the XSD for the persisted event data schema, eg: "tracking-session-login-1\_0\_2.xsd".
- **schemaKeys** List of comma separated Strings that represent the keys of the schema, eg: "application, date, item"

## Method Summary

void	dispatch(javax.servlet.http.HttpServletRequest request) Dispatch a TrackingEvent.
------	--

## Method Detail

### dispatch

public void dispatch(javax.servlet.http.HttpServletRequest request)

throws P13nControlException Dispatch a TrackingEvent. The type of event is specified on the control property.

#### Parameters:

request – The HttpServletRequest from the Page Group.

#### Throws:

P13nControlException – if request is null, or if errors encountered creating and dispatching event.

# Rule Event Control

This control dispatches a RuleEvent to the Portal Behavior Tracking System.

## Overview

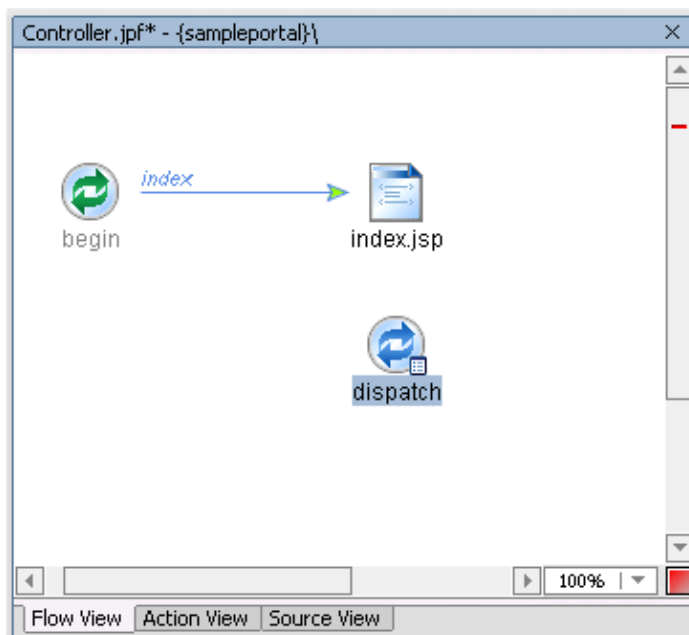
This control dispatches a 'SessionLoginEvent' to the Portal Behavior Tracking System. The Request object may be obtained from a Page Flow using the following code:

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

## Using the control

Place this control inside a Page Flow if you want to fire a session event from within a specific portlet. For example, to associate this control with a button on an index.html page, take the following steps:

1. Create a new Basic Page Flow.
2. From Flow View, select **Add→Portal Controls → Rule Event Control**.
3. From the Data Palette, drag the dispatch() method into the Flow View.



4. Double-click on the dispatch action and edit the variable names:
  - ◆ request – the HttpServletRequest object, cannot be null
  - ◆ rulesetName – The name of the rule set containing the rule
  - ◆ ruleName – The name of the rule
5. Double-click on **Index.html**, and from the Palette, drag a NetUI element onto the Design View. Type in a label name for the button in the Value field, select Button as the Type, and then select dispatch for the Action. Click **OK**.
6. The new GUI element will appear in the design view of the **Index.html** with the proper variables filled in.

### Related Topics

[Tutorial: Creating a Login Control Page Flow Using the Wizard](#)

[Overview: Working with Portal Control](#)

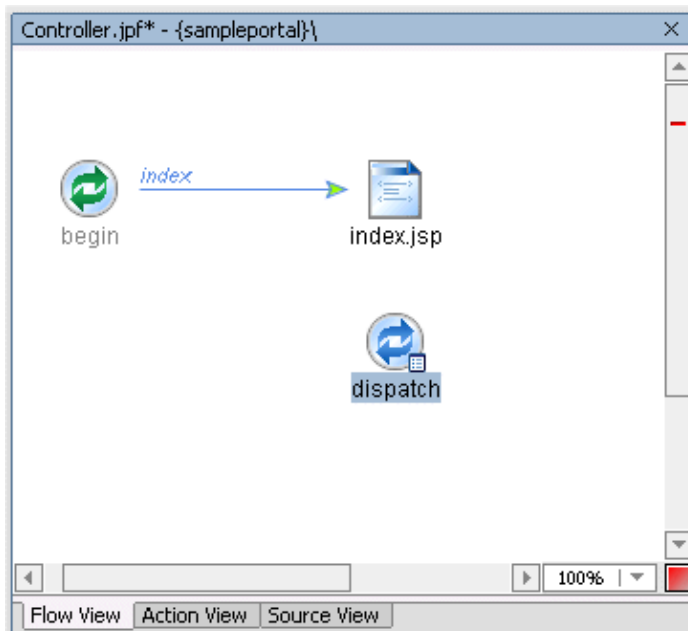
# Session Login Event Control

This control dispatches a 'SessionLoginEvent' to the Portal Behavior Tracking System.

## Using the control

Place this control inside a Page Flow if you want to fire a session event from within a specific portlet. For example, to associate this control with a button on an Index.html page, take the following steps:

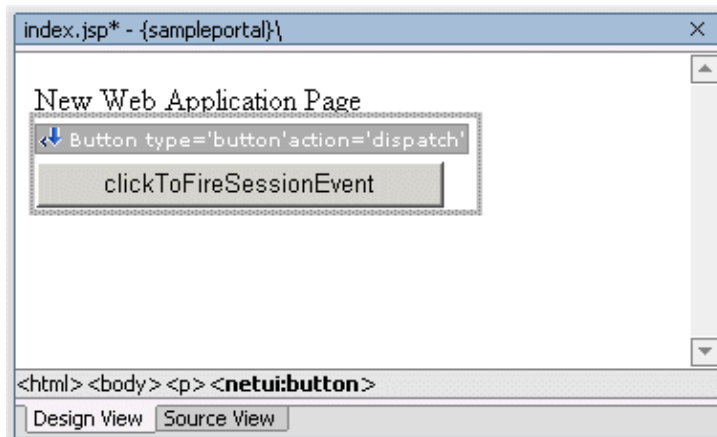
1. Create a new Basic Page Flow.
2. From Flow View, select **Add->Portal Controls -> Session Event Control**.
3. From the Data Palette, drag the dispatch() method into the Flow View.



4. Double-click on **Index.html**, and from the Palette, drag a NetUI Button onto the Design View. Type in a label name for the button in the Value field, select Button as the Type, and then select dispatch for the Action. Click **OK**.

## Developing Portal Applications

5. The new button will appear in the design view of the Index.html with the proper variables filled in.



### Related Topics

[Tutorial: Creating a Login Control Page Flow Using the Wizard](#)

[Overview: Working with Portal Control](#)

# User Login Control

**NOTE:** This control can be placed in a Page Flow using the Wizard.

## Overview

Placing this control on a Page Flow action allows a user to login using a portlet. The form component sends authentication information to the UserLogin control. If the login is successful, access to user profile information is granted. If not, an exception is thrown.

This control is used by the portal GUI components to send authentication information to the portal site. It allows a site visitor to log in to the portal, and gives indication as to whether the login is successful. The control also provides access to the user's profile information, if login successful. It does not register the user.

## Annotations

The *saveAnonymous* property specifies that properties and tracking data set by the user before registering should be saved after registration. If set to true, then any properties the user may have set during this Session before registering are added to the new user's properties. Defaults to true.

*public interface **UserLoginControl***  
*extends weblogic.jws.control.Control*

Method Summary	
<code>com.bea.p13n.usermgmt.profile.ProfileWrapper</code>	<b>login</b> ( <code>java.lang.String</code> username, <code>java.lang.String</code> password, <code>javax.servlet.http.HttpServletRequest</code> request) Log this user into the Portal application
<code>void</code>	<b>logout</b> ( <code>javax.servlet.http.HttpServletRequest</code> request) Log out current user

## Method Detail

### login

```
public com.bea.p13n.usermgmt.profile.ProfileWrapper login( java.lang.String username,
                                                           java.lang.String password,
                                                           javax.servlet.http.HttpServletRequest request)
    throws P13nControlException
```

Log this user into the Portal application

**Parameters:**



## Developing Portal Applications

username – The user's login name

password – The user's password

**Returns:**

a ProfileWrapper representing the user's stored information

**Throws:**

Pl3nControlException – if user cannot be logged in, or if there is a remote error on the UserManager EJB authenticating or retrieving user profile.

### logout

```
public void logout(javax.servlet.http.HttpServletRequest request)
```

Log out current user

### Related Topics

[Tutorial: Creating a Login Control Page Flow Using the Wizard](#)

[Adding Portal Controls to Java Page Flows](#)

# User Profile Control

This control exposes the user profile information to a Page Flow portlet. This is useful if you need to get all properties for a user, or only a subset of properties. Obviously, in order to obtain access to this information, a user would need to login with appropriate privileges. For this reason, a Page Flow that uses the User Profile Control would be a good candidate for a nested page flow.

**NOTE:** The User Profile Control does not allow the user to register. If the user does not have a valid account, an anonymous or tracked profile will be used.

## Overview

This control is backed by the UserManager EJB, which is deployed into every Portal application created in WebLogic Workshop.

To use this control, drag the User Profile Control onto the Page Group being edited, and then refer to its instance using the Form Editor.

## Security

The caller must be in the role of "PortalSystemAdministrator" or "Admin" to invoke method 'getAllUserNames()

Related Topics

[Tutorial: Creating a Control Page Flow Using the Wizard](#)

[Adding Portal Controls to Java Page Flows](#)

## ***User Registration Event Control***

This control dispatches a 'UserRegistrationEvent' to the Portal Behavior Tracking System. The Request object may be obtained from a Page Flow by:

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

**NOTE:** This event is fired automatically by `<um:createUser>` and the *CreateUserController*.

## Method Summary

void	<b>dispatch</b> ( javax.servlet.http.HttpServletRequest request, java.lang.String userId) Dispatch a UserRegistrationEvent to the Event Service.
------	--

### dispatch

```
public void dispatch(javax.servlet.http.HttpServletRequest request,  
                    java.lang.String userId)  
    throws P13nControlException
```

Dispatch a UserRegistrationEvent to the Event Service.

**Parameters:**

request – the HttpServletRequest object, cannot be null  
userId – The id (name) of the new registered user

**Throws:**

P13nControlException – if the EventService is misconfigured or if remote exceptions are encountered while accessing the event service.

#### Related Topics

[Tutorial: Creating a Control Page Flow Using the Wizard](#)

[Adding Portal Controls to Java Page Flows](#)

# ***User Information Query Control***

This utility control returns the list of roles for a particular user and also the list of immediate parent groups.

## **Overview**

This utility control returns the list of roles for the current user for a particular resource. It will also return the list of groups to which that user belongs.

It does not allow the determination of roles for containers that aren't Page Groups, nor does it allow the determination of roles for an arbitrary user. This control only interrogates the current user.

## **Annotations**

The *resourceType* property specifies whether the resource is one of the following types:

- GlobalRoleResource
- EnterpriseRoleResource (default value)
- WebappRoleResource
- HierarchyRoleResource
- LeafRoleResource

## **Security**

The caller must be in the role of "PortalSystemAdministrator" or "Admin" to invoke method 'getAllGroupNames()'.

# Creating a New EJB Control

Enterprise Java Bean (EJB) controls make it easy for you to use an existing, deployed EJB from within an application. This topic describes how to create a new EJB control and configure it to access a local EJB or a remote EJB.

## Creating a New EJB Control

1. If you are not in Design View, click the *Design View* tab.
2. From the *Insert* menu, choose *Insert-->Portal-->EJB Control*. The *Control – Insert EJB* dialog opens.
3. In the *Variable name for this control* field, type the name for your EJB control.
4. Select the *Create a new EJB control to use with this service* radio button.
5. In the *New JCX name* field, type the name of the new file.
6. Decide whether you want to make this a control factory and select or clear the *Make this a control factory that can create multiple instances at runtime* check box. For more information about control factories, see Control Factories: Managing Collections of Controls.
7. In the Step 3 pane, next to the *jndi-name* field, click *Browse*. The *Database Browser* dialog appears.
8. Select the appropriate EJB from the list and click *Select*. The name appears in the *jndi-name* field, and the interfaces used by this EJB appear in the *home interface* and *bean interface* fields.
9. Click *Create*.

*Note:* Before an EJB control will function, the EJB it represents must be deployed in WebLogic Server. To deploy an EJB, please refer to the WebLogic Server documentation, or refer to your system administrator.

## Accessing Remote EJBs

A local EJB is an EJB that is deployed on the same instance of WebLogic Server that is running your application. A remote EJB is one that is deployed on a different server.

You can access remote EJBs using the EJB control, provided the server hosting the EJB control and the server to which the target EJB is deployed are in the same domain. You access remote EJBs by using special JNDI syntax in the *jndi-name* attribute. For example:

```
jndi://username:password@host:7001/my.resource.jndi.object
```

*Note:* Accessing a remote EJB in a different domain via the EJB control requires advanced transaction configuration. Please consult the "Configuring Domains for Inter-Domain Transactions" section of the Managing Transactions topic in the WebLogic Server documentation on [edocs.bea.com](http://edocs.bea.com).

Related Topics

Portal Control Declaration

Using WebLogic Built-In Controls

Using an EJB Control

# ***Entity Property Manager Control***

EntityPropertyManager is the remote interface for a session bean that manages persistence of ConfigurableEntity EJB's and their properties. The default implementation uses the WLPS database as its backing store, other implementations may use other datastores such as an LDAP server.

Each ConfigurableEntity must have an ejb-ref in its deployment descriptor that can be used to identify the correct EntityPropertyManager to use.

This control requires that the EntityProperty EJB has been deployed to the application. The EntityProperty EJB is contained in p13n\_ejb.jar, and is automatically deployed as part of a Portal application.

- Locator – a PropertyLocator identifying the entity to modify.
- PropertySet – the property set containing the property to modify.
- PropertyName – the name of the property to modify.
- Value – the value to persist for the given property and entity.

The following methods are available in this control:

- createUniqueId
- getDynamicProperties
- getEntityNames
- getProperties
- getProperty
- getPropertyLocator
- getUniqueId
- removeEntity
- removeProperties
- removeProperty
- setProperty

Related Topics

Tutorial: Creating a Control Page Flow Using the Wizard

Overview: Working with Portal Control

# ***Event Service Control***

Place this control in a Page Flow and pass it events that will be handled by registered listeners.

## **Overview**

Listeners register themselves for this service via the WebLogic management console; classes that implement the `EventListener` interface may add themselves as listeners using the Configuration tab for the Event Service. Those classes express interest in certain Event types, and when an event of that is dispatched via this service, it is forwarded to the listener.

This control interacts with the `EventService EJB`, which must be deployed to the application. (The `EventService EJB` is contained in `p13n_ejb.jar`, and is deployed with every Portal application.)

### Related Topics

Overview: Working with Portal Control

Tutorial: Creating a Control Page Flow Using the Wizard



# Group Manager Control

This control is a remote interface for the GroupManager session bean, and depends on the following EJB environment in the ejb deployment descriptor:

***ejb/GroupProfileManager***: an ejb-ref pointing to the default GroupProfileManager deployment.

***ProtectedGroupNames***: an env-entry of type String containing a comma-separated list of group names (case insensitive) that cannot be deleted through the GroupManager. These are optional, required only if you wish to protect group names from deletion. Note that the special WLS groups everyone, users, and Administrators are always protected – this list can be used to extend that protection to other groups.

***Any methods*** that create or remove groups delegate to the WebLogic provider selected by RealmHelper.getProviderMBean. If multiple providers are configured, see RealmHelper for information about which of those providers will be used.

## Security

The caller must be in the role of "PortalSystemAdministrator" or "Portal" to invoke most of these methods.

Related Topics

Tutorial: Creating a Control Page Flow Using the Wizard

Adding Portal Controls to Java Page Flows

# Group Profile Manager Control

This control is a remote interface for the ProfileManager, a stateless session bean used to access profile values.

## Using the control

To provide a successor to the methods which take one, use the profile name of the successor profile. Each profile manager uses only one type as the successor type; for example, when specifying a successor to a UserProfileManager, it assumes the name is a group profile name. Property mapping is done through the deployment descriptor of the ProfileManager session bean. First, the default EntityPropertyManager to be used must have an ejb-ref named ejb/EntityPropertyManager. Any other EntityPropertyManager's to be used must also have ejb-refs.

The following instructions explain how to map properties to a non-default EntityPropertyManager:

**To map an entire property set:** create an environment entry called PropertyMapping/<property set name> which is a String that holds the name of the ejb-ref to use for that property set. For example, PropertyMapping/Ldap might have the value ejb/LdapPropertyManager, which is an ejb-ref pointing to an EntityPropertyManager that goes to an ldap server.

**To map a single property:** create an environment entry called PropertyMapping/<property set name>.<property name> which is a String that holds the name of the ejb-ref to use for that property. For example, PropertyMapping/Ldap.fax might have the value ejb/CustomerPropertyManager, which is an ejb-ref pointing to an EntityPropertyManager that goes to a customer database table.

If a property set is mapped to an EntityPropertyManager, and another entry maps one property of that property set to a different EntityPropertyManager, the single property mapping will override the property set mapping. In the previous examples, the LdapPropertyManager would be used for all properties in the Ldap property set except for "fax", which would be retrieved from the CustomerPropertyManager.

## Security

The caller must be in the role of "PortalSystemAdministrator" or "Admin" to invoke most of these methods.

### Related Topics

Tutorial: Creating a Control Page Flow Using the Wizard

Adding Portal Controls to Java Page Flows

# ***Property Set Manager Control***

PropertySetManager is a stateless session bean that provides access to property sets, through PropertySetRepositories. It also provides translation between XML and property set objects for use in the data synchronization framework.

## **Overview**

The methods provided by this control are as follows:

- getPropertySet – Get the PropertySet identified by the given type and name.
- getPropertySetXml – Retrieves the XML representation of a property set.
- getPropertySets – Gets all PropertySets of a given type.

## **Security**

The caller must be in the role of "PortalSystemAdministrator" or "Admin" to invoke these methods.

Related Topics

Tutorial: Creating a Control Page Flow Using the Wizard

Adding Portal Controls to Java Page Flows

# ***Realm Configuration Control***

This is the public interface for keeping personalization profile records in sync with the WLS realm.

The following methods are provided for this control:

- cleanupMisconfiguredGroups
- cleanupMisconfiguredUsers
- getGroupProfileNames
- getMisconfiguredGroups
- getMisconfiguredUsers
- getRealmGroups
- getRealmUsers
- getUserProfileNames
- isManageableRealm

## **Security**

The caller must be in the role of "PortalSystemAdministrator" or "Admin" to invoke these methods.

Related Topics

[Tutorial: Creating a Control Page Flow Using the Wizard](#)

[Adding Portal Controls to Java Page Flows](#)

# User Manager Control

This Control is the Remote Interface for the UserManager session bean.

## Overview

This is the synchronization point between user profile support and WebLogic security. Any user management operations should be done here, rather than through the security APIs directly.

This class depends on the following EJB environment in the ejb deployment descriptor:

- ***jdbc/commercePool***: a resource-ref to a TxDataSource for the commercePool database connection pool containing user profile data.
- ***ejb/UserProfileManager***: an ejb-ref pointing to the default UserProfileManager deployment.
- ***ejb/ProfileType/...***: any alternative profile types should have an ejb-ref entry of the form `ejb/ProfileType/<profile type name>` that points to the appropriate UserProfileManager for that type. These are optional, applicable only if other profile types are used.
- ***ReservedUserNames***: an env-entry of type String containing a comma-separated list of user names (case insensitive) that are restricted: the UserManager will not create users with these names. These are optional, required only if you wish to restrict creation of any user names.
- ***ProtectedUserNames***: an entry of type String containing a comma-separated list of user names (case insensitive) that cannot be deleted through the UserManager. These are optional, required only if you wish to protect user names from deletion.  
Any methods that create or remove users delegate to the WebLogic provider selected by `RealmHelper.getProviderMBean`. If multiple providers are configured, see `RealmHelper` for information about which of those providers will be used.

### Related Topics

[Tutorial: Creating a Control Page Flow Using the Wizard](#)

[Adding Portal Controls to Java Page Flows](#)

# ***User Profile Control***

This control exposes the user profile information to a Page Flow portlet. This is useful if you need to get all properties for a user, or only a subset of properties. Obviously, in order to obtain access to this information, a user would need to login with appropriate privileges. For this reason, a Page Flow that uses the User Profile Control would be a good candidate for a nested page flow.

**NOTE:** The User Profile Control does not allow the user to register. If the user does not have a valid account, an anonymous or tracked profile will be used.

## **Overview**

This control is backed by the UserManager EJB, which is deployed into every Portal application created in WebLogic Workshop.

To use this control, drag the User Profile Control onto the Page Group being edited, and then refer to its instance using the Form Editor.

### Related Topics

[Tutorial: Creating a Control Page Flow Using the Wizard](#)

[Adding Portal Controls to Java Page Flows](#)

# 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

## Developing Portal Applications

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 (current service pack) domain, copy the tools' webflow files into the newly-created /webapps/tools700 directory.

If you do not have WebLogic Portal 7.0 installed, you can download it from BEA's download site.

- c. Open the newly copied webflow-extensions.wfx file and delete the following block:

```
<process executor="examples.petflow.layout.LayoutProcessor" name="layoutman">
  <node-processor-input name="header" required="true"/>
  <node-processor-input name="content" required="true"/>
  <node-processor-input name="footer" required="true"/>
</process>
```

- d. Save and close webflow-extensions.wfx.
  - e. 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 browser 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.

## Developing Portal Applications

- ◆ 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:

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

## Developing Portal Applications

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

### **Required**

<b>Start Date and Stop Date</b>	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

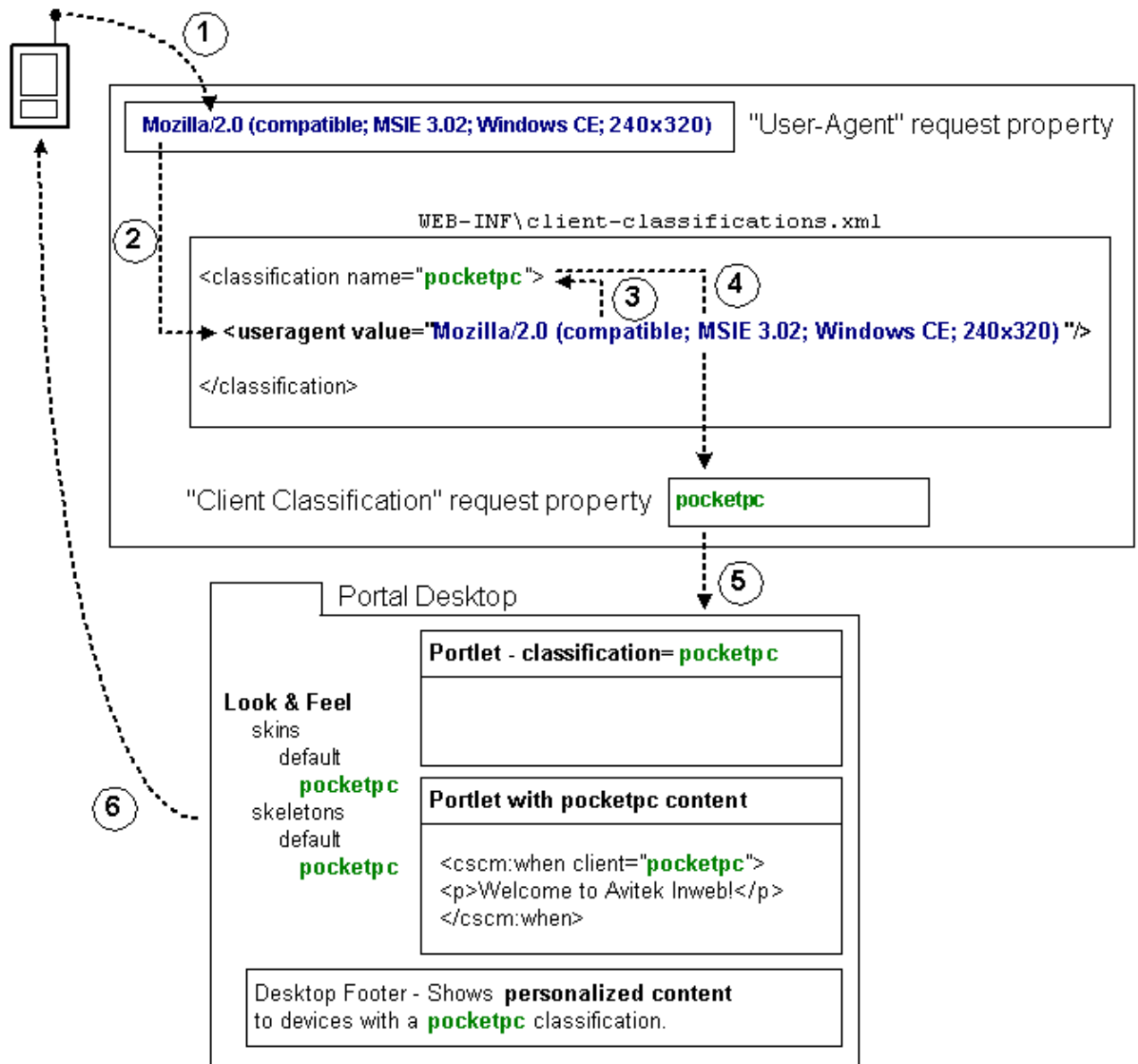
There are many types of Web-enabled mobile devices that can access your portals. Since these devices have different interfaces and different-sized viewing areas, each has a unique requirement for the type of content they display.

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.



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.

**I** 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.

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
3. Designing Interaction Management
4. Creating Personalization Conditions
5. Personalizing Portal Applications

Related Topics

Portal JSP Tags

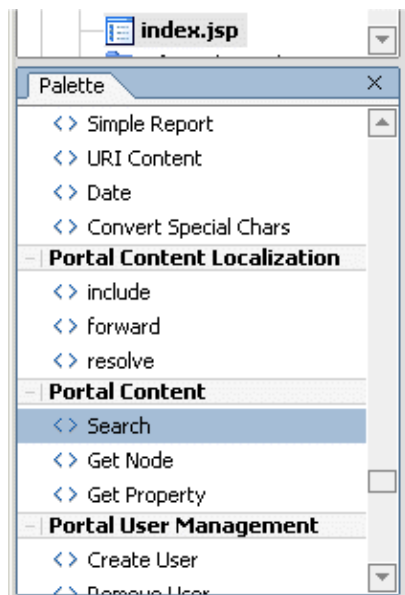
# Using Portal JSP Tags

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.

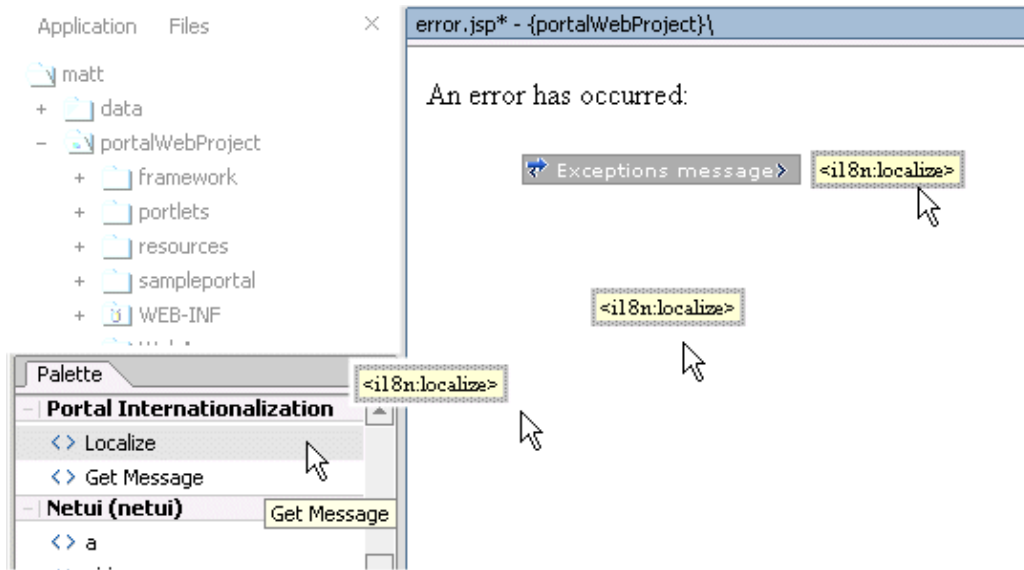
## Using JSP Tags

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



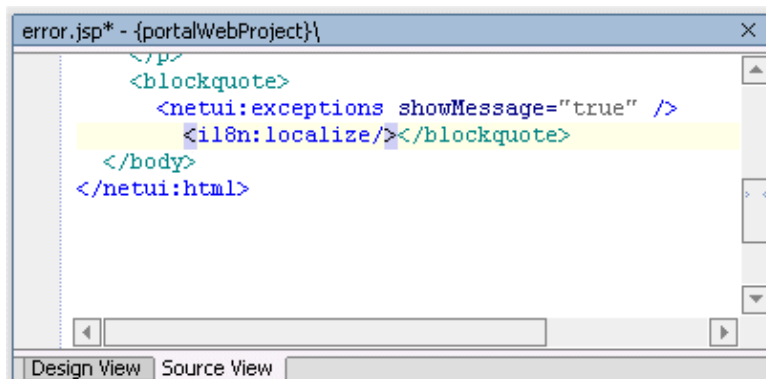
To use a tag, simply drag it into the JSP designer, use the Source View to edit the code directly, and use the Properties Designer to set properties.

## Developing Portal Applications



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.



**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 from 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?

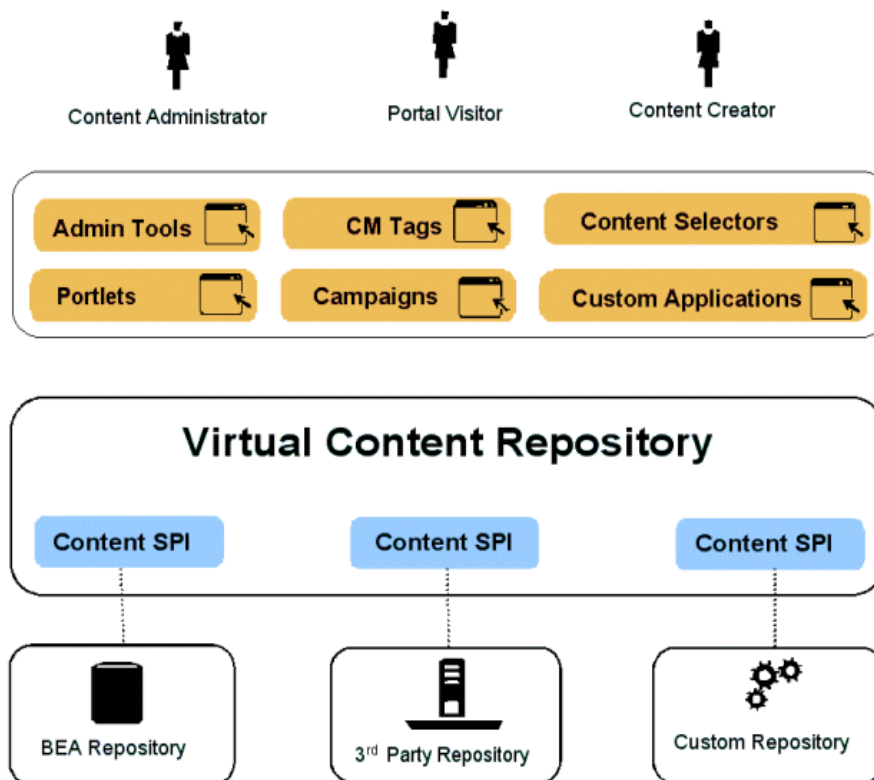
# Overview of Content Management

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 may 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

Designing Interaction Management

Creating Personalization Conditions

Personalizing Portal Applications



# Setting up Unified User Profiles

A Unified User Profile provides the capability to leverage user data from external sources such as LDAP servers, legacy systems and databases. This allows for the use of a single profile to access user data from many different sources.

**Note:** If you are using WebLogic's default user store, unified user profile functionality is already configured.

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

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.

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.UnsupportedOperationException` 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 then 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 at <http://edocs.bea.com/wlp/docs81/javadoc/index.html>.)
- 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:

- 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. For more information, refer to Modifying the

Existing ProfileManager Deployment Configuration.

- 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. For more information, refer to Configuring and Deploying a New ProfileManager.

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 ldaprofile.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, as 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>
  </reference-descriptor>
</weblogic-enterprise-bean>
```

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-name>
    </ejb-reference-description>
  </reference-descriptor>
</weblogic-enterprise-bean>
```

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 `META-INF/application.xml` for your enterprise application 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`.

Your new Unified User Profile type is ready to use. You can use the WebLogic Administration Portal to create a user of type "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 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`.

## Developing Portal Applications

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>${APPNAME}.BEA_personalization.MyEntityPropertyManager</jndi-name>
    </ejb-reference-description>
  </reference-descriptor>
  <jndi-name>${APPNAME}.BEA_personalization.MyProfileManager</jndi-name>
```

## Developing Portal Applications

```
</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 `META-INF/application.xml` for your enterprise application 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`.

Your new Unified User Profile type is ready to use. You can use the WebLogic Administration Portal to create a user of type "UUPEXampleUser," 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`.

**Note:** Tell your administrators that when they create a user in the WebLogic Administration Portal, they must select the new user type.

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

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. Deploy the ldaprofile.jar component within your application.

The LdapPropertyManager EJB in ldaprofile.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.

This is done by editing ejb-jar.xml to 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. -->

<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>
```

## Developing Portal Applications

```
<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 ldaprofile.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 ldaprofile.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. Start the server and deploy the application.
4. Start the WebLogic Server Administration Console for the active domain.

You can set a default profile type for each Web project by setting a context parameter in web.xml for DEFAULT\_USER\_PROFILE\_TYPE. For example:

```
<context-param>
```

## Developing Portal Applications

```
<param-name>DEFAULT_USER_PROFILE_TYPE</param-name>  
  <param-value>WLCS_Customer</param-value>  
</context-param>
```

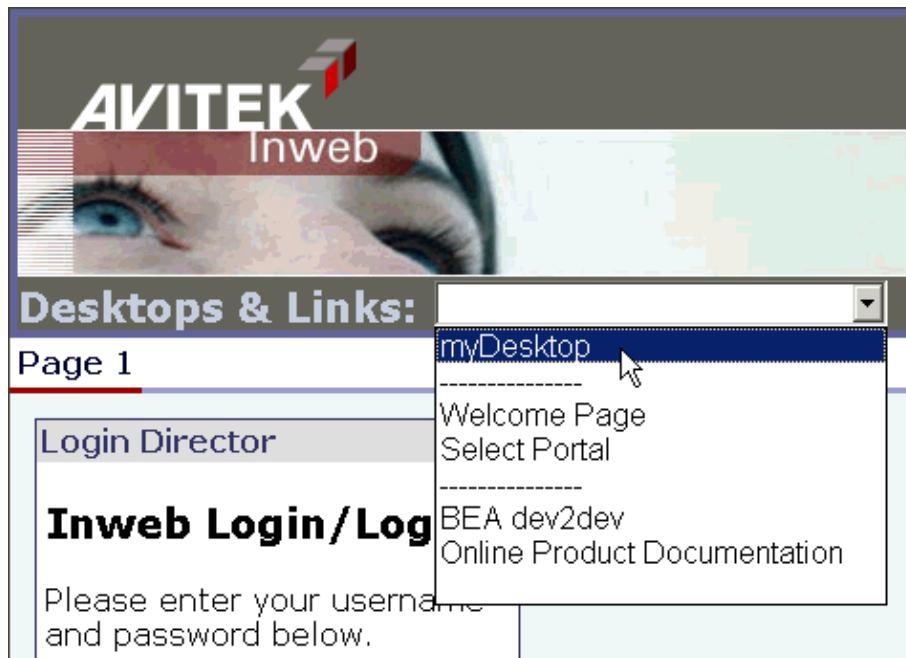
To create a custom user profile property set, see [Creating User Profile Properties](#).

# 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:

<i>Import or copy this</i>	<i>to this directory</i> (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/



## Developing Portal Applications

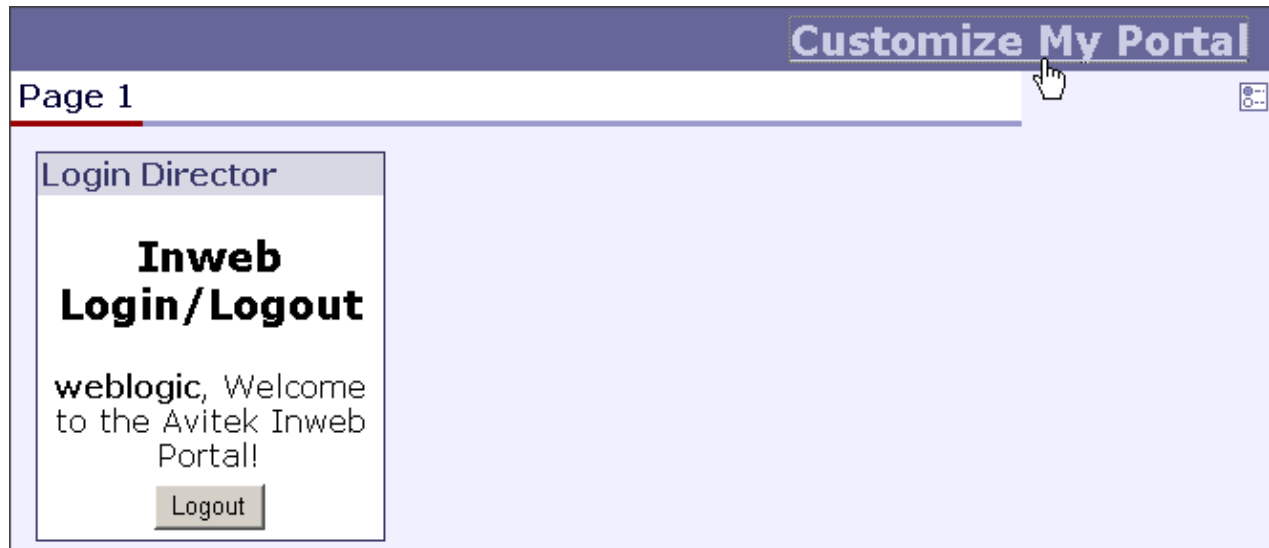
4. Open `<PORTAL_APP>/<project>/portlets/header/header.jsp` in WebLogic Workshop and replace the string `sampleportal` with the name of your project.
5. Create a shell and make `<PORTAL_APP>/<project>/portlets/header/header.jsp` the header content.
6. In a `.portal` file open in the Portal Designer,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

## Developing Portal Applications

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

# 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 JSPs. 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.

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:

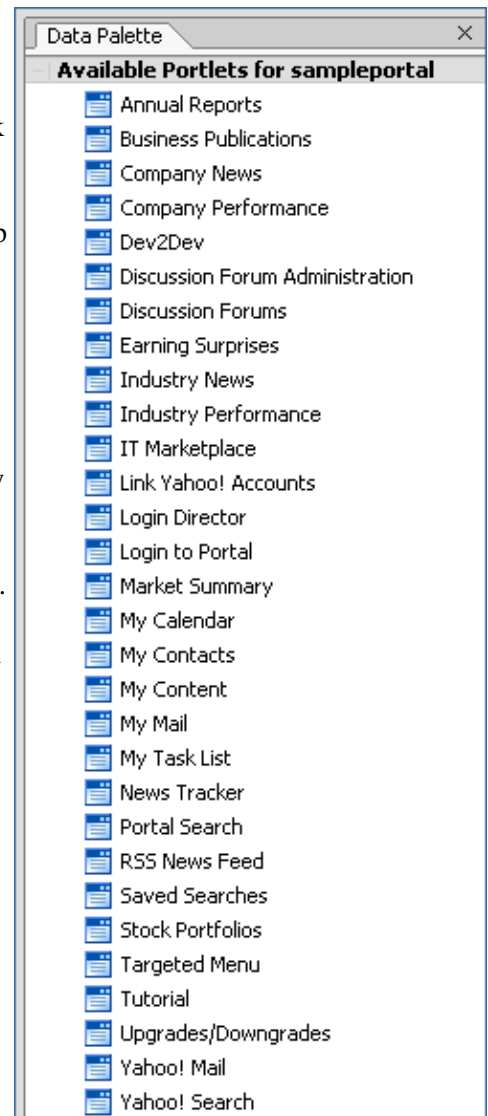
<BEA\_HOME>/<WEBLOGIC\_HOME>/samples/portal/portalApp

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.

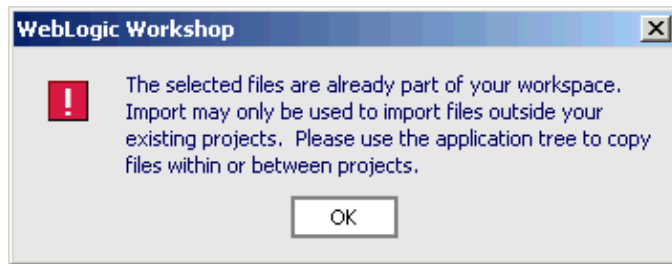
## Developing Portal Applications

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

<b><i>Import or copy this</i></b>	<b><i>to this directory</i></b> (create if it does not exist)
<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/login/director.portlet	<PORTAL_APP>/<project>/portlets/login/director
<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/login/director.jsp	<PORTAL_APP>/<project>/portlets/login/director
<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB-INF/src/examples/login/DirectorBacking.java	<PORTAL_APP>/<project>/WEB-INF/src/examples/login
<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/WEB-INF/src/examples/login/DirectorUtil.java	<PORTAL_APP>/<project>/WEB-INF/src/examples/login

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 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](#)

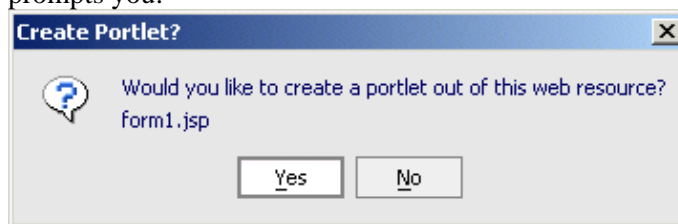
[Customizing Portlets](#)

# 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.

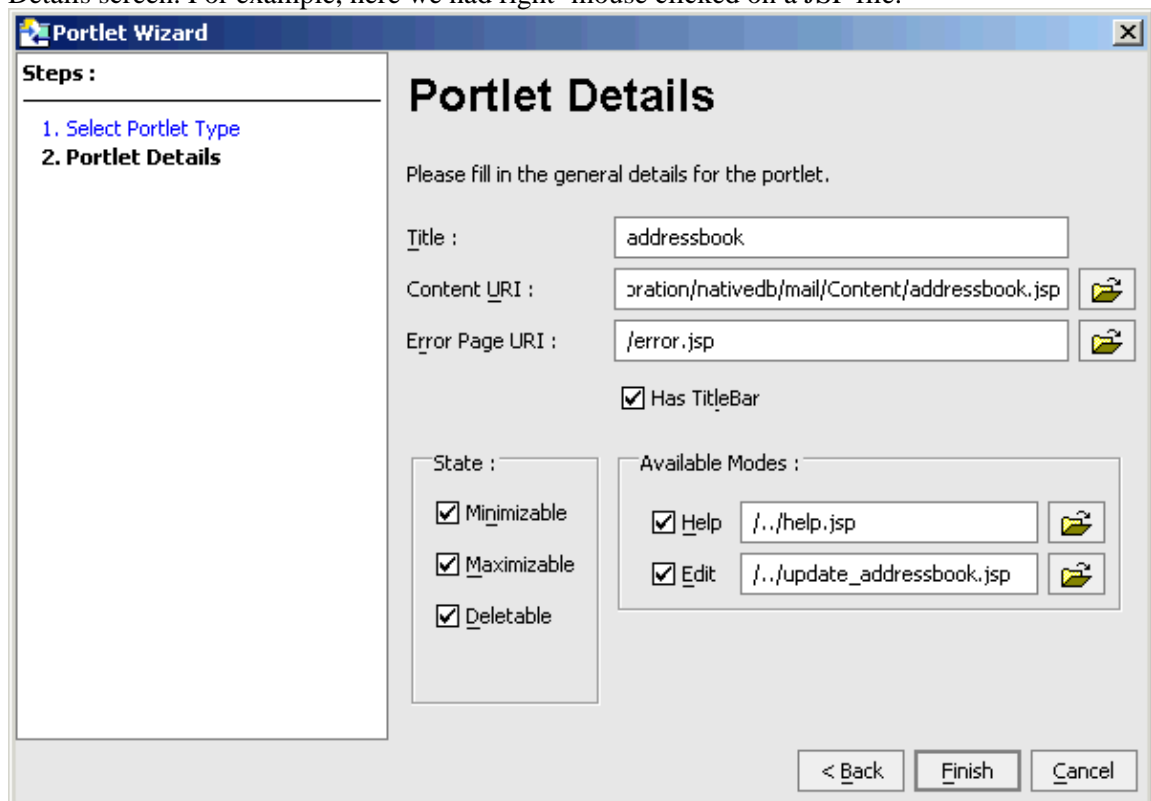
A new portlet can be associated with an existing resource, or created first and associated with a resource later. The Portlet Wizard is invoked in the WebLogic Workshop IDE anytime you perform one of these operations:

- 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:



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.





## 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.

<i>Type</i>	<i>Description</i>
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.

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
- In addition, you can create portlets that use web services. See [Creating a Web Services Portlet](#).

Related Topics

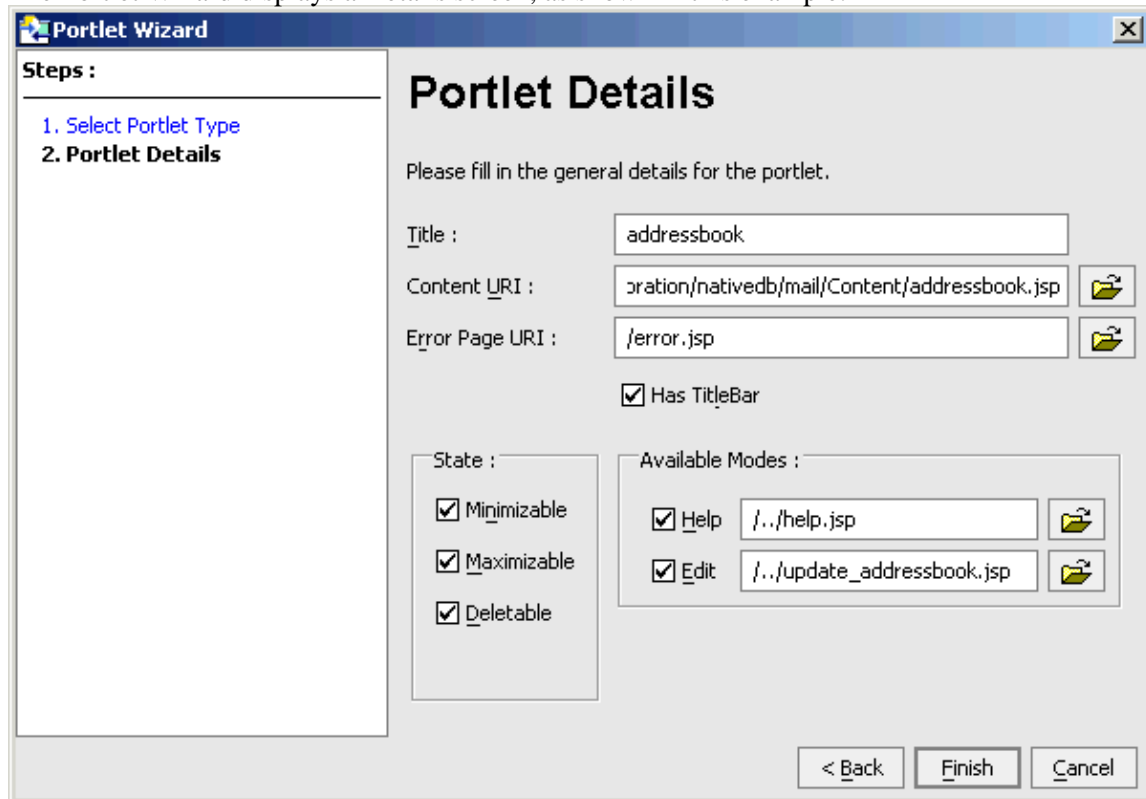
Building Portlets

# 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:



The screenshot shows the 'Portlet Wizard' dialog box, specifically the 'Portlet Details' step. The title bar reads 'Portlet Wizard'. On the left, a 'Steps' pane shows '1. Select Portlet Type' and '2. Portlet Details' (the current step). The main area is titled 'Portlet Details' and contains the instruction 'Please fill in the general details for the portlet.'.

Fields and options include:

- Title :** Text box containing 'addressbook'.
- Content URI :** Text box containing 'pration/nativedb/mail/Content/addressbook.jsp' with a folder icon to its right.
- Error Page URI :** Text box containing '/error.jsp' with a folder icon to its right.
- ☒ **Has TitleBar**
- State :** A group box containing three checked options: ☒ **Minimizable**, ☒ **Maximizable**, and ☒ **Deletable**.
- Available Modes :** A group box containing two checked options: ☒ **Help** with text box '/../help.jsp' and folder icon, and ☒ **Edit** with text box '/../update\_addressbook.jsp' and folder icon.

At the bottom right are three buttons: '< Back', 'Finish', and 'Cancel'.

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 Exit 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

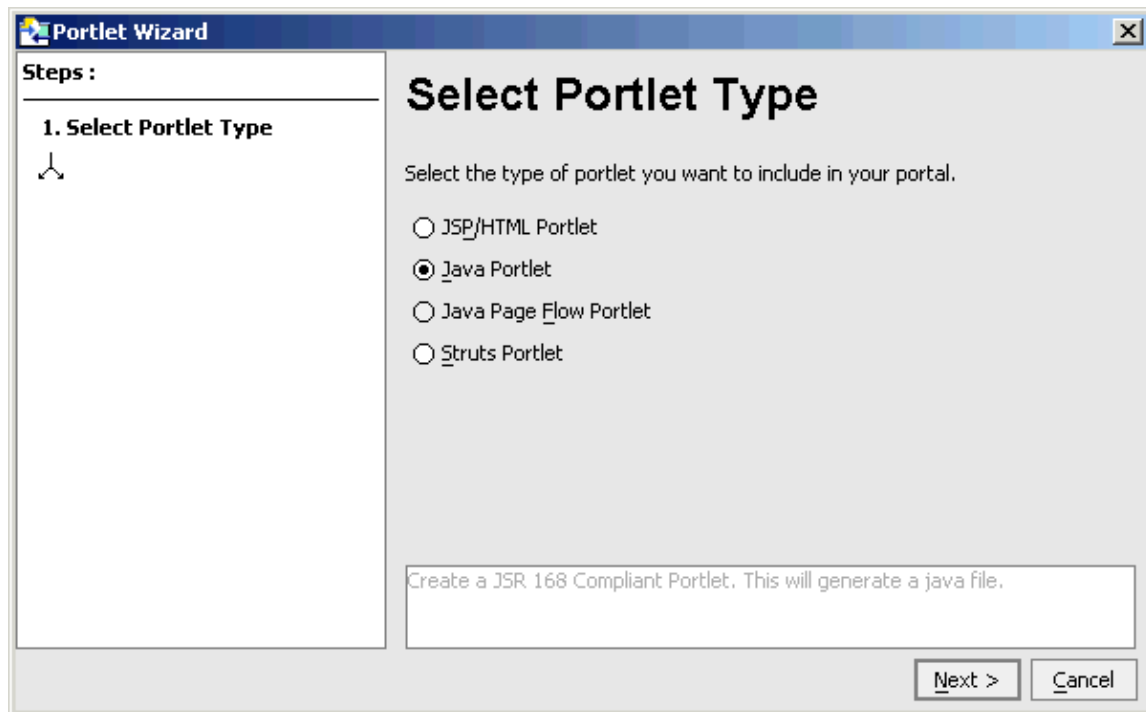
JSR 168 (Java Portlet) is a Java specification that aims at establishing portability between portlets and portals. One of the main goals of the specification is to define a set of standard Java APIs for portal and portlet vendors. These APIs will cover areas such as presentation, aggregation, security, 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.

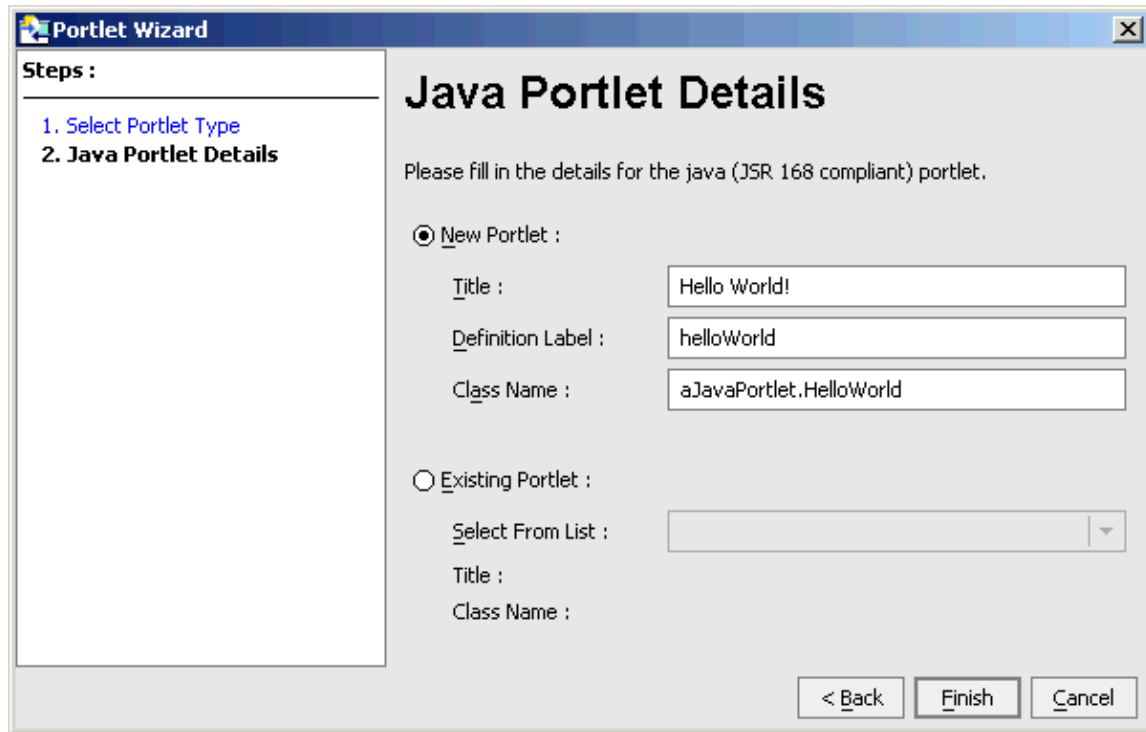
In the WebLogic Workshop IDE, you can use the Portlet Wizard to create a new Java Portlet. For example, in the /portalApp/sampleportal web project, there is a /portlets folder. In the IDE Application pane, right-mouse click on the portlets folder and select **New > Folder...**

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:



Click the Next button. The Portlet Wizard displays this screen, on which we have already filled in three values:



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.com
  <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.

## Developing Portal Applications

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:

The screenshot shows the WebLogic Workshop interface. On the left, a portlet window titled 'helloWorld.portlet - {John}\aJavaPortlet\'. It contains a blue header bar with the text 'Hello World!' and a table with the following properties:

Name	helloWorld
Description	Description goes here
Class	aJavaPortlet.HelloWorld
States	
Modes	
Cache Expiration	
Mime Types	

On the right, the 'Property Editor' pane is open, showing 'Java Portlet - Java Portlet Properties'. It is divided into several sections:

- 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](#)

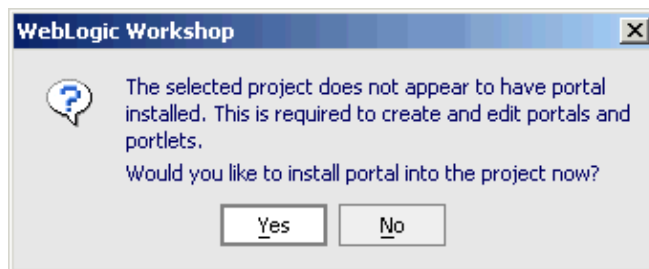
[Creating Portlets](#)

# Building Java Page Flow Portlets

You can use the Portlet Wizard to build 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.jspf 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:



If you receive this prompt, click the Yes button. When the portal libraries are installed, the Portlet Wizard displays this dialog:

**Portlet Wizard**

**Steps :**

1. Select Portlet Type
2. Portlet Details

**Portlet Details**

Please fill in the general details for the portlet.

Title :

Content URI :

Error Page URI :

☒ Has TitleBar

State :

- ☐ Minimizable
- ☐ Maximizable
- ☐ Deletable

Available Modes :

- ☐ Help
- ☐ Edit

< Back   Finish   Cancel

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 Exit 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

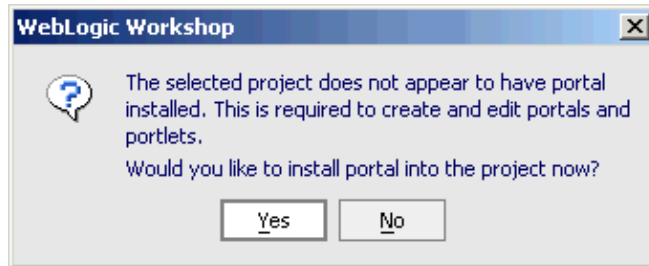


# Building Struts Portlets

You can use the Portlet Wizard to generate a portlet based on a Struts Module.

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:



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.

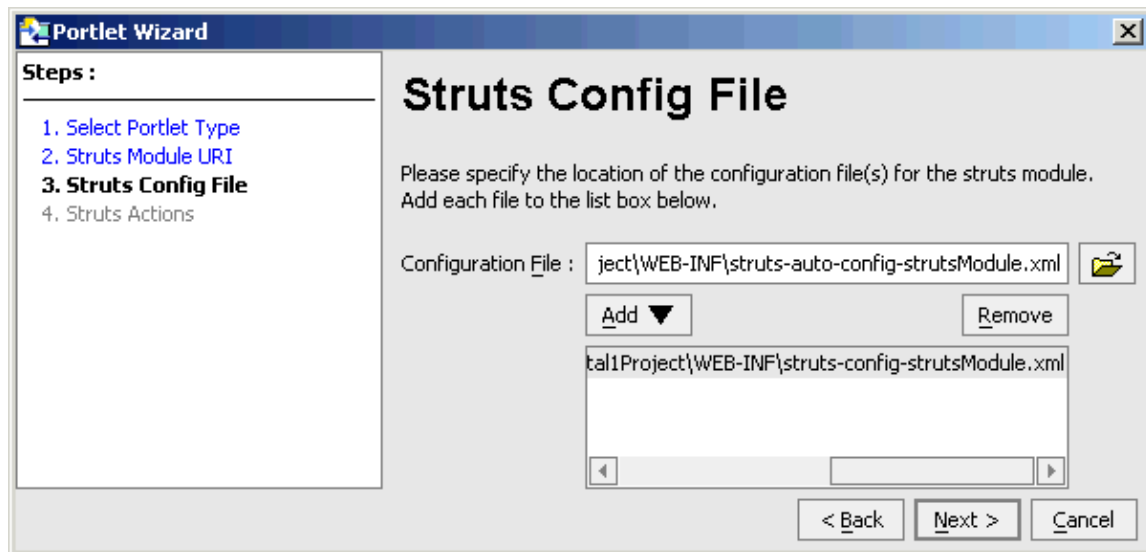
On the wizard's Struts Module URI screen, specify the folder that will contain the <portlet-name>.portlet file.

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.

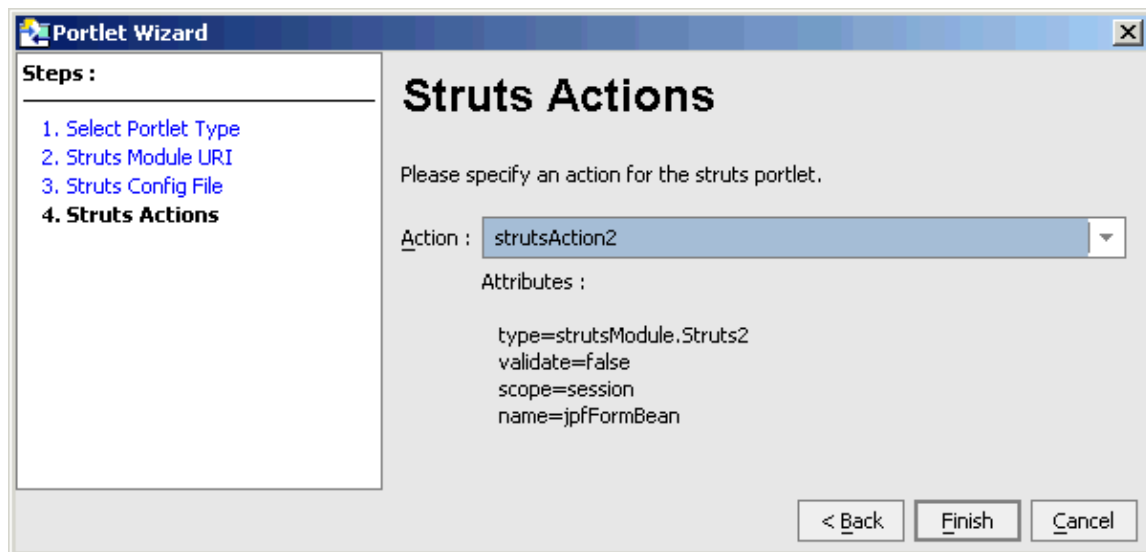
The following files that are under the <WEBLOGIC\_HOME>/samples/workshop/SamplesApp/WebApp/ folder were copied into our Portal application:

```
/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:



On the final screen, Struts Actions, we specified an action for the Struts Portlet:



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

A portlet's titlebar serves two purposes:

- It is a configurable feature of portlets that enables you to place a text title at the top of the window into which your portlet is rendered.
- It also serves as the container for Mode controls such as Help, Edit, Maximize, Minimize, and Deletable. A portlet must have a titlebar in order to expose Mode controls to the user.

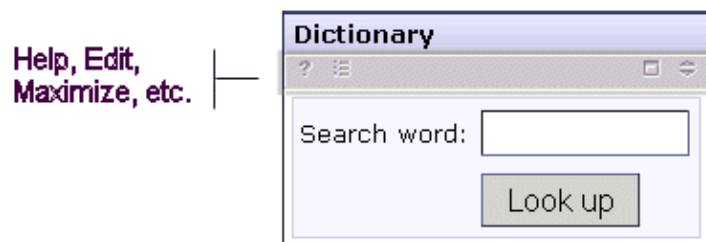
When you define a portlet with the Portlet Wizard, one of the options is to enable or disable the titlebar. For example:

The screenshot shows the 'Portlet Wizard' dialog box, specifically the 'Portlet Details' step. The 'Steps' pane on the left shows '1. Select Portlet Type' and '2. Portlet Details'. The main area contains the following fields and options:

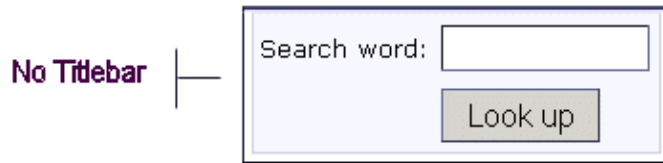
- Title :** addressbook
- Content\_URI :** oration/nativedb/mail/Content/addressbook.jsp
- Error Page URI :** /error.jsp
- ☒ Has TitleBar
- State :**
  - ☒ Minimizable
  - ☒ Maximizable
  - ☒ Deletable
- Available Modes :**
  - ☒ Help: ../help.jsp
  - ☒ Edit: ../update\_addressbook.jsp

At the bottom are buttons for '< Back', 'Finish', and 'Cancel'.

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

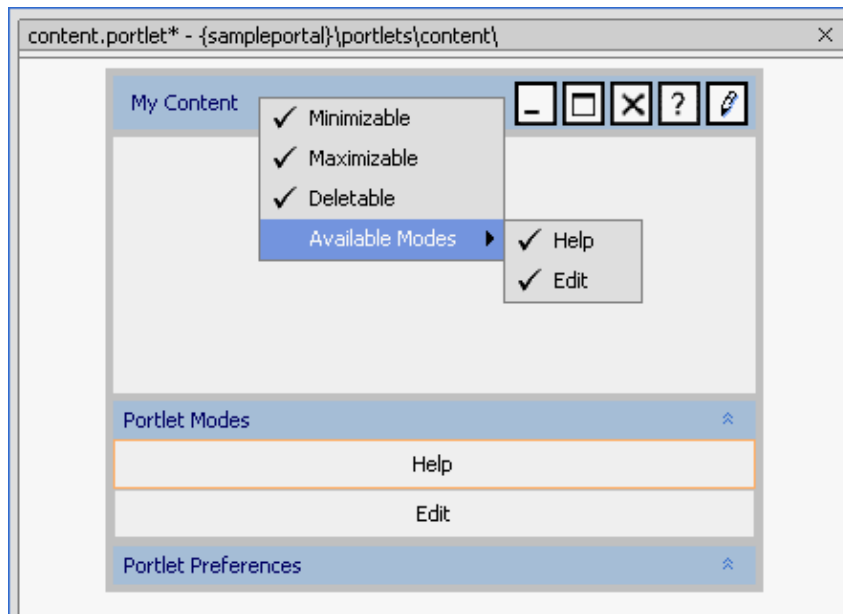
### About the Icons Used in Portlet Titlebars

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	<b>{overflow-y:auto};{height:250px}</b>

Where **Content Presentation Style** = **{overflow-y:auto};{height:250px}**

The result looks like the following figure:



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.jl.jlController.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 message )
{
    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.

## Developing Portal Applications

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 ***YourWebapp/YourPortalName.portal***.

### Related Topics

[Tutorial: Using Page Flows Inside Portlets](#)

[Portal Key Concepts and Architecture](#)

[Developing Portal Applications](#)

[Handling Exceptions in Page Flows](#)