Oracle® Fusion Middleware

Portlet Development Guide for Oracle WebLogic Portal

10*g* Release 3 (10.3.2)

E14244-04

April 2011

ORACLE

Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal, 10*g* Release 3 (10.3.2)

E14244-04

Contents

6 Building Java Portlets

7 Creating Clipper Portlets

8 Working With JSF-Java Portlets

9 Developing Portlets

10 Optimizing Portlet Performance

13 Interportlet Communication Example With Event Handling

14 Adding the Content Presenter Portlet

15 Adding a Third-Party Portlet

16 Using the Collaboration Portlets

List of Examples

List of Figures

List of Tables

Preface

The primary focus of this guide is developing and administering portlets. Topics like interportlet communication, analytics, and integration of third-party portlets are discussed. In addition, the Content Presenter portlet is discussed. The Content Presenter portlet allows users to retrieve and display different kinds of content in a portal in real time, without assistance from the IT Department or software developers.

Audience

This guide is intended for developers and portal administrators. Developers use Oracle Enterprise Pack for Eclipse (OEPE) to create portlets, to write the business logic used by portlets, to create web applications, and to perform other related tasks, like implementing data transfer and interportlet communication. Portal administrators use the WebLogic Portal Administration Console to assemble and configure assemblages of portlets called desktops.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Related Documents

For more information, see the following documents in the WebLogic Portal documentation set:

- Oracle WebLogic Portal Portal Development Guide
- Oracle WebLogic Portal Federated Portals Guide
- Oracle WebLogic Portal Security Guide
- Oracle WebLogic Portal Tutorials

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction

This chapter introduces Oracle WebLogic Portal portlet concepts and describes the content of this guide.

This chapter includes the following sections:

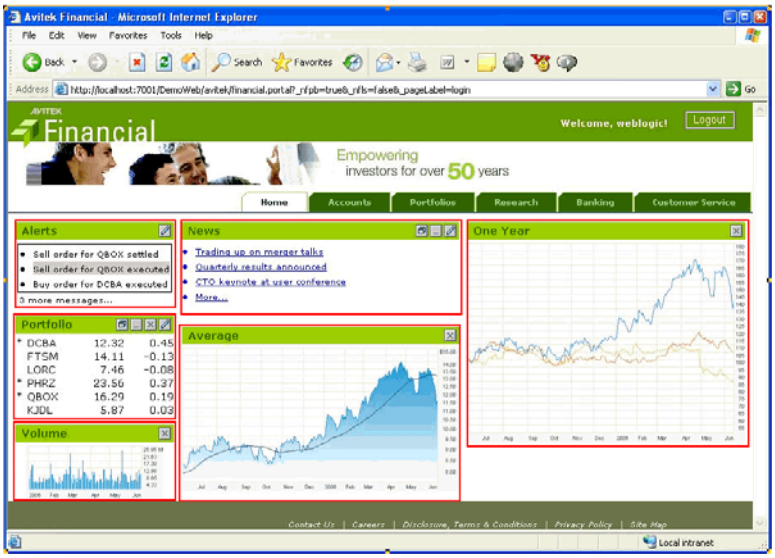
- Section 1.1, "Portlet Overview"
- Section 1.2, "Portlet Development and the Portal Life Cycle"

1.1 Portlet Overview

Portlets are modular panes within a web browser that surface applications, information, and business processes. Portlets can contain anything from static HTML content to Java controls to complex web services and process-heavy applications. Portlets can communicate with each other using events and other techniques. A single portlet can also have multiple instances—in other words, it can appear on a variety of different pages within a single portal, or even across multiple portals if the portlet is enabled for Web Services for Remote Portlets (WSRP). You can customize portlets to meet the needs of specific users or groups.

Figure shows an example portal desktop with its associated portlets outlined in red.

Figure 1–1 Portal Desktop with Portlets



WebLogic Portal supports the development of portlets through Oracle Enterprise Pack for Eclipse, which is a client-based tool. You can develop portals without Oracle Enterprise Pack for Eclipse through coding in any tool of choice such as JBuilder, VI or Emacs; portlets can be written in Java or JSP, and can include JavaScript for client-side operations. However, to realize the full development-time productivity gains afforded to the WebLogic Portal customer, you should use Oracle Enterprise Pack for Eclipse as your portal and portlet development platform.

For a description of each type of portlet that you can build using WebLogic Portal, refer to Chapter 3, "Portlet Types."

1.2 Portlet Development and the Portal Life Cycle

The tasks in this guide are organized according to the portal life cycle, which includes best practices and sequences for creating and updating portals. For more information about the portal life cycle, refer to the *Oracle Fusion Middleware Overview for Oracle WebLogic Portal*. The portal life cycle contains four phases: architecture, development, staging, and production.

1.2.1 Architecture

During the architecture phase, you plan the configuration of your portal. For example, you can create a detailed specification outlining the requirements for your portal, the specific portlets you require, where those portlets will be hosted, and how they will communicate and interact with one another. You also consider the deployment strategy for your portal. Security architecture is another consideration that you must keep in mind at the portlet level.

The chapters describing tasks within the architecture phase include:

- Chapter 2, "Portlet Planning"
- Chapter 3, "Portlet Types"

1.2.2 Development

Developers use Oracle Enterprise Pack for Eclipse to create portlets, pages, and books. During development, you can implement data transfer and interportlet communication strategies.

In the development stage, careful attention to best practices is crucial. Wherever possible, this guide includes descriptions and instructions for adhering to these best practices.

The chapters describing tasks within the development phase include:

- Chapter 4, "Understanding Portlet Development"
- Chapter 5, "Creating Portlets"
- Chapter 7, "Creating Clipper Portlets"
- Chapter 10, "Optimizing Portlet Performance"
- Chapter 11, "Monitoring and Determining Portlet Performance"
- Chapter 12, "Configuring Local Interportlet Communication"
- Chapter 14, "Adding the Content Presenter Portlet"
- Chapter 15, "Adding a Third-Party Portlet"

1.2.3 Staging

Oracle recommends that you deploy your portal, including portlets, to a staging environment, where it can be assembled and tested before going live. In the staging environment, you use the WebLogic Portal Administration Console to assemble and configure desktops. You also test your portal in a staging environment before propagating it to a live production system. In the testing aspect of the staging phase, there is tight iteration between staging and development until the application is ready to be released.

The chapters describing tasks within the staging phase include:

- Chapter 17, "Assembling Portlets into Desktops"
- Chapter 18, "Deploying Portlets"

1.2.4 Production

A production portal is live and available to end users. A portal in production can be modified by administrators using the WebLogic Portal Administration Console and by users using Visitor Tools. For instance, an administrator might add additional portlets to a portal or reorganize the contents of a portal.

The chapter describing tasks within the production phase is:

- Chapter 19, "Managing Portlets in Production"

1.3 Getting Started

This section describes the basic prerequisites to using this guide and lists guides containing related information and topics.

1.3.1 Prerequisites

In general, this guide assumes that you have performed the following prerequisite tasks before you attempt to use this guide to develop portlets:

- Review the Section 1.3.2, "Related Guides" and become familiar with the basic operation of the tools used to create portals, portlets, and desktops,
- Review the Oracle Enterprise Pack for Eclipse tutorials and documentation to become familiar with the Eclipse-based development environment and the recommended project hierarchy.
- Complete the tutorial *Oracle Fusion Middleware Tutorials for Oracle WebLogic Portal*.

1.3.2 Related Guides

Oracle recommends that you review the following guides:

- *Oracle Fusion Middleware Overview for Oracle WebLogic Portal*
- *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*

Whenever possible, this guide includes cross references to material in related guides.

Part I

Architecture

During the architecture phase, you plan the configuration of the portlets that comprise your portal.

For a detailed description of the architecture phase of the portal life cycle, see the *Oracle Fusion Middleware Overview for Oracle WebLogic Portal*.

Part I contains the following chapters:

2 Portlet Planning

Proper planning is essential to portlet development. A properly planned portlet structure and organizational model can provide a cohesive and consistent portal interface, flexible scalability, and high performance without requiring frequent adjustments within your production system.

This chapter focuses on planning considerations and decisions that should precede the development of your portlets. Global portal-wide planning information is provided in the *Oracle Fusion Middleware Overview for Oracle WebLogic Portal*, which summarizes the types of issues to consider in the architecture phase across your portal environment. The various WebLogic Portal feature guides, such as the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*, describe architectural issues in detail for each feature area.

This chapter includes the following sections:

2.1 Portlet Development in a Distributed Portal Team

If you will be creating portlets within an environment that includes a remote (distributed) development team, you must carefully plan your implementation. Considerations for team development include:

- **Using shared resources** – You can have common portlets, such as the login portlet.
- **Sharing a common domain** – You can have a common domain among team members with different Oracle home directories.
- **Integrating remotely developed portlets into the portal** – You need to manage settings that are common to the portal application, which must match across the entire development project.

Team development of a WebLogic Portal web site revolves around well-designed source control and a correctly configured shared domain for development. For detailed instructions on setting up your development environment, refer to the Team

Development chapter of the *Oracle Fusion Middleware Production Operations Guide for Oracle WebLogic Portal*.

2.2 Portlets in a Non-Portal Environment

In some cases, you might want to expose portlets in a web page even though that web application is not based on WebLogic Portal. For example, you might want to expose portlets with WSRP from a producer environment that does not include any WebLogic Portal components. You might be running a Struts web application in a basic WebLogic Server domain, or a Java page flow application in a basic Oracle Enterprise Pack for Eclipse domain. In either case, WebLogic Portal is not part of the server configuration. The exposed portlets can then be consumed by remote portlets running in a regular WebLogic Portal domain.

Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

For more information on developing portlets for a non-WebLogic Portal environment, see "Configuring a WebLogic Server Producer" in the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

2.3 Planning Portlet Instances

In the Development phase, you use Oracle Enterprise Pack for Eclipse to create portlets and place them onto a portal. In the Staging phase, you use the Administration Console to add portlets to portal desktops. Each time you add a portlet to a desktop, you create an *instance* of that portlet. Portlet instances allow for multiple variations of the same portlet definition. By using portlet instances, portal users and administrators can configure multiple views of the same portlet through the use of portlet preferences, and reduce the overall number of distinct portlets; this portlet reuse improves portal performance and management efficiency. A common example of portlet instances is a stock watch portlet in which there is a single or multi-valued preference for ticker symbols such as ORCL, which would configure the portlet to display Oracle stock information.

Try to plan your portal hierarchy to reuse portlets when practical. For more information about portlet instances and how portlet instances are related to portlets in the Administration Console's portlet library, refer to Section 17.1, "Portlet Library."

2.4 Security

You can control access to portlet resources for two categories of users:

- **Portal visitors** – You control access to portal resources using *visitor entitlements*. Visitor access is determined based on visitor entitlement roles.
- **Portal administrators** – You control portal resource management capabilities using *delegated administration*. Administrative access is determined based on delegated administration roles.

During the architecture phase, you plan how to organize security policies and roles, and how that fits into your system-wide security strategy. You implement your security plans by setting up delegated administration and visitor entitlements using the WebLogic Portal Administration Console.

For an overall look at managing security for your portal environment, refer to the *Oracle Fusion Middleware Security Guide for Oracle WebLogic Portal*. Specific security considerations for feature areas are contained in those documents; for example, recommendations for security in WSRP-enabled environments are contained in the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

2.5 Interportlet Communication

Interportlet communication (IPC) allows multiple portlets to use or react to data. You can use interportlet communication within a single portal web application, or within federated portal applications.

For more information on interportlet communication within a single portal web application, refer to Chapter 12, "Configuring Local Interportlet Communication." For more information on interportlet communication within federated portal applications, refer to the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

2.6 Performance Planning

Try to plan for good performance within your portlet architecture to minimize the fine-tuning that is required in a production environment.

Here are some examples of performance optimizations that you can plan into your overall portal strategy:

- **Portlet caching –** You can cache the portlet within a session instead of retrieving it each time it recurs during a session (on different pages, for example).
- **Remote portlets –** With remote portlets, any portal controls within the application (portlet) that you are retrieving are rendered by the producer and not by your portal. The expense of calling the control life cycle methods is borne by resources not associated with your portal. You must balance this advantage against the delay that might be caused by network latency issues.
- **Customized portlet properties –** Customizing your portlet settings can help you improve performance; for example, you can set process-expensive portlets to be processed in a multi-threaded (forkable) environment.
- **Asynchronous portlet rendering** - Asynchronous portlet rendering allows you to render the content of a portlet independently from the surrounding portal page. You can use either AJAX technology or IFRAME technology to implement asynchronous rendering.

Plan your performance optimizations before you begin developing portlets so that you can implement any pre-requisites that are required. For detailed instructions on developing high-performance portlets, refer to Chapter 10, "Optimizing Portlet Performance." For post-development WebLogic Portal performance recommendations, refer to the *Performance Tuning Guide*.

3

Portlet Types

As part of your portlet implementation plan, Oracle recommends that you examine the different types of portlets that are available in WebLogic Portal and decide which types are best suited for the tasks that you want to accomplish.

> **Tip:** Java Server Faces (JSF) technology provides the best interoperative facilities with the rest of the Oracle Fusion Middleware and WebCenter products.

If you are looking for a way to interface with Java controls, use Struts-based infrastructure, and deliver rich navigation elements, then you might choose to implement Java Page Flow or Struts portlets. If you are looking for a simple portlet or you want to convert an existing JSP page into a portlet, you might consider using a JSP portlet. If you work for an independent software company or other enterprise that is concerned with portability across multiple portal vendors, then you might choose to use JSR 286-compliant Java portlets whenever possible. If you want to implement asynchronous portlet rendering in your portal, you can use nearly any of the portlet types described in this chapter.

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

This chapter differentiates the various portlet types to help you in your decision-making process. This chapter contains the following sections:

3.1 Java Server Faces (JSF) Portlets

The Java Server Faces (JSF) specification, JSR 127, defines a user interface framework that simplifies development and maintenance of Java applications that run on a server and are displayed and used from a client.

> **Note:** Oracle recommends JSF as the best approach to implementing portal applications. As one of the active members of the JSF expert group, Oracle has committed itself to evolve and support JSF.

According to the Java Server Faces Specification, available from the Java Community Process web site at `http://jcp.org/aboutJava/communityprocess/final/jsr127/index2.html`.

JSF's core architecture is designed to be independent of specific protocols and markup. However it is also aimed directly at solving many of the common problems encountered when writing applications for HTML clients that communicate via HTTP to a Java application server that supports servlets and JavaServer Pages (JSP) based applications. These applications are typically form-based, and are comprised of one or more HTML pages with which the user interacts to complete a task or set of tasks. JSF tackles the following challenges associated with these applications:

- Managing UI component state across requests
- Supporting encapsulation of the differences in markup across different browsers and clients
- Supporting form processing (single multi-page form, or more than one form per page)
- Providing a strongly typed event model that allows the application to write server-side handlers (independent of HTTP) for client generated events
- Validating request data and providing appropriate error reporting
- Enabling type conversion when migrating markup values (Strings) to and from application data objects (which are often not Strings)
- Handling error and exceptions, and reporting errors in human-readable form back to the application user
- Handling page-to-page navigation in response to UI events and model interactions.

For instructions on building Java Server Faces portlets, refer to Section 5.4.2, "Building JSF Portlets."

3.2 Java Server Page (JSP) and HTML Portlets

JSP portlets and HTML portlets point to JSP or HTML files for their content. These portlets can be simple to implement and deploy, and they provide basic functionality quickly. However, this type of portlet does not enforce separation of business logic and the presentation layer. As the application grows, the portlet often becomes harder to maintain as you try to update the web application and share code. JSP portlets are not well-suited for advanced portlet navigation.

When using JSP pages as part of a page flow portlet, you must make sure that requests adhere to WebLogic Portal scoping requirements. For more information about JSP

portlets and page flow scoping, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

For instructions on building JSP portlets, see Section 5.4.1, "Building JSP and HTML Portlets."

3.3 Java Portlets

JSR 268 (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 cover areas such as presentation, aggregation, security, and portlet life cycle.

A Java portlet is expressed as a Java class. This type of portlet accommodates portability across platforms, and does not require the use of portal server-specific JSP tags. The behavior is similar to a servlet. Java portlets produced using WebLogic Portal can be used universally by any other vendor's application server container that supports JSR 268.

For instructions on building Java portlets, refer to Chapter 6, "Building Java Portlets."

3.4 Java Page Flow Portlets

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

> **Note:** Oracle recommends you choose an alternative supported Portlet type, such as JSF, over Java Page Flow / Apache Beehive technology. WLP will continue to support Page Flows and Apache Beehive, however the technology will be deprecated over time and will not see significant new enhancements.

A Java page flow portlet uses Apache Beehive page flows to retrieve its content. This portlet type allows you to separate the user interface code from navigation control and other business logic, and provides the ability to implement both simple and advanced portlet navigation.

The Page Flow framework is built on top of the Struts application framework. The Struts framework is a popular, reliable standard that is widely used to quickly create robust and navigable web applications. The page flow framework adds valuable data binding facilities to the Struts standard, and the portal framework provides a scoping capability for page flow portlets so that multiple page flows can be supported in a single portal. You can use resources such as Java controls and web services.

For instructions on building Java page flow portlets, refer to Section 5.4.8, "Building Java Page Flow Portlets."

3.5 Struts Portlets

Note: Apache Struts is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Struts portlets are based on the Struts framework, which is an implementation of the Model-View-Controller (MVC) architecture. The MVC architecture provides a model for separating the different components and roles of the application logic. This development framework helps you create portlets that are easier to maintain over time.

Typically, native Struts development requires management and synchronization of multiple files for each action, form bean, as well as the Struts configuration file. Even in the presence of tools that help edit these files, developers are still exposed to all the underlying plumbing, objects, and configuration details.

For instructions on building Struts portlets, refer to Section 5.4.6, "Building Struts Portlets."

3.6 Browser (URL) Portlets

Browser portlets display HTML content from an external URL. Unlike other portlet types that are limited to displaying data contained within the portal project, browser portlets display URL content that is external from the portal project.

An advantage of browser portlets is that no development tasks are required to implement it, either from the Oracle Enterprise Pack for Eclipse workbench or from the WebLogic Portal Administration Console. However, keep in mind that WebLogic Portal does not provide a mechanism to develop content for this type of portlet; the definition of the portlet merely contains the external URL to display. For example, no mechanisms exist to dynamically influence the external content's URL; no support exists for portlet preferences, portlet modes, and so on. Browser portlets do not track the URL through the user's interaction with remote content – page refreshes cause the content of the URL specified in the portlet definition to be displayed.

WebLogic Portal implements a browser portlet using an IFRAME. You can override the default implementation mechanism using more advanced development techniques.

The content of the browser portlet is completely disconnected from the portal. The embedded application must manage the navigational state of the portlet.

For instructions on building Browser portlets, refer to Section 5.4.4, "Building Browser Portlets."

3.7 Clipper Portlets

Clipping is an easy technique for including content in your portal. You can clip all or part of another web site. Users can effectively view and interact with content from

another web site without leaving the portal. For detailed information on creating clipper portlets, see Chapter 7, "Creating Clipper Portlets."

3.8 Remote (Proxy) Portlets

WebLogic Portal supports the Web Services for Remote Portlets (WSRP) standard, a product of the OASIS standards body. Portlets that are written to meet this standard, which includes a WSDL portlet description, can be hosted within a producer application, and surfaced in a consumer application. Moreover, the WebLogic Portal Administration Console facilitates access to WSRP producer applications in a local portal.

WebLogic Portal can act as either a WSRP remote producer or as a consumer. When acting as a consumer, WebLogic Portal's remote—or proxy—portlets are WSRP-compliant. These portlets present content that is collected from WSRP-compliant producers, allowing you to use external sources for portlet content, rather than having to create its content or its structure yourself.

Because setting up a remote portlet is a fundamental task in creating a federated portlet environment, the task of creating a remote portlet is described in detail within the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

3.9 Portlet Type Summary Table

Table 3–1 summarizes the characteristics of each portlet type so that you can quickly determine the advantages and disadvantages of each type.

Table 3–1 Portlet Type Summary Table

Type	Advantages	Disadvantages
JSF	Allows component-based development of pages that can handle their own intra-page events. Simplifies separation of the user interface code from navigation control and other business logic. Provides the ability to implement both simple and advanced portlet navigation. Allow you to quickly leverage Java controls, web services, and business processes. **Note:** Oracle recommends JSF as the best approach to implementing portal applications. As one of the active members of the JSF expert group, Oracle has committed itself to evolve and support JSF.	All postbacks to a JSF application are expected to be done using a POST; the GET method is not supported.
JSP/HTML	Simple to implement and deploy. Provides basic functionality without complexity.	Does not enforce separation of business logic and presentation layer. Not well-suited for advanced portlet navigation.
Java (JSR 286)	Accommodates portability across platforms. Does not require the use of portal server-specific JSP tags. Behavior is similar to a servlet	Lack of advanced portlet features that are available with some other portlet types. Requires a deeper understanding of the J2EE programming model.

Table 3–1 (Cont.) Portlet Type Summary Table

Type	Advantages	Disadvantages
Java Page Flow	Allows separation of the user interface code from navigation control and other business logic. Provides the ability to implement both simple and advanced portlet navigation. Allow you to quickly leverage Apache Beehive controls, web services, and business processes. Provides a visual environment to build rich applications based on struts. **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.	Implementation is more complex. Advanced page flow features are not necessary for static or simple, one view portlets.
Browser	Allows a portlet to display content from a URL that is outside the portal project. Provides a "no development needed" portlet for quick implementation.	Less control over formatting. Lacks certain features of other portlet types, such as Content Path and Error Path. No interportlet communication support.
Clipper	Lets you subset or modify the contents of a remote web page. The portal can potentially access the clipper portlet's content.	Clipped content is included directly in the portal page, allowing the potential for overlapping with other parts of the portal.
Struts	Provides a flexible control layer based on standard technologies like Java Servlets, JavaBeans, ResourceBundles, and XML. Provides a more structured approach for creating and maintaining complex applications. Useful for importing existing applications. **Note:** Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.	Not quite as robust as page flow portlets, which are based on Apache Beehive. For new development, page flow portlets provide a better solution.
Remote	Allows you to functionally and operationally de-couple applications within your portal. Allows you to leverage external sources for portlet content. Depending on the environment, might improve performance.	Implementation is more complex. Your application's features might not be able to be as robust; for example, some Javascript might not perform correctly. Depending on the environment, might have a performance cost. For more about performance with remote portlets, refer to Section 10.3, "Remote Portlets."

Part II

Development

During the development phase, you use Oracle Enterprise Pack for Eclipse to create portlets, pages, and books. During development, you can implement federation and interportlet communication strategies. In the development stage, careful attention to best practices is crucial.

For a detailed description of the architecture phase of the portal life cycle, see the *Oracle Fusion Middleware Overview for Oracle WebLogic Portal*.

Part II contains the following chapters:

4

Understanding Portlet Development

This chapter provides conceptual and reference information that you might find useful as you begin to develop portlets with WebLogic Portal. For a detailed description of the components that are involved in portlet design, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*. For instructions on how to create each type of portlet, refer to Section 5.4, "How to Build Each Type of Portlet."

This chapter contains the following sections:

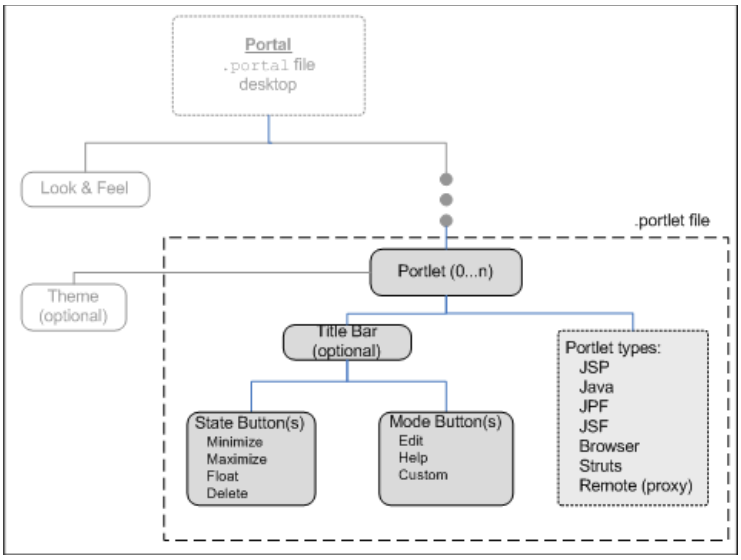
- Section 4.1, "Portlet Components"
- Section 4.2, "Resources for Creating Portlets"
- Section 4.3, "Portlet Rendering"
- Section 4.4, "JSP Tags and Controls in Portlets"
- Section 4.5, "Backing Files"
- Section 4.6, "Support for Apache Portals Bridges"

4.1 Portlet Components

Portlets are modular panes within a web browser that surface applications, information, and business processes. Portlets can contain anything from static HTML content to JSF applications to complex web services and process-heavy applications. Within a portal application, a portlet is represented as an XML file with a `.portlet` file extension. As you build portlets using Oracle Enterprise Pack for Eclipse, the XML elements and attributes are automatically built.

Figure 4–1 shows the components that make up a portlet, which are located in the `.portlet` file. Objects shown in gray text are described in more detail within the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Figure 4–1 Portlet Components



This section includes the following topics:

- Section 4.1.1, "Portlet Properties"
- Section 4.1.2, "Portlet Title Bar, Mode, and State"
- Section 4.1.3, "Portlet Preferences"
- Section 4.3.1, "Render and Pre-Render Forking"
- Section 4.3.2, "Asynchronous Portlet Content Rendering"
- Section 4.3.3, "Portlets as Popups (Detached Portlets)"

For more information about Look & Feel components, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

4.1.1 Portlet Properties

Portlet properties are named attributes of the portlet that uniquely identify it and define its characteristics. Some properties—such as title, definition label, and Content URI—are required; many other optional properties allow you to enable specific functions for that portlet such as scrolling, presentation properties, pre-processing (such as for authorization) and multi-threaded rendering. The specific properties that you use for a portlet vary depending on your expected use for that portlet.

For detailed information on portlet properties and how to set them, refer to Section 9.1, "Portlet Properties."

4.1.2 Portlet Title Bar, Mode, and State

When you create a portlet, you can choose whether or not it should have a title bar. Also, all portlets created with WebLogic Portal support modes and states. Modes affect the portlet's content; edit, help, and custom modes are available. States affect the rendering of the portlet; minimize, maximize, normal, float, and delete states are available.

You must enable the title bar on a portlet if you want to set modes and states for that portlet.

In certain situations your selection of a mode and state for a portlet might affect your ability to set up other portlet features, such as interportlet communication. For example, if you are setting up an event handler that listens to a portlet, you can select to execute the event handler only if the portlet to which it is listening is in a window that is *not minimized*.

For detailed instructions on setting portlet modes and states, refer to Section 9.5, "Portlet Appearance and Features."

4.1.3 Portlet Preferences

Portlets are distinct applications that you can reuse in a given portal. Once you create a portlet, you can instantiate it several times.

Along with the ability to create multiple instances of portlets, WebLogic Portal allows you to specify preferences for portlets. You use preferences to cause each portlet instance to behave differently yet use the same code and user interface. Portlet preferences provide the primary means of associating application data with portlets; this feature is key to personalizing portlets based on their usage.

Plan a portlet implementation that allows portlets to be as reusable as possible; planning for reuse simplifies your development and testing efforts because you can differentiate generic portlets by setting unique preferences.

For detailed instructions on setting portlet preferences, refer to Section 9.2, "Portlet Preferences."

4.2 Resources for Creating Portlets

Although the Portlet Wizard provides an easy way to create portlets, you might find that it is not your primary means of creating them. You can create a portlet in many ways, such as duplicating existing portlets or generating a portlet based on an existing JSP or JSF module. Many resources can provide the raw material for a portlet, including the following:

- **Portlets in J2EE Shared Libraries** – Portlets are provided with WebLogic Portal, which you can copy into your project and modify for your use. For example, you can add the Collaboration Portlets (pre-built portlets that are supplied with WebLogic Portal) to your Portal Web Project, and have access to Calendar, Task, Address Book, Discussion, and Mail portlets. For more information on the Collaboration portlets, including installation instructions, see Chapter 16, "Using the Collaboration Portlets."
- **Third-party portlets** – Special-purpose portlets provided as separate products by partner companies.
- **Existing JSPs, Struts modules, and Page Flows** – Existing resources that you can drag onto a portal page to automatically generate a portlet.

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

You can find detailed instructions on how to use these resources as the basis for a portlet in Chapter 5, "Creating Portlets."

4.3 Portlet Rendering

Portlet rendering consists of two life cycle stages:

- **Pre-rendering** – The background work to obtain necessary data or to perform pre-processing
- **Rendering** – The actual drawing of the portlet onto the portal page

General rendering topics are covered in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*. This section contains the following portlet-specific rendering topics:

- Section 4.3.1, "Render and Pre-Render Forking"
- Section 4.3.2, "Asynchronous Portlet Content Rendering"

4.3.1 Render and Pre-Render Forking

By default, pre-rendering and rendering for each portlet on a page is performed in sequence, and the portal page is not displayed until processing is complete for every portlet. This sequence can cause a noticeable delay in displaying the web page and might cause a user to think there is a problem with the web site. To prevent this situation, you can set up your portlets so that they perform pre-rendering and rendering tasks in parallel using multi-threaded *forked* processing.

Forking portlets at the rendering stage is supported for all portlet types. Pre-render forking is supported for the following portlet types:

- JSP
- Page Flow
- Java (JSR286)
- WSRP (consumer portlets only)

For detailed instructions on implementing forked portlets, refer to Section 10.4, "Portlet Forking."

Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

4.3.2 Asynchronous Portlet Content Rendering

Asynchronous portlet rendering allows the content of a portlet to be rendered independently of the surrounding portal page. When using asynchronous portlet rendering, a portlet is rendered in two HTTP requests. The first phase is the normal portal page request during which the portlet's non-content areas, such as the title bar, are rendered; a second request causes the portlet's content to render in place.

For detailed instructions on implementing asynchronous content rendering, refer to Section 10.5, "Asynchronous Portlet Content Rendering."

Tip: You can also enable asynchronous rendering for an entire portal desktop by setting a portal property in either Oracle Enterprise Pack for Eclipse or the WebLogic Portal Administration Console. For more information on asynchronous desktop rendering, see the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

4.3.3 Portlets as Popups (Detached Portlets)

WebLogic Portal supports the use of detached portlets. Detached portlets provide popup-style behavior.

For detailed instructions on using detached portlets, refer to Section 9.10, "Detached Portlets."

4.4 JSP Tags and Controls in Portlets

WebLogic Portal provides JSP tags that you can use within JSPs. Portlets can use JSPs as their content nodes, enabling reuse and facilitating personalization and other programmatic functionality. When you use the Palette view in Oracle Enterprise Pack for Eclipse, you can view available JSP tags and then drag them into the Source View of your JSP, and use the Properties view to edit elements of the code.

JSP tag libraries appear in the Design Palette whenever the JSP editor is open. If you do not see this palette, select **Window** > **Show View** > **Design Palette**. Select Tag Libraries from the palette's drop down menu to show only the tag libraries.

WebLogic Portal also provides custom Apache Beehive controls that make it easy for you to quickly add pre-built modules to your portal; custom Apache Beehive controls exist for event management, Visitor Tools, Community management, and so on. For example, most user management functionality can be easily exposed with a User Manager Control on a page flow.

Note: The term control is also used to refer to the portal (netuix) framework controls, such as desktop, book, page, and so on. These controls are referred to in the text as portal framework controls.

For information about the classes associated with WebLogic Portal's JSP tags, refer to the *Oracle Fusion Middleware Java API Reference for Oracle WebLogic Portal*.

For more information about using Apache Beehive controls within portlets, see Section 9.7, "JSP Tags and Controls in Portlets."

Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

4.5 Backing Files

The most common means of influencing portlet behavior within the control life cycle is to use a portlet backing file. A portlet backing file is a Java class that can contain methods corresponding to Portal control life cycle stages, such as init() and preRender(). You can use a portlet's backing context, an abstraction of the portlet control itself, to query and alter the portlet's characteristics. For example, in the init() life cycle method, a request parameter might be evaluated, and depending on the

parameter's value, the portlet backing context can be used to specify whether the portlet is visible or hidden.

Backing files can be attached to portals either by using Oracle Enterprise Pack for Eclipse or coding them directly into a `.portlet` file.

For detailed instructions on implementing backing files, refer to Section 9.4, "Backing Files."

4.6 Support for Apache Portals Bridges

WLP supports Apache Portals Bridges, a technology that provides support for JSR 168 compliant portlet development using common web frameworks like Struts, JSF, PHP, and others.

If using the Apache Portals Bridges, the WLP implementation of the ServletContextProvider interface is available as com.bea.portlet.container.ServletContextProviderImpl. For more information on Portals Bridges, see `http://portals.apache.org/bridges`.

5 Creating Portlets

This chapter describes the most common ways to create portlets, including the Portlet Wizard and the use of out-of-the-box portlets. This chapter also contains instructions for building each type of portlet that is supported by WebLogic Portal.

Before you begin, be sure you are familiar with the concepts associated with creating portlets, as described in Chapter 4, "Understanding Portlet Development."

This chapter contains the following sections:

5.1 Supported Portlet Types

The following portlet types are supported by WebLogic Portal:

- **Java Server Faces (JSF) Portlets** – JSF portlets produced using WebLogic Portal conform to the JSR 127 specification.
- **Java Server Page (JSP) and HTML Portlets** – JSP portlets and HTML portlets point to JSP or HTML files for their content.
- **Java Portlets (JSR 286)** – Java portlets produced using WebLogic Portal can be used universally by any vendor's application server container that supports JSR 286.
- **Browser (URL) Portlets** – Browser portlets display HTML content from an external URL; no development tasks are required to implement them.
- **Clipper Portlets** – A clipper portlet is a portlet that renders content from another web site. A clipper portlet can include all or a subset of another web site's content using a process called "web clipping." Clipper portlets are discussed in Chapter 7, "Creating Clipper Portlets."
- **Remote Portlets** – WebLogic Portal's remote portlets conform to the WSRP standard; they can be hosted within a producer application, and surfaced in a consumer application.

- **Java Page Flow Portlets** – Java page flow portlets use Apache Beehive page flows to retrieve their content. Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
- **Struts Portlets** – Struts portlets are based on the Struts framework, which is an implementation of the Model-View-Controller (MVC) architecture. Apache Struts is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

For a detailed discussion of each portlet type, refer to Chapter 3, "Portlet Types."

Note: WLP supports Apache Portals Bridges. For more information, see Section 4.6, "Support for Apache Portals Bridges."

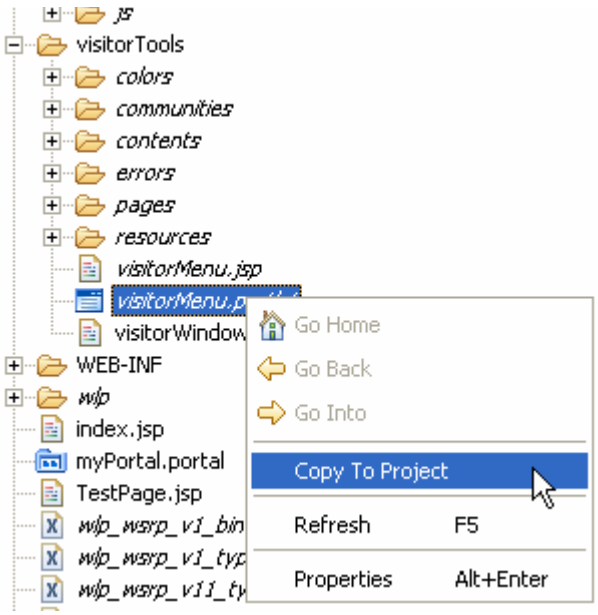
5.2 Portlets in J2EE Shared Libraries

You can copy portlets or other resources from a J2EE Shared Library into your portal application and modify them as needed. A portlet existing in your project will supersede a portlet of the same name in a J2EE Shared Library. To see a list of available portlets, you can use the Merged Projects View of the workbench; resources contained in J2EE Shared Libraries are shown in italic print. You can expand the tree to see the resources that are stored in the various modules. For a reference list of all the J2EE Libraries and their locations on your file system, you can select **Window** > **Preferences** > **WebLogic** > **J2EE Libraries**.

After you locate a portlet that you want to use, you can right-click the portlet in the Merged Projects View and select the **Copy to Project** option. Figure 5–1 shows an example of a J2EE Shared Library portlet in the Merged Projects view with the Copy to Project option selected.

Caution: Copying and modifying resources from installed J2EE Shared Libraries can present a problem if you upgrade WLP or install a patch in the future. For example, you might have to perform manual steps to incorporate product changes that affect those resources. For future upgrades and patch installations, WLP cannot fully support configurations where J2EE library resources have been copied to the project and modified.

Figure 5–1 Portlet Being Copied to a Project from Merged Projects View



For more information about J2EE Shared Libraries, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

5.3 Portlet Wizard Reference

An important tool that you can use to create portlets from scratch is the WebLogic Portal Portlet Wizard. The following sections describe the Portlet Wizard in detail:

- Section 5.3.1, "Order of Creation - Resource or Portlet First"
- Section 5.3.2, "Starting the Portlet Wizard"
- Section 5.3.3, "New Portlet Dialog"
- Section 5.3.4, "Select Portlet Type Dialog"
- Section 5.3.5, "Portlet Details Dialogs"

In general, you choose the portlet type on the first dialog of the wizard; when generating a portlet based on an existing resource, the Portlet Wizard automatically detects the portlet type whenever possible.

5.3.1 Order of Creation - Resource or Portlet First

This section provides an overview of the two methods you can use to begin creating a portlet—creating the portlet resource information/file first or creating the portlet itself first.

5.3.1.1 Creating the Resource First

You might already have a JSP file, for example, that you want to use as the basis for a portlet. (In addition to JSP files, you can drag other resources onto the portal (such as content selectors) to automatically start the portlet wizard.)

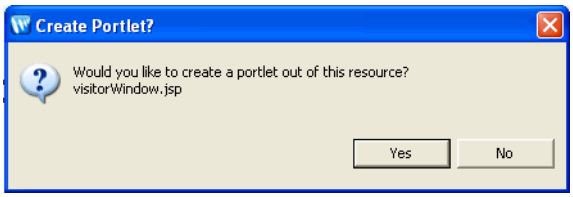
If you have an existing resource that you want to use as the basis of a portlet, follow these steps:

1. Create or open a portal's `.portal` file in Oracle Enterprise Pack for Eclipse.

2. Drag the resource, such as a JSP file, into one of the portal's placeholder areas in the design view in the editor.

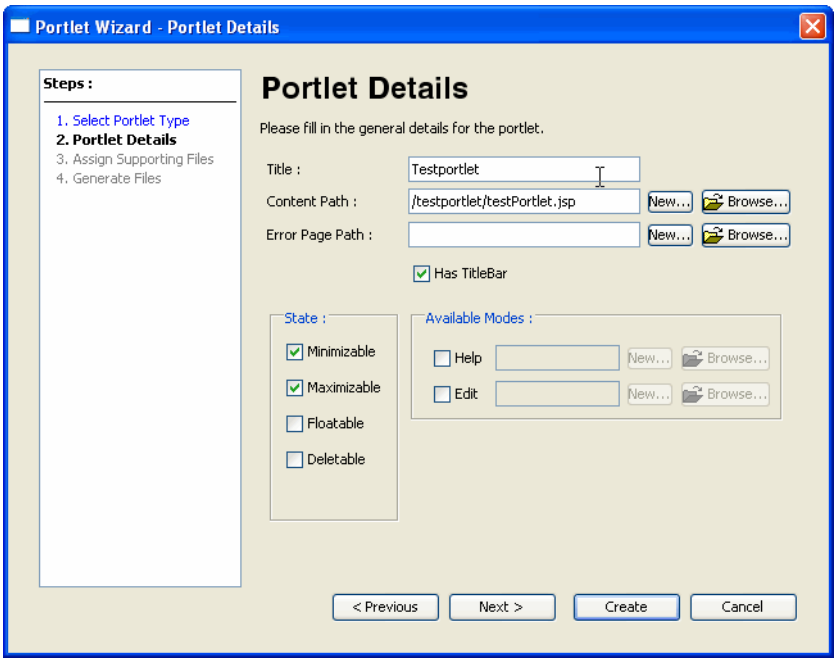
   Oracle Enterprise Pack for Eclipse prompts you with a dialog similar to the example in Figure 5–2.

Figure 5–2 Portlet Wizard Prompt Following Drag and Drop of a Resource



If you click **Yes**, the Portlet Wizard uses information from the resource file to begin the process of creating a portlet, and displays the Portlet Details dialog. Figure 5–3 shows an example:

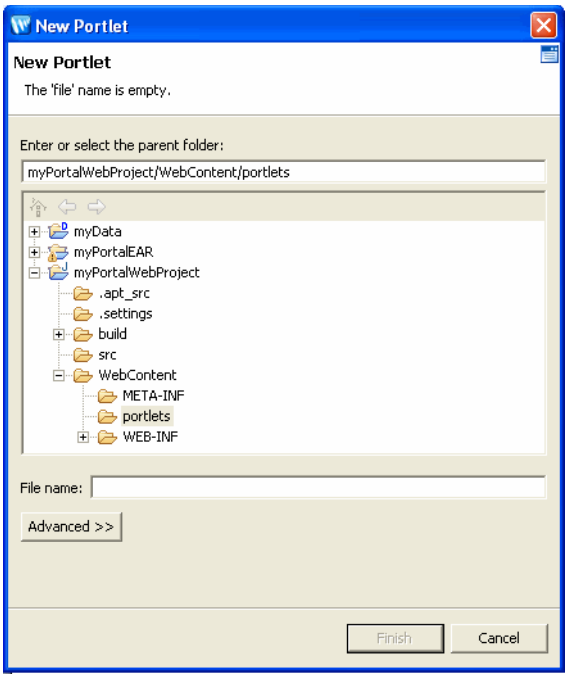
Figure 5–3 Example Portlet Wizard Details Dialog Following Drag and Drop of a Resource



5.3.1.2 Create the Portlet First

If you do not have an existing source file to start with, you can create the portlet using the New Portlet dialog and the Portlet Wizard. To do so, right-click a folder in your portal web project and select **New** > **Portlet.** Figure 5–4 shows an example of the New Portlet dialog.

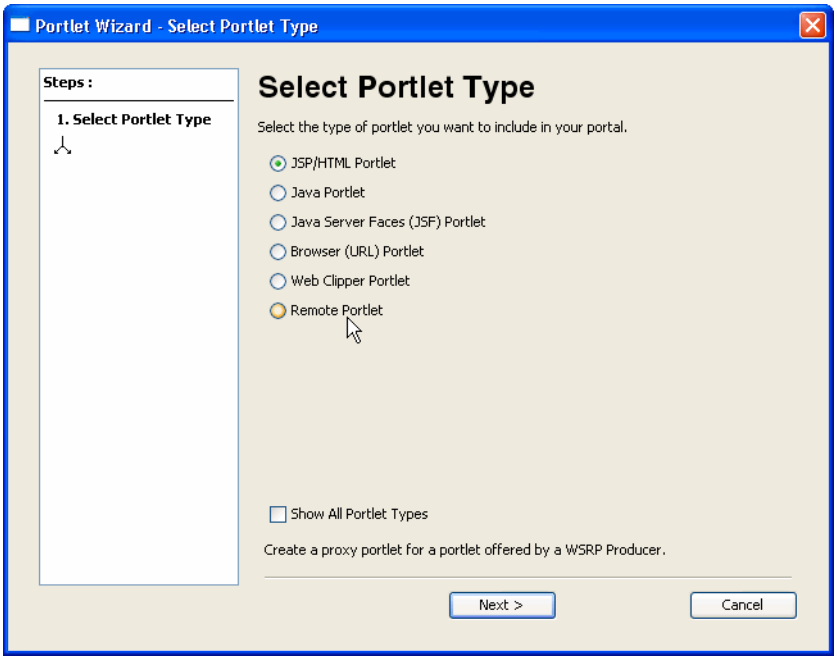
Figure 5–4 Portlet Wizard New File Dialog



After you confirm or change the parent folder, name the portlet, and click **Finish**, the Portlet Wizard begins and displays the Select Portlet Type dialog. Figure 5–5 shows an example dialog.

Detailed instructions for creating each type of portlet are contained in Section 5.4, "How to Build Each Type of Portlet."

Figure 5–5 Portlet Wizard - Select Portlet Type Dialog

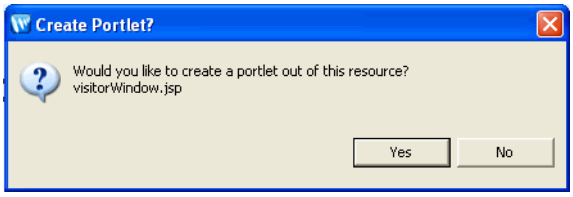


5.3.2 Starting the Portlet Wizard

Oracle Enterprise Pack for Eclipse invokes the Portlet Wizard any time you perform one of these operations:

- Select **File** > **New** > **Portlet** from Oracle Enterprise Pack for Eclipse's top-level menu, or right-click a folder in your web application, and select **New** > **Portlet**. After you name the portlet and click **Next**, the Portlet Wizard starts.
- Drag and drop a resource such as a JSP from the Package Explorer view onto a placeholder area of an open portal (in other words, a *portal_name*.portal file is open in the editor view of the workbench.) Oracle Enterprise Pack for Eclipse prompts you with a dialog similar to the example in Figure 5–6.

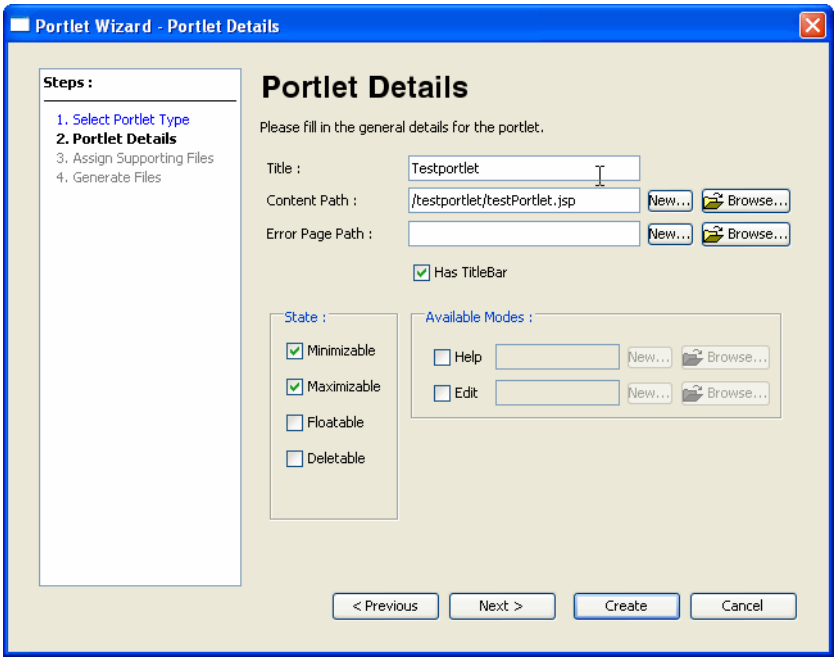
Figure 5–6 Portlet Wizard Prompt Following Drag and Drop of a Resource



If you click **Yes**, the Portlet Wizard uses information from the resource file to begin the process of creating a portlet.

- Right-click an existing resource such as a JSP file, a portal placeholder, or a portal content selector; then select **Generate Portlet** from the context menu. The Portlet Wizard displays the Portlet Details dialog. Figure 5–7 shows an example of a dialog after right-clicking a JSP file.

Figure 5–7 Portlet Wizard - Portlet Details Example for JSP Resource



5.3.3 New Portlet Dialog

When you use **File** > **New** > **Portlet** to create a new portlet, a New Portlet dialog displays before the Portlet Wizard begins. Figure 5–4 shows an example of the New Portlet dialog.

The parent folder defaults to the location from which you selected to add the portlet.

This dialog requires that you select a project and directory for the new portlet, and provide a portlet file name. (The file name appears in the Package Explorer view after you create the portlet.) The **Next** button is initially disabled; the button enables when you select a valid project/directory and portlet name. If you select an invalid portal project in the folder tree on this dialog, an error message appears in the status area near the top of the dialog explaining that the project is not a valid portal project. You cannot continue until you have selected a valid project (if one is available).

Note: With WebLogic Portal Version 9.2 and later versions, the option to convert a non-portal project to a portal project is not offered. For information on how to integrate portal J2EE Shared Libraries into an already existing project, see the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

5.3.4 Select Portlet Type Dialog

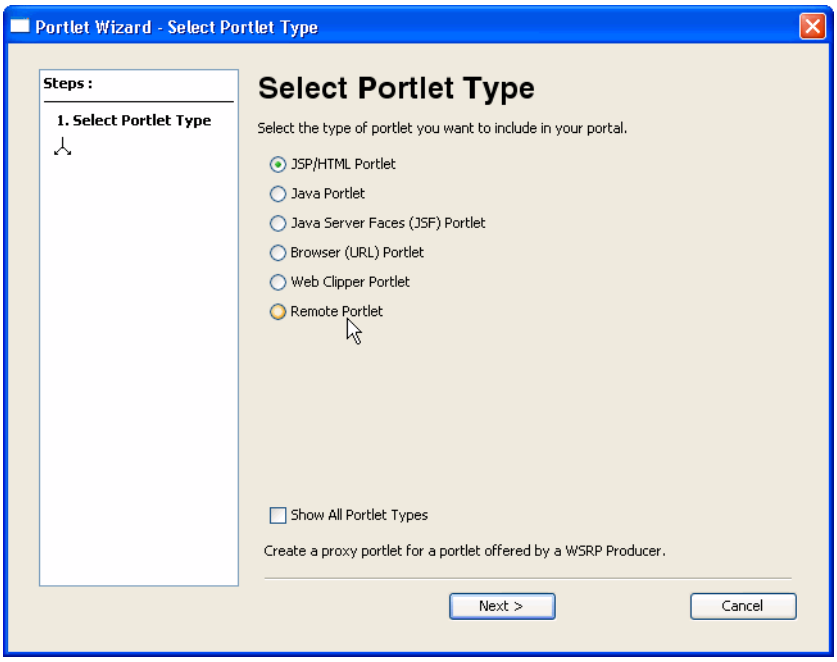
When the Portlet Wizard starts, it determines the valid portlet types to offer on the Select Portlet Type dialog, based on the type of project that you are working in.

For example, if you are working within a Portal Web Project that has only the WSRP-Producer and Portal Framework Struts facets installed, it does not have the full set of portal libraries. In this case, only the Struts portlet type is available a valid selection; the other portlet types are not included in the Select Portlet Type dialog. If you were to install the optional Apache Beehive facets, then the Java Page Flow portlet option would appear.

If no valid portlet types exist based on the project type, an informational message displays.

Figure 5–8 shows the default configuration of the Select Portlet Type dialog (assuming the default WebLogic Portal facets are installed in the web project).

Figure 5–8 Portlet Wizard - Select Portlet Type Dialog



The **Show All Portlet Types** option forces all portlet types to appear in the Select Portlet Type dialog even if their modules were not installed. For information on installing the optional Page Flow and/or Struts support, see "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Caution: If you create and publish portlets that require modules that have not been properly installed, unexpected behavior and server runtime errors can occur.

5.3.5 Portlet Details Dialogs

The Portlet Details dialogs that display after you select a portlet type vary according to the type of portlet you are creating. The portlet-building tasks that are described in Section 5.4, "How to Build Each Type of Portlet" contain detailed information about each data entry field of the portlet details dialogs.

5.4 How to Build Each Type of Portlet

The following sections describe how to create each type of portlet supported by WebLogic Portal:

- Section 5.4.1, "Building JSP and HTML Portlets"
- Section 5.4.2, "Building JSF Portlets"
- Section 5.4.3, "Building Java Portlets"
- Section 5.4.4, "Building Browser Portlets"
- Section 5.4.5, "Building Clipper Portlets"
- Section 5.4.6, "Building Struts Portlets"
- Section 5.4.7, "Building Remote Portlets"

- Section 5.4.8, "Building Java Page Flow Portlets"

5.4.1 Building JSP and HTML Portlets

JSP portlets are very similar to simple JSP files. In most cases you can use existing JSP files to build portlets from them. JSP portlets are recommended when the portlet is simple and doesn't require the implementation of complex business logic. Also, JSP portlets are ideally suited for single page portlets.

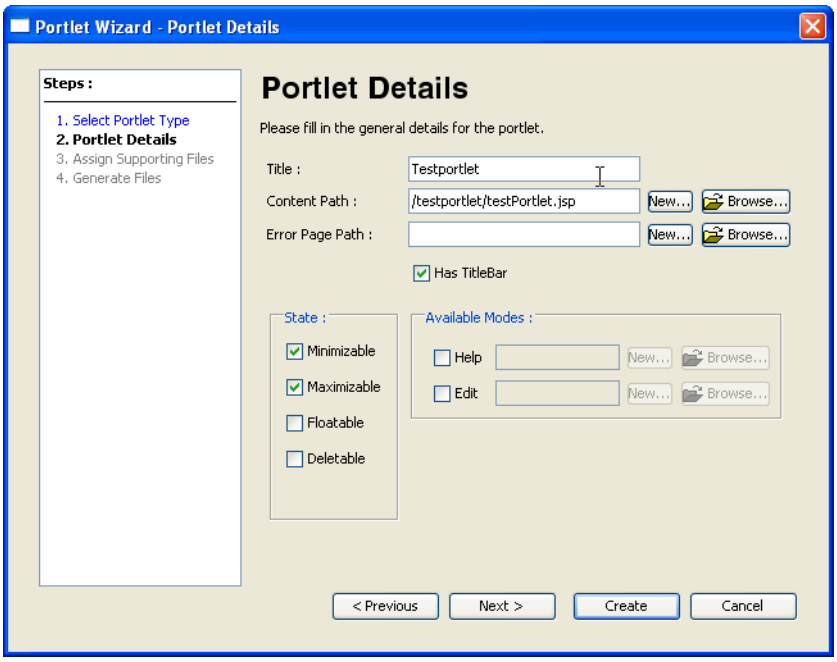
There are several ways to invoke the Portlet Wizard, as explained in the section Section 5.3.2, "Starting the Portlet Wizard." This description assumes that you create a portlet based on an existing JSP file.

To create a JSP portlet, follow these steps:

1. Right-click a JSP file and select **Generate Portlet** from the menu.

   The Portlet Wizard displays the Portlet Details dialog; Figure 5–9 shows an example.

Figure 5–9 Portlet Wizard - JSP Portlet Details Dialog



2. Specify the values you want for this portlet, following the guidelines shown in Table 5–1.

Table 5–1 Portlet Wizard - JSP Portlet Data Entry Fields

Field	Description
Title	The value for the Title might already be filled in.You can change the value if you wish.
Content Path	The value for the Content URI (location of the JSP) is probably already filled in. You can change this value if you wish either by entering the path to a JSP or browsing to it. You can also create a new JSP on the fly either by entering a name in the field or by choosing the New button.
Error Page Path	To designate a default error page to appear in case of an error, indicate the path to the desired URI. You can also create a new error JSP on the fly either by entering a name in the field or by choosing the New button.

Table 5–1 (Cont.) Portlet Wizard - JSP Portlet Data Entry Fields

Field	Description
Has Titlebar	If you want your portlet to have a title bar, check this box. The displayed title matches the value in the Title field. In order for a portlet to have changeable states or modes, the portlet must have a title bar. See also Section 9.5, "Portlet Appearance and Features."
State	Select the desired check boxes to allow the user to minimize, maximize, float, or delete the portlet. For a more detailed description of portlet states, refer to Section 9.5.5, "Portlet States."
Available Modes	The basic modes are Help and Edit. You can enable access to Help from the portlet or you can allow the user to edit the portlet. To enable an option, select the desired check box and provide the path to the JSP page that will provide the appropriate function. You can either browse to an existing file or click **New** to create a new JSP. For a more detailed description of portlet modes, refer to Section 9.5.2, "Portlet Modes."

3. Click **Create** to create the portlet, or click **Next** to assign supporting files. For more information on assigning supporting files see Section 5.5, "Assigning Supporting Files."

   The Oracle Enterprise Pack for Eclipse window updates, adding the *Portlet_Name*.portlet file to the display tree; by default, Oracle Enterprise Pack for Eclipse places the portlet file in the same directory as the content file.

5.4.2 Building JSF Portlets

As a portlet developer, it is important to know which version of JSF your web application is configured to use: JSF 1.2 (the default for WLP 10.3.2) or JSF 1.1. The JSF version determines the underlying technology used for the JSF portlet, and it also affects how you develop your portlet.

By default, WLP 10.3.2 web applications are configured to use Java Server Faces (JSF) 1.2. In this case, the IDE uses the Java 2.0 Portlet (JSR-286) with the JSR-329 Faces Portlet Bridge. This bridge allows the JSF portlet to operate within a JSR-286 compliant portlet container. The resulting portlet is a standard Java 2.0 portlet (JSR-286). The portlet includes all of the JSR-286 portlet features and properties. The Faces front page is referenced by the portlet `<init-param>` property. This Java portlet also uses the `GenericFacesPortlet` class, which is the standard Java base class for Faces portlets. The portlet editor indicates that this JSR- 286 portlet is a JSF portlet by placing *bridge to <faces home page JSP>* in the title bar. For an in-depth discussion, see Chapter 8, "Working With JSF-Java Portlets." For more information on Java portlets and their capabilities, see Chapter 6, "Building Java Portlets."

If your portal web application is configured to use JSF 1.1, the non-standard or native WLP bridge for JSF portlets is used instead. If the `facelet` facet is installed, the default behavior changes to always use the native WLP bridge for JSF portlets without regard to what version of JSF (1.1 or 1.2) is selected. The IDE uses the native WLP JSF portlet bridge and constructs a JSP-backed portlet with a `<facesContent>` element. This non-standard bridge is the same bridge used in previous versions of WLP. For an in-depth discussion on these types of portlets, consult the chapter "Working With JSF Portlets" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal* 10.3.0 at `http://download.oracle.com/docs/cd/E13155_01/wlp/docs103/portlets/jsf.html`.

For an in-depth discussion of JSF portlet development including best practices, see Chapter 8, "Working With JSF-Java Portlets." For more information on Java portlets and their capabilities, see Chapter 6, "Building Java Portlets."

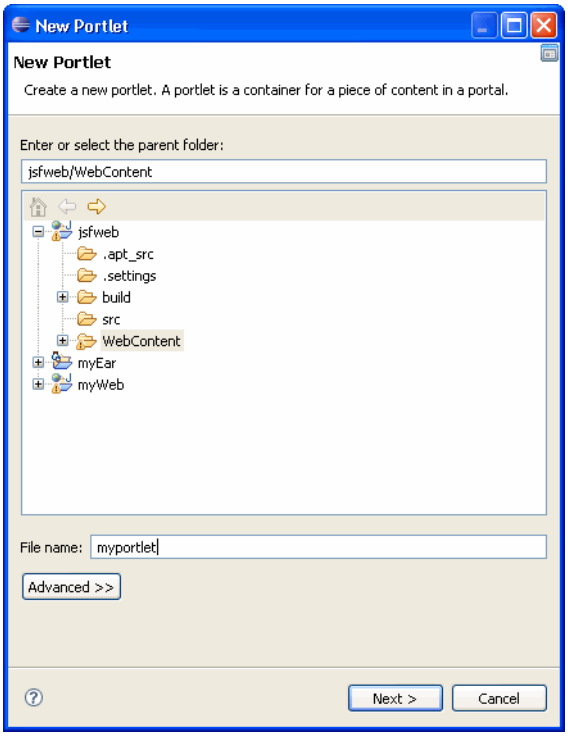
To create a JSF portlet, follow these steps:

Note: The portlet creation wizard differs somewhat depending on whether you are using JSF 1.2 or 1.1. In describing the wizard in this section, these differences are highlighted.

1. Right-click in the Package Explorer view, within the web content directory, and select **New** > **Portlet** from the menu.

   The New Portlet dialog displays. Figure 5–10 shows an example of the New Portlet dialog.

Figure 5–10 Portlet Wizard - New Portlet Dialog



   The parent folder defaults to the location from which you selected to add the portlet.

2. Edit the parent folder field if needed to indicate the project and directory for the new portlet.

   The **Next** button is initially disabled; the button enables when you select a valid parent folder and portlet name. If you select an invalid portal project in the folder tree on this dialog, an error message appears in the status area near the top of the dialog explaining that the project is not a valid portal project.

3. Type a file name for the new portlet.

4. Click **Next** to continue.

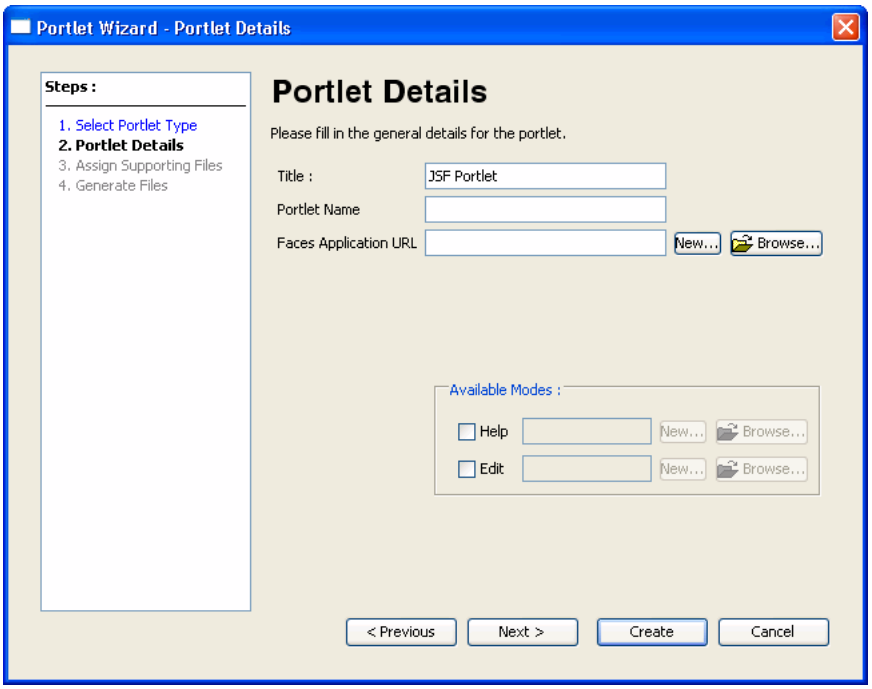
   The Portlet Wizard displays the Select Portlet Type dialog.

5. Select the JSF portlet option. If your web application is configured with JSF 1.2 (the default), select the Java Server Faces (JSF) Portlet option. If the web application uses JSF 1.1 or facelets, select the **Java Server Faces (JSF) Content** option. The

wizard automatically displays the correct option depending on your JSF configuration.

The Portlet Wizard displays the Portlet Details dialog; Figure 5–11 shows an example.

Figure 5–11 Portlet Wizard - JSF Portlet Details Dialog for JSF 1.2 Configuration



6. Specify the values you want for this portlet. If your web application is configured to use JSF 1.2, follow the guidelines in Table 5–2. If your web application uses JSF 1.1, follow the guidelines in Table 5–3.

Table 5–2 Portlet Wizard - JSF Portlet Data Entry Fields for a JSF 1.2 Configuration

Field	Description
Title	The value for the portlet title, which displays in the title bar if enabled.
Portlet Name	A unique name for the portlet.
Faces Application URL	Enter or browse to the inital URL of the Faces Application to be portletized. This sets the javax.portlet.faces.defaultViewId.view `<init-param>` which can be later changed through the portlet editor.
Available Modes	The basic modes are Help and Edit. You can enable access to Help from the portlet or you can allow the user to edit the portlet. To enable an option, select the desired check box and provide the path to the JSP page that will provide the appropriate function. You can either browse to an existing file or click **New** to create a new JSP. For a more detailed description of portlet modes, refer to Section 9.5.2, "Portlet Modes."

Table 5–3 Portlet Wizard - JSF Portlet Data Entry Fields for a JSF 1.1 Configuration

Field	Description
Title	The value for the portlet title, which displays in the title bar if enabled.
Content Path	Enter or browse to the front page JSP file or facelet (.xhtml) file of your Faces application. Click New to create a new Faces JSP or facelet. This file must be placed within the same web project that contains the JSF portlet. If your web application uses JSF 1.1 or the Facelet facet, follow the guidelines in Table 5–3.
Error Page Path	**Note:** Error pages are not supported with JSF portlets.
Has Titlebar	If you want your portlet to have a title bar, check this box. The displayed title matches the value in the Title field. In order for a portlet to have changeable states or modes, the portlet must have a title bar.
State	Select the desired check boxes to allow the user to minimize, maximize, float, or delete the portlet. For a more detailed description of portlet states, refer to Section 9.5.5, "Portlet States."
Available Modes	You can enable access to Help from the portlet or you can allow the user to edit the portlet. To enable an option, select the desired check box and provide the path to the file that will provide the appropriate function. For a more detailed description of portlet modes, refer to Section 9.5.2, "Portlet Modes."

7. Click **Create**, or click **Next** to assign supporting files. For more information on assigning supporting files see Section 5.5, "Assigning Supporting Files."

   The Oracle Enterprise Pack for Eclipse window updates, adding the *Portlet_Name*.portlet file to the display tree. The portlet automatically opens in the portlet editor. Figure 5–12 shows the portlet editor for a portlet that is configured for JSF 1.2. The portlet shown is a Java 2.0 portlet that uses the JSF 329 bridge.

Figure 5–12 Portlet Editor for a JSF 1.2 Configuration Portlet

5.4.3 Building Java Portlets

Java portlets are based on the JSR 286 specification that establishes rules for portlet portability. Java portlets are intended for software companies and other enterprises that are concerned with portability across multiple portlet containers.

For detailed information on creating and configuring Java portlets for WLP, see Chapter 6, "Building Java Portlets."

5.4.4 Building Browser Portlets

Browser portlets, also called Content URI portlets, are basically HTML portlets that use URLs to retrieve their content. Unlike other portlet types that are limited to displaying data contained within the portal project, browser portlets can display URL content that is outside from the portal project.

Tip: A clipper portlet also lets you include remote web content in a portal page. For information on clipper portlets and how they differ from browser portlets, see Chapter 7, "Creating Clipper Portlets."

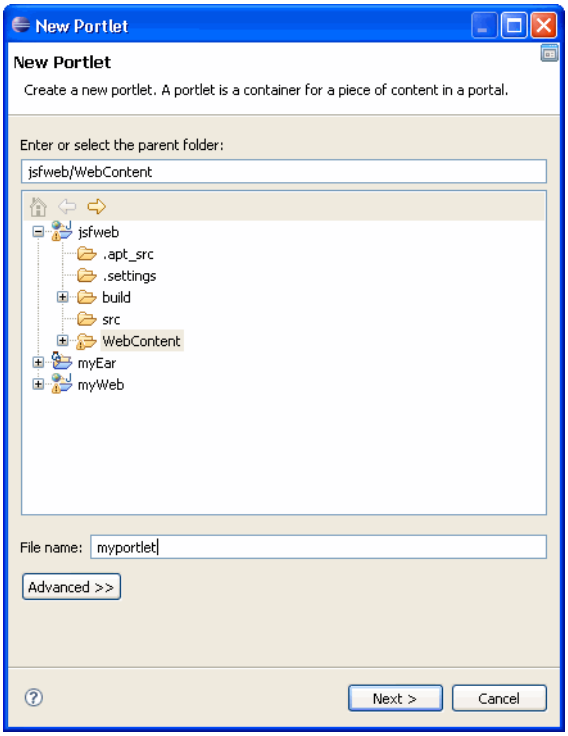
There are several ways to invoke the Portlet Wizard, as explained in the section Section 5.3.2, "Starting the Portlet Wizard." This description assumes that you right-click in the Package Explorer view tree within a portal project and select **New** > **Portlet** from the menu.

To create a browser portlet, follow these steps:

1. Right-click in the Navigation tree within a portal project and select **New** > **Portlet** from the menu.

   The New Portlet dialog displays. Figure 5–13 shows an example of the New Portlet dialog.

Figure 5–13 Portlet Wizard - New Portlet Dialog



   The parent folder defaults to the location from which you selected to add the portlet.

2. Edit the parent folder field if needed to indicate the project and directory for the new portlet.

   The **Finish** button is initially disabled; the button enables when you select a valid parent folder and portlet name. If you select an invalid portal project in the folder tree on this dialog, an error message appears in the status area near the top of the dialog explaining that the project is not a valid portal project.

3. Type a file name for the new portlet.

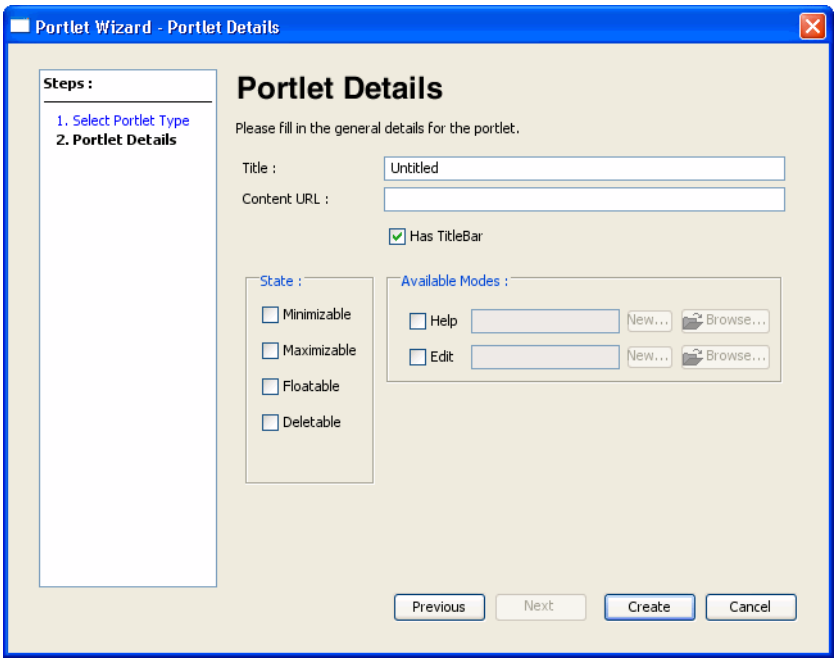
4. Click **Finish** to continue.

   The Portlet Wizard displays the Select Portlet Type dialog.

5. Click **Browser (URL) Portlet** and then click **Next**.

   The Portlet Wizard displays the Portlet Details dialog; Figure 5–14 shows an example.

Figure 5–14 Portlet Wizard - Browser Portlet Details Dialog



6. Specify the values you want for this portlet, following the guidelines shown in Table 5–4.

Table 5–4 Portlet Wizard - Browser Portlet Data Entry Fields

Field	Description
Title	The title for the portlet. This value appears in the title bar of the portlet in the editor view of the Oracle Enterprise Pack for Eclipse workbench.
Content URL	The value for the Content URL (external URL) that the portlet should use to retrieve its information. A validator checks the format of the URL that you enter, and a message notifies you if the URL is not properly formatted. You can either change the URL or ignore the warning and continue with the URL as is.
Has Titlebar	If you want your portlet to have a title bar, check this box. The displayed title matches the value in the Title field. In order for a portlet to have changeable states or modes, the portlet must have a title bar.
State	Select the desired check boxes to allow the user to minimize, maximize, float, or delete the portlet. For a more detailed description of portlet states, refer to Section 9.5.5, "Portlet States."
Available Modes	The basic modes are Help and Edit. You can enable access to Help from the portlet or you can allow the user to edit the portlet. To enable an option, select the desired check box and provide the path to the JSP page that will provide the appropriate function. You can either browse to an existing file or click **New** to create a new JSP. For a more detailed description of portlet modes, refer to Section 9.5.2, "Portlet Modes."

7. Click **Create**.

The Oracle Enterprise Pack for Eclipse window updates, adding the *Portlet_Name*.portlet file to the display tree; by default, Oracle Enterprise Pack for Eclipse places the portlet file in the same directory as the content file.

Note: The internal implementation of Browser portlets depends on asynchronous portlet content rendering; because of this, the portlet attribute **Async Content** that is displayed in the Properties view is set to none and is read-only. For more information about asynchronous content rendering, refer to Section 10.5, "Asynchronous Portlet Content Rendering."

5.4.5 Building Clipper Portlets

A clipper portlet is a portlet that renders content from another web site. A clipper portlet can include all or a subset of another web site's content using a process called "web clipping." Clipper portlets are discussed in Chapter 7, "Creating Clipper Portlets."

5.4.6 Building Struts Portlets

Note: Apache Struts is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Use the Portlet Wizard to generate a portlet based on a Struts module, as explained in this section.

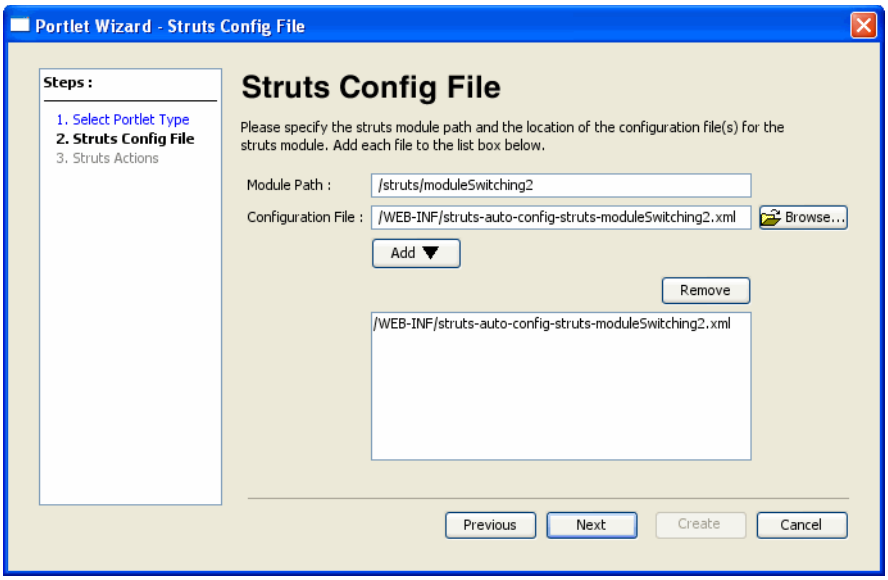
Before you can create a Struts portlet, you must first integrate your existing Struts application into your portal web application. For detailed information on integrating Struts applications into WebLogic Portal, see "Integrating Existing Web Applications into WebLogic Portal" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

> **Tip:** It is highly recommended that you fully develop and test a Struts application before attempting to host it within a portal. This helps to separate the complexities of developing a working Struts application from the additional issues involved in putting the Struts application into a portlet.

To create a Struts portlet, follow these steps:

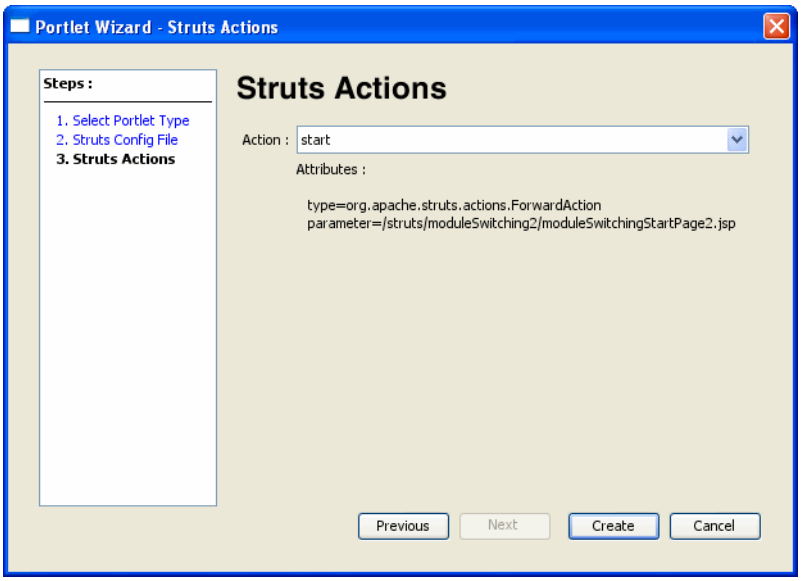
1. Right-click the Struts application module's XML configuration file located in the `WEB-INF` directory of the portal web application.
2. Select **Generate Portlet** from the menu. The wizard automatically collects and displays the module path and configuration file name(s) in the Struts Config File dialog. An example is shown in Figure 5–15. Use the **Browse** and **Add** buttons to locate and add additional configuration files, if applicable.

Figure 5–15 Struts Config File Dialog



3. Click **Next**.
4. In the Struts Actions dialog, specify an action for the Struts portlet. The actions that appear in the drop-down menu are based on entries in the configuration file(s) that were added previously.

Figure 5–16 Struts Actions Dialog



5. Click **Create**.

The Oracle Enterprise Pack for Eclipse window updates, adding the `Portlet_Name.portlet` file to the display tree; by default, Oracle Enterprise Pack for Eclipse places the portlet file in the directory that you specified in the Struts Module Path dialog of the wizard.

Configuring Multi-Part Form Data Support for a Struts Portlet

You can configure your Struts portlet to handle a multi-part struts action form after a server request has been posted.

Before you can create a Struts portlet, you must first integrate your existing Struts application into your portal web application. For detailed information about integrating Struts applications into WebLogic Portal, see "Integrating Existing Web Applications into WebLogic Portal" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

To add the multi-part form data support to a struts portlet:

1. In Oracle Enterprise Pack for Eclipse, in your portal web application, create a module for the Struts portlet, as described earlier in this section.
2. In your portal web application, create a JSP page that contains an action form with the attribute `enctype= "multipart/form-data"`.
3. Open the Strut module's XML configuration file, `struts-auto-config.xml`, located in the `WEB-INF` directory of your portal web application.
4. Configure your struts module to point to the newly created JSP page, and save `struts-auto-config.xml`.
5. In `struts-auto-config-upload.xml`, add the following entries:

   ```
   <controller inputForward="true"
   processorClass="com.bea.struts.adapter.action.AdapterRequestProcessor"
   multipartClass=" com.bea.struts.adapter.action.ScopedMultipartRequestHandler"/>
   ```

6. Save `struts-auto-config-upload.xml`.
7. To access the struts application directly, open the browser and use the following URL: `http://localhost:port/struts_webapp/module.do`.

   Where, *`localhost:port`* are the host name and port number where WebLogic Portal is deployed, *`struts_webapp`* is the name of your portal web application containing the struts module, and *`module.do`* is the module in which you want to implement the multi-part form data support.

   For example: `http://localhost:7001/StrutsUploadWeb13/upload.do`

   Note: If you want to run the struts portlet through WebLogic Portal, make sure your portlet points to the WebLogic Portal tag library, which is specified in the import statements at the beginning of a JSP file. If you do not use WebLogic Portal's HTML tab library, the page gets redirected outside of the portal page. As a result, the strut portlet's JSP page takes over the entire page.

5.4.7 Building Remote Portlets

Because remote portlet development is a fundamental task in a federated portlet environment, the task of creating remote portlets is described in detail within the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

The following types of portlets can be exposed with WSRP inside a WebLogic portal:

- JavaServer Faces (JSF) portlets
- JavaServer Pages (JSP) portlets
- Struts portlets

- Java portlets (JSR286; supported only for complex producers)
- Page Flow portlets

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

5.4.8 Building Java Page Flow Portlets

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

> **Note:** Oracle recommends you choose an alternative supported Portlet type, such as JSF, over Java Page Flow / Apache Beehive technology. WLP will continue to support Page Flows and Apache Beehive, however the technology will be deprecated over time and will not see significant new enhancements.
>
> You can edit the tag attributes manually in the JSP editor's Source tab or in the Properties view.
>
> For information about the Apache Beehive Netui documentation, see `http://beehive.apache.org/docs/1.0.2/`.
>
> For information about the Netui tags, see `http://beehive.apache.org/docs/1.0.2/netui/tags/index.html`.

You can use the Portlet Wizard to built a portlet that uses Apache Beehive Page Flows to retrieve its content.

To create a page flow portlet, follow these steps:

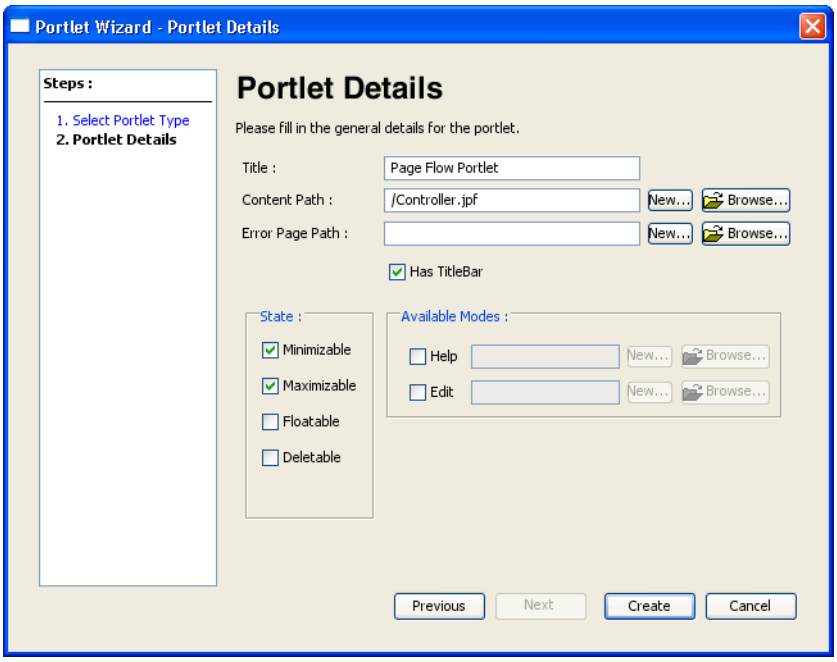
1. Right-click the folder where you want to store the page flow portlet. (The folder must be within the WebContent directory.)
2. Select **New** > **Portlet**.

   The **New Portlet** dialog displays.
3. Enter a name for the portlet and click **Next**.

   The Portlet Wizard displays the Select Portlet Type dialog.
4. In the Select Portlet Type page, select the **Show All Portlet Types** checkbox.
5. Select the Java Page Flow Portlet radio button and click **Next**.

   The Portlet Wizard displays the Portlet Details dialog; Figure 5–17 shows an example.

Figure 5–17 Portlet Wizard - JPF Portlet Details Dialog



6. Specify the values you want for this portlet, following the guidelines shown in Table 5–5.

Table 5–5 Portlet Wizard - JPF Portlet Data Entry Fields

Field	Description
Title	The title for this portlet, which displays in the title bar if you select to include one.
Content Path	The Page Flow Request URI. You can type a value here, or click the Browse button to open a class picker and select the appropriate class. If you use the class picker to choose a page flow class, this fully-qualified class name is converted to a URI path of a JPF. The JPF is referred to by the `.portlet` file when the portlet is created. If you enter or navigate to a `.java` that has no corresponding class in the project or J2EE Shared Libraries, the Portlet Wizard creates the `.java` file for the page flow. If multiple project source directories exist, then the wizard prompts you to store the new `.java` file in the source directory of your choice.
Error Page Path	To designate a default error page to appear in case of an error, check the box and indicate the path to the desired URI.
Has Titlebar	If you want your portlet to have a title bar, check this box. The displayed title matches the value in the Title field. In order for a portlet to have changeable states or modes, the portlet must have a title bar.
State	Select the desired check boxes to allow the user to minimize, maximize, float, or delete the portlet. For a more detailed description of portlet states, refer to Section 9.5.5, "Portlet States."
Available Modes	The basic modes are Help and Edit. You can enable access to Help from the portlet or you can allow the user to edit the portlet. To enable an option, select the desired check box and provide the path to the JSP page that will provide the appropriate function. You can either browse to an existing file or click **New** to create a new JSP. For a more detailed description of portlet modes, refer to Section 9.5.2, "Portlet Modes."

7. Click **Create**.

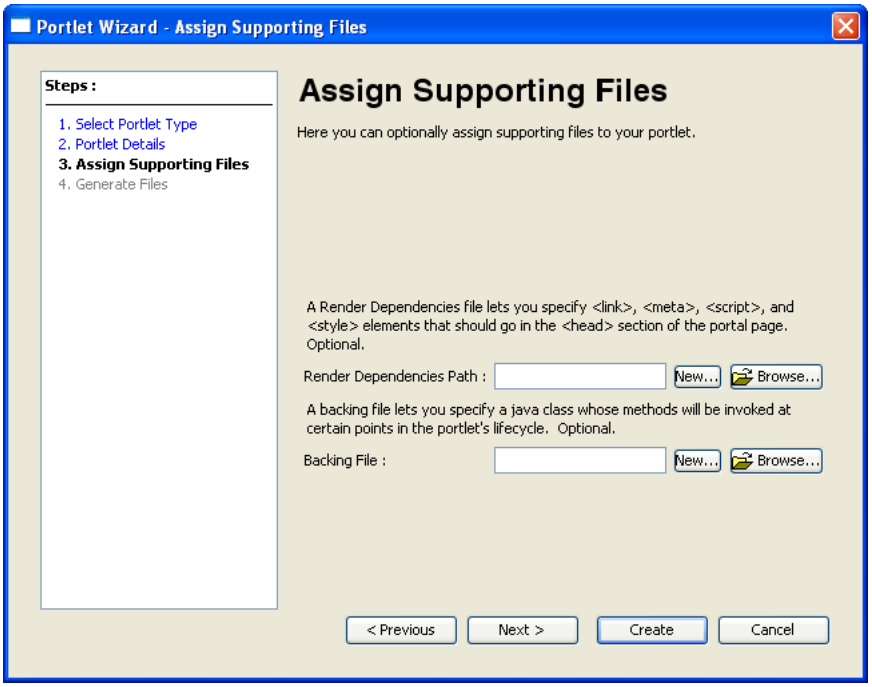
The Oracle Enterprise Pack for Eclipse window updates, adding the *Portlet_Name*.portlet file to the display tree; by default, Oracle Enterprise Pack for Eclipse places the portlet file in the same directory as the content file.

In order to fully understand the process of creating a page flow portlet, you should be familiar with the concept of Page Flows. For more information on using page flows with WebLogic Portal, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

5.5 Assigning Supporting Files

Some portlet types allow you to attach supporting files to the portlet. For these portlets, the Portlet Wizard provides an optional Assign Supporting Files step, shown in Figure 5–18. This step lets you add a backing file and/or a render dependencies file to your portlet.

Figure 5–18 Assign Supporting Files Dialog



5.5.1 Adding a Render Dependencies File

Render dependencies allow you to specify resources, such as CSS or JavaScript, that are required for rendering a portlet in the portal page. For details on render dependencies (also called portlet dependencies), see Section 9.5.1, "Portlet Dependencies."

The Assign Supporting Files part of the Portlet Wizard lets you create a new dependency file or choose an existing one to associate with the portlet. Click **New** to bring up the New Dependencies File dialog, and specify a name and location for the file. Click **Browse** to locate an existing dependency file.

5.5.2 Adding a Backing File

The most common means of influencing portlet behavior within the control life cycle is to use a portlet backing file. Backing files are a way for you to provide Java code that

will run at different points during the portlet lifecycle. A backing file is a Java class that implements com.bea.netuix.servlets.controls.content.backing.JspBacking. For details on backing files, see Section 9.4, "Backing Files."

The Assign Supporting Files part of the Portlet Wizard lets you create a new backing file or choose an existing one to associate with the portlet. Click **New** to bring up the New Backing File Class dialog, and specify a source folder, package, and name for the file. Click **Browse** to locate an existing backing file.

5.6 Adding a Portlet to a Portal

In the development phase of the portal life cycle, you add portlets to a portal using the Oracle Enterprise Pack for Eclipse workbench.

Note: A page must have a layout before you can add a portlet to it. The vertical or horizontal placement of portlets in a placeholder is determined by the selected layout for the page.

Follow these steps:

1. In the Package Explorer view, double-click the portal (`.portal` file) to which you want to add the portlet.

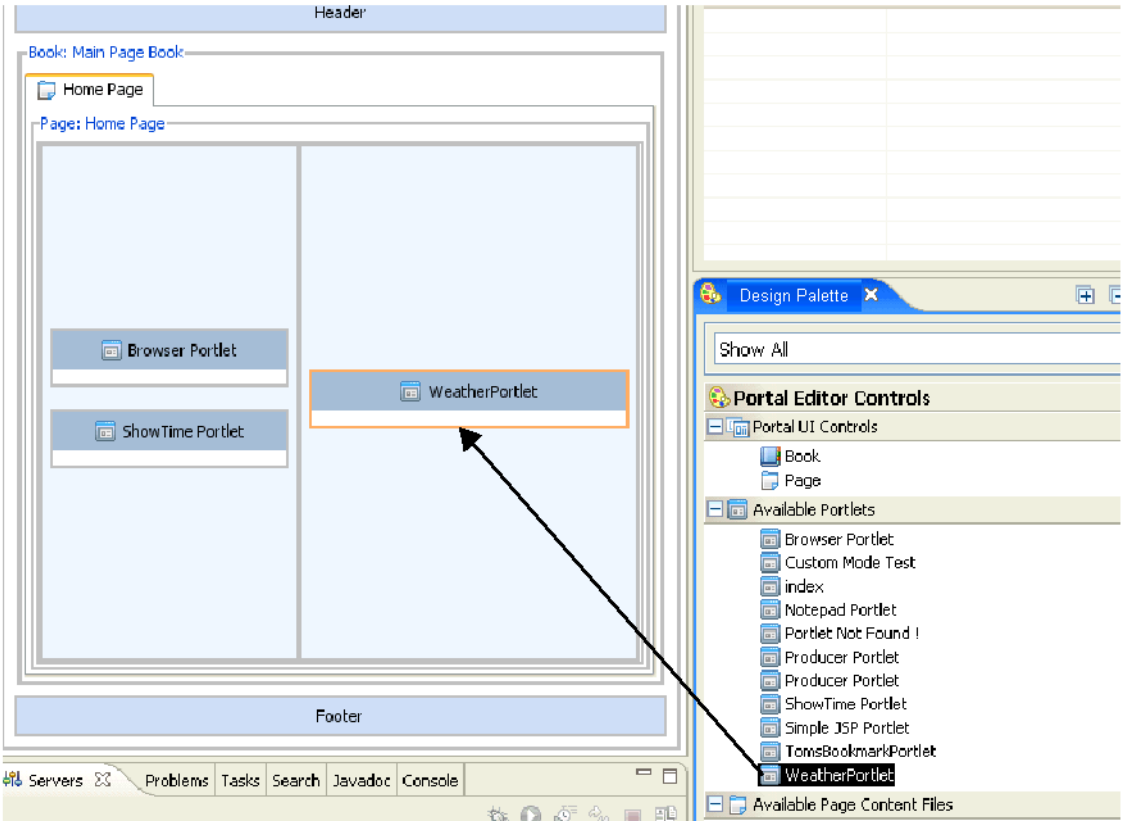
   The portal displays in the editor.

2. If your portal has multiple pages, click the desired page to select it.

3. From the Design Palette view, drag the portlet (the `.portlet` file) onto the portal page at the desired location.

   Figure 5–19 shows an example of this step.

Figure 5–19 Dragging a Portlet from the Palette onto a Portal Page in Editor View



With the portlet selected, you can use the Properties view to customize desired portlet properties.

For detailed information about portlet properties, refer to Section 9.1, "Portlet Properties."

When you add a portlet to a page in the workbench editor, a reference to that portlet is added to the `.portal` file. You can use the `.portal` file as a template for creating desktops in the WebLogic Portal Administration Console. When a portal administrator creates a desktop based on that template, the portlet is added to the portal resource library where it can be added to pages in streaming desktops. For an overview of file-based portals compared with streaming portals, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

In the Staging phase of the portal life cycle, you use the WebLogic Portal Administration Console to configure portlets on desktops. A single portlet definition can be associated with one or more portals (desktops) by creating instances of the portlet. Each of these portlet instances can have its own "personality" and behavior as specified by a variety of different configuration options.

For details in adding a portlet to a portal desktop in the WebLogic Portal Administration Console, refer to Section 17.2.5, "Managing Portlets on Pages."

5.7 Deleting Portlets

To remove a portlet from a portal without deleting the portlet from your portal web project, right-click the portlet in the Oracle Enterprise Pack for Eclipse workbench *editor* and click **Delete**.

To delete a portlet from your portal web project, right-click the portlet in the Package Explorer view and choose **Delete**.

To remove a portlet after you have assembled portlet instances into portal desktops using the Administration Console, refer to Section 17.2.4, "Deleting a Portlet."

6 Building Java Portlets

Java portlets are based on the JSR 286 specification, which provides an API and establishes rules for portlet portability. Java portlets are intended for software companies and other enterprises that are concerned with portability across multiple portlet containers.

> **Note:** JSR 286 or Version 2.0 of the Portlet Specification is based on Version 1.0 that was defined in JSR 168. WLP supports the Version 2.0 Java Portlet Specification.

WebLogic Portal provides capabilities for Java portlets beyond those listed in the JSR 286 spec. For example, you can set threading options, use a backing file, and so on. To implement these additional features, WebLogic Portal uses a combination of the typical `.portlet` file—which you create in the same way that you create other portlet types—as well as the standard `portlet.xml` file and the `weblogic-portlet.xml` file.

This chapter includes these topics:

- Section 6.17, "Exporting Java Portlets for Use on Other Systems"
- Section 6.18, "Importing Java Portlets"
- Section 6.19, "JSR-286/JSR-168 Portlet Compatibility"
- Section 6.20, "Adding an Icon to a Java Portlet"

6.1 Building a Java Portlet

To create a Java portlet, follow these steps:

1. Right-click the folder where you want to store the portlet and select **New** > **Portlet.**

   The **New Portlet** dialog displays.

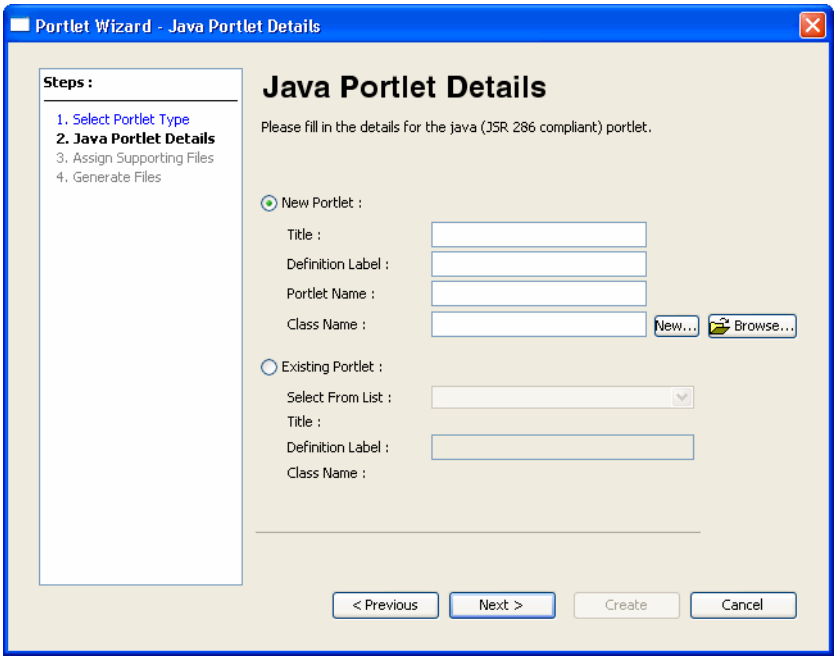
2. Enter a name for the portlet and click **Next**.

   The Portlet Wizard displays the Select Portlet Type dialog.

3. Select the Java Portlet radio button and click **Next**.

   The Java Portlet Details dialog displays. Figure 6–1 shows an example.

Figure 6–1 Portlet Wizard - Java Portlet Details Dialog



4. In the Java Portlet Details dialog, choose whether you want to create a new portlet or create a new instance of an existing portlet by selecting the appropriate radio button.

   New Portlet – If you are creating a new portlet, WebLogic Portal uses the information that you enter in the wizard to perform these two tasks:

   - Create a new `.portlet` file
   - Either create a new `portlet.xml` file (if this is the first Java portlet that you are creating in the project), or add an entry in the `portlet.xml` file, which is located in the `WEB-INF` directory.

Existing Portlet – If you choose to refer to an existing portlet in the wizard, the wizard lets you pick a portlet already defined in the `portlet.xml` file. This option allows you to create a new `.portlet` file and associate it with an existing entry in the `portlet.xml` file. For the Existing Portlet option, a new portlet will be created that has a unique definition label, but that references an existing portlet name in the `portlet.xml` file. The Existing Portlet option allows you to create multiple configurations of one portlet to be surfaced as different portlet instances.

5. Specify the values you want for this portlet, following the guidelines shown in Table 6–1. All fields are required.

Table 6–1 Portlet Wizard - Java Portlet Data Entry Fields

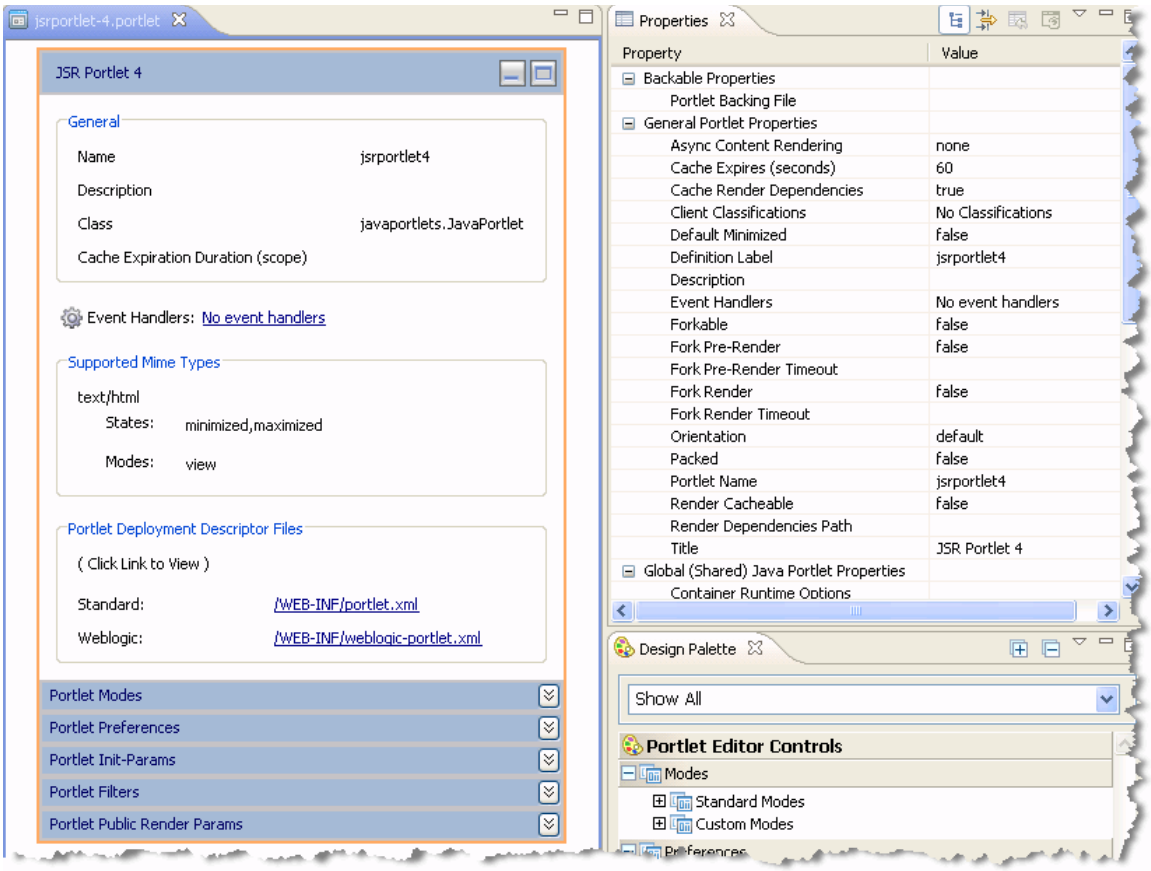
Field	Description
New Portlet – Title	The value for the Title maps to the `<title>` element in the file `portlet.xml`. The title in the `.portlet` file takes priority over the one in the `portlet.xml` file.
New Portlet – Definition Label	Specifies the definition label for the portlet. The value must be unique. The wizard automatically creates a unique ID if you enter one that is already in use. The definition label is the portlet's unique identifier.
New Portlet – Portlet Name	This string provides a value to the `<portlet-name>` element in the `portlet.xml` file and the `portletName` attribute of the `<netuix:javaPortlet>` element in the `.portlet` file. **Note:** If there is no `portletName` attribute value specified in the `.portlet` file, then the Definition Label is assumed to be the portlet name. In this case, there must be a `<portlet-name>` element in the `portlet.xml` file that corresponds to the Definition Label.
New Portlet – Class Name	Enter a valid class name or click **Browse** to navigate to the location of a Java class. This value maps to the `<portlet-class>` element. The class must implement javax.portlet.Portlet. You can also create a new class by clicking **New** and using the New Java Portlet Class dialog to create the Java portlet class (the dialog automatically creates a class that implements javax.portlet.Portlet.) If you enter a class name that does not currently exist, the wizard will create the javax.portlet.Portlet class when you click **Create**.
Existing Portlet – Select From List	The dropdown menu is populated from the `portlet.xml` file and contains the values from the `<portlet-name>` elements. When you select an existing portlet, the Title and Class Name display in read-only fields. **Note:** If you import an existing Java portlet into Oracle Enterprise Pack for Eclipse, you do not need to add an entry in the `web.xml` file for the WebLogic Portal implementation of the JSR286 portlet taglib; this taglib is declared implicitly. Be sure to use `http://java.sun.com/portlet` as the taglib URI in your JSPs.
Existing Portlet – Definition Label	Enter a unique definition label. If the label you enter is not unique, a unique one is created for you automatically. The definition label is used in the `.portlet` file and allows the WLP Framework to uniquely identify the new portlet instance. The definition label is not part of the `portlet.xml` file.

6. Click **Create** to create the portlet or, click **Next** to assign supporting files. For more information on assigning supporting files see Section 5.5, "Assigning Supporting Files."

   Based on the values that you entered, the Wizard creates a `.portlet` file, and adds an entry to `/WEB-INF/portlet.xml`, if it already exists, or creates the file if needed.

   Oracle Enterprise Pack for Eclipse displays the newly created portlet and its current properties. Figure 6–2 shows an example of a Java portlet's appearance and properties as displayed in the portlet editor.

Figure 6–2 Java Portlet Appearance and Properties



After you create the portlet, you can modify its properties in the Properties view, or double-click the portlet in the editor to view and edit the generated Java class. For more information on setting properties, see Section 9.1, "Portlet Properties."

Note: If you delete a `.portlet` file, the corresponding entry remains in the `portlet.xml` file. You might want to clean up the `portlet.xml` file periodically; these extra entries do not cause problems when running the portal but do result in error messages in the log file.

6.2 Java Portlet Deployment Descriptor

The `portlet.xml` deployment descriptor file for Java portlets is located in the `WEB-INF` directory. In addition, the `weblogic-portlet.xml` file is an optional Oracle-specific file that you can use to inject some additional features.

Example 6–1 shows an example of how entries might look in the `portlet.xml` file:

Example 6–1 Example of a portlet.xml file for a Simple Hello World Java Portlet

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app version="2.0"
   xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <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>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>view</portlet-mode>
      </supports>
   </portlet>
</portlet-app>
```

6.3 Portlet Modes and States

As with other types of portlets, you can configure portlet modes and states for a Java portlet. Modes allow you to affect the end user's ability to edit the portlet or display Help for the portlet. You can also create custom modes. States determine the end user's ability to affect the rendering of a portlet, such as to maximize or minimize the portlet. For more information, see Section 9.5.2, "Portlet Modes" and Section 9.5.5, "Portlet States." For information on custom modes for Java portlets, see Section 6.14, "Adding Custom Portlet Modes."

6.4 Portlet Preferences

As with other types of portlets, you can configure portlet preferences for Java portlets. Portlet preferences provide the primary means of associating application data with portlets. This feature is key to personalizing portlets based on their usage. For more information, see Section 9.2, "Portlet Preferences."

6.5 Portlet Initialization Parameters

Initialization parameters are specified with the `<init-param>` element. The `<init-param>` element contains a name/value pair as an initialization parameter of the portlet. You can use the getInitParameterNames() and getInitParameter() methods of the javax.portlet.PortletConfig interface to return these initialization parameter names and values in your portlet code. Initialization parameters are described in the JSR 286 specification.

You can add init-params to your Java portlet by dragging a New Init-Param icon from the Design Palette onto the Java portlet in the portlet editor. Then, click on the **Portlet Init-Param** section of the portlet to display the parameter's properties in the Property view. In the Property view, you can enter the following initialization parameter data:

- Description
- Name
- Value

For example, if you created an initialization parameter called "Color" and set the default value to "green," the following entry will be made in the `portlet.xml` file:

```
<init-param>
    <description>My init param</description>
    <name>Color</name>
    <value>green</value>
</init-param>
```

6.6 Portlet Filters

Portlet filters are a major new feature added to the Java Portlet Specification Version 2 (JSR 286). As the specification explains, filters are Java components that allow on-the-fly transformations of information in both the request to and the response from a portlet. A portlet filter is a reusable piece of code that can transform the content of portlet requests and portlet responses. Filters do not generally create a response or respond to a request as portlets do, rather they modify or adapt the requests, and modify or adapt the response.

> **Tip:** For an in-depth discussion of portlet filters, refer to the Java Portlet Specification, Version 2 (JSR 286).

To add a portlet filter(s) to a Java portlet:

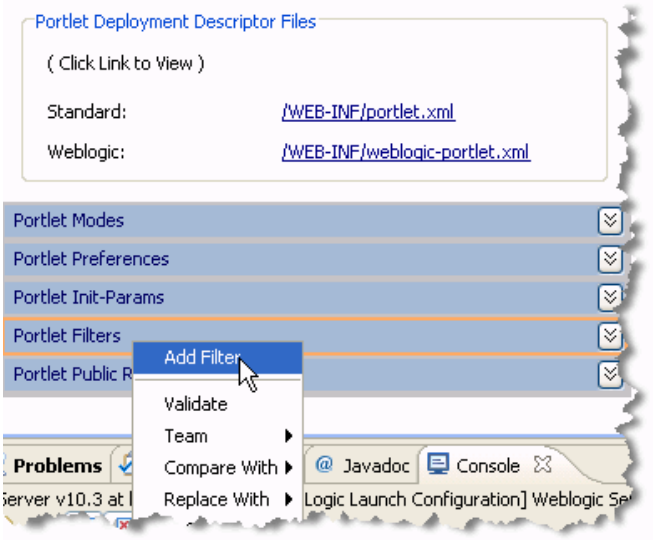
1. Create a Java portlet. (See Section 6.1, "Building a Java Portlet.")
2. Create a filter class. The Portlet Wizard provides a convenient mechanism for creating a Java portlet class. Click the **New** button in the Java Portlet Details part of the Portlet Wizard. This feature creates a class that implements the appropriate interfaces, as described in the following note.

> **Note:** The class must implement at least one of these interfaces in the javax.portlet.filter package, depending on the portlet lifecycle(s) to which the filter will be applied: ActionFilter, EventFilter, RenderFilter, or ResourceFilter. Each filter implements a doFilter() method that is processed when its corresponding life cycle method is called. For example, the ActionFilter.doFilter(...) method is called when the processAction life cycle method is invoked by the portlet container. Refer to the Java Portlet Specification Version 2 for more information.

3. Assign the filter to the portlet. Right click the **Portlet Filters** part of the portlet in the portlet editor, and select **Add Filter**, as shown in Figure 6–3. (Or, you can double-click the **New Filter** item in the Design Palette.)

> **Tip:** You can also drag and drop filters from the Design Palette view onto the Portlet editor. You can drag and drop either existing filters or the **New Filter** item. When you drop a filter, you can choose its position in the filter list. See Section 6.7, "Order of Portlet Filters."

Figure 6–3 Adding a Filter



4. Complete the Define or Choose a Portlet Filter dialog. The dialog requires that you provide a name and a class for a new filter or pick an existing one. The filter class must implement one or more of the javax.portlet.filter classes, which include ActionFilter, EventFilter, RenderFilter, and ResourceFilter.

   If you create a new filter, you must select the life cycle(s) with which to associate it. To do this, click the **Edit** button next to the Lifecycles field. For instance, if the filter class implements RenderFilter interface, you would select the **Render Phase**. Figure 6–4 shows the Select Portlet Lifecycle(s) dialog where you pick the life cycle entities that are implemented in the filter class.

   > **Tip:** If you click the **New** button next to the Class field after using the Select Portlet Lifecycle(s) dialog to pick one or more phases, the Java Class dialog is automatically populated with the correct interface(s) corresponding to those selected phases.

   You can optionally enter a Display Name and Init Params (initialization parameters). The Display Name is the name that is displayed in the IDE. Init Params are name/value pairs that let you pass values to the init() method of the Java class. Click the **Edit** button next to the Init Params field to add them.

   The Define or Choose a Portlet Filter dialog also lets you pick an existing filter to use with a new portlet. Use the Filter Definitions dropdown menu to select a filter to associate with the portlet. This menu is populated with a list of the filters that are specified in the `portlet.xml` file. You can then optionally pick a Display Name for the filter and add a description. The Display Name is only associated with the new portlet. After a filter is associated with a portlet, you can edit the display name and description in the Properties view when you select the filter in the Design view or Outline view.

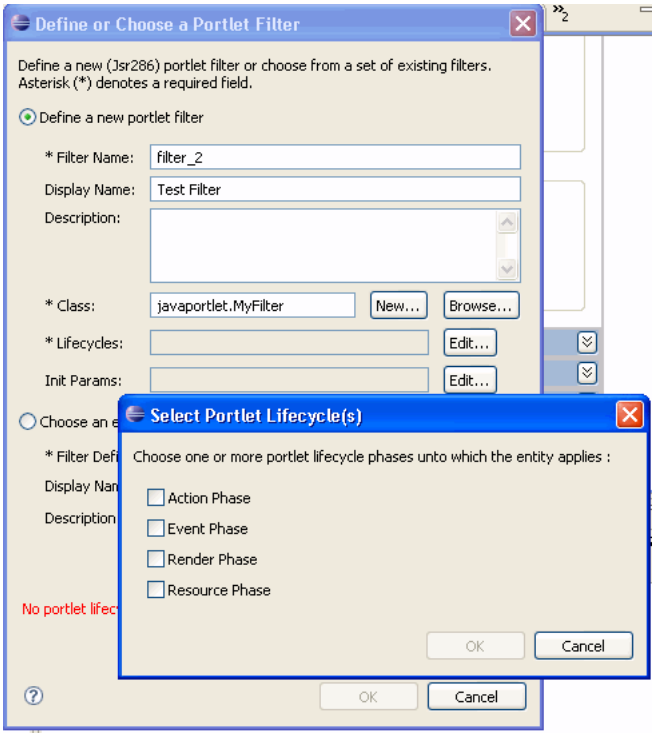
   > **Note:** When you associate an existing filter with a new portlet, you are potentially sharing that filter with one or more other portlets. When you share a filter, you need to remember that the order of filters as specified in the `portlet.xml` file is significant. See Section 6.7, "Order of Portlet Filters."

5. Click **OK** to add the filter to the portlet. The `portlet.xml` file is only saved to the disk when you save the `.portlet` file.

   The `<filter>` and `<filter-mapping>` elements are automatically added to the `portlet.xml` descriptor file. These elements specify the filter name, class, life cycle phase(s), and mapping. The mapping element allows multiple portlets to share a single filter definition. For example:

```
<filter>
    <display-name>Test Filter</display-name>
    <filter-name>filter_1</filter-name>
    <filter-class>javaportlets.MyFilter</filter-class>
    <lifecycle>ACTION_PHASE</lifecycle>
    <lifecycle>RENDER_PHASE</lifecycle>
</filter>
<filter-mapping>
    <filter-name>filter_1</filter-name>
    <portlet-name>jsrportlet4</portlet-name>
</filter-mapping>
```

Figure 6–4 Defining a Portlet Filter



6.7 Order of Portlet Filters

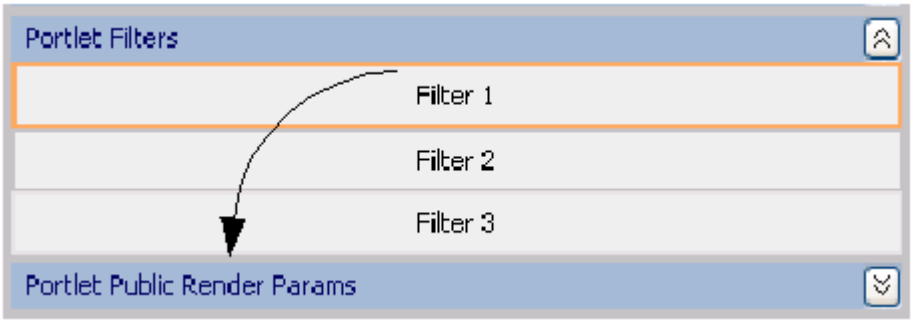
The Portlet editor presents a list of the filters that have been added to a Java portlet (see Figure 6–5). The editor shows you the names of the filters and lists them in the order in which they are defined in the `portlet.xml` file. This order is significant because it specifies the order in which the filters are applied to the portlet.

You can easily change the order of the filters. One way to change the order is to drag and drop the filters in the Portlet editor (see Figure 6–5). Another way to reorder the

portlet filters is to right-click a filter and select Move Up or Move Down from the context menu.

> **Note:** Remember that filters can be mapped to multiple portlets. If you have multiple portlets mapped to (sharing) a filter, arbitrarily changing the filter order can produce undesired side effects. For example, if you change the order in which filters are applied in one portlet, the reordering will apply to all other portlets that share the filter.

Figure 6–5 You Can Drag and Drop Filters to Reorder Them



6.8 Public Render Parameters

Public render parameters are a JSR 286 feature that allows portlets to share parameter values with other portlets, allowing a form of interportlet communication. For instance, if portlet A and portlet B are both configured to use a particular public render parameter, any changes in the parameter's value made by portlet A will be seen by portlet B. Unlike render parameters, which can't be read during the processAction lifecycle, public render parameters can be accessed in all lifecycles of the portlet: processAction, processEvent, render, and serveResource. Public render parameters automatically work with the asynchronous desktop rendering feature (for details, see Section 10.5, "Asynchronous Portlet Content Rendering"). Public render parameter values can also be shared with non-JSR-286 portlets, where they are called "shared parameters". For more information, see Section 9.3, "Using Shared Parameters.".

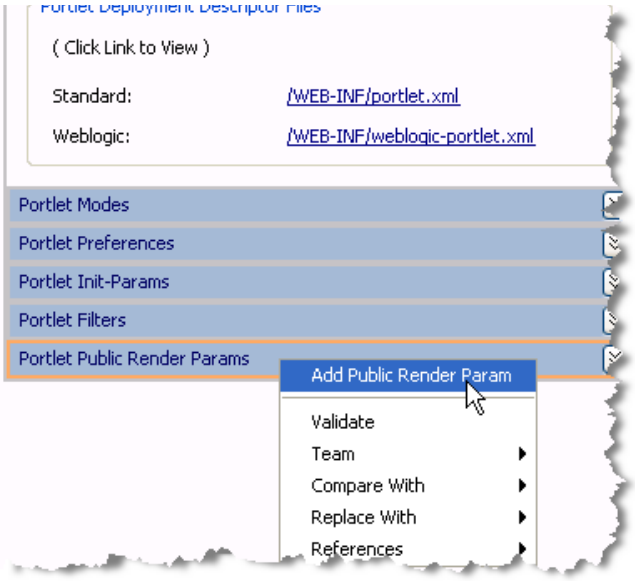
> **Tip:** For an in-depth discussion of public render parameters and how to access them, refer to the Java Portlet Specification, Version 2 (JSR 286).

To add public render parameters to a Java portlet:

1. Bring up the Define or Choose a Portlet Public Render Param dialog. To do this, right-click the **Portlet Public Render Params** part of the Java portlet in the portlet editor and select **Add Public Render Param**, as shown in Figure 6–6. Or, you can double-click on the **New Public Render Param** item in the Design Palette.

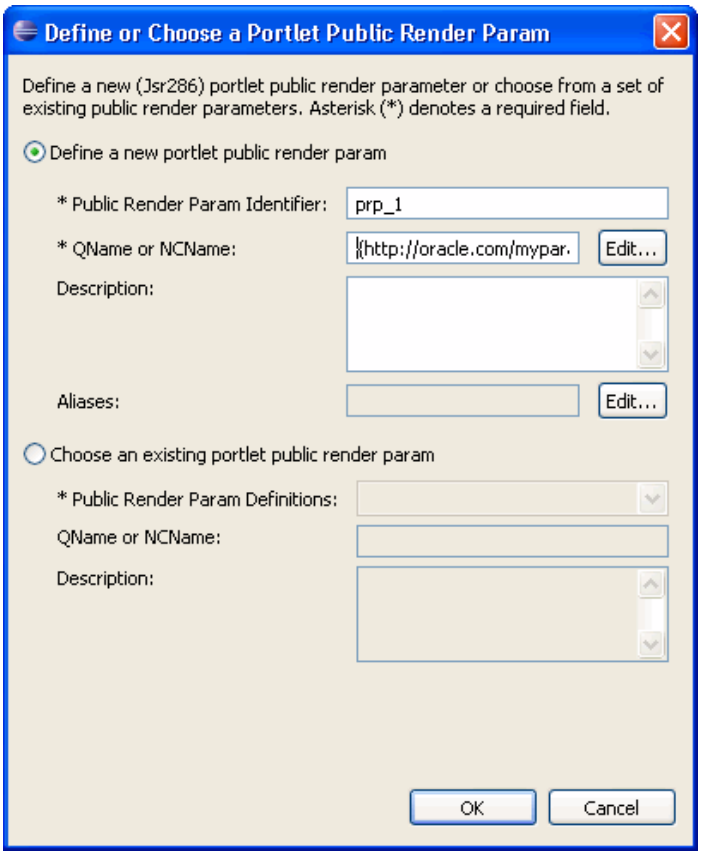
   > **Tip:** You can also drag and drop public render parameters from the Design Palette view onto the Portlet editor. You can drag and drop either existing render parameters or the **New Public Render Param** item. The order in which the parameters appear in the Portlet editor is not significant.

Figure 6–6 Adding a Public Render Parameter



2. The Define or Choose a Portlet Public Render Param dialog lets you define a new parameter or pick an existing one to associate with the portlet. This dialog is shown in Figure 6–7.

Figure 6–7 Define or Choose a Portlet Public Render Param Dialog



To create a new parameter, first enter an identifier as shown in Figure 6–7. The identifier is a string that identifies the public render parameter within the `portlet.xml` deployment descriptor. This identifier is portlet application scoped. Any portlet that wishes to share the parameter can do so by referencing this identifier in the `portlet.xml` file, and can access the public parameter's value at runtime using the identifier as the parameter name.

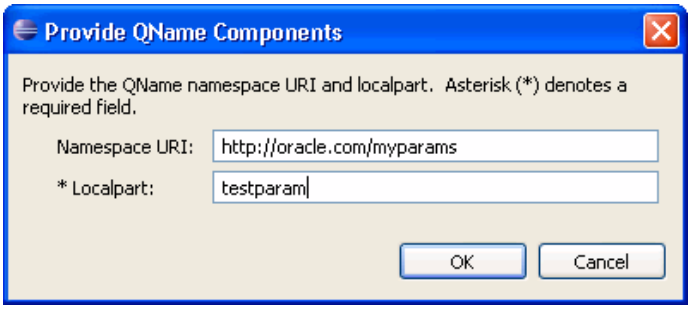
The next required field is the QName, or qualified name, that uniquely identifies the public render parameter. The QName of a public render parameter is used as the unique identifier when parameter values are being distributed between different portlet applications, such as portlets consumed through WSRP. For example, if portlet A from producer X has a public render parameter it accesses with an identifier (name) of `zip`, and portlet B from producer Y has a public render parameter it accesses with an identifier (name) of `zipcode`, these two portlets will automatically share the appropriate values as long as the QNames for their parameters are identical.

The QName consists of a required local part and a namespace URI. If you do not provide the namespace URI, the default namespace URI for the portal application is used. For information on QNames and NCNames, see Section 12.12, "About QNames and Aliases." See also Section 6.12, "Using Global (Shared) Properties" for information on setting the default namespace. Click **Edit** next to the QName field to bring up the Provide QName Components dialog, as shown in Figure 6–8.

> **Tip:** The Provide QName Components dialog automatically forms the namespace/local part identifier in the standard syntax, for example: `{http://oracle.com/myparams}testparam`. If you enter the QName directly in the Define or Choose a Portlet Public Render Param dialog, you must use this syntax.

The Define or Choose a Portlet Public Render Param dialog also lets you pick an existing public render parameter to use with a new portlet. Use the Public Render Param Definitions dropdown menu to select a parameter to associate with the portlet. This menu is populated with a list of the parameters that are specified in the `portlet.xml` file. The QName and NCName, as well as the description, are not editable from this dialog. To change these values, you can use the Properties view after the public render parameter is mapped to the portlet. For more information, see Section 12.12, "About QNames and Aliases."

Figure 6–8 Provide QName Components Dialog

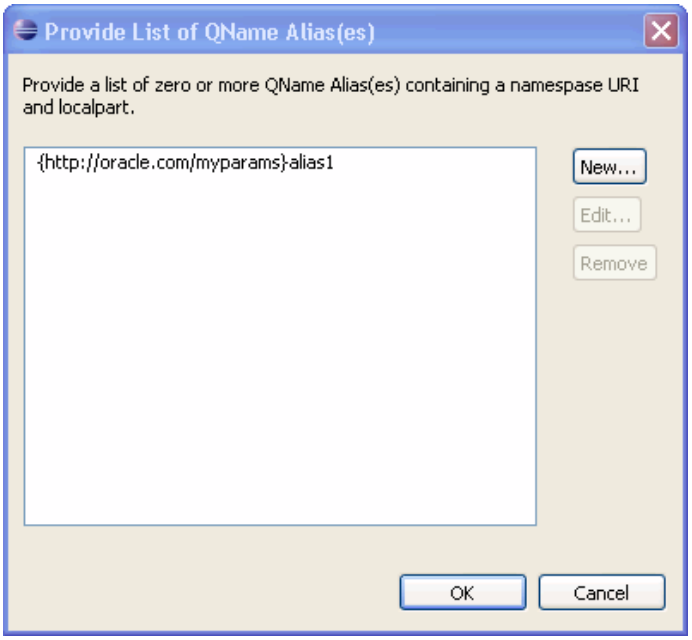


You can optionally specify an alias name. For more information, see Section 12.12, "About QNames and Aliases."

To add an alias, click **Edit** and in the next dialog, click **New**. The Provide QName Components dialog appears (see Figure 6–8. Enter an optional Namespace URI

and a Local Part for the QName and click **OK**. The alias definition appears in the Provide List of QName Alias(es) dialog (Figure 6–9). You can add as many public render parameter aliases to a portlet as you like.

Figure 6–9 Provide List of QName Alias(es) Dialog



You can add more than one public render parameter to a portlet. Note that the `portlet.xml` file is only updated and saved to the disk when you save the `.portlet` file.

3. Click **OK** to create the parameter.

As a result of performing the steps in this section, the `portlet.xml` file and the `.portlet` file are both updated as follows:

Changes to the portlet.xml File:

First, an application-scoped `<public-render-parameter>` element is added to the file. For example:

```
<portlet-app ...>
    <portlet> ... </portlet>
    <public-render-parameter>
        <identifier>prp_1</identifier>
        <qname xmlns:x="http://oracle.com/myparams">x:testparam</qname>
    </public-render-parameter>
</portlet-app>
```

In addition, the `<supported-public-render-parameter>` element is added to the `<portlet>` element in `portlet.xml` (the definition of the portlet to which the render parameter was added). This element configures the portlet to share the parameter. In the Java portlet class, the parameter can be set with a call like this:

```
public class JavaTestPortlet extends GenericPortlet {
     . . .
     public void processAction(ActionRequest req, ActionResponse res)
         throws IOException, PortletException {
     . . .
            res.setRenderParameter("prp_1", testparam);
```

```
    }
    . . .
}
```

6.8.1 Public Render Parameter Example

The following example provides code from two portlets that share a public render parameter called "selectedBook".

> **Note:** This example uses java portlet tags, not WLP render tags.

Portlet A

The `.portlet` file for Portlet A has a public render parameter specified with the identifier "selectedBook". The `.jsp` file has an anchor tag that uses the renderURL tag in the Java portlet tag library.

```
<pz:contentSelector rule="ListBooks" id="listOfBooks"/>
<utility:notNull item="${listOfBooks}">
    <h4>Books You Might Like</h4>
    <utility:forEachInArray array="${listOfBooks}" id="node"
type="com.bea.content.Node">
        <portlet:renderURL var="selectedBookUrl">
            <portlet:param name="selectedBook" value="${node.path}"/>
        </portlet:renderURL>
        <a href="${selectedBookUrl}">
            <cm:getProperty id="node" name="xTitle" conversionType="html"/>
        </a>
    </utility:forEachInArray>
</utility:notNull>
```

Portlet B

The `.portlet` file for Portlet B has a public render parameter specified with the identifier "selectedBook". The `.jsp` file includes the following logic:

```
<c:choose>
    <c:when test="${param['selectedBook'] != null}">
        <cm:getNode id="bookNode" path="${param['selectedBook']}"/>
        <div><ad:render id="bookNode" /></div>
    </c:when>
    <c:otherwise>
        <p>No book was selected</p>
    </c:otherwise>
</c:choose>
```

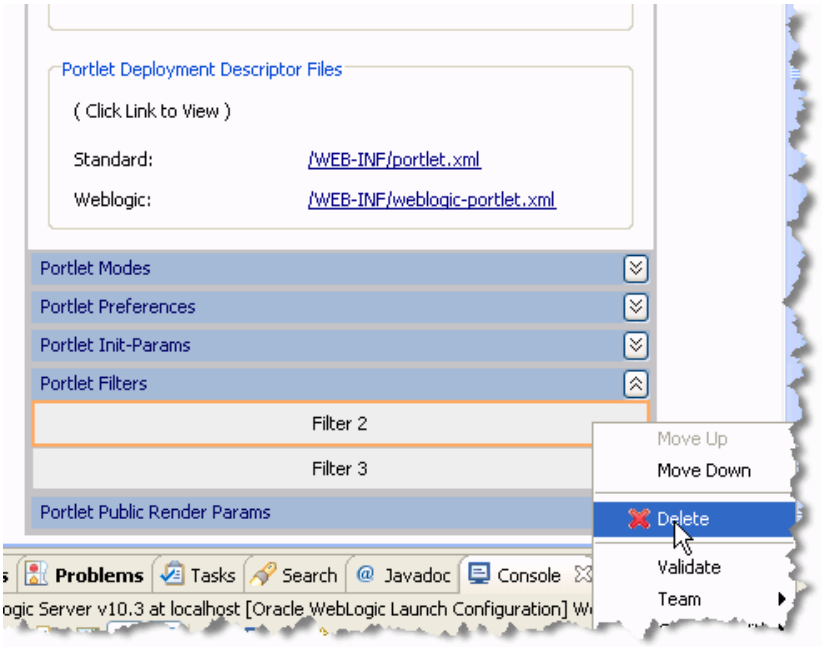
6.9 Event Handling with Java Portlets

For details on using event handling with Java Portlets, see Section 12.7, "Events in Java Portlets."

6.10 Deleting Java Portlet Features

The procedure for deleting Portlet Modes, Portlet Preferences, Portlet Init-Params, Portlet Filters, and Portlet Public Render Params is the same. In the portlet editor, expand the feature menu, right-click the item you want to delete, and select Delete from the menu, as illustrated in Figure 6–10:

Figure 6–10 Deleting a Filter



If you are deleting a *shared* filter, public render parameter, or event, a confirmation dialog appears, and you must choose one of the following options:

- **Remove (dissociate) the shared [feature] from this portlet only.**

  In this case, the feature definition remains in the `portlet.xml` file. Only the mapping to the current portlet is removed. Choose this case if other portlets reference the feature. For example, if multiple portlets reference the same filter, you would want to leave the filter definition in the `portlet.xml` file.

- **Delete the shared [feature] from the portlet.xml.**

  In this case, the entire feature and all mappings to the feature are removed from the `portlet.xml` file. Only pick this option if you know that no other portlets require the feature definition.

For example, Example 6–2 shows an excerpt from a `portlet.xml` file where a filter and filter mappings to several portlets are defined. If you delete `filter_2` from the portlet named `jsrportlet4` using the first delete option, the element named `jsrportlet4` will be removed from the `<filter-mapping>` element, as shown in Example 6–3. Note that the other portlets will continue to reference the filter. If you select the second delete option, the entire `<filter>` definition and the `<filter-mapping>` will be removed. In this case, no portlets will reference the filter.

Example 6–2 Filter and Filter Mapping Elements Before Delete Operation

```
<filter>
   <display-name>Filter 2</display-name>
   <filter-name>filter_2</filter-name>
   <filter-class>javaportlets.MyFilter</filter-class>
   <lifecycle>RENDER_PHASE</lifecycle>
</filter>
<filter-mapping>
    <filter-name>filter_2</filter-name>
    <portlet-name>jsrportlet4</portlet-name>
    <portlet-name>jsrportlet5</portlet-name>
```

```
    <portlet-name>jsrportlet6</portlet-name>
</filter-mapping>
```

Example 6–3 Filter and Filter Mapping Elements After Delete Operation

```
<filter>
   <display-name>Filter 2</display-name>
   <filter-name>filter_2</filter-name>
   <filter-class>javaportlets.MyFilter</filter-class>
   <lifecycle>RENDER_PHASE</lifecycle>
</filter>
<filter-mapping>
    <filter-name>filter_2</filter-name>
    <portlet-name>jsrportlet5</portlet-name>
    <portlet-name>jsrportlet6</portlet-name>
</filter-mapping>
```

6.11 Using Container Runtime Options

Container runtime options provide a way to change the runtime behavior of the portlet container. You can declare these options either at the portlet level or at the portlet application level. You can set container runtime options at the application level using the Global Shared Properties feature (see Section 6.12, "Using Global (Shared) Properties"). You can also set container runtime options at the portlet-level. For details on this technique, see Section 6.13, "Setting Portlet-Level Container Runtime Options."

If set at the application level, an option will apply to all portlets in the application, except for portlets which explicitly set a different value for the container runtime option at the portlet level.

Container runtime options can have one or more values, though most runtime options use only a single value. This section describes the four standard container runtime options defined in the JSR 286 specification and additional ones supported by `WLP`.

6.11.1 Standard Container Runtime Options

This section describes four standard container runtime options that are defined in the JSR 286 specification.

- **`javax.portlet.escapeXml`**
  - Affects how URLs are XML-encoded when using the JSR 286 tag library tags `actionUrl`, `renderUrl`, and `resourceUrl`. See the JSR 286 specification, section 10.4.1 for a complete description. Note that this container runtime option does *not* have any affect on the return value of PortletURL.toString() or `ResourceURL.toString()`; in both cases, for JSR 286, the output will use only ampersand characters. To use ampersand entities or be able to specify the XML encoding to use when generating URLs not using the tag libraries, see the `BaseURL.write()` methods. The setting of this container runtime option affects only the default behavior of the URL tags; you can override it on a per-tag basis using the `encodeXml` attribute.
  - Valid Values:
    * `true` – The JSR 286 tag library `actionUrl`, `renderUrl` and `resourceUrl` tags will default to XML-encoding the resulting URLs (using ampersand entities for parameter separation) by default.
    * `false` – The default behavior is changed to not XML-encode (and use just ampersand characters) by default.

- **javax.portlet.servletDefaultSessionScope**
  - Specifies the scope of the Session object provided to servlets or JSPs which are included or forwarded to from a Java portlet. See the JSR 286 specification, section 10.4.3 for a complete description. The default behavior (if this container runtime option is not set at all) is to map the portlet session with application scope, equivalent to setting the runtime option value to `APPLICATION_SCOPE`. To map instead to the portlet session scope, you can set the value for this runtime option to `PORTLET_SCOPE`.
  - Valid Values:
    * `APPLICATION_SCOPE` – Maps the session to the application scope
    * `PORTLET_SCOPE` – Maps the session to the portlet scope.
- **javax.portlet.actionScopedRequestAttributes**
  - Stores request attributes that are set by the portlet code in "scopes", generally starting at a `processAction` and continuing until the next `processAction`, and provides these request attributes back to the portlet when the next lifecycle method is called. See the JSR 286 specification section 10.4.4 for a complete description and examples. The JSR 286 specification requires all portlet containers to support this container runtime option. The `com.oracle.portlet.excludedActionScopeRequestAttributes` container runtime option described below can be used to limit which request attributes are stored in the scopes.
  - Valid Values:
    * `true` – Store and provide request attribute values between calls to the portlet. If `true` is the first value for this container runtime option, you may further specify the number of attribute scopes the portlet container should cache by making the second value the string `numberOfCachedScopes`, and the third value the number of scopes to cache.
    * `false` – Do not store request attributes.
- **javax.portlet.renderHeaders**
  - This option is not currently supported by WebLogic Portal. It is used to indicate that the portlet supports having its `render()` method called twice, setting any header information, portlet title and next possible portlet modes in the first call, and rendering the body of the portlet in the second call. See the JSR 286 specification sections 10.4.2 and 11.1.4.3 for complete details.
  - Valid Values:
    * `true` – Indicates the portlet supports the two-phase render.
    * `false` – Indicates that the portlet does not support two-phase render.

6.11.2 Other Container Runtime Options Supported by WLP

This section describes several other container runtime options supported by WLP.

- **com.oracle.portlet.compatibilityMode**
  - Used to invoke one of the compatibility modes available in WebLogic Portal's Java portlet container. See Section 6.19, "JSR-286/JSR-168 Portlet Compatibility" for details.
  - Valid Values:

    * owlp168 – Invokes the WebLogic Portal JSR 168 compatibility mode; also automatically sets the com.oracle.portlet.disallowResourceServing runtime option to true and the com.oracle.portlet.streamingOptimized runtime option to true, if no other values for those options were specified.

    * owc168 – Invokes the Oracle WebCenter JSR 168 compatibility mode; also automatically sets the com.oracle.portlet.disallowResourceServing runtime option to true if no other value for that option is specified.

- **com.oracle.portlet.disallowResourceServing**
  - Used to disable portlets from serving resources, which may be a security concern. See the note in Section 6.19, "JSR-286/JSR-168 Portlet Compatibility" for details.
  - Valid Values:
    * true – Portlets will not be allowed to serve resources. This is useful if JSR 168 portlets are run in the JSR 286 container, as any JSR 168 portlet extending javax.portlet.GenericPortlet will automatically inherit the JSR 286 functionality, which automatically forwards resource requests to a file in the webapp named after the resource ID, creating a potential security problem. For the same security reason, JSR 286 portlets that do not serve resources are safest to disallow resource serving.
    * false – Portlets will be allowed to serve resources.
- **com.oracle.portlet.streamingOptimized**
  - Indicates the portlet is optimized to run in streaming mode, which may enhance performance. There are side effects to using this option:
    * The RenderResponse.reset() and RenderResponse.resetBuffer() methods do nothing; they will not clear the buffer or consistently throw IllegalStateExceptions. Besides cases where portlet code calls reset() or resetBuffer() directly, this also affects portlets that write data out before doing a forward using the PortletRequestDispatcher. When in streaming mode, the output written before the forward will not be cleared.
    * RenderResponse.getBufferSize() will always return 0.
    * RenderResponse.setBufferSize() has no effect.
    * Headers, cookies and HTML HEAD elements set by the portlet during render (in one-phase render) or during the RENDER_MARKUP render phase (in two-phase render) may not be rendered in the portal response to the client, if the underlying portal response is already committed.

      To avoid the underlying portal response from being committed, use the avoid-response-commit setting in the WEB-INF/wlp-framework-common-config.xml, as described Section 9.13, "Avoiding Committing Responses."
    * RenderResponse.isCommitted() will return a value consistent with the portlet's behavior, but not reflective of the underlying response. For example, when the render phase is started for the portlet, isCommitted() will return false, but if MimeResponse.flushBuffer() is called, the isCommitted()

method will then return true, even if the underlying response is not yet committed. This behavior is necessary to support `PortletRequestDispatcher.forward()` calls, which require the response to not be committed. This behavior also means that the portlet has no way of knowing whether headers being set actually make it back to the client.

* `RenderResponse.setProperty()` calls will *not* clear previous values for the property being set, but will behave the same as `RenderResponse.addProperty()` would. This is required to allow multiple portlets being rendered to the same page to aggregate header values, rather than resetting other portlets' header values.

Note: Portlets should never depend on the side-effects noted above. For example, if portlet caching is used, even when streaming mode is turned on some renders may occur using the default (buffering) `MimeResponse` object if the output is destined to be cached.

– Valid Values:

* `true` – The portlet supports being rendered in a streaming manner.

* `false` – The output of the portlet should be buffered.

- **`com.oracle.portlet.minimumWsrpVersion`**

  – Specifies minimum required WSRP version for the portlet to work. If the WSRP version being used is less than the value specified, the portlet will not be included in a WSRP `GetServiceDescription`, `GetPortletDescription`, `GetMarkup` and other WSRP responses. This may be useful for JSR 286 portlets that require the use of events, public render parameters or portlet-served resources, as these features are not present in the WSRP 1.0 specification but are supported in WSRP 2.0.

  – Valid Values:

  * 1 – Any version of WSRP is acceptable.

  * 2 – At least WSRP version 2.0 is required.

- **`com.oracle.portlet.offerPortletOverWsrp`**

  – Used to indicate whether a portlet should be offered in the WSRP producer's service description. If this container runtime option is not specified at all, the default value specified in the `WEB-INF/producer-config.xml` will be used. In addition, if a Java portlet has an associated `.portlet` file, the `offerRemote` setting in the `.portlet` file will override the setting specified in this container runtime option.

  – Valid Values:

  * `true` – The portlet is offered in the WSRP producer's service description.

  * `false` – The portlet will not be included in the service description.

- **`com.oracle.portlet.wsrpHeaderMode`**

  – Used only when portlets are being rendered as WSRP remote portlets, to indicate where cookies and headers should be put in the WSRP SOAP response as a hint to the WSRP consumer for the header or cookie's intended final destination. When portlets are run locally (not over WSRP), headers and

cookies set by portlets are always assumed to go to the client. Setting this container runtime option sets a default value for the `PortletRequest` attribute `com.oracle.portlet.wsrpHeaderMode`, which can still be overridden by the portlet at runtime on a per-header basis. If no value for this container runtime option is set, the portlet container will assume the default value for the producer as specified in the `WEB-INF/wsrp-producer-config.xml` file.

- Valid Values:
  - `client` – Headers and cookies set by the portlet will be directed to go to the client. (e.g. browser)
  - `consumer` – Headers and cookies set by the portlet will be directed to go to the consumer and not passed on to the client.
  - `both` – Headers and cookies set by the portlet will be directed to go to the client and the consumer.

- **`com.oracle.portlet.suppressWsrpOptimisticRender`**
  - Suppresses the optimistic render of a portlet after the action and/or event lifecycles if the portlet is being run over WSRP. Normally, if a WSRP portlet receives a WSRP `PerformBlockingInteraction` request (`processAction` in JSR 168/JSR 286 portlets) and the portlet does not send any events as a result, the WSRP producer will render the portlet and return the portlet's markup in the response of the `PerformBlockingInteraction` SOAP message. This markup may be cached by the consumer until the consumer's page renders, and if nothing else affecting the state of the portlet happens (such as the portlet receiving an event), the cached markup can be used by the consumer, eliminating the need for a second SOAP call to `GetMarkup`. This assumes that the portlet's render phase is idempotent, which is always a best practice. However, if the portlet expects to receive an event, or rendering the portlet is more costly than a second SOAP message for `GetMarkup`, you can use this container option to suppress the optimistic render of the portlet after a `processAction` or `handleEvent` call. The portlet will still be rendered normally when the producer receives the WSRP `GetMarkup` request.
  - Valid Values:
    - `true` – Optimistic render is always suppressed.
    - `false` – Optimistic render may be performed.

- **`com.oracle.portlet.externalScopeRequestAttributes`**
  - This is a multi-valued property, with each value being a regular expression. Request attributes which match any of the regular expressions are considered outside of portlet scope, and are shared with the underlying portal request. If the `javax.portlet.actionScopedRequestAttributes` option is used, any request attributes matching the regular expressions declared in `externalScopeRequestAttributes` are not stored in the action scope request attributes.
  - Default Values:
    - `com\.bea\.netuix.*`
    - `com\.bea\.wlw\.runtime.*`
    - `com\.bea\.wsrp.*`

    * `javax\.servlet.*`
    * `weblogic\.servlet.*`
    * `com\.bea\.p13n.*`

- **`com.oracle.portlet.excludedActionScopeRequestAttributes`**
  - This is a multi-valued property, with each value being a regular expression. Request attributes which match any of the regular expressions are not stored as action-scoped request attributes if the `javax.portlet.actionScopedRequestAttributes` container runtime option is used, in addition to any request parameters whose values match the regular expressions defined in the `com.oracle.portlet.externalScopeRequestAttributes` container runtime option.
  - Default Values:
    * `javax\.portlet.*`
    * `oracle\.portlet.*`
    * `com\.oracle\.portlet.*`
- **`com.oracle.portlet.allowEventPayloadsWithoutJAXBBindings`**
  - Causes the JSR286 container to send a redirect to the browser to the portlet's render URL after a processAction is run (and after events are handled) so that reloading the resulting page will not result in another processAction. The default value is "false".
  - Valid Values:
    * `true` – A redirect will be issued after every portlet action.
    * `false` – No redirect will be automatically issued after portlet actions.
- **`com.oracle.portlet.redirectAfterAction`**
  - Allows event payload types declared in `portlet.xml` and event payload objects sent from JSR 286 portlets to bypass the JSR 286 specification requirement that these types have a valid JAXB binding. This container runtime option is valid only at the portlet application level and is ignored if specified at the portlet level. The default value is `false`.
  - Valid Values:
    * `true` – Event payloads without valid JAXB bindings are allowed.
    * `false` – Event payloads without valid JAXB bindings are not allowed, per the JSR 286 specification.
- **`com.oracle.portlet.wsrpPortletHandle`**
  - Allows the specification of the WSRP portlet handle to be used for the portlet, which must be unique within the webapp. This container runtime option is only valid at the portlet level and will be ignored if specified at the portlet-application level. The default value is the portlet name from portlet.xml.
  - Valid Value:
    * A unique string conforming to WSRP portlet handle requirements.
- **`com.oracle.portlet.requireIFrame`**

  - Specifies whether the portlet needs to be rendered inside an IFrame. The default value is "false".
  - Valid Values:
    * `true` – The portlet will be rendered inside an IFrame.
    * `false` – The portlet will not be forced to be rendered in an IFrame.

- **`com.oracle.portlet.allowWsrpExport`**
  - Specifies whether the WSRP export-portlets operation should be supported for the webapp. If "false", this will override the setting in WEB-INF/wsrp-producer-config.xml and turn off the WSRP export-portlets operation. This option is used mainly for backward-compatibility; it is preferred to control the export-portlets operation through the wsrp-producer-config.xml setting. This container runtime option is valid only at the portlet-application level and will be ignored if specified at the portlet level. The default value is "true".
  - Valid Values:
    * `true` – Export-portlets will be allowed.
    * `false` – Export-portlets will not be allowed.

- **`com.oracle.portlet.useWsrpUserContextForUserAuthenticationInfo`**
  - Specifies whether the PortletRequest methods getRemoteUser(), getUserPrincipal() and isUserInRole() are based on the WSRP user context information or on standard J2EE security. The default value is "false".
  - Valid Values:
    * `true` – The user information will be based on the WSRP user context. This can be a security problem so this option should be used with care.
    * `false` – The user information will be based on the J2EE authenticated user.

- **`com.oracle.portlet.defaultServedResourceRequiresWsrpRewrite`**
  - Specifies the default WSRP `requiresRewrite` flag to use when generating ResourceURLs for portlet-served resources. This setting is used for all ResourceURLs created by the portlet, unless overridden by the `oracle.portlet.server.resourceRequiresRewriting` request attribute when the ResourceURL methods write() or toString() are called. This setting is also used to specify the WSRP `requiresRewriting` flag on the served resource response, but can be overridden by the presence of the `oracle.portlet.server.resourceRequiresRewriting` request attribute when the portlet's serveResource() method returns.
  - Default Values:
    * If unspecified – The `requiresRewrite` URL flag will not be given a value, and the `requiresRewriting` response flag for a `serveResource` operation will be based on the MIME type of the response.
    * `true` – The `requiresRewrite` URL flag and `requiresRewriting` response flag will be set to `true`, which directs the consumer to overwrite the resource.
    * `false` – The `requiresRewrite` URL flag and `requiresRewriting` response flag will be set to `false`, indicating that consumer does not need

to rewrite the resource, though the consumer may choose to rewrite the resource.

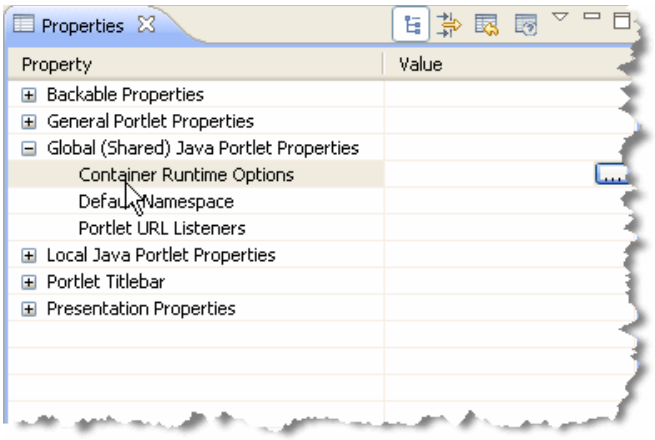
- **com.oracle.portlet.defaultProxiedResourceRequiresWsrpRewrite**
  - Specifies the default WSRP `requiresRewrite` flag to use when encoding URLs for resources not served by the portlet. This setting is used for all urls returned by the PortletResponse.encodeURL() method, unless overridden by the `oracle.portlet.server.resourceRequiresRewriting` request attribute when the PortletResponse.encodeURL() method is called.
  - Valid Values:
    * `true` – The `requiresRewrite` flag is set to `true`, which directs the consumer to overwrite the resource.
    * `false` – The `requiresRewrite` flag is set to false, indicating that the consumer does not need to rewrite the resource.
- **com.oracle.portlet.eventPayloadsXmlType**
  - Specifies optional XML schema types for JAXB-bindable Java object event payloads if events are sent over WSRP. This is a multi-valued runtime option; each value consists of a Java class name / QName pair, separated by a colon (":") and with the QName specified using the standard Java String representation of a QName ("`\{namespace\}localpart`"). For each value specified, the QName is used as the XML schema type to annotate WSRP event payloads of the specified Java object type after the object has been marshalled to XML. This setting is useful when using events to communicate across portlets on multiple producers from different vendors.

6.12 Using Global (Shared) Properties

Global (Shared) Java Portlet Properties are properties that are available to all Java portlets within the web application. WLP supports three global shared properties: Container Runtime Options, Default Namespace, and Portlet URL Listeners. The properties and their values are stored in the `portlet.xml` file. Refer to the Java Portlet Specification Version 2 for detailed information on accessing and using these global shared properties.

To add or modify global shared properties for a Java portlet, locate the Global (Shared) Java Portlet Properties section in the Properties view and select the property you wish to create or modify, as shown in Figure 6–11. Note that because these properties are shared among multiple portlets, you need to be careful when adding and deleting them to prevent unwanted side effects. For example, if you change the values for a shared property in one portlet, the value is changed for all portlets. If you open another portlet, you will see the changed value reflected there.

Figure 6–11 Global Shared Properties



You can set these global shared properties:

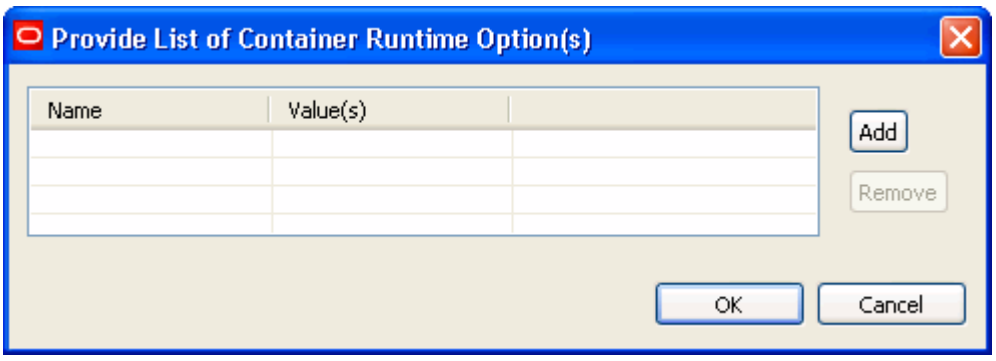
- **Container Runtime Options**

  You can specify certain container runtime options for global shared parameters. For details on container runtime options, see Section 6.11, "Using Container Runtime Options." When container runtime options in the global shared properties are set, the options will take effect at the portlet application level, applying to all portlets declared in the portlet application.

  > **Tip:** You can override container runtime option settings for global shared parameters for individual portlets by declaring container runtime options at the portlet level. For more information, see Section 6.13, "Setting Portlet-Level Container Runtime Options."

  To bring up a dialog for adding runtime option name/value pairs, select Container Runtime Options in the Properties view and click **Edit**. The dialog is shown in Figure 6–12.

Figure 6–12 Provide List of Container Runtime Option(s) Dialog



The specified runtime option(s) are added to the `portlet.xml` file. The global container options apply to all portlets registered in the `portlet.xml` file. For example:

```
<container-runtime-option>
    <name>javax.portlet.actionScopedRequestAttributes</name>
    <value>true</value>
</container-runtime-option>
```

You can also set container runtime options for individual portlets. See Section 6.13, "Setting Portlet-Level Container Runtime Options." Refer to the Java Portlet Specification Version 2 for more information on the container runtime options.

- **Default Namespace**

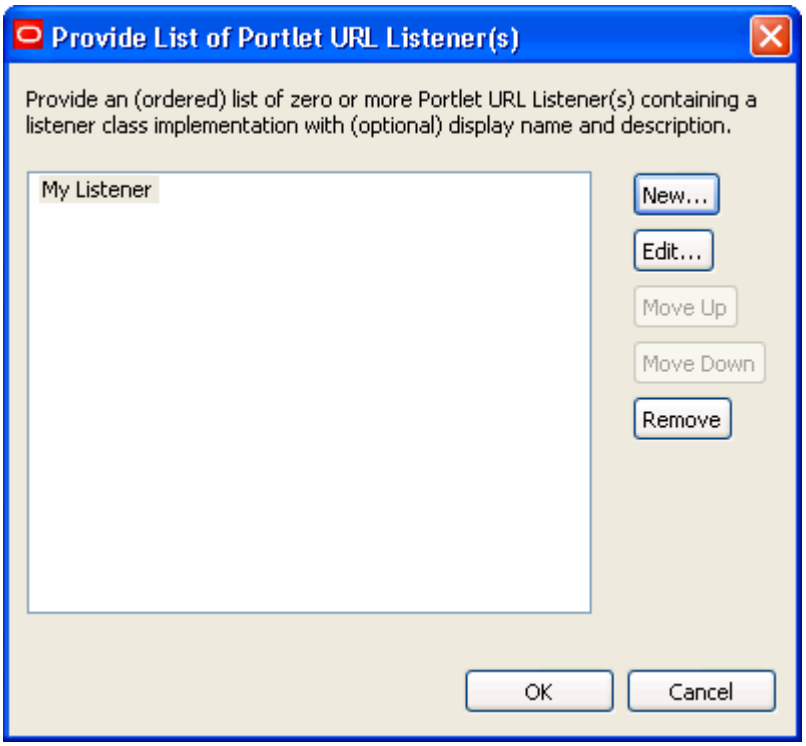
  Specifies a default namespace for the web application. The default namespace is used in cases where a public render parameter or an event is defined with only a local name. The specified namespace is added to the `portlet.xml` file. For example:

  ```
  <default-namespace>http:example.com/events</default-namespace>
  ```

- **Portlet URL Listeners**

  Portlets can register portlet URL listeners to filter URLs before they are generated. To add a listener, click the **Edit** button next to the Portlet URL Listeners property. The Provide List of Portlet URL Listener(s) dialog appears, as shown in Figure 6–13. Click the **New** button to add a new listener, or **Edit** to modify the selected listener. In the dialog that appears, provide a Class name and, optionally, a display name and description for the listener.

Figure 6–13 Provide List of Portlet URL Listener(s) Dialog



To receive a callback from the portlet container before a portlet URL is generated the listener class needs to implement the PortletURLGenerationListener interface and register it in the `portlet.xml` file. For example:

```
<listener>
    <display-name>MyURLListener</display-name>
    <listener-class>javaportlets.URLListener</listener-class>
</listener>
```

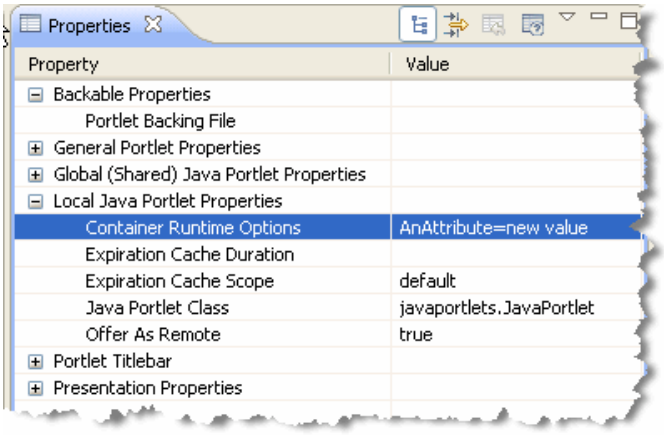
The URL listeners are called in the order they are defined in the `portlet.xml` file. The dialog box for adding URL listeners lets you move listeners up or down to change the order in which they are called.

6.13 Setting Portlet-Level Container Runtime Options

You can specify container runtime options for individual portlets. These options override any application-level settings (if any) of the same container runtime options specified at the portlet-application level.

In the Java portlet Properties view, select the **Container Runtime Options** item under Local Java Portlet Properties, as shown in Figure 6–14. Just click in the value field to bring up a dialog for adding runtime option name/value pairs. For more information on runtime options, see the Java Portlet Specification Version 2, Section 6.12, "Using Global (Shared) Properties," and Section 6.11, "Using Container Runtime Options."

Figure 6–14 Local Container Runtime Options



6.14 Adding Custom Portlet Modes

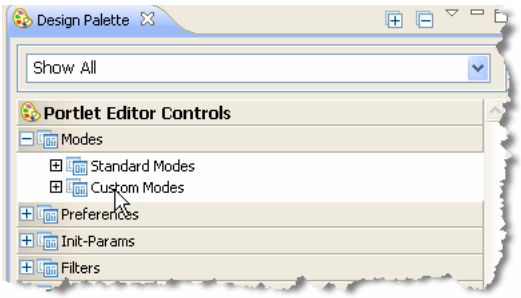
A portlet mode indicates the function a portlet is performing when the portlet is rendered. The Java Portlet Specification includes three required modes: View, Edit, and Help. In addition, the specification allows for custom modes, which provide the ability to define additional modes. Custom portlet modes fall into two categories: portal managed and not portal managed. See the Java Portlet Specification Version 2 for more information.

To add a custom portlet mode:

1. Double-click the New Custom Mode node in the Portlet Editor Controls part of the Design Palette, as shown in Figure 6–15.

   Tip: You can also drag and drop Custom Portlet Modes from the Design Palette view onto the Portlet editor. You can drag and drop either existing modes or the **Custom Modes** > **New Custom Mode** item. The position of a mode in the list of modes in a portlet is not significant.

Figure 6–15 Selecting Custom Modes

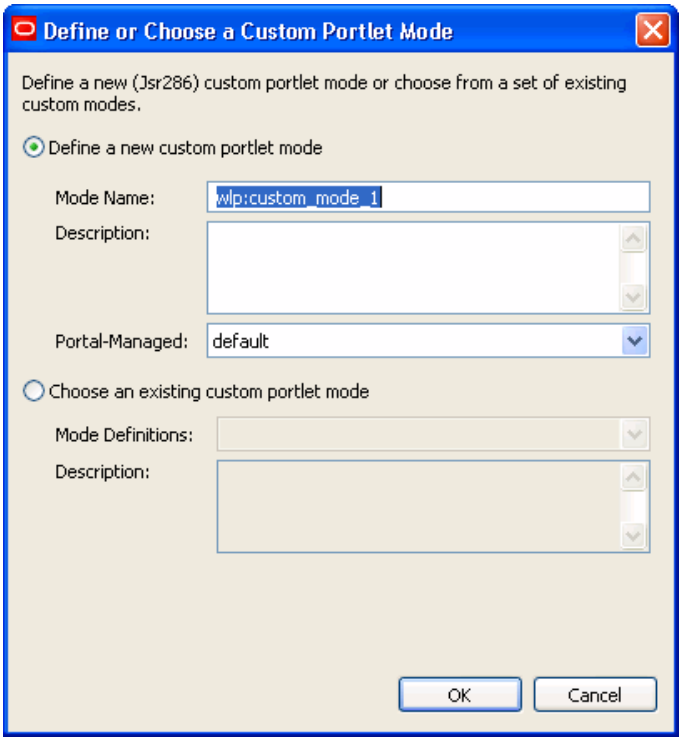


2. Fill out the Define or Choose a Custom Portlet Mode dialog. This dialog lets you specify a new custom mode or select an existing one for modification.

   The Portal-Managed drop down menu lets you select whether or not the custom mode is portal managed or not. Select default to use the server's default. In the case of WebLogic Portal server, this default is true. Select true if you want the mode to be portal managed regardless of the server's default. Select false to if you never want the mode to be portal managed.

   See Figure 6–16.

Figure 6–16 Define or Choose a Custom Portlet Mode Dialog



Custom modes are inserted into the `portlet.xml` file and can be shared by multiple portlets. Here is a sample custom mode definition:

```
<custom-portlet-mode>
    <description>My custom mode</description>
    <portlet-mode>wlp:custom_mode_1</portlet-mode>
</custom-portlet-mode>
```

6.15 Using Special Portlet Request Attributes

The WLP Java portlet container supports customization of some behaviors at runtime through the use of special portlet request attributes. Set these attributes in the portlet Java code before the operation you wish to affect. Because the request values don't change until you explicitly change them, the best practice is to explicitly change them. for example:

```
request.setAttribute("oracle.portlet.server.resourceRequiresRewriting",
Boolean.TRUE);

String url = response.encodeURL(pathToResourceForRewriting);

request.removeAttribute("oracle.portlet.server.resourceRequiresRewriting");
```

These special portlet request attributes are:

- **`oracle.portlet.server.resourceRequiresRewriting`**
  - If this request attribute is set when a call to `PortletResponse.encodeURL()` is made, the value of the request attribute is used to determine whether the resource being referred to by the URL requires WSRP rewriting on the consumer, if the portlet is being run over WSRP. If the portlet is not being run over WSRP, this request attribute has no effect. Resource proxying is the standard way to retrieve resources with WSRP: a proxy on the consumer is used to retrieve non-portlet artifacts from the producer, such as icons or images that the producer portlet may refer to. In most cases, these proxied artifacts do not contain URLs to other things on the producer (which would also need to go through the consumer's proxy to work properly), for example, an image. By default, if this request attribute is not set or is set to `Boolean.FALSE`, the resource URLs will not be rewritten on the proxy. However, for proxied resources which contain URLs that need to be re-written to also go through the consumer's resource proxy, setting this request attribute to `Boolean.TRUE` will ensure that the consumer's resource proxy rewrites URLs in the resource itself.
  - Valid values are `java.lang.Boolean` objects:
    * `java.lang.Boolean.TRUE` – Indicates that the URL encoded with `PortletResponse.encodeURL()` is to a resource which itself requires URL rewriting on the consumer.
    * Any other value – Indicates that the URL encoded with `PortletResponse.encodeURL()` is to a resource which does not itself require URL rewriting on the consumer.
- **`com.oracle.portlet.wsrpHeaderMode`**
  - If the portlet is being rendered on a WSRP producer, the value of this attribute is used to determine where the WSRP container should place the header or cookie being set on the WSRP SOAP message, as a hint to the consumer of its intended final destination. The container runtime option of the same name is used by the portlet container to set an initial value for this request attribute on each PortletRequest object, if the portlet is being rendered on a WSRP producer. If the portlet is not being rendered over WSRP, this request attribute is ignored and all headers and cookies set are assumed to be sent to the client.
  - Valid values are `java.lang.String objects`:
    * `client` – Indicates that the header or cookie is intended to go to the client.

* `consumer` – Indicates that the header or cookie is intended to be stored on the consumer.
* `both` – Indicates that the header or cookie should be sent to the client and stored on the consumer.

6.16 Using Portlet-Served Resource Links

JSR 286 portlets can create two different kinds of links to resources: direct links, and portlet-served resource links. Direct links were supported by the JSR 168 specification, while portlet-served resources are new in the JSR 286 specification.

6.16.1 Using Direct Links

Direct links are links to a resource in the same web application as the portlet, and are constructed by the portlet and encoded with the PortletResponse.encodeURL() method. Resources accessed through direct links do not go through the portal server and will not have the portlet context available. In general, use direct links when access to the portlet context or going through the portal is not needed, as direct links are more efficient than portlet-served resource links, which go through the portal framework. Examples of resources that may be best served through direct links include static images and icons that the portlet may wish to include in its markup.

6.16.2 Using Portlet-Served Resource Links

Unlike direct links, portlet-served resource links point back to the portlet, and require the portlet itself render the resource.

> **Tip:** For more information on portlet-served resources, see the JSR 286 specification, Section 13.

You can create portlet-served resource links using the `javax.portlet.ResourceURL` class, or the tag library's resourceUrl tag. When one of the portlet-served resource links is loaded by a client, the portlet container invokes a Java portlet's serveResource() method to have the portlet serve the resource. Using portlet-served resource links, the resource may be protected by portal security and has access to the portlet context. There are many examples of resources that may be best served through portlet-served resource links:

- A resource which reflects the portlet's state, such as a graph or chart based on data the portlet is displaying
- Any resource to which the portlet conditionally restricts access
- Portlet-context dependent JSON or HTML fragments for use in Ajax use-cases

> **Note:** Both direct and portlet-served resource links, when properly encoded through the PortletResponse.encodeURL() method or the appropriate ResourceURL methods, will not necessarily result in a valid URL. The portal framework may instead provide a token that is turned into a proper URL to the resource when the token is rewritten on the client. The portlet should therefore never modify a link provided by the PortletResponse.encodeURL(), ResourceURL.toString(), or ResourceURL.write() methods before writing the link to the portlet's markup.

For portlet-served resources, the portlet container acts as a proxy for accessing the resource; the portlet itself has full control over the resource response. Calls to the portlet's serveResource() method usually occur after a call to render(), if the portlet's output from the render contained any portlet-served resource links. During serveResource(), the portlet has read-only access to the portlet's current render parameters, shared render parameters, mode and state. Portlets can serve more than one resource by using a resource ID: an identifier specified on the portlet-served resource link that is provided to the portlet during a call to serveResource() in the ResourceRequest.

Caution: The `javax.portlet.GenericPortlet` class, which is extended by most Java portlets, has a default implementation of serveResource(). This default implementation simply forwards the resource request to a file in the same web application identified by the resource ID. If a portlet does not override this default behavior, it may be possible for a malicious user to construct a portlet-served resource URL which serves up any file in the portlet's web application, bypassing any security constraints set in the `web.xml`. For this reason, Oracle recommends that portlets either provide their own implementation of serveResource() or disable portlet-served resources entirely using the container runtime option `com.oracle.portlet.disallowResourceServing`. For more information, see Section 6.11, "Using Container Runtime Options."

6.17 Exporting Java Portlets for Use on Other Systems

WebLogic Portal produces Java portlets that conform to the JSR 286 specification and can be used universally across portals that support JSR 286 portlet containers. Oracle Enterprise Pack for Eclipse lets you export Java portlets to a supported archive file (WAR, JAR, or ZIP) that can be deployed on any supported server.

Tip: You can also use the Import feature to import archive files containing Java portlets into your Oracle Enterprise Pack for Eclipse workspace. For details, see Section 6.18, "Importing Java Portlets."

Note: Throughout this section, supported archive files refer to WAR, JAR, and ZIP files.

By default, Oracle Enterprise Pack for Eclipse exports the `portlet.xml`, any Java class files required by the portlet, and any Java source files. Also, if any class or source Java files are found within a JAR or ZIP archive, that archive is also exported. You can optionally specify additional files to be exported.

To export Java portlets to a supported archive file:

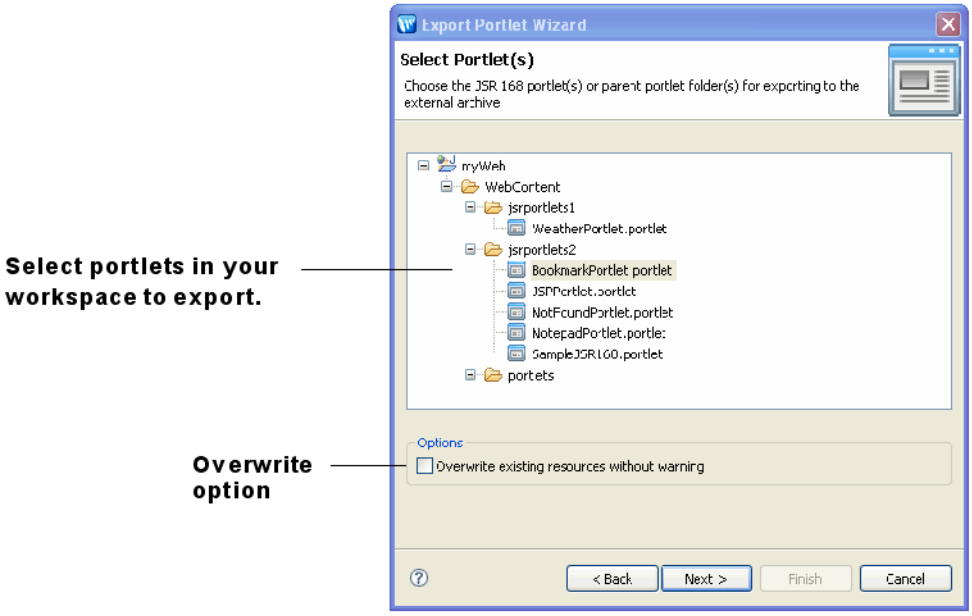
1. Select **File** > **Export**. You can also right-click in the Project Explorer and select **Export** > **Export...**
2. In the Export page of the export wizard, open the WebLogic Portal folder and select **Portlet(s) to Archive**.
3. Click **Next**.
4. In the Select Portlets page of the export wizard, expand the web project that contains the Java portlet(s) you want to export. You can select the parent folder

that contains the portlet(s) or drill down to select individual portlets, as shown in Figure 6–17. Any portlets and/or parent folders that were selected in the Project Explorer will be pre-selected by default.

Select the **Overwrite existing resources without warning checkbox** to force the export tool to overwrite duplicate files automatically.

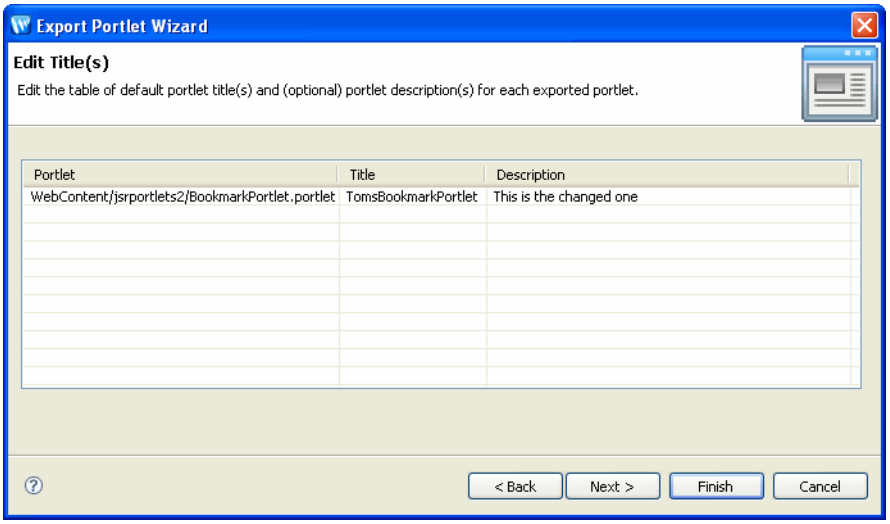
Note: All selected portlets must exist within the same web project. You cannot select portlets for export across different web projects.

Figure 6–17 Select Portlet(s) to Export Dialog



5. Click **Next**. The Edit Title(s) dialog appears, as shown in Figure 6–18.

Figure 6–18 Edit Title(s) Dialog



6. In the Edit Title(s) dialog, you can add or modify the Title and/or Description of an exported portlet. These fields are written to the exported portlet's `portlet.xml` file. Click **Next** to continue.

   Note: **Next** button is disabled until you supply a title.

7. In the Select Archive page of the export wizard, enter a full path and name for the archive file, or use the **Browse** button to specify the path, and click **Next**. If the archive does not exist, the wizard will prompt you to create it.
8. In the Select Format page of the export wizard, pick an archive format from the dropdown list. WLP provides two choices:
   - **Oracle WebLogic Server (Generic)** – Outputs just the standard Java Portlet files to the archive, like `portlet.xml` and the Java Portlet class files. This is the default option.
   - **Oracle WebLogic Portal** – In addition to exporting the standard Java Portlet files, this formatter also exports standard WLP files, such as `.porlet` files, backing file classes, and `.dependencies` files.
9. In the Select Files dialog, select any optional supporting files, such as JSPs, that you wish to include in the supported archive file. Any files that are included in the selected archive format (such as `portlet.xml`) are automatically selected in the dialog. You can associate a Target Path path with any selected files. Those files will be placed in the specified target path within the archive file. By default, all files are stored relative to the root directory of the archive.
10. Click **Finish**. The archive file is created in the location you specified.

6.18 Importing Java Portlets

WLP provides two tools for importing Java (JSR 168/286) portlets. One lets you import Java portlets into the your Oracle Enterprise Pack for Eclipse workspace. You can then add the imported portlets to a portal application, modify them with the portlet editor, or perform other development functions. The other tool lets you import and deploy Java portlets using the Portal Administration Console. This section discusses these tools and techniques for importing JSR 168/286 portlets into a WLP application. This section includes these topics:

- Section 6.18.1, "Importing Java Portlets Into Your Eclipse Workspace"
- Section 6.18.2, "Importing and Deploying JSR 286 Portlets in the Administration Console"

6.18.1 Importing Java Portlets Into Your Eclipse Workspace

This section explains how to use the Oracle Enterprise Pack for Eclipse to import portlets and includes these topics:

- Section 6.18.1.1, "Starting the Import Wizard"
- Section 6.18.1.2, "Using the Import Wizard"
- Section 6.18.1.3, "Accessing the Portlets"
- If you are importing JSR 286 portlets into an existing web application that contains JSR 168 portlets, see Section 6.19, "JSR-286/JSR-168 Portlet Compatibility" for important guidelines.

6.18.1.1 Starting the Import Wizard

To start the import wizard, do the following:

1. Open Oracle Enterprise Pack for Eclipse.
2. Open the portal application into which you want to import the JSR 286 portlets.
3. Select **File** > **Import** or right-click on the portal application and select **Import**.

6.18.1.2 Using the Import Wizard

This section explains how to use the import wizard to import JSR 286 WAR files into an enterprise application.

1. On the first page of the import wizard, select **WebLogic Portal** > **Portlet(s) from Archive**. Click **Next** to continue
2. On the **Select Project** page of the import wizard, select the project into which the JSR 286 WAR will be imported. The list is automatically populated with the projects in the current workspace. If you would like to replace an earlier version of the portlet, select **Overwrite existing resources without warning**. Click **Next** to continue

   > **Note:** If the selected project does not include a portlet deployment descriptor, one will be generated when the JSR 286 WAR file is imported.

3. On the **Select Archive** page of the import wizard, enter the path to the WAR file containing the JSR 286 compliant portlets to deploy. You can also use the **Browse** button to navigate to the WAR file. Click **Next** to continue
4. On the **Select Portlet(s)** page of the import wizard, choose the portlets to import from the JSR 286 WAR selected on the previous page. By default, the portlet(s) will be imported into the root folder of the portal project chosen in step 2. To import a portlet into a different location, select the portlet and enter the path in the Target Path field or click **Browse** to navigate to the folder. Click **Next** to continue
5. The **Select Files** page of the import wizard lists the artifacts that will be included in the portlet project. By default, all artifacts specified as required in the Import Template are selected, including the `portlet.xml` file and any class files used by the portlet. Select any additional supporting portlet artifacts to include in the portlet project (you can select directories or individual files). By default, the files will be imported into the portlet project in the structure shown. To import an artifact into a different location, select the file or directory and enter the path in the Target Path field or click **Browse** to navigate to the folder. Click **Next** to continue
6. Click **Finish** to execute the import. If the selected project does not include a portlet deployment descriptor, an error will appear to notify you that a new descriptor file will be generated.

6.18.1.3 Accessing the Portlets

After the WAR file is imported, you can add the portlet(s) to your portal. You can use them in file based `.books`, `.pages`, and `.portals`, or in desktop streaming mode using the WebLogic Administration Console once they have been published to the server from the IDE. The imported portlets can be served over WSRP if your portlet producer project is configured to do so. After a web application is added as a producer, you can incorporate the application's portlets as you would with any WSRP producer using the WebLogic Portal Administration Console. See the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal* for details.

Tip: If your producer and consumer applications share the same server, it is recommended that you enable local proxy mode. Local proxy support allows co-located producer and consumer web applications to short-circuit network I/O and "SOAP over HTTP" overhead. See the section "Using Local Proxy Mode" in the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

6.18.2 Importing and Deploying JSR 286 Portlets in the Administration Console

The WebLogic Portal Administration Console provides a utility for automatically deploying JSR 286 portlets that are packaged in JSR 286 WAR files. This utility lets you import JSR 286 WAR files containing JSR 286 portlets, and expose the portlets in WSRP producers. For detailed information on this utility, see the chapter "Deploying Portal Applications" in the *Oracle Fusion Middleware Production Operations Guide for Oracle WebLogic Portal*.

6.19 JSR-286/JSR-168 Portlet Compatibility

When you use Oracle Enterprise Pack for Eclipse to import JSR 286 portlets into an existing web application that contains JSR 168 portlets, the existing `portlet.xml` is upgraded to the JSR 286 schema. (Note that if you import portlets manually, by editing the schema files, this upgrade will not occur automatically.) In most cases, the JSR 168 portlets will continue to work exactly as they did before. However, there are a few cases in which JSR 168 portlets will behave differently in JSR 286; these portlets must invoke a JSR 168 compatibility mode to run under JSR 286.

Note: JSR 168 portlets run in a JSR 286 container without JSR 168 compatibility mode invoked may be subject to a potential security hole through the new portlet-served resource feature in JSR 286. Because the `GenericPortlet` base class implements the `serveResource()` method by simply looking for a file in the web application with the name specified in the resource ID, malicious users could gain access to files not intended to be served. When any JSR 168 compatibility mode is used, the WLP JSR 286 container disables resource serving for these portlets by default using the `com.oracle.portlet.disallowResourceServing` container runtime option. If JSR 168 portlets are included in a JSR 286 `portlet.xml` file without specifying a JSR 168 compatibility mode, Oracle recommends that you use the `com.oracle.portlet.disallowResourceServing` container runtime option to disable resource serving. For information on setting this container runtime option for individual portlets, see Section 6.13, "Setting Portlet-Level Container Runtime Options."

The WLP JSR 286 portlet container invokes a JSR 168 compatibility mode for a portlet if either of the following are true:

- The `portlet.xml` in which the portlet is defined has an XML namespace that does not contain portlet-app_2_0 (it is a JSR168 `portlet.xml` file). If a file exists in the web application named `/WEB-INF/oracle-portlet.xml`, the WebCenter JSR 168 compatibility mode is invoked; otherwise, the WLP JSR 168 compatibility mode is not invoked.
- The container runtime option `com.oracle.portlet.compatibilityMode` is set to `owlp168` (WLP JSR 168 compatibility mode) or `owc168` (WebCenter JSR 168

compatibility mode). This allows JSR 286 portlets and JSR 168 portlets from WLP and/or WebCenter to co-exist in the same web application.

- Any `hide-portlet` entries with a value of `true` will automatically be converted to a `false` value for the container runtime option `com.oracle.portlet.offerPortletOverWsrp` if no value for that container runtime option is specified in the `portlet.xml` file.

If a portlet has a JSR 168 compatibility mode invoked, the portlet container will make the modifications described in the following sections for the portlet to remain backwards-compatible with JSR 168.

6.19.1 Generic JSR 168 Compatibility Modifications

To be compliant with the JSR 168 specification, the following modifications will be made if any JSR 168 compatibility mode is invoked:

WLP JSR 168 Compatibility Mode Functionality	Standard JSR 286 Functionality
BaseURL.setParameter(String name, String value) will throw an IllegalArgumentException if value is null.	The parameter will be removed if it was already set on the BaseURL
BaseURL.setParameter(String name, String[] values) will throw an IllegalArgumentException if values is null.	The parameter will be removed if it was already set on the BaseURL
BaseURL.setParameters(Map<String, String[]>) will throw an IllegalArgumentException if any of the map values is null.	The parameter will be ignored.
When using a PortletRequestDispatcher to dispatch to a servlet or JSP, the provided request.getProtocol() method will return null.	The provided request.getProtocol() method will return "HTTP/1.1"
RenderResponse.getWriter() and RenderResponse.getOutputStream() will throw an IllegalStateException if RenderResponse.setContentType() has not been called previously.	Setting the content type is not required before calling getWriter or getOutputStream.
PortletContext.getMajorVersion() will return 1.	PortletContext.getMajorVersion() will return 2.
The container runtime option `com.oracle.portlet.disallowResourceServing` will be set to `true` for the portlet if no other value has been specified for that container runtime option.	The container runtime option `com.oracle.portlet.disallowResourceServing` will be left at the default of `false` for the portlet if no other value has been specified for that container runtime option.

6.19.2 WebLogic Portal JSR 168 Compatibility Modifications

The following modifications will be made in addition to the generic modifications if the WLP JSR 168 compatibility mode is invoked:

WLP JSR 168 Compatibility Mode Functionality	Standard JSR 286 Functionality
PortalURL.toString() will use the web application setting for ampersand entity encoding.	BaseURL.toString() must return a URL that does not encode ampersands as XML entities.
When using the JSR 168 tag library, the URLs generated by the actionUrl and renderUrl tags will always inherit the default ampersand encoding for the web application.	URLs must not encode ampersands as XML entities.
PortletResponse.encodeURL() will use the web application setting for ampersand entity encoding.	PortletResponse.encodeURL() will always use the ampersand character, not the ampersand entity.

WLP JSR 168 Compatibility Mode Functionality	Standard JSR 286 Functionality
The portlet container will generate errors if render is called on a portlet and the portlet does not implement the methods render(), doDispatch(), doView() (if portlet mode is VIEW), doEdit() (if portlet mode is EDIT), or doHelp() (if portlet mode is HELP).	The portlet's render() method will be invoked regardless.
The portlet container will generate errors if an action is invoked on a portlet and the portlet does not implement processAction().	The processAction() method will be called regardless.
The container will automatically set the `com.oracle.portlet.streamingOptimized` runtime option to "true" if no other value for that option is specified, as this was the previous behavior of JSR168 portlets in WLP.	The container `com.oracle.portlet.streamingOptimized` runtime option will be left at the default of "false" if no other value for that option is specified.

6.19.3 WebCenter JSR 168 Compatibility Modifications

The following modifications will be made in addition to the generic modifications if the WebCenter JSR 168 compatibility mode is invoked:

WebCenter JSR 168 Compatibility Mode Functionality	Standard JSR 286 Functionality
PortalURL.toString() will always use the XML ampersand entity.	BaseURL.toString() must return a URL that does not encode ampersands as XML entities.
When using the JSR 168 tag library, the URLs generated by the actionUrl and renderUrl tags will always inherit the default ampersand encoding for the web application.	URLs must not encode ampersands as XML entities.
PortletResponse.encodeURL() will use the XML ampersand entity for encoding the URL.	PortletResponse.encodeURL() will always use the ampersand character, not the ampersand entity.

WebCenter JSR 168 Compatibility Mode Functionality	Standard JSR 286 Functionality
If present, the `WEB-INF/oracle-portlet.xml` file will be parsed and the following configuration information will be honored: ▪ `minimum-wsrp-version` entries will automatically get converted to com.oracle.portlet.minimumWsrpVersion container runtime options ▪ `requires-iframe` will be converted to com.oracle.portlet.requiresIFrame container runtime options, if no such options are specified in the portlet.xml file. This will be honored both when the portlet is run locally and produced over WSRP. ▪ `navigation-parameters` will automatically get converted to JSR286 public render parameters. ▪ `hide-portlet` entries with a value of `true` will automatically get converted to a `false` value for the container runtime option `com.oracle.portlet.offerPortletOverWsrp` if no value for that container runtime option is specified in the `portlet.xml` file ▪ `portlet-extension portlet-id` entries are automatically converted to `com.oracle.portlet.wsrpPortletHandle` container runtime options, if no such options are specified in the `portlet.xml` file. The value of the container runtime option will be "E:i" + portlet-id + ":default". ▪ `portlet-app-extension allow-export` entry will be converted to a `com.oracle.portlet.allowWsrpExport` container runtime option at the portlet-app level, if no such option is specified in the `portlet.xml` file. ▪ `portlet-app-extension strict-authentication` will be converted to a `com.oracle.portlet.useWsrpUserContextFor UserAuthenticationInfo` container runtime option; if `strict-authentication` is `true`, the container runtime option will be `false`, otherwise the container runtime option value will be `true`.	
The container runtime option com.oracle.portlet.useWsrpUserContextForUserAuthenticationInfo will be set to "true" unless the oracle-portlet.xml file exists and has a portlet-app-extension "strict-authentication" element value of "true".	The container com.oracle.portlet.useWsrpUserContextForUserAuthenticationInfo runtime option will be left at the default value of "false" if no other value for that option is specified.

6.20 Adding an Icon to a Java Portlet

To add an icon to a Java portlet, you need to edit the `weblogic-portlet.xml` file, as described in this section.

1. Place the icon in the images directory of the skin that the portal is using. For example, if the skin name is avitek, icons must be placed in:

   ```
   myPortal/skins/avitek/images
   ```

2. In the Application panel, locate and double-click the `weblogic-portlet.xml` file to open it. This file is located in the portal's WEB-INF folder, for example:

   ```
   myPortal/WEB-INF/weblogic-portlet.xml
   ```

3. Add the following lines to the `weblogic-portlet.xml` file:

   ```
   <portlet>
     <portlet-name>myPortlet</portlet-name>
   ```

```
  <supports>
    <mime-type>text/html</mime-type>
    <titlebar-presentation>
      <icon-url>myIcon.gif</icon-url>
    </titlebar-presentation>
  </supports>
</portlet>
```

4. Make these substitutions:
   - Change *myPortlet* to the name of the portlet that is specified in WEB-INF/portlet.xml
   - Be sure the mime-type also matches the mime-type found in WEB-INF/portlet.xml
   - Change *myIcon.gif* to the name of the icon you wish to add

7

Creating Clipper Portlets

A clipper portlet is a portlet that renders content from another web site. A clipper portlet can include all or a subset of another web site's content using a process called "web clipping." This chapter explains how to create and configure clipper portlets.

This chapter includes these topics:

7.1 Introduction

Clipping is an easy technique for including content in your portal. You can clip all or part of another web site. Users can effectively view and interact with content from another web site without leaving the portal.

Note that another WLP feature, the browser portlet, also lets you include remote web page contents in a portal. For information on browser portlets, see Section 5.4.4, "Building Browser Portlets." A clipper portlet differs from a browser portlet in the following ways:

- A browser portlet uses an IFrame, while a clipper portlet includes the content of the clipped web page into the same page as the rest of the portal. An advantage of using an IFrame is that it isolates the remote content, preventing it from overlapping other parts of the portal. Disadvantages are that the portal cannot access the IFrame's content and portal and IFrame sessions are maintained separately.

- A browser portlet returns the entire remote page, while a clipper portlet lets you subset or modify the contents of a remote web page.

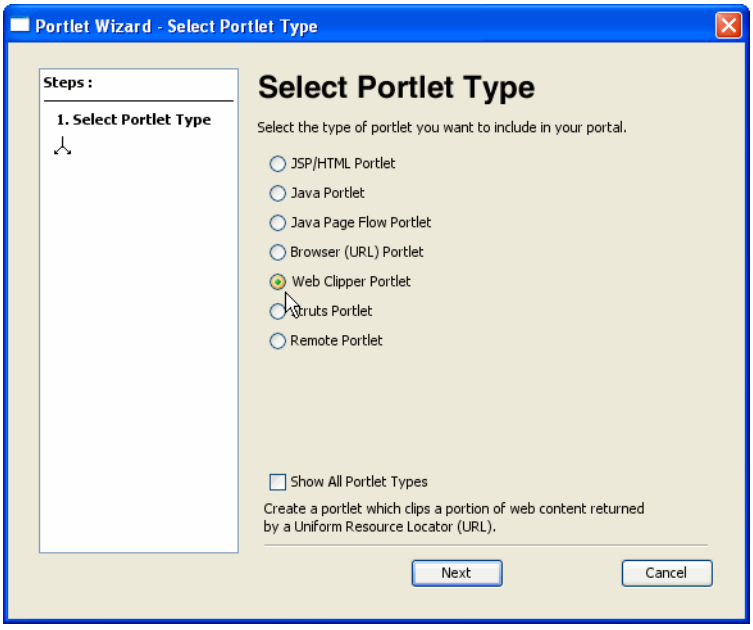
7.2 Creating a Clipper Portlet

You create a clipper portlet using the Portlet Wizard. The steps are similar to those of creating other types of portlets.

> **Note:** No post-processing is performed on the text of a clipped web page, unless a clipCustomClass preference is specified as described in Section 7.4, "Modifying the Appearance of a Clipper Portlet." Clipped text is written verbatim to the response. If the original web page contains syntax errors, the errors may also appear in the consumer browser when the clipper portlet is rendered.

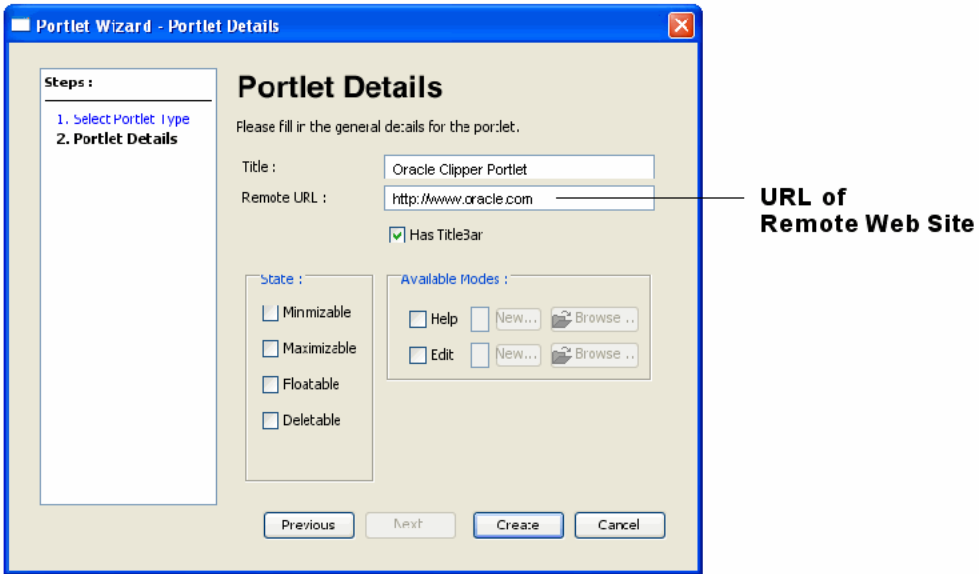
1. If it is not currently open, open the Portal Perspective.
2. Select **File** > New > Portlet.
3. In the New Portlet dialog, enter a name for the portlet, and click Finish. The Portlet Wizard opens.
4. In the Portlet Wizard, select **Web Clipper Portlet**, as shown in Figure 7–1 and click **Next**.

Figure 7–1 Selecting Web Clipper Portlet



5. In the Portlet Details dialog, enter a title for the portlet and enter the URL of the web site you want to clip in the Remote URL field, as shown in Figure 7–2.

Figure 7–2 Specifying the URL of a Remote Web Site



6. Click **Create** to create the new clipper portlet.
7. Modify the clipper portlet, if you want, by adding and editing preferences, as explained in Section 7.3, "Modifying Clipper Portlet Properties."

Note: By default, the entire web site is included in the clipper portlet's contents, including the <HEAD> element of the remote site.

7.3 Modifying Clipper Portlet Properties

By setting certain portlet properties, you can change the appearance of a clipper portlet, subset the content of a web page that appears in a clipper portlet, and provide authentication. There are two primary ways to modify a clipper portlet's properties: through the Properties editor and manually.

This section includes these topics:

- Section 7.3.1, "Using the Properties Editor"
- Section 7.3.2, "Setting Clipper Properties Manually as Preferences"

7.3.1 Using the Properties Editor

You can use the Properties editor to edit the common set of portlet properties, such as the title bar and presentation properties. Clipper portlets share most of these properties with other types of portlets, and the procedure for changing them is the same. See Section 9.1, "Portlet Properties" for detailed information on editing portlet properties through the Properties editor.

7.3.2 Setting Clipper Properties Manually as Preferences

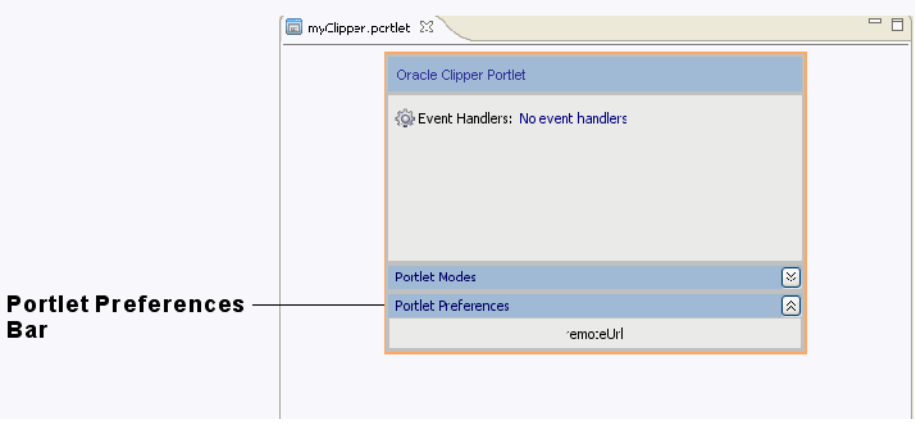
Clipper portlets also include a set of properties that do not appear in the Properties editor and which must be set manually. The easiest way to modify clipper portlet properties manually is to add and set them as portlet preferences.

Tip: WLP provides preferences for controlling the extent of a clipped page and for authentication. See Section 7.4, "Modifying the Appearance of a Clipper Portlet" and Section 7.5, "Authenticating a Clipper Portlet" for specific information on those tasks.

To set portlet preferences, do the following:

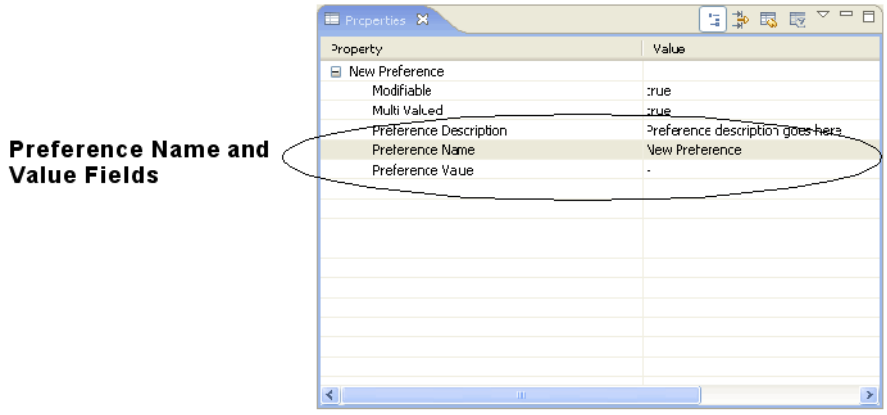
1. Open the Portlet editor for the clipper portlet. To do this, right click the portlet name in the Project Explorer and select Open With > Portlet Editor.
2. Right click the Portlet Preferences bar in the portlet editor and select Add Preference. The Portlet Preferences bar is shown in Figure 7–3.

Figure 7–3 Portlet Preferences Bar



3. In the Properties editor, enter the Preference Name and Preference Value in the appropriate fields, as shown in Figure 7–4.

Figure 7–4 Preference Name and Value Fields



For more information on setting portlet preferences, see Section 9.2, "Portlet Preferences."

7.4 Modifying the Appearance of a Clipper Portlet

You can set portlet preferences to specify which portion of a web page to clip. You can also modify the text that is clipped by implementing a Java class and specifying it as a portlet preference. Table 7–1 lists and describes the set of clipper portlet preferences

that you may set manually to accomplish these tasks. For details on how to set preferences, see Section 7.3, "Modifying Clipper Portlet Properties."

> **Note:** The preferences clipXPath, clipStartText/clipEndText, and clipCustomClass (listed in Table 7–1) are exclusive. The system looks for clipCustomClass first. If that class is not present, the system looks for clipXPath. If clipXPath is not present, the system looks for clipStartText/clipEndText.

Table 7–1 Preferences for Determining the Text to Clip

Property Name	Property Value
clipXPath	(Optional) An XPath that is applied to the remote page. The remote page is required to be well-formed XML. If you set this option, the system will apply the XPath to the remote page and put the text of the first node found in the clipper portlet output. This option provides a convenient way to clip a specific chunk of text. For example, suppose this preference value is html/body[1], which is an XPath expression that selects the <body> element of the web page's text. You can also use this option to specify div elements to clip. For example, //div[@id="barracuda"] clips out a element.
clipStartText, clipEndText	(Optional) Specifying regular expressions that are used to locate the beginning and the end of the web page text to clip. For example, if the page you want to clip looks like: Some web site text... <abc> text to clip </abc> Some more web site text... and you want to clip the text between <abc> and </abc> inclusive, enter the following properties and values: clipStartText = <abc> clipEndText = </abc> The left angle bracket needs to be escaped if you enter the values directly in the XML .portlet file. For example: <netuix:preference name="clipStartText" value="<abc>" modifiable="false"/> <netuix:preference name="clipEndText" value="<abc>" modifiable="false"/>
clipCustomClass	(Optional) This preference specifies the name of a class that implements com.bea.netuix.clipper.IClipStrategy. This interface lets you define your own clipping logic. The interface has one method to implement: String clip(String markup); Your implementation must have a no-argument constructor. The clipCustomClass preference registers your implementation with the portlet. An IClipStrategy class can be used to selectively rewrite a web page; for example, you can substitute text in the page, or suppress certain elements.

7.5 Authenticating a Clipper Portlet

This section explains how to configure authentication for a clipper portlet. Once configured, clipper portlet authentication is automatic. WebLogic Portal supports two kinds of clipper portlet authentication:

- Form-based authentication
- Basic HTTP authentication

Both of these methods are described in this section.

7.5.1 Form-Based Authentication

Form-based authentication is performed through a server-side form request on the remote site. You configure this type of authentication by setting preferences on the portlet. The procedure for setting preferences is described in Section 7.3, "Modifying Clipper Portlet Properties."

> **Note:** There are current security limitations associated with form-based authentication. See Section 7.12, "Clipper Portlet Limitations."

To set up form-based authentication:

1. Set the preference: authenticationType = Form. This preference enables form-based authentication.
2. Tell the server how to build up the HTTP request that performs the authentication by setting the preferences listed in Table 7–2.
3. Provide the authentication credentials by setting the preferences listed in Table 7–2.

Table 7–2 HTTP Request Preferences

Preference Name	Preference Value
loginFormUrl	(Required) The ACTION attribute in the HTML <FORM> element. This is the URL to which the authentication request is made.
loginFormMethod	(Required) The METHOD attribute in the HTML <FORM> element. The value must be either GET or POST.
loginFormUserParam	(Required) The name of the request parameter that holds the login name.
loginFormPasswordParam	(Required) The name of the request parameter that holds the login password.
loginFormExtraParams	(Optional) A string to append to the request query. Use this string to specify custom parameters that might need to be set. For example, if the form also has COLOR and SHAPE parameters, you can set them with: loginFormExtraParams= COLOR=PURPLE&SHAPE=DIAMOND

The following preferences are used to provide the authentication credentials.

Table 7–3 Authentication Credential Preferences

Preference Name	Preference Value
groupUsername	Specifies shared user name.
groupPassword	Specifies the password for the shared user name.
personalUsername	Specifies a user name on a per-user basis. Ignored if groupUsername is set.
personalPassword	Specifies the password on a per-user basis. Ignored if groupUsername is set.

Example 7–1 shows example preferences for form-based authentication.

Example 7–1 Example Form-Based Authentication Preferences

```
<netuix:preference name="remoteUrl" value="http://some.site.com"
modifiable="false"/>
<netuix:preference name="loginFormUrl" value="http://some.site.com/login.action"
modifiable="false"/>
```

```
<netuix:preference name="authenticationType" value="Form" modifiable="false"/>
<netuix:preference name="loginFormMethod" value="POST" modifiable="false"/>
<netuix:preference name="loginFormUserParam" value="os_username"
modifiable="false"/>
<netuix:preference name="loginFormPasswordParam" value="os_password"
modifiable="false"/>
<netuix:preference name="loginFormExtraParams" value="os_destination=abc"
modifiable="false"/>
<netuix:preference name="groupUsername" value="your_username" modifiable="false"/>
<netuix:preference name="groupPassword" value="your_password" modifiable="false"/>
```

7.5.2 Basic HTTP Authentication

To set up basic HTTP authentication:

1. Set the preference: authenticationType = BasicHTTP. This preference enables form-based authentication.
2. Provide the authentication credentials by setting the preferences listed in Table 7–4.

Table 7–4 Authentication Credential Preferences

Preference Name	Preference Value
groupUsername	Specifies shared user name.
groupPassword	Specifies the password for the shared user name.
personalUsername	Specifies a user name on a per-user basis. Ignored if groupUsername is set.
personalPassword	Specifies the password on a per-user basis. Ignored if groupUsername is set.

Example 7–2 shows example basic HTTP authentication preferences.

Example 7–2 Example Basic HTTP Authentication Preferences

```
<netuix:preference name="authenticationType" value="BasicHTTP"
modifiable="false"/>
<netuix:preference name="groupUsername" value="your_username" modifiable="false"/>
<netuix:preference name="groupPassword" value="your_password" modifiable="false"/>
```

7.6 Configuring URL Rewriting

This section explains how to configure the way clipper portlets rewrite navigable links and resource URLs.

This section includes these topics:

- Section 7.6.1, "Navigable Link Configurations"
- Section 7.6.2, "Resource URL Configurations"
- Section 7.6.3, "URL Rewriting Configuration Techniques"

7.6.1 Navigable Link Configurations

Navigable links, such as anchor links, can be configured as follows:

- Rewrite the link so that the resulting page displays in the portlet. This is the default.

- Do not rewrite the link. In this case, if you click the link, the linked page opens in another browser window, not in the portlet.
- Block the link. Because a clipper portlet embeds the URL that defines a page to clip in the portal request, it is possible to manually change the URL so the portlet clips an arbitrary web page. This presents a security risk, because a user could browse web pages from the WLP server, which may be behind a firewall and thus allow access to pages that aren't authorized for the given user. Clipper portlets can be configured to block this security risk.

See Section 7.6.3, "URL Rewriting Configuration Techniques" for more information.

7.6.2 Resource URL Configurations

Resource URLs point to images, stylesheets, scripts, and so on. You can configure a clipper portlet so that it either does or does not rewrite resource links so that they are proxied through the WLP server.

By default, resources are proxied, because cookies for clipped pages are stored on the WLP server. For example, if you clip a page behind a firewall, your browser will not have access to resources on the remote page. In this case, it is necessary to route resource requests through the WLP server. However, this proxying can affect WLP server performance; therefore, you have the option to turn proxying off if you don't need it.

See Section 7.6.3, "URL Rewriting Configuration Techniques" for more information.

7.6.3 URL Rewriting Configuration Techniques

You can configure the way URLs are rewritten by implementing a Java class called IClipperUrlFilter or by setting portlet preferences.

7.6.3.1 Implementing IClipperUrlFilter

The SPI interface com.bea.netuix.clipper.IClipperUrlFilter is available for you to define your link rewriting rules. This interface has three methods, listed in Example 7–3. Your implementation must have a no-argument constructor. Register your implementation with the portlet using the `urlFilter` portlet preference. For example:

```
<netuix:preference name="urlFilter" value="my.package.MyUrlFilterImpl"
modifiable="false"/>
```

For more information on setting portlet preferences, see Section 7.3.2, "Setting Clipper Properties Manually as Preferences" and see "Portlet Preferences" in the *Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal*.

Example 7–3 IClipperUrlFilter Methods

```
/** Should the url be reachable from the clipper portlet?
   * If this method returns false, rewritten links containing this url will have
   * empty values (for example, a link <a href="forbidden.site.com">
   * would be rewritten to <a href="">, and a request to clip this url would
   * receive a 404 response.
   */
  boolean allowUrl(String url);

  /**
   * Should the url be rewritten to stay within portal context? If this methods
   * returns false, clicking on a link
   * to this url will take the user straight to the target url, which will render
```

```
     * the new page in the full browser, and not inside the clipper portlet.
     *
     * This method applies to navigable urls only: links in anchors, form actions,
     * etc.
     */
    boolean rewriteClickableUrl(String url);

    /**
     * For resource urls only, e.g. image, script, and style tags.
     *
     * Should the resource be proxied through the wlp server, or should the resource
     * link point
     * directly to the original resource in the remote page?
     */
    boolean rewriteResourceUrl(String url);
```

7.6.3.2 Using Portlet Preferences

If you don't want to define your own class to control link rewriting, you can use these portlet preferences. For more information on setting portlet preferences, see Section 7.3.2, "Setting Clipper Properties Manually as Preferences" and see "Portlet Preferences" in the *Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal*.

- **allowedUrlRegex** – Set the value of this portlet preference to a regular expression. WLP tries to match URLs against this expression, and if the match fails, the link will be blocked. For example:

  ```
  <netuix:preference name="allowedUrlRegex" value=".*allowedlink.*"
  modifiable="false"/>
  ```

- **proxyResourceUrls** – If this portlet preference is set to false, resource URLs will not be rewritten to run through the WLP server. The default value is true. The following example turns off rewriting for resource links:

  ```
  <netuix:preference name="proxyResourceUrls" value="false*" modifiable="false"/>
  ```

  Suppose the clipped page has an image tag . If the proxyResourceUrls preference value is false, then the clipper page will have the exact same image link:

  ```
  <img src="abc.com/myImage.gif"/>
  ```

  But if the value is set to true, the link will look like this:

  ```
  <img src="myportal.com?[BEA internal link representation]" />
  ```

7.7 Clipper Portlets and HTTPS

This section discusses how to handle clipper portlets with HTTPS URLs.

When an HTTPS link is clipped, the link shows up as an HTTPS link in the portal page.

If you click on a clipped link that causes a redirect on the remote site from an HTTP URL to an HTTPS URL, the portal request is redirected from an HTTP URL to an HTTPS URL, as expected. However, note the following exception to this case: the initial request to the portal for a given browser session is never redirected to HTTPS. The following cases illustrate this exception:

The page `www.xyz.com` has a link to `xyz.com/mail`. This link points to `http://mail.google.com/mail`, and clicking that link redirects you to `https://www.google.com/accounts/ServiceLogin`.

If you clip `http://www.xyz.com` into your portal at `http://myportal.com`, start your browser, and click the mail link in your clipped portal, the portal request will be redirected to `https://myportal.com`. The clipped page will be redirected to, for example, `https://www.xyz.com/accounts/ServiceLogin`, and you will see that page.

If you clip `http://mail.xyz.com/mail`, and you start your browser and open the portal, then you will not be redirected to HTTPS. The clipped page will still follow the redirect to, for example, `https://www.zyz.com/accounts/ServiceLogin`, so the page contents will look fine, but the route from the browser to the WLP server will not use HTTPS. Note that the "Sign In" form on that page has an HTTPS action URL, so the action for the clipped form will point to `https://myportal.com`.

Likewise, if you clip `https://www.xyz.com/accounts/ServiceLogin`, and you start your browser and go to `http://myportal.com`, then you will not be redirected to HTTPS on that initial request.

7.8 Certificates and WebLogic Server

For WLS to make an HTTPS request to a site, it must have a certificate for that site in its keystore. If the certificate is not available, you will see exceptions such as:

```
[Security:090477]Certificate chain received from aaa.bbb.com - 10.123.45.67 was
not trusted causing SSL handshake failure.
```

For detailed information on configuring SSL in WLS, see "Configuring Identity and Trust" in *Oracle Fusion Middleware Securing Oracle WebLogic Server*. The basic steps to configure a clipper portlet to use HTTPS correctly are:

1. Obtain a security certificate for the site you are trying to clip.

   Tip: One way to obtain the certificate is to use the Firefox plug-in called "Cert Viewer Plus." This plug-in lets you view and save the security certificate.

2. Open a command shell and navigate to the root directory of your domain.
3. Locate the trust keystore; for example, `DemoTrust.jks`.
4. Obtain the password for the keystore.
5. Use the Java keytool program to import the key. For example:

   ```
   keytool -import -file my_certificate_file -keystore
   DemoTrust.jks -alias some_unique_alias
   ```

   The alias value is an alias unique to that .jks file. You can view the aliases with the command:

   ```
   keytool -list -keystore DemoTrust.jks
   ```

7.9 Refreshing the Original Clipper Portlet Page

By default, if a user clicks a link inside a clipper portlet, then visits another page within the portal, and then returns to the page with the clipper portlet, the clipper

portlet "remembers" and displays the last URL rendered within the clipper portlet. It is possible, however, to change this default behavior.

The refresh feature discussed in this section allows you to configure a clipper portlet to return to its original state in response to an `onActivation` portal event instead of rendering the portlet with the currently cached URL for the clipper portlet. This type of event is fired by a page change or when the portlet is minimized/restored.

> **Tip:** The refresh feature automatically returns to the URL specified by the `remoteURL` portlet preference for the clipper portlet. This preference specifies the initial URL for the clipper portlet. You specify this URL when you create the clipper portlet. You can change it by clicking the **Portlet Preferences** bar in the clipper portlet editor.
>
> Note that another `remoteURL` property is cached with the clipper portlet's state in the session. This cached property keeps track of the last URL accessed by the user in the clipper portlet. This cached property is not necessarily the same as the remote URL specified by the portlet preference.

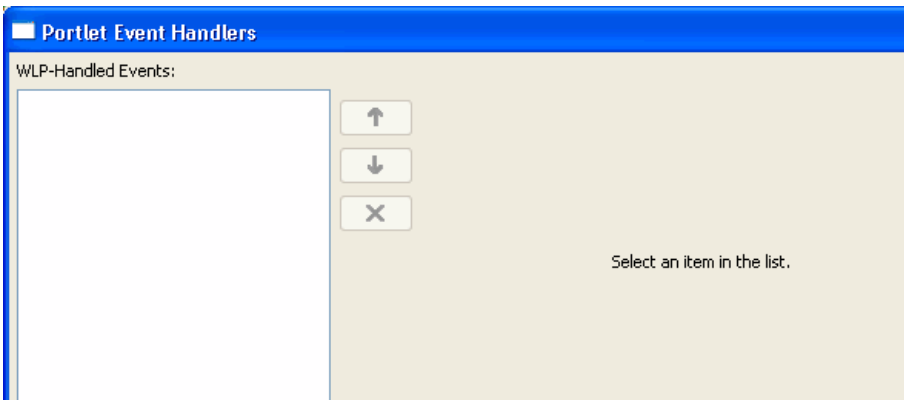
The following steps illustrate how to configure a clipper portlet to return to its original state in response to an `onActivation` portal event.

1. In the portlet editor, click the Event Handlers link. If no handlers have been created previously, the link is called **No Event Handlers**, as shown in Figure 7–5.

Figure 7–5 Event Handlers Link

2. In the Portlet Event Handlers dialog click **Add Handler**, as shown in Figure 7–6.

Figure 7–6 Portlet Event Handlers Dialog Box

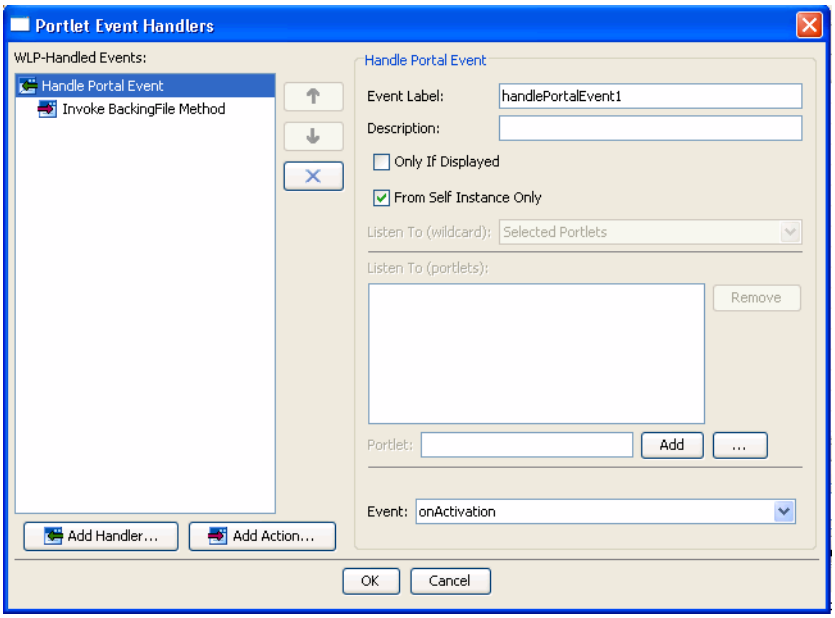


3. From the drop down list, select **Handle Portal Event**.

   The Portlet Event Handlers dialog box expands to allow entry of more details.

4. Uncheck the **Only If Displayed** checkbox.
5. Check the **From Self Instance Only** checkbox.
6. From the Event dropdown menu, select **onActivation**. See Figure 7–7.

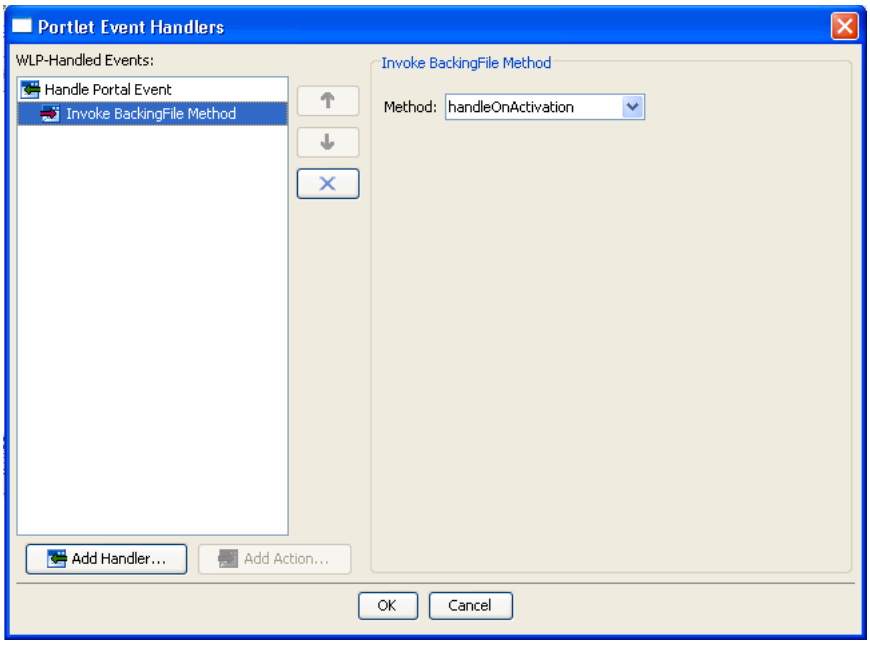
Figure 7–7 Event Handler Dialog Box Expanded



7. Click **Add Action** and select **Invoke Backing File**.
8. Set the backing file method to `handleOnActivation`, as shown in Figure 7–8.

Note: All clipper portlets reference the backing file `com.bea.netuix.clipper.ClipperBacking`. You can see this backing file referenced in the clipper portlet properties editor and in the `.portlet` file. This backing file class has the method called `handleOnActivation()`, which is called in response to the `onActivation` event. This method handles the details of restoring the clipper portlet to its original state. See also Section 7.10, "Using Backing Files with Clipper Portlets."

Figure 7–8 Specifying the Backing File Method



9. Click **OK**.

The event handler is added, and now the clipper portlet will be restored to its original state whenever an `onActivation` event is fired.

Example 7–4 shows a properly-configured sample `.portlet` file that reflects the result of the above configuration steps.

Example 7–4 Configured .portlet File Sample

```
<netuix:portlet backingFile="com.bea.netuix.clipper.ClipperBacking" definitionLabel="myClipper"
title="Myclipper">
    <netuix:handlePortalEvent event="onActivation"
         eventLabel="handlePortalEvent1"
         fromSelfInstanceOnly="true" onlyIfDisplayed="false">
         <netuix:invokeBackingFileMethod method="handleOnActivation"/>
    </netuix:handlePortalEvent>
    <netuix:titlebar>
        <netuix:minimize/>
    </netuix:titlebar>
    <netuix:content>
        <netuix:jspContent contentUri="/clipper/clipper.jsp"/>
    </netuix:content>
    <netuix:preference modifiable="false" name="remoteUrl"  value="http://www.oracle.com"/>
</netuix:portlet>
```

7.10 Using Backing Files with Clipper Portlets

The clipper portlet comes with its own backing file. For detailed information on backing files, see "Backing Files" in the *Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal*.

To add your own backing file to a clipper portlet, do the following:

- Create your own backing class that extends com.bea.netuix.clipper.ClipperBacking.

- For any of the backing file methods you override, call the super class on that method.
- Set the `backingFile` attribute in your `.portlet` file to your backing class.

7.11 Updating Portlet Preferences While the Server is Running

If you change the preferences for a clipper portlet in the `.portlet` file, the changes are not picked up at runtime unless you set the following attribute in the `WEB-INF/netuix-config.xml` file:

```
<propagate-preferences-on-deploy propagate-to-instances='true' master='file'/>
```

Preference changes that are made in the Administration Console are picked up automatically.

7.12 Clipper Portlet Limitations

The following are known limitations on the clipper portlet feature. These limitations may or may not apply to future releases.

- Authentication preferences are not encrypted.
- JavaScript in remote pages is not fully supported. Sites with that use JavaScript may work, but it is not guaranteed.
- Persistent cookies are not supported. Remote cookies only last as long as the main portal session.
- The current remote URL for a clipper portlet is stored in the session. This means that to reset your clipper window, you need to close and restart your browser.

8

Working With JSF-Java Portlets

This chapter discusses procedures and best practices for developing and configuring JSF portlets within a JSR-286 (Java 2.0) portlet using the Standard JSF Bridge (JSR-329). For using the WLP native JSF bridge, see the chapter "Working With JSF Portlets" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal* 10.3.0 available at `http://download.oracle.com/docs/cd/E13155_01/wlp/docs103/portlets/jsf.html`.

This chapter includes the following sections:

8.1 Overview

In a portal environment, an application implemented using Java Server Faces technology can be surfaced as a portlet by creating a JSF portlet. Oracle WebLogic Portal has supported the use of JSF for the implementation of portlets by developing

the WLP native bridge since a standards-based bridge did not exist. Starting in WLP 10.3.2, the support for JSF portlets continues to be enhanced with the added support for the JSR-329 standards-based JSF bridge. The new JSR-329 bridge is now the default for JSF portlets which continues to leverage all of the powerful features of WLP.

This section includes the following topics:

8.1.1 Supported Portlet Bridges

The standard JSR-286 portlet with the standard JSR-329 compliant bridge is new in WLP 10.3.2 and allows for JSF portlets to interoperate consistently across all Java portlet (JSR-286) compliant containers. This type of JSF portlet is now the default for WLP 10.3.2.

> **Note:** For detailed information about Java portlet features supported by WLP, see Chapter 6, "Building Java Portlets."

WLP continues to provide a native JSF portlet bridge implementation for use in web applications that are configured to use JSF 1.1 and/or the Facelet facet. This option creates a JSP-style portlet with a `<facesContent>` XML element, which uses the native bridge functionality. In other words, a portlet with a `<facesContent>` element uses the WLP native JSF bridge. To add the Facelet facet, see the section "Portal Web Project Wizard" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*, which includes instructions on how to add and remove facets. The Portal Web Project Wizard does not let you configure JSF 1.2 with Facelets, so you must choose JSF 1.1.

> **Note:** Before WLP 10.3.2, the native JSF portlet bridge was the only JSF portlet bridge available in WebLogic Portal. If you are developing a new project, Oracle recommends that you avoid using the native JSF bridge because it is less likely to be compatible with third-party JSF toolkits.

The Portlet Creation Wizard automatically detects the version of JSF that is used in a web application and provides the correct portlet type for you to select. When JSF 1.2 is the configured version, the JSF portlets use Java Portlet 2.0 (JSR-286) with the JSR-329 standard portlet bridge. When JSF 1.1 is the configured version, the WLP native portlet bridge is used.

Table 8–1 summarizes the types of portlets that are created depending on the version of JSF you are using. As a developer, it is important to know what kind of portlet is created. For instance, a standard Java 2.0 (JSR-286) portlet has many characteristics that do not pertain to the native Faces Content Portlet. The type of portlet created can affect development choices, like interportlet communication, event handling, and portlet preferences.

Table 8–1 Resulting Portlet Types for JSF Portlets

JSF Version	Facelet Facet Added to Project	Type of Portlet Created	Portlet Bridge Used	Comments
1.1	N/A	Faces Content Portlet	Native WLP Bridge (non-standard)	JSF 1.1 always uses the WLP native bridge
1.2	Yes	Faces Content Portlet	Native WLP Bridge (non-standard)	
1.2	No	Java Portlet Version 2.0 (JSR-286)	JSR-329 Bridge (standard)	

The version of JSF is configured in the web application as a facet. For more information on facets, see the section "Adding Facets to an Existing Web Project" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

For details on creating JSF 1.1 portlets using the native WLP bridge, see the chapter "Working With JSF Portlets" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal* available at `http://download.oracle.com/docs/cd/E13155_01/wlp/docs103/portlets/jsf.html`. The remaining of this chapter describes configuration and architecture of Java 2.0 portlets using JSF 1.2 and the JSR-329 standard bridge.

8.2 Creating Java 2.0-JSF 1.2 Portlets

For detailed information on creating a Java-JSF standard portlet using the Oracle Enterprise Pack for Eclipse, see Section 5.4.2, "Building JSF Portlets." At this point you should have your first Java-JSF portlet created. The artifacts are as follows:

- The portlet. When opened with an XML editor you should confirm that it is a Java portlet. This is evident by the following XML snippet:

  ```
  <netuix:javaPortlet  definitionLabel="firstPortlet" portletName="firstPortlet"
  title="First Portlet"/>
  ```

- `portlet.xml` in `WEB-INF`

  This file should contain a new entry for the newly created portlet. In addition, it should have an `<init-param>` that has a value for `javax.portlet.faces.defaultViewId.view`. The value should be the JSP or Faces page that this portlet should render for the view. Note that the portlet class defaults to `javax.portlet.faces.GenericFacesPortlet`. You may extend this class to override default behavior for the Java portlet and change the entry in `portlet.xml` accordingly. Many of the examples in this section assumes you will be doing this.

- The JSP or Faces file to which the `view` points.

  Faces Application URL that was selected in the Portlet Creation Wizard. This is the beginning page to the portlet. From here the portlet is built using components similar to building a normal Faces application. The difference is that your JSP should not contain `<html>`,`<head>`,`<body>`,`<title>` tags since these are only used when you own the entire web page, and inside a portlet, you do not. (The IDE may put these there for you, take them out!) The next sections cover in more details the points you should be aware of about a Faces application within a portal container. Your JSP or Faces file may contain plain HTML tags as well as Faces

tags. If it contains no Faces tags, your portlet runs as a Java portlet without the use of the Faces bridge.

Example 8–1 Sample JSF-JSP

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f"  uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h"  uri="http://java.sun.com/jsf/html"%>
<f:view>
    <h:form id="helloForm" >
       <h:outputText value="Hello World" />

    </h:form>
</f:view>
```

8.3 JSR-286 and JSR-329 Architecture

The portlet bridge acts as the translation engine between the portlet environment and the Faces environment. A portlet is an application that provides a specific piece of content (for example, HTML, XHTML, WML) to be included as part of a larger picture called a portal page. A Faces portlet uses the Faces design paradigm within a Java portlet paradigm. The bridge allows for the blending of the Java portlet lifecycle with that of Faces lifecycle.

Reading the JSR-329 spec combined with the JSR-286 spec will greatly improve your understanding about the JSR-286 and JSR-329 architecture. The following are the salient features of the architecture:

The `javax.portlet.faces.GenericFacesPortlet` in JSR-329 extends `javax.portlet.GenericPortlet` from the JSR-286 specification. A portlet developer may often want to extend `GenericFacesPortlet` to handle special `init` or destroy processing or override any of the interface methods to implement custom behavior before calling into the Faces bridge. The `GenericFacesPortlet` overrides the `init`, `destroy`, `doDispatch`, `doEdit`, `doHelp`, `doView`, `processAction` and `processEvent` methods of `GenericPortlet` and defines several new methods to handle JSF processing.

Chapter 5, "Bridge Request Lifecycle Requirements" of the JSR-329 specification at `http://www.jcp.org/en/jsr/detail?id=329` explains the portlet lifecycle combined with the Faces lifecycle in more detail. The high-level summary is that the portlet enters its lifecycle when the client (typically a browser) generates a request (`HttpRequest`) to display or interact with the web page which contains the portlet. The portlet is then asked to render, process an action, process a given event, or serve a resource based on that client request. Note that when processing an action or an event, these may trigger more events before a render is completed. In a Faces portlet, these portlet lifecycles must be coordinated to the Faces system of invoking the managed bean, processing validations, handling error messages, rendering its components, and processing the navigation rules. To handle the coordination of these two different lifecycles is the purpose of the JSR-329 bridge.

Each JSF-Java portlet on the page gets its own instance of the bridge. By default, each JSF-Java portlet created through the Eclipse IDE uses the base class of the `GenericFacesPortlet`. Keep in mind that the portlet container and the Faces engine are loosely coupled through the portlet bridge. That is, they are not directly tied to each other. They can communicate indirectly via the underlying request (`HttpRequest`) from the client (browser).

The biggest challenge and source of confusion for a Java-JSF portlet developer is the fact that Faces has a single request model that is split between action and rendering. The Java portlet model has a multi-request lifecycle split between action and/or event and rendering. In addition, in the Java portlet 2.0 model, the action process can spawn one or more events before issuing a render request. The JSR-329 bridge manages this complexity, but it is also important for the developer to understand the phases to help aid in writing the appropriate code in the appropriate places for best results.

8.4 Understanding WLP and JSF Rendering Life Cycles

This section explains WLP and JSF rendering life cycles:

- Section 8.4.1, "WLP and JSF Life Cycles"
- Section 8.4.2, "Invocation Order of WLP and JSF Life Cycle Methods"

8.4.1 WLP and JSF Life Cycles

Both WebLogic Portal and JSF frameworks support the concept of component trees that define the rendering of the HTML page. Both also rely on the concept of rendering life cycles. Each component tree is traversed multiple times during the execution of the request. Each traversal is called a life cycle or phase.

When developing JSF portlets, it is helpful to understand how those life cycles interact. For more information on the phases of the portal life cycle, see Chapter 4, "Understanding Portlet Development" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

8.4.2 Invocation Order of WLP and JSF Life Cycle Methods

The following represents the merged life cycle execution order across the Portal, Java Portlet, and JSF containers:

- On portal app deploy
  - PortalContainer:init
  - GenericFacesPortlet.init()
- On portal app undeploy
  - PortalContainer:destroy
  - GenericFacesPortlet.destroy()
- PortalContainer:preRender
- PortalContainer:render
  - GenericFacesPortlet.doView and/or GenericFacesPortlet.serveResource
  - JSFContainer:RestoreView (only on first render of portlet instance)
  - JSFContainer:RenderResponse
- Portlet link or form submit processing a Portlet.actionUrl
  - GenericFacesPortlet.processAction

  Then, if a JSF file is the view:
  - JSFContainer:RestoreView
  - JSFContainer:ApplyRequestValues

  - JSFContainer:ProcessValidations
  - JSFContainer:UpdateModelValues
  - JSFContainer:InvokeApplication
    * JSFContainer:invoke Action Listeners
    * JSFContainer:invoke Action method
- PortalContainer:raiseEvents / JSR-286 Events
  - GenericFacesPortlet.processEvent() or annotated event method
  - JSFContainer:RestoreView

Note: Unlike for a native JSF portlet, a backing file for Java 2.0 portlet is typically unnecessary since the portlet is already based on a Java portlet lifecycle paradigm of `init`, `processAction`, `processEvent`, `render`, `serveResource`, `destroy`. These lifecycles allow you to implement your backing code into the appropriate Java method in your Java portlet. However, it is possible to add a backing file to the portlet to handle certain limited situations, such as the need to have code process in the `preRender` phase of the portlet since the Java portlet lifecycle does not have a one to one mapping of this lifecycle phase. See Section 8.9, "Converting Native JSF Portlets to Standard Java JSF Portlets."

8.5 Accessing WLP Context Objects from JSF Managed Beans

To enable portlets to programmatically interact with the portal framework, a set of context objects is available from the JSF managed bean.

Table 8–2 shows what portlet context objects are in scope for different managed bean methods for different JSF life cycles. This chart is useful when implementing a managed bean that needs to obtain WLP context information. The key point is that property getters and setters need to be coded so that they work properly with either context object since they may be called during either an action or a render phase. Generally, the `PortletPresentationContext` is only available during render, and the `PortletBackingContext` is only available while processing an action.

Table 8–2 Scope of Portlet Context Objects

JSF Life Cycle	Managed Bean Method	Portlet Backing Context	Portlet Presentation Context	Portal Use Case
PROCESS_VALIDACTIONS	get property	Yes	No	Portlet receives a postback, input is being validated.
PROCESS_VALIDACTIONS	set property	Yes	No	Portlet receives a postback, input is being validated.
UPDATE_MODEL_VALUES	set property	Yes	No	Portlet receives a postback, input has been validated.
INVOKE_APPLICATION	action method	Yes	No	Portlet is the target of a postback.

Table 8–2 (Cont.) Scope of Portlet Context Objects

JSF Life Cycle	Managed Bean Method	Portlet Backing Context	Portlet Presentation Context	Portal Use Case
RENDER_RESPONSE	get property	No	Yes	Portlet is being rendered.
RENDER_RESPONSE	set property	No	Yes	Portlet is being rendered.

8.6 Understanding Scopes and JSF Portlets

This chapter covers scoping topics that apply to JSF portlets.

- Section 8.6.1, "Conceptual Scopes for Standard JSF Applications"
- Section 8.6.2, "Conceptual Scopes for Portal Applications"
- Section 8.6.3, "Implementation Patterns for Portal Scopes"
- Section 8.6.4, "Reinterpretation of the JSF Session and Request Scopes"
- Section 8.6.5, "Global Session and Portlet Group Session Scopes"

8.6.1 Conceptual Scopes for Standard JSF Applications

The standard JSF scopes are interpreted differently in a portal environment. This section discusses the differences and includes the following:

- Section 8.6.1.1, "JSF Standard Scopes"
- Section 8.6.1.2, "View Scope"
- Section 8.6.1.3, "Pageflow/Conversation Scope"

8.6.1.1 JSF Standard Scopes

JSF managed beans have well-defined scopes in the JSF specification. The JSF 1.2 specification provides three scopes for managed beans:

- **Application** - Bean state is accessible by all users in the web application.
- **Session** - Bean state is accessible to any view for the given user, across the life span of all requests within the session.
- **Request** - Bean state is accessible for the duration of a single request.

In addition to these, several other scopes exist. Specifically, JSF 2.0 adds a **View** scope, and many web frameworks provide a **Pageflow** scope, as described below. However, these are not supported in JSF 1.2.

8.6.1.2 View Scope

View scope is included in the JSF 2.0 specification. It enables managed beans to be attached to a specific view across multiple HTTP requests. Once a user navigates to a different view, the bean state is destroyed. This is a helpful pattern for scoping state to a single instance of a view for a user, but is not supported in JSF 1.2.

8.6.1.3 Pageflow/Conversation Scope

A Pageflow (also called a conversation) is a subset of the views and controller logic within a web application that pertains to a logical task or business process. Multiple

pageflows can exist within a web application, and each one usually carries state that should only be scoped to that pageflow. With Pageflow scope, managed bean state is accessible across the life span of all requests within the session, limited to the time in which a user is interacting with the set of views within that Pageflow. This is not supported in JSF 1.2.

8.6.2 Conceptual Scopes for Portal Applications

Because of the composite nature of a portal user interface, there are more conceptual scopes for portals than for standard JSF applications.

The list of portal scopes includes:

- **Application** - Bean state is accessible by all users in the web application.
- **Global Session** - Bean state is accessible to any portlet for the given user, across the life span of all requests within the web application.
- **Portlet Group Session** - Bean state is accessible to any view within a group of portlet instances or definitions for the given user, across the life span of all requests within the web application. This use case can be important for interportlet communication.
- **Portlet Instance Session** - Bean state is accessible to any view within a single portlet instance for the given user, across the life span of all requests within the web application.
- **Pageflow** - Bean state is accessible to any view within a Pageflow within a single portlet instance for the given user, across the life span of all requests within the Pageflow. This is not supported in JSF 1.2.
- **View** - Bean state is accessible for as long as the user is interacting with the current view across multiple requests within a single portlet instance. If the user is interacting with another portlet, the bean state is retained. This is not supported in JSF 1.2.
- **Portal Aware Request** - Bean state is accessible with the portlet instance for the duration of a single request. If the user is interacting with another portlet instance, the bean state is retained until the next request in which the user interacts with the portlet.

8.6.3 Implementation Patterns for Portal Scopes

Table 8–3 describes how the standard JSF scopes map to the WLP scopes, and how the unrepresented JSF scopes are supported. These implementation strategies are explained in the following sections.

Table 8–3 Managed Bean Scope Implementation Strategies

Portal Managed Bean Scope	Implementation Strategy for JSF Portlets
Application	faces-config.xml scope = application
Global Session	faces-config.xml scope = session, plus custom code
Portlet Group Session	faces-config.xml scope = session, plus custom code
Portlet Instance Session	faces-config.xml scope = session
Pageflow	Use an alternate navigation controller
View	New scope with JSF 2.0 (not supported in this release)

Table 8–3 (Cont.) Managed Bean Scope Implementation Strategies

Portal Managed Bean Scope	Implementation Strategy for JSF Portlets
Portal Aware Request	faces-config.xml scope = request

8.6.4 Reinterpretation of the JSF Session and Request Scopes

Table 8–4 compares JSF managed bean scoping levels between a JSF application and a WLP JSF portlet.

Table 8–4 Comparison of Scoping Levels

Faces-Config.xml Scope Label	Conceptual Scope for JSF Application	Conceptual Scope for JSF Portlet
Application	Application	Application
Session	Global Session	Portlet Instance Session
Request	HttpRequest	Portal Aware Request

Because a managed bean declared with session scope in `faces-config.xml` is interpreted as Portlet Instance Session scoped with the portlet bridge, it is possible to put multiple instances of that portlet on a page and not have conflicts. Each portlet instance that uses the managed bean is provisioned with a distinct instance of the bean.

Also, the different interpretation of request scope prevents a JSF portlet from breaking if the user interacts with a second portlet while interacting with the first JSF portlet.

8.6.5 Global Session and Portlet Group Session Scopes

The remaining scopes for managed beans cannot be expressed in `faces-config.xml` alone. However, the remaining scopes can be achieved using code patterns that involve `HttpSession`. For details, see Section 8.7, "State Sharing."

8.7 State Sharing

This section includes the following:

- Section 8.7.1, "State Sharing Concepts"
- Section 8.7.2.1, "HttpSession Versus HttpServletRequest"
- Section 8.7.2.2, "Single Portlet Pattern"
- Section 8.7.2.3, "Multiple Portlet Pattern"

8.7.1 State Sharing Concepts

JSF managed beans are intended to be the storage containers for application state within a JSF application. In general, this works well even within a portal environment. However, this standard JSF pattern is not always sufficient. There are cases where state needs to be shared with something outside of the JSF portlet. For example:

- A different portlet instance's JSF managed bean.
- A portlet instance in a remote servlet container via WSRP.
- A non-portal object, such as a servlet or servlet filter.

But, there are limitations that must be heeded when working within the JSF container:

- A JSF managed bean may not invoke any method on any portlet instance's backing file or Java portlet code, including its own.
- A JSF managed bean may not invoke any method on a JSF managed bean in another portlet instance.
- A portlet backing file or Java portlet code may not invoke any method on any portlet instance's JSF managed bean.

Despite these limitations, there are ways to handle these types of situations as explained in the remaining of this section.

8.7.2 State Sharing Patterns

This section includes the following subsections:

8.7.2.1 HttpSession Versus HttpServletRequest

This section includes the following:

8.7.2.1.1 Store state in the HttpSession In a portal environment, the lifecycle of the request is not always straightforward for the following reasons:

- WSRP can involve multiple requests - When a user interacts with a portlet which is being consumed over WSRP, the handling of that interaction and the render of the portlet can occur over two requests (`performBlockingInteraction` and `getMarkup` requests). Attributes set during the interaction are not available in the subsequent `getMarkup` request. In this case, storing state inside an `HttpServletRequest` will not behave appropriately.
- JSR-286 can have multiple request phases as well, for example, first the action request, and/or event request phase, and then a subsequent render request phase.
- Scoped requests - A portlet in which a code is executed often does not have access to the actual `HttpServletRequest`. Usually, the portlet is a scoped object. The portlet often does not have access to the actual `HttpServletRequest` during code execution. Usually the `HttpServletRequest` is a scoped object of the lifecycle phase and attributes set on a scoped request are not visible to other portlets. This makes sharing state between portlets unfeasible using the request object.

Therefore, it is often preferred to set the state in the `HttpSession` and there are JSR-286 mechanisms (Single Portlet Pattern and Multiple Portlet Pattern) that can help you. However, storing state in `HttpSession` has some drawbacks:

- Attributes set into the `HttpSession` must be Serializable so that the session can be replicated within a cluster. Not all Java objects are easily serializable, so this may be an issue. One way around this issue is to mark that Java object as transient.
- WebLogic Server distributes or stores the attributes by serializing the attributes. Adding more attributes to the `HttpSession` creates higher overhead for the replication facility. Again it could be that you won't need this attribute replicated, and therefore, could mark the Java object that is being stored as an attribute as

transient to avoid this replication overhead, since transient objects are not serialized.

- If an attribute is appropriate only for the current request, the `HttpSession` attribute must be removed by the managed bean after it is finished with it.
- When used for multiple portlet patterns, the approach is fragile. This is discussed in more detail in Section 8.7.2.3, "Multiple Portlet Pattern."

8.7.2.2 Single Portlet Pattern

This pattern is defined as sharing state amongst components of a single portlet instance with other components affiliated with that portlet instance, for example, the Java Portlet itself and corresponding JSF managed bean. In the case of single pattern, the state can--depending on the data structure of the state--be passed through as response render parameters or be stored as request attributes. The best namespace for setting attributes is using the portlet's instance label so that if the same portlet exists in more than one location in a portal, its attributes are not mixed. These portlet components have access to the portlet instance label.

The `javax.portlet.actionScopedRequestAttributes` container runtime option handles the request attributes for you, so when they are set, they can be seen in the next lifecycle phase. See Section 6.11, "Using Container Runtime Options" for more details.

8.7.2.3 Multiple Portlet Pattern

Often, it is necessary to have multiple portlets share data between them. Although it seems intuitive to store this data in the `HttpSession` object as well, it is not recommended because the order in which the portlets get and set the state cannot be predicted. Therefore, `HttpSession` object should not be used to store data when users are allowed to move portlets on a page. This limitation applies to WSRP environment as well.

Oracle recommends using JSR-286 events or JSR-286 public render parameters for this use case. The major benefits of events and public render parameters are:

- Layout independent
- Support for WSRP environments
- Designed into the JSR-286 portlet architecture. The JSR-286 specification has implemented Public Render Parameters with the intent on sharing view states across portlets. In addition, the specification introduced the JSR-286 events with payloads to enable reacting to state changes. Events add more flexibility, but can cause extra overhead.

8.8 Using JSF in Java Portlets

You are now familiar with the general architecture. This section focuses on the coding details of common situations a portlet developer needs to know when writing the portlet.

It will be helpful if you already have an understanding of the two specifications: *JSR-286 for the Java portlet architecture* and *JSR-329 for the JSF bridge architecture*.

`javax.faces.context.ExternalContext`, as described in the JSF documentation "*contains all of the per-request state information related to the processing of a single JavaServer Faces request, and the rendering of the corresponding response.*" It is here where you will be able to access the `PortletRequest` and `PortletResponse` objects.

```
FacesContext fc = FacesContext.getCurrentInstance();
ExternalContext ec = fc.getExternalContext();
PortletRequest preq   = ec.getRequest();
PortletResponse presp = ec.getResponse();
```

In the WLP native JSF bridge, the architecture environment was different, thus, the `ExternalContext` returned an `HttpServletRequest` and `HttpServletResponse` instead. This is a fundamental concept to understand when converting your portal application from using native JSF portlets to JSF-Java portlets or just writing JSF-Java portlets in general. With Java portlets, there is another layer between the actual `HttpServletRequest/HttpServletResponse` and the Faces environment. This Java portlet layer provides for a standards-based way to developing your portlet.

The following sections explain in more detail common situations using the PortletRequest/Response and HttpServletRequest/Response objects:

- Section 8.8.1, "Servlet Request And Servlet Response"
- Section 8.8.2, "PortletPreferences"
- Section 8.8.3, "PortletPresentationContext"
- Section 8.8.4, "Using JSPs in JSF Portlets"

8.8.1 Servlet Request And Servlet Response

Generally the PortletRequest and PortletResponse objects give you the necessary access to the client request and response to do most of your development work. However, there are times when you need access to one layer deeper and need the actual `ServletRequest` and `ServletResponse`.

For example, when the `ServletAuthentication` class on Oracle WebLogic Server is used to acquire an authenticated user session, it requires the `ServletRequest` and `ServletResponse` objects. In the native WLP bridge, the following code returned an instance of these objects:

```
FacesContext fc = FacesContext.getCurrentInstance();
HttpServletRequest req =
(HttpServletRequest)fc.getExternalContext().getRequest();
HttpServletResponse resp =
 (HttpServletResponse)fc.getExternalContext().getResponse();
```

This above code gets a `ClassCastException` when using Java portlets, because now the `getExternalContext().getRequest()` call returns a `PortletRequest` and `getExternalContext().getResponse()` returns a `PortletResponse`.

In the Java portlet code, it is best to get a handle to the `ServletRequest` and `ServletResponse` objects via calling the `PortletRequestDispatcher` from the `PortletContext` object that is returned from the `facesContext.getExternalContext()`, like shown below:

```
FacesContext fc = FacesContext.getCurrentInstance();
Object obj = fc.getExternalContext().getContext();
if (obj instanceof PortletContext){
   PortletContext pc = (PortletContext) obj;
   PortletRequestDispatcher prdispatcher =
        pc.getNamedDispatcher("myServletName");
```

`PortletRequestDispatcher` enables you to dispatch control to a resource such as a servlet or JSP and provides the servlet or JSP a handle to the `ServletRequest` and

ServletResponse. (In this example a named dispatcher was used, but you can also get an unnamed dispatcher and use a servlet path instead.)

It is also possible to use a non-standard way of accessing the attributes to gain access to the HttpServletRequest and HttpServletResponse like this:

```
FacesContext fc = FacesContext.getCurrentInstance();
ExternalContext ec = fc.getExternalContext();
PortletRequest portletrequest = ec.getRequest();

portletrequest.getAttribute("javax.servlet.request")
portletrequest.getAttribute("javax.servlet.response")
```

Although this is possible, it is not recommended because it does not take into consideration the portlet-level scoping of attributes and it is non-standard so it may not work across all portlet containers the same way. However, there are times that dispatching control to a servlet or JSP is overkill and this may be more convenient. But realize that dispatching control to a servlet allows for a more flexible architecture and keeps specialized code inside a servlet and outside of the portlet environment.

See JSFJavaPortletHelper Class in Appendix: JSFJavaPortletHelper for further implementation details.

8.8.2 PortletPreferences

In the WLP native JSF bridge, preferences are handled through the PortletBackingContext, which could be referenced from a backing file. With Java portlets there is a standard API call for getting the PortletPreferences object.

The following code is an example of how to access the PortletPreferences object from within a JSF managed bean method. It acquires the PortletRequest object through which the standard API call of getPreferences() and returns the PortletPreferences object.

```
FacesContext fc = FacesContext.getCurrentInstance();
PortletRequest pr = (PortletRequest);
fc.getExternalContext().getRequest();
PortletPreferences prefs = pr.getPreferences();
```

To learn more about PortletPreferences, see *Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal* at http://download.oracle.com/docs/cd/E15919_01/wlp.1032/e14244/configure.htm#i10731138.

8.8.3 PortletPresentationContext

The PortletPresentationContext is a WLP class that represents the portal desktop environment. In the WLP native JSF bridge, the JSP file in use typically gets the PortletPresentationContext during the render phase. You can continue to access the PortletPresentationContext through a JSP by dispatching control from your Java portlet to the JSP as described in Section 8.8.1, "Servlet Request And Servlet Response." Doing it this way keeps WLP specific code more isolated.

If you have a need to access the PortletPresentationContext from a JSF managed bean method, you must provide the actual ServletRequest object to the PortletPresentationContext factory object. This technique is shown in the JSFJavaPortletHelper Class example.

8.8.4 Using JSPs in JSF Portlets

In a Java portlet, you can use simple JSP pages that do not contain Faces component tags. These pages behave like a basic Java portlet and are processed without going through the Faces bridge. Using non-Faces component JSP pages may be preferred in some cases such as handing file upload, since the JSF 1.1 and JSF 1.2 implementations did not include a JSF component for this common web use case. See Section 8.14, "Code Examples for Common Use Cases" for an example.

8.9 Converting Native JSF Portlets to Standard Java JSF Portlets

For the most part, the JSF views that worked in the WLP native JSF bridge should continue to work in a Java portlet using the JSR-329 standards bridge. However, there are some differences that you should take into account when converting from the Native JSF Portlets in the Standard Java JSF Portlets.

This section includes the following:

- Section 8.9.1, "Backing Files"
- Section 8.9.2, "NamingContainer"
- Section 8.9.3, "Events"
- Section 8.9.4, "Preferences"
- Section 8.9.5, "Localization"
- Section 8.9.6, "Error Pages"
- Section 8.9.7, "Portlet Modes"
- Section 8.9.8, "ServletRequest/ServletResponse"

8.9.1 Backing Files

If you used backing files in your native JSF portlets, then Oracle recommends moving the source code that was in there to the appropriate Java portlet methods. Although, there is not a complete one-to-one correspondence with the backing file lifecycle and the portlet lifecycle, it is possible, with the use of events, to create similar behavior. This will help reduce complexity of your portlet code.

To do this, you will want to subclass the `GenericFacesPortlet` class.

Example 8–2 GenericFacesPortlet

```
public class MyCustomPortlet extends GenericFacesPortlet {
      @Override
      public void processAction(ActionRequest actionRequest, ActionResponse
      actionResponse)
      {
      //code from the backing file's handlePostBackData() method goes here
      super.processAction(actionRequest, actionResponse);
       }
      @Override
      public void init()
      {
      /*code to execute when the portlet is first initialized. Note this has
      different timing than a backing file init() method.
     */
          super.init();
      }
```

```
      @Override
public void destroy()
{
/*code to execute when the portlet is destroyed. Note this has different timing
than a backing file destroy() method.
*/
          super.destroy();
      }
```

Example 8–3 Changing entry in portlet.xml

```
<portlet>
      <portlet-name>MySpecialPortlet</portlet-name>
      <portlet-class>oracle.samples.wlp.jsf.portlets.MyCustomPortlet
      </portlet-class>
      <init-param>
         <name>javax.portlet.faces.defaultViewId.view</name>
         <value>/demo/display.jsp</value>
      </init-param>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>view</portlet-mode>
      </supports>
</portlet>
```

Please note the following differences in the lifecycle between the backing file methods and the Java portlet methods:

- `Backingfile.handlePostBackData()` - this is always called, even if the portlet is not the target.
- `MyCustomPortlet.processAction()` - this is only called if the portlet is the target.

If code existed in your backing file's `handlePostBackData()` method that always needs to be executed, then in a Java portlet paradigm, the portlet should subscribe to the appropriate event for your situation.

For example, the portal `OnRefresh` event is sent to all portlets that are about to be rendered as well as after a `processAction` has occurred. So, listening for this event in your subclassed Java portlet would be recommended in order to handle setting visibility if that setting of visibility was something that was in the `handlePostbackData()` method of the backing file. Review the portal events in the section "Event Types" of *Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal* for more information on the types of events available to you.

8.9.2 NamingContainer

For the JSF-JSP files used in the JSF Native Portlets, remove all references to the WLP's `<jsf-naming:namingContainer>`. This is because JSR-329 has its own way of handling namespacing; using WLP's namespacing causes errors in a Java based JSF Portlet.

8.9.3 Events

Convert the WLP proprietary events to the standards-based Java Portlet events model that is new in JSR-286. See Section 12.4, "Portlet Event Handling" for further details.

See Section 12.3, "Differences Between Portal Events and Java Portlet Events" for more details on how to access portal events within your Java Portlet.

8.9.4 Preferences

If using preferences, change how the `PortletPreference` object is obtained by using the Java Portlet API supplied in the `PortletRequest` object.

8.9.5 Localization

If using localization, set up the Java Portlet localization for the portlet title.

8.9.6 Error Pages

The native JSF portlet includes an attribute for the error path, which redirects your portlet when an error occurs. See Section 8.10.1.2, "Portlet Error Page" for how to handle errors with Java Portlets.

8.9.7 Portlet Modes

In the JSF native portlets, the help/edit mode `.jsp` files could not contain any JSF tags, this is no longer the case with Java-JSF portlets. JSF tags may exist in the help or edit or custom modes.

8.9.8 ServletRequest/ServletResponse

If you are using `ServletRequest` and `ServletResponse` in your native JSF portlets, you may want to move this code to its own servlet so that it can get a handle to the `servletRequest` and `servletResponse` objects. Or, create a helper class to consolidate the coding effort to get the `servletRequest` and `servletResponse` from the `PortletRequest` object. See Section 8.8, "Using JSF in Java Portlets" for further details.

8.10 Using Common WLP Features With JSF Portlets

This section describes how commonly used WebLogic Portal features are used in an environment with JSF portlets.

This section includes the following:

- Section 8.10.1, "Portlet Container Features"
- Section 8.10.2, "Portal Container Features and JSF Portlets"
- Section 8.10.3, "Ajax Enablement"

8.10.1 Portlet Container Features

This section discusses the following portlet container features:

- Section 8.10.1.1, "Portlet Modes"
- Section 8.10.1.2, "Portlet Error Page"
- Section 8.10.1.3, "Portlet Preferences"
- Section 8.10.1.4, "Portlet Dependencies"

8.10.1.1 Portlet Modes

For more information about portlet modes, see Section 9.5.2, "Portlet Modes." Java-JSF Portlets support portlet modes and custom modes just as a Java portlet does. The `portlet.xml` file identifies which modes it supports and what the `defaultViewId` should be.

8.10.1.2 Portlet Error Page

A common WLP feature for portlets is the ability to set an error path in the event of an error. This works differently for standard Java portlets. The Java portlet specification requires that the portlet throws a `PortletException`.

Since Java portlet is the underlying technology for the Java-JSF portlet, a `PortletException` is thrown in the event a portlet cannot process an operation successfully. Therefore, the error page processing is handled in your subclass of `GenericFacesPortlet` by catching the error in a try/catch block and sending the appropriate error page instead. A more generic approach is to write a portlet filter that watches all responses and handles `PortletException` in a consistent manner. See Section 6.6, "Portlet Filters" for more details.

In the JSF native portlets, the error page could not contain any JSF tags, this is no longer the case with JSR-286 Faces portlets. JSF tags may exist in the error `.jsp` file.

8.10.1.3 Portlet Preferences

Portlet preferences provide the primary means of associating application data with portlets. Portlet preferences are accessible through the WLP portlet context objects. For detailed information on portlet preferences in JSR-286 portlets, see Section 9.2.2.3, "Getting and Setting Preferences for Java Portlets Using the Preferences API."

In the context of a JSF portlet, note that after setting preferences values for the portlet, `store()` must be called. This call is best accomplished by setting preferences in JSF managed bean property setters, and then calling `store()` in an action method.

To illustrate this technique, consider a JSF portlet that lets a user get a stock quote. The last quote the user obtains is persisted using portlet preferences. The JSF view is shown in Example 8–4. This view retrieves portlet preference values from a JSF managed bean. The managed bean, shown in Example 8–5 sets and gets the preference values from WLP.

Example 8–4 A JSF View

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri='http://bea.com/faces/adapter/tags-naming' prefix='jsf-naming' %>
<f:view>
<h:panelGrid columns="4" width="100%">
<h:form id="stockForm">
    <h:panelGroup>
        <h:outputText value="Stock Quote:"/>
    </h:panelGroup>
    <h:panelGroup>
        <h:inputText id="ticker" value="#{WLPPrefsRequestBean.ticker}"
         required="true"/>
    </h:panelGroup>
    <h:panelGroup>
        <h:inputText id="shares" value="#{WLPPrefsRequestBean.shares}"
         required="true"/>
    </h:panelGroup>
```

```
    <h:panelGroup>
        <h:outputText value="#{WLPPrefsRequestBean.currentValue} "/>
    </h:panelGroup>
    <h:panelGroup></h:panelGroup>
    <h:panelGroup></h:panelGroup>
    <h:panelGroup>
        <h:commandButton action="#{WLPPrefsRequestBean.getQuote}"
         id="quote" value="Get Quote"/>
    </h:panelGroup>
    <h:panelGroup>
        <h:commandButton action="#{WLPPrefsRequestBean.resetQuote}"
         id="reset" value="Reset"/>
    </h:panelGroup>
</h:form>
</h:panelGrid>
</f:view>
```

The managed bean is listed in Example 8–5.

> **Note:** Example 8–5 uses a utility class described in Section 8.18, "Appendix: JSFJavaPortletHelper."

Example 8–5 The JSF Managed Bean

```
package oracle.samples.wlp.jsf.beans;
import java.io.Serializable;
import javax.portlet.PortletPreferences;
import oracle.samples.wlp.jsf.JSFJavaPortletHelper;
/**
 * An example that shows how to use WLP preferences with a JSF managed bean.
 * This example makes the following assumptions for the preference writes to work
properly:
 *  1. The bean is request scoped
 *  2. The user is authenticated
 *  3. The portal is a streaming desktop, not a .portal
 */
public class WLPPrefsRequestBean implements Serializable {
    private static final long serialVersionUID = 1L;
    private static final String TICKER = "ticker";
    private static final String TICKER_DEF = "ORCL";
    private static final String SHARES = "shares";
    private static final String SHARES_DEF = "1000";

    /**
     * The request scoped preferences object.
     */
    private PortletPreferences prefs;

    /**
     * Constructor. Initializes the preference object.
     */
    public WLPPrefsRequestBean () {
        prefs = JSFJavaPortletHelper.getPreferencesObject();
    }

    // ACTION METHODS

    /**
     * Updates the preference values set by the user.
```

```
     * @return always null, to retain the same JSF view
     */
    public String getQuote() {
        // all the setting work is done in the setters
        // what is left is to store the new prefs into the database
        JSFJavaPortletHelper.storePreferences(prefs);
        return null;
    }

    /**
     * Resets the preferences back to their defaults.
     * @return always null, to retain the same JSF view
     */
    public String resetQuote() {
        JSFJavaPortletHelper.setPreference(prefs, TICKER, TICKER_DEF);
        JSFJavaPortletHelper.setPreference(prefs, SHARES, SHARES_DEF);
        JSFJavaPortletHelper.storePreferences(prefs);
        return null;
    }

    // GETTERS AND SETTERS

    public String getShares() {
        return JSFJavaPortletHelper.getPreference(prefs, SHARES, SHARES_DEF);
    }

    public void setShares(String shares) {
        JSFJavaPortletHelper.setPreference(prefs, SHARES, shares);
    }

    public String getTicker() {
        return JSFJavaPortletHelper.getPreference(prefs, TICKER, TICKER_DEF);
    }

    public void setTicker(String ticker) {
        JSFJavaPortletHelper.setPreference(prefs, TICKER, ticker);
    }

    public int getCurrentValue() {
        // convert the String preference into an integer
        String sharesStr = getShares();
        int shares = 0;
        try { shares = Integer.parseInt(sharesStr); }
        catch (Exception e) {}
        // compute some bogus value
        return shares * 52;
    }
}
```

8.10.1.4 Portlet Dependencies

JavaScript use is widespread. While it is possible to add Javascript to any portlet, JSF or other types, there are important considerations to think about when introducing Javascript inside your portlet.

The primary concern is the mechanism by which a developer can introduce custom JavaScript into a portlet. While it is certainly possible to inject arbitrary JavaScript using script tags in the `<body>` or `<view>` of the page, this causes several problems:

- It may cause multiple copies of the same JavaScript file to be downloaded.

- The files are inserted under the BODY element, not the HEAD, which technically is invalid HTML.
- The JavaScript is not namespaced to an instance, causing potential for collisions of functions and variables between portlets.

WLP provides elegant solutions to these issues. For more information, see Section 9.5.1, "Portlet Dependencies."

While it is important to read that documentation, here is a quick summary of the features:

Portlets can identify which Javascript files (`.js` files) they depend upon. This dependency is configured by creating a Render Dependencies Configuration file for the portlet. Inside the Javascript files, the Javascript variables and functions can be prefixed with a token "`wlp_rewrite_`" automatically.

An example of using WLP's Portlet Dependencies feature to inject Javascript inside a portlet in order to perform DOM manipulation can be found in the section "Section 8.10.3, "Ajax Enablement." Another common reason for injecting Javascript inside your portlet is for client-side form validation.

8.10.2 Portal Container Features and JSF Portlets

This section discusses the following portal container features:

- Section 8.10.2.1, "Locale Provider"
- Section 8.10.2.2, "Skeleton Files"

8.10.2.1 Locale Provider

WLP provides the ability to implement a `LocaleProvider` to determine the most appropriate locale for the portal. For more details, see "Section 8.15.5, "Localizing JSF Portlets" in Section 8.15, "Preparing JSF Portlets for Production."

8.10.2.2 Skeleton Files

Skeleton files control the rendering of each component of the WLP page. They are implemented as JSPs. The use of JSF components in these JSPs is not supported. The WLP framework renders the skeletons using standard JSP technology. For more information about skeletons, see the chapter "User Interface Development" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal* at `http://download.oracle.com/docs/cd/E15919_01/wlp.1032/e14243/develop_ui_lookfeel.htm`.

8.10.3 Ajax Enablement

Many web sites are Ajax enabled to provide a rich user experience. Ajax allows the browser to emulate the interactivity of traditional desktop applications. There are three primary patterns for using Ajax in a portal. This section describes each of those patterns, and explains how to implement the pattern with JSF portlets in WebLogic Portal.

- Section 8.10.3.1, "Partial Page Rendering Pattern"
- Section 8.10.3.2, "Stateless API Request Pattern"
- Section 8.10.3.3, "Portlet Aware API Request Pattern"

8.10.3.1 Partial Page Rendering Pattern

The Partial Page Rendering (PPR) is the case in which an `XmlHttpRequest` is issued to the server, and the server responds with visual markup (XHTML or HTML). This markup is written directly into the browser page, updating the content that the user sees. There are several challenges related to this capability:

- The Ajax request must enter through a proper Portal entry point, so that state such as portlet preferences and previous view state can be honored.
- It requires the server to be able to return a subset of the markup for the visible page.
- Some portal-like interactions, like interportlet communication, require the server to respond with more markup than the client expected (e.g. markup for multiple portlets).

Because of these complex issues, WLP supports official techniques for implementing PPR with WLP portlets - Asynchronous Desktop and Asynchronous Portlet modes. These facilities are available to all portlet types, and are transparent to the portlet developer. Thus, any Java-JSF portlet can utilize these facilities without changes to the portlet implementation. These modes are configured using simple configuration settings.

Asynchronous Desktop Mode Asynchronous Desktop mode causes all portlets on the portal to become asynchronous. The WLP framework rewrites all URLs on the portal pages such that they invoke Ajax requests (`XMLHttpRequests`). This facility supports all WLP framework features, including interportlet communication. This setting is either on or off for the entire desktop.

For more information, see the section "Asynchronous Desktop Rendering" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Asynchronous Portlet Mode - Ajax or IFrame Asynchronous Portlet mode is similar to Asychronous Desktop mode except that it is configured at the portlet level. This allows more fine-grained configuration of what is Ajax-enabled by the framework. This mode can also be configured to use IFrames instead of Ajax. The primary limitation with this mode is that interportlet communication is not supported.

For more information, see the section "Asynchronous Portlet Content Rendering" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

8.10.3.2 Stateless API Request Pattern

Some use cases work better when the server does not serve up presentation markup, but raw data. In these cases, the client has the necessary code to process that data, and render the presentation markup appropriately. The first type of this pattern is that of a stateless API. Such an API is not specific to the portal or even portlet; it is a general purpose API. In these cases, the `XmlHttpRequest` must pass all of the necessary information for the server to process the request. The classic example of this type of API is a search box autocomplete API. In this use case, after a user types a character in a search box, and `XmlHttpRequest` is sent to the server with the contents of the search box. The server responds with a list of possible matches, and JavaScript on the client renders them in a drop down list. Such an API need not be aware of any portal context to fulfill the request. Therefore, any WebLogic Portal portlet (JSF or otherwise) may use such an API without restriction. The Ajax invocation in this case is fulfilled outside the context of the portal environment, and thus there are not any unique issues related to WLP. There are many examples available on the web. Searches for "javascript autocomplete example" will yield many results.

8.10.3.3 Portlet Aware API Request Pattern

The third pattern is for Ajax requests that target data service APIs that have access to the portlet context. The service may need access to the portlet instance's preferences, for example. For these cases, WebLogic Portal's framework must be involved in the request processing. In support of this use case, WebLogic Portal must provide the following:

- Instancing support - The ability to differentiate which portlet instance on the client issued the request.
- Context support - The ability to provide portlet instance context, such as its instance label or preferences.
- IPC awareness - If the Ajax invocation changes the state of the portlet, other portlets might also need to be updated.

WebLogic Portal implements a wrapper for the standard `XmlHttpRequest` to provide these features. The following example demonstrates its use.

Portlet Aware Data API Example

This example shows how to invoke an API via Ajax that operates within the JSF portlet's context. Meaning, the API implementation has access to portlet instance context such as preferences. The API can return the data in any textual format it chooses, with JSON, XML or simple text being common choices.

The example is shown below. It consists of a simple portlet with just a link. Clicking on the link on the portlet will cause an Ajax request to target a data API implemented in the JSF portlet.

Figure 8–1 A view of the example portlet

1. Create the JavaScript function that issues the Ajax request using the WLP `XmlHttpRequest` wrapper. Allow the caller to pass the URL in to the function. A more generic DOM manipulation may be necessary based on URL passed in, but this example is made for ease in understanding. For more information on the use of `wlp_rewrite_ token` used, see the section "Portlet Dependencies" in *Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal*.

Example 8–6 The JavaScript Function That Invokes the WLP-Specific XMLHttpRequest

```
// ajaxPortlet.js
function wlp_rewrite_issueAjax(dataUrl) {
      var xmlhttp = new bea.wlp.disc.io.XMLHttpRequest();
      xmlhttp.onreadystatechange = function() {
         if ((xmlhttp.readyState == 4) && (xmlhttp.status == 200)) {
             var data = xmlhttp.responseText;
            // MODIFY: do something with the data here
             document.getElementById
               ("wlp_rewrite_xprDiv").innerHTML = data;
         }
      };
      xmlhttp.open('GET', dataUrl, true);
      xmlhttp.send();
}
```

2. Create the JSF JSP. The URL that is invoked in the Ajax request is retrieved from a JSF backing bean via the EL expression `#{JavaScriptPortletSessionBean.ajaxDataURL}`. The backing bean implementation is discussed later. The portlet instance label is prepended to the function due to the use of the `wlp_rewrite_ token` in the JavaScript file above.

Example 8–7 The JSF JSP That Triggers the Ajax Call Using an 'onclick' Handler

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:namespace var="portletns"/>

<f:view>
   <h:form id="ajaxDemoForm">
     <h:outputLink value="#"
       onclick="#{JavaScriptPortletSessionBean.portletInstanceLabel}_issueAjax
        ('#{JavaScriptPortletSessionBean.ajaxDataURL}'); return false;">

       <h:outputText value="Invoke Portlet Data API via Ajax"/>
     </h:outputLink>
     <f:verbatim> <br></br>
     </f:verbatim>
     <div id="<%=portletns%>_xprDiv" style="background-color: #ffcccc;">replace me
     </div>

   </h:form>
</f:view>
```

3. Implement a JSF managed bean that provides the core logic for implementing the Ajax solution. The `getPortletInstanceLabel()` method can remain unchanged, but the `getAjaxDataURL()` method need to be modified to suit your needs. Note the resource Id set here. Remember to add this managed bean to the `faces-config.xml` file.

Example 8–8 The JSF managed bean that provides the Ajax data API

```
package oracle.samples.wlp.jsf.beans.ajaxdemo;
import javax.faces.context.FacesContext;
import javax.portlet.RenderResponse;
import javax.portlet.ResourceURL;
import oracle.samples.wlp.jsf.JSFJavaPortletHelper;

/*
 * Demo faces managed bean
 */
public class JavaScriptPortletSessionBean {
    ResourceURL ajaxUrl = null;
    // PORTLET INSTANCE LABEL
    public String getPortletInstanceLabel() {
      return JSFJavaPortletHelper.getInstanceLabel();
    }
    // AJAX DATA API URL
    public String getAjaxDataURL() {
      if (ajaxUrl == null){
         RenderResponse rresponse = null;
         FacesContext fc = FacesContext.getCurrentInstance();
         Object obj = fc.getExternalContext().getResponse();
         if (obj instanceof RenderResponse){
```

```
                rresponse = (RenderResponse) obj;
                ajaxUrl = rresponse.createResourceURL();
                ajaxUrl.setResourceID("ajax-getData");
            }
        }
        if(ajaxUrl != null) {
          return ajaxUrl.toString();
        }
        else {
             //throw error here
             return "error";
        }
    }
}
```

4. Create a Java class that extends the `GenericFacesPortlet` class.

Example 8–9 The Java Portlet with serveResource Implemented

```
package oracle.samples.wlp.jsf.portlets.ajaxdemo;
import java.io.IOException;
import javax.portlet.PortletException;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.ResourceRequest;
import javax.portlet.ResourceResponse;
import javax.portlet.faces.GenericFacesPortlet;
public class AjaxEnabledPortlet extends GenericFacesPortlet {
    public void serveResource(ResourceRequest request, ResourceResponse response)
      throws PortletException, IOException{
        String resourceid = request.getResourceID();
        if ("ajax-getData".equals(resourceid)){
        PortletRequestDispatcher prd = null;
        //demo servlet providing the data service,
        request.setAttribute("dataRqst", "getDemoData");
        prd = getPortletContext().getNamedDispatcher("MyDataService");
        prd.include(request,response);
     }
     else{
          super.serveResource(request, response);
     }
  }
}
```

5. Create the portlet dependencies file (`.dependencies`) to link in the `ajaxPortlet.js` JavaScript file from above. Note the use of `content-uri=` in order for `wlp_rewrite_ tokens` to be rewritten. For more information on creating dependencies files, see the section "Portlet Dependencies" in *Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal*.

Example 8–10 Dependencies Sample

```
<?xml version="1.0" encoding="UTF-8"?>
<p:window
xmlns:p="http://www.bea.com/servers/portal/framework/laf/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/portal/framework/laf/1.0.0
laf-window-1_0_0.xsd ">
   <p:render-dependencies>
     <p:html>
       <p:scripts>
```

```
        <p:search-path>
          <p:path-element>.</p:path-element>
        </p:search-path>
        <p:script type="text/javascript" content-uri="ajaxPortlet.js" />
      </p:scripts>
    </p:html>
  </p:render-dependencies>
</p:window>
```

6. Create the portlet (`.portlet` file) using the Portlet Wizard (See Section 5.4.2, "Building JSF Portlets"). Use the JSF JSP file created in Step 2 as the "Faces Application URL" in the "Portlet Details" dialog box and enter the path to the dependencies file created in Step 5 in the "Render Dependencies Path" field on "Assign Supporting Files" dialog box while using the Portlet Wizard. (You can also attach or change the dependencies file used by the portlet using the IDE after the portlet has been created.)

   Once the `.portlet` file is created, open the `WEB-INF/portlet.xml` file to change the name of the `<portlet-class>` for the portlet just created, to the class name created in Step 4. In this example that is `AjaxEnabledPortlet`. This is necessary since the `GenericFacesPortlet` was subclassed in this example to add custom resource serving.

   Was:

   ```
   <portlet>
   .
   .
   .
       <portlet-class>
        javax.portlet.faces.GenericFacesPortlet
       </portlet-class>
   .
   .
   .
   </portlet>
   ```

   Change to:

   ```
   <portlet>
   .
   .
   .
       <portlet-class>
        oracle.samples.wlp.jsf.portlets.ajaxdemo.AjaxEnabledPortlet
       </portlet-class>
   .
   .
   .
   </portlet>
   ```

7. Add the portlet to a portal (`.portal` file). This portal must have Disc enabled. You can enable Disc in the portal property sheet in the IDE for `.portal` files, or in the Portal Administration Tool for streaming desktops. For information about how to enable Disc, see `http://download.oracle.com/docs/cd/E15919_01/wlp.1032/e14229/disc.htm`.

8. Create the servlet (called `MyDataService` in Example 8–11) that the portlet dispatches to in its `serveResource()` method.

Example 8–11 A Fake DataService Servlet Returning Data for the Request

```
package oracle.samples.wlp.jsf.portlets.ajaxdemo;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MyDataService extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            doDataService(request,response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            doDataService(request,response);
    }

    private String doDataService(HttpServletRequest givenRequest,
HttpServletResponse givenResponse){
    // AJAX DATA API SERVICE IMPLEMENTATION
    // Could return any text format, like JSON, XML, simple text
    // Data is hardcoded in this example, but could easily be
    // accessing a database and be dynamic.
       try {
             Object attrObj =  givenRequest.getAttribute("dataRqst");
             if (attrObj != null) {
                String attr = (String) attrObj;
                if (attr.equals("getDemoData")) {
                   PrintWriter pw = givenResponse.getWriter();
                   pw.println("The answer is 42.");
                   pw.flush();
                   pw.close();
             }
         }
     }
     catch (IOException ioe) {
     //MODIFY: log the io exception
     System.out.println("IOException occured." + ioe.getStackTrace());
     }
     //MODIFY: put your error message here
     return "data request could not be processed.";
    }
}
```

9. Remember to add that servlet mapping to your `web.xml` file. For security reasons, it is advisable to make such things only available through a named request dispatcher.

Example 8–12 Servlet Mapping

```
<servlet>
<!-- should only be accessible through a named request dispatcher -->
<!--  for ajax demo -->
<servlet-name>MyDataService</servlet-name>
      <servlet-class>
           oracle.samples.wlp.jsf.portlets.ajaxdemo.MyDataService
      </servlet-class>
</servlet>
```

Now you have the pieces in place to compile and restart your server so you can run `AjaxEnabledPortlet` just created. The portlet should display a link "Invoke Portlet Data API via Ajax" and a `<div>` that says "replace me".

When the link "Invoke Portlet Data API via Ajax" is clicked, the "replace me" should be replaced by the data, "The answer is 42", coming back from the "MyDataService" servlet.

This shows one way to invoke a data service using Ajax within a JSF portlet.

8.11 Understanding Navigation Within a JSF Portlet

This section discusses navigation between views within a JSF portlet:

- Section 8.11.1, "Navigating Within a Portlet with the JSF Controller"
- Section 8.11.2, "Support for Redirects"

8.11.1 Navigating Within a Portlet with the JSF Controller

This section discusses the use case in which a user interacts with a portlet, triggering an update to that portlet.

It is standard to use *command buttons* and *command links* tied to actions to control navigation through the JSF application. Example 8–13 uses a navigation rule and in Example 8–14 a `commandButton` uses that rule. With the JSF navigation controller, three different approaches may be used when defining the value for the `action` attribute on the command button or the command link:

- The `action` attribute is not specified on the component. This indicates that the navigation is a postback to the same view.
- The `action` attribute is a hard-coded name of a navigation path to follow. This name is mapped in the controller configuration in the `faces-config.xml` file linked to an actual view in Example 8–13.
- The `action` attribute contains Expression Language (EL) that invokes a backing bean method to evaluate a navigation path or null to indicate postback.

Each of these approaches is valid when a JSF application is operating as a JSF portlet. The JSF portlet bridge ensures the URLs are written properly to maintain this behavior correctly within a portal.

Example 8–13 Navigation Configuration in faces-config.xml

```
<navigation-rule>
   <from-view-id>/portlets/sample/page2.jsp</from-view-id>
   <navigation-case>
      <from-outcome>gotoPage1</from-outcome>
      <to-view-id>/portlets/sample/page1.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

Example 8–14 Command Button That Uses the Navigation Rule

```
<h:commandButton action="gotoPage1" id="gotoPage1Button"
                 value="Goto Page 1">
</h:commandButton>
```

8.11.2 Support for Redirects

The JSF navigation controller supports HTTP redirect as an annotation on a navigation case. The JSF controller issues an HTTP redirect to the client for the target page of the navigation case if it is configured to do so. Example 8–15 illustrates such a redirect navigation case. In Example 8–15, note the use of the explicit `<redirect/>` element to indicate a redirect. The navigation case does not work properly when the redirect configuration is used within a JSF portlet in WLP.

Example 8–15 Redirect in a Navigation Case

```
<navigation-rule>
   <from-view-id>/portlets/redirect/page2.jsp</from-view-id>
   <navigation-case>
      <from-outcome>gotoPage1</from-outcome>
      <to-view-id>/portlets/redirect/page1.jsp</to-view-id>
      <redirect/>
   </navigation-case>
</navigation-rule>
```

Depending on the context in which it is being called, the above redirect navigation case will either be ignored by the portal, or the rule will execute and send the user to the target page via an entire web page redirect, thus taking the user out of the portal context entirely. This behavior occurs because the redirect URL is not rewritten to remain within the portal environment. Therefore, you should avoid specifying `redirect/` in the JSF navigation rules since it tries to redirect the entire web page and not just the portlet itself.

Another way to do a redirect to the same page is to use the Java portlet `setEvent` to redirect from a managed bean. For instance, after a user has successfully authenticated, the portlet should be redirected so the portal page is re-rendered with the authentication information. This can be done like this:

```
FacesContext fc = FacesContext.getCurrentInstance();
obj = fc.getExternalContext().getResponse();
ActionResponse ar = (ActionResponse) obj;
 ar.setEvent(new QName("urn:com:oracle:wlp:netuix:event:portal",
 "framework.redirectBeforeRender"),null);
```

This triggers a portal event to inform the container that it should refresh its components before rendering. In the authentication use case, this provides for the correct user experience by re-rendering the portal components to display any personalization effects for the authenticated user.

8.12 Interportlet Communication with JSF Portlets

The example described in this section assumes there are two JSF Java portlets set up in your portal. The example will explain how Interportlet Communication (IPC) can be set up between them, allowing one portlet (the sending portlet) to notify another portlet (the receiving portlet) of a user interaction. This is just a simple overview of one possible use case for using IPC in JSF -Java portlets; it is not a complete step-by-step example.

In this example, a user clicks a link in the sending portlet and as a result, the receiving portlet displays the URL tied to that link. So this example is about one portlet triggering a navigation event in another portlet. It may be helpful to brush up on the details of Interportlet Communication (IPC) with Java portlets, see Chapter 12,

"Configuring Local Interportlet Communication." Note that events described in this section are Java Portlet Events and not the native WLP events.

In order to have a Java event handled by the Java Portlet and not automatically sent through to the JSR-329 bridge as a Faces event, the following `<context-param>` must be added to the `web.xml` file. This is documented in the JSR-329 spec, but noted here as well. In this snippet of XML as shown in Example 8–16, the `javax.portlet.faces.autoDispatchEvents` application parameter is set to `false`. This is a web application level setting, not a portlet-level setting.

Example 8–16 autoDispatchEvents Set to False

```
<!--  tells the bridge to let the jsr286 events process -->
<context-param>
    <param-name>javax.portlet.faces.autoDispatchEvents</param-name>
    <param-value>false</param-value>
</context-param>
```

This setting allows for Java portlets to have their annotated events called. Without it, all events are assumed to be JSF events and the portlet container is bypassed.

Receiving Portlet Setup

The Java portlet code for the portlet receiving the Java event is shown in Example 8–17. In this example, a subclass called `ExampleIPCPortlet` extends `GenericFacesPortlet` so that code can be added to handle the event and do customized processing in the `doView()` method. For more information about these methods, see Chapter 6, "Building Java Portlets."

Example 8–17 Java Portlet Example

```
public class ExampleIPCPortlet extends GenericFacesPortlet {
    private static String CURRENTVIEWID = "currentViewId";
    @Override
    protected void doView(RenderRequest request, RenderResponse response)
        throws PortletException, java.io.IOException
    {
        String view = (String) request.getParameter(CURRENTVIEWID);
        if (view != null) {
           request.setAttribute("javax.portlet.faces.viewId", view);
        }
        super.doView(request, response);
     }
     @ProcessEvent(qname="{http://oracle.com/sampleEvents}display.change")
     public void processViewChange(EventRequest eRequest, EventResponse eResponse)
       throws PortletException, IOException
     {
        Event event = eRequest.getEvent();
        String eventValue = (String) event.getValue();
        // set the new view id
        eResponse.setRenderParameter(CURRENTVIEWID, eventValue);
    }
}
```

The `portlet.xml` (Example 8–18) file contains the definition of the event to be dispatched to the receiving portlet. The event is defined like this:

```
<event-definition>
     <qname xmlns:x="http://oracle.com/sampleEvents">x:display.change</qname>
     <value-type>java.lang.String</value-type>
```

```
</event-definition>
```

The portlet must be set up to listen for this event. The `portlet.xml` file snippet, shown in Example 8–18, contains an example of the receiving portlet definition. The `<supported-processing-event>` indicates the events to which the receiving portlet listens. Also note `<portlet-class>` for the receiving portlet is not `GenericFacesPortlet`, but the subclass name `oracle.samples.wlp.jsf.portlets.ExampleIPCPortlet` instead. The subclassing of `GenericFacesPortlet` was required to create a customized `doView()` method and the `processViewChange()` method shown above in the Example 8–17, "Java Portlet Example".

Example 8–18 portlet.xml

```
<portlet>
     <portlet-name>display</portlet-name>
     <display-name>display</display-name>
     <portlet-class>oracle.samples.wlp.jsf.portlets.ExampleIPCPortlet
     </portlet-class>
     <init-param>
         <name>javax.portlet.faces.defaultViewId.view</name>
         <value>/demo/display.jsp</value>
     </init-param>
     <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>view</portlet-mode>
     </supports>
     <portlet-info>
        <title>Display</title>
        <short-title>Display</short-title>
     </portlet-info>
     <supported-processing-event>
         <qname xmlns:x="http://oracle.com/sampleEvents">x:display.change</qname>
     </supported-processing-event>
</portlet>
```

When the link in the sending portlet is clicked, the `display.change` is fired and the receiving portlet (`ExampleIPCPortlet`) is set up to listen for that event. As a result, the receiving Java portlet sets a render parameter when processing the event. This render parameter is accessed when the `doView()` method is called, thus changing the receiving portlet's view when the portlet container completes the render request for the portlet.

Sending Portlet Setup

To create the sending portlet, which sends the event to the receiving portlet, a regular GenericFacesPortlet with a regular JSF-JSP file that uses a JSF managed bean to create a URL can be used. The `.jsp` file should contain a link that looks something like Example 8–19. In addition, the managed bean for this JSF view will contain the `myURL` method like shown in Example 8–20. This method sends an event with the URL it wants the portlet receiving the event to change its view to. (Hardcoding URLs like this example is not recommended in a production environment, but is here for demonstration purposes only.)

The Example 8–18 shows the XML necessary for the portlet to identify that it will send this Java portlet event. Note the `<supported-publishing-event>` added to its portlet definition.

```
<supported-publishing-event>
```

```
      <qname
       xmlns:x="http://oracle.com/sampleEvents"> x:display.change
      </qname>
</supported-publishing-event>
```

Example 8–19 .jsp File to Call a Bean

```
… //This is an example of a commandLink using a faces bean to create the action
url //
 <t:commandLink action="#{ExampleBean.myURL}">
               <h:outputText value="my IPC example" />
 </t:commandLink>
```

Example 8–20 Faces Bean Method for "ExampleBean" That Sends the Event

```
public String myURL() {
   sendEvent("/demo/examples/myNext.jsp");
   return null;
}
public void sendEvent(String viewId){
       FacesContext fc = FacesContext.getCurrentInstance();
       Object obj = fc.getExternalContext().getResponse();
       if (obj instanceof ActionResponse){
          ActionResponse ar = (ActionResponse) obj;
          ar.setEvent
            (new QName("http://oracle.com/sampleEvents",
                       "display.change"),
                       viewId );
     }
}
```

To summarize the flow of events, when a user clicks the link in the sending portlet, the sending portlet uses the URL created at render time from `myURL()` in the bean code, which in turn calls `sendEvent(viewId)` with the appropriate URL. The `sendEvent` method then creates a Java portlet event which the receiving portlet is configured to listen, as shown above in the `portlet.xml` previously described. Upon receiving the event, the receiving portlet should display the "`/demo/examples/myNext.jsp`".

Now you have a portal example that uses JSF for its components and Java portlet eventing to trigger Interportlet Communication that invokes a navigational change of the listening portlet.

8.13 Namespacing

This section discusses namespacing issues that must be addressed when building JSF portlets. Namespacing often comes into play when multiple instances of the same portlet exist within the portal. When there is more than one instance of a portlet, or two different portlets are using similar Javascript, then each instance needs to label its component uniquely. This is commonly done with putting a unique instance Id in front of the portlet's components in order to uniquely identify them through code. The following sections describe ways to do this:

- Section 8.13.1, "Client ID Namespacing with the View Components"
- Section 8.13.2, "Client ID Namespacing with the WLP NamingContainer"
- Section 8.13.3, "Javascript Namespacing with Portlet Tag Library"

8.13.1 Client ID Namespacing with the View Components

When a JSF portlet is rendered onto a portal page, it can coexist with other JSF portlets on that same page. In JSF, like other web frameworks, it is critical that the Id attribute on each element of the X/HTML browser page is unique. JSF has the concept of a naming container, which provides an Id namespace to all components it contains. The most common naming container is the View component (the `f:view` tag).

Because all JSF portlets contain an `f:view` tag as the root component (otherwise, it's not really a JSF portlet but just a Java Portlet at that point), WebLogic Portal and the JSR-329 bridge, for most cases, can correctly introduce a namespace for all component Ids in each portlet.

8.13.2 Client ID Namespacing with the WLP NamingContainer

In the native JSF bridge, Oracle recommended the use of the `NamingContainer` component to strengthen the scoping mechanism within the portal. For JSF with Java Portlet technology, this practice is no longer necessary and causes problems if used. If migrating from the WLP native bridge to the JSR-329 bridge, remove any `NamingContainer` components from the JSF views.

8.13.3 Javascript Namespacing with Portlet Tag Library

There are scenarios when you may find it helpful to use the standard portlet tag library and the namespace tag, as shown in Example 8–21. The Java portlet specification mandates that this tag library be available in the portlet container to give `.jsp` files an access to the portlet-specific elements. In this example, the `<div id …>` is namespaced, so that each instance of this portlet contains a unique `<div id …>`, and allows code to access it accordingly by getting the instance label of the portlet. This is not unique to JSF portlets; a feature of the Java Portlet standard.

Example 8–21 Javascript Namespacing Example

```
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>

<portlet:namespace var="portletns"/>
.
.
.
<div id="<%=portletns%>_xprDiv" style="background-color: #ffcccc;">replace me
</div>
.
.
.
```

The WLP API also allows access to the portlet instance label for namespacing purposes via PortletPresentationContext and the PortletBackingContext. The method in JSFJavaPortletHelper Class called `getInstanceLabel` implements an example of one way to do this, and can be called from within a managed bean. The Ajax Enablement example shows how this can be used to change a `<div>` through an Ajax call.

8.14 Code Examples for Common Use Cases

These code examples are illustrated here for learning purposes. These examples describe *one of the many ways* to implement common use cases. Your business application may require an expansion to these concepts.

- Section 8.14.1, "Uploading Images"
- Section 8.14.2, "Login/Logout Example"
- Section 8.14.3, "Login Portlet Implementation"
- Section 8.14.4, "Putting Login Portlet Into A Portal environment"

8.14.1 Uploading Images

This section discusses uploading a file and serving the uploaded file (or any resource) to the browser. JSF 1.1 and JSF 1.2 did not come with an equivalent JSF component tag for the HTML file upload tag of:

```
<input type="file" size="50" name="file" accept="image/png,image/jpeg,image/gif"/>
```

However, the third party tools such as Tomahawk provide such tags. In a JSR-286-based portlet, you can use multiple ways to upload files. One way is to create a pure JSR-286 solution, without using JSF because JSF does not provide the appropriate tag.

This section includes the following:

- Section 8.14.1.1, "File Upload with HTML tags"
- Section 8.14.1.2, "File Upload with Tomahawk tags"

8.14.1.1 File Upload with HTML tags

If you are not interested in using a third party library, but want to do a file upload, then you need to create a normal HTML JSP and parse the form manually.

1. Create the JSP file for uploading files. This should contain plain HTML elements since JSF 1.2 does not provide a similar component.

Example 8–22 JSP File For Uploading Files Using Plain HTML Elements

```
%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ page import="javax.portlet.*"%>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<portlet:defineObjects />
Example of multiple file upload.
<form id="uploadForm" method="post" action="<portlet:actionURL/>"
enctype="multipart/form-data">
<table border="0">
<tr>
    <td/>
</tr>
<tr>
   <td>Select your picture:</td>
   <td><input type="file" size="50" name="file"
accept="image/png,image/jpeg,image/gif" /></td>
</tr>
<tr>
   <td/>
</tr>
```

```
<tr>
    <td>Select your picture:</td>
    <td><input type="file" size="50" name="file"
accept="image/png,image/jpeg,image/gif" /></td>
</tr>
<tr>
    <td><input type="submit" value="Upload" /></td>
</tr>
</table>
</form>
<%
    String message = renderRequest.getParameter("renderMessage");
    if (message != null) {
  %>
<pre>
<%= message %>
</pre>
<%
  }
%>
```

2. Create the Java class to parse your form. Example 8–23 uses the `javax.mail.*` classes to process the content from the multipart form. It uses a class called `FormDataHelper`. This class takes an `ActionRequest` from the `processAction` JSR-286 method.

Example 8–23 FormDataHelper

```
package oracle.samples.wlp;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.activation.DataSource;
import javax.mail.BodyPart;
import javax.mail.MessagingException;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMultipart;
import javax.portlet.ActionRequest;
import javax.portlet.PortletException;

/**
 * This uses the javax.mail classes to take the content from the multipart form.
 * This is called from the 'processAction' method of a JSR286 portlet.
 * To construct this class, send in the actionRequest from the processAction
   method.
 *
 */
public class FormDataHelper {
  // The MIME prefix for all multipart media types
  private static final String MULTIPART_PREFIX = "multipart/";
  // The MIME content type for the file upload form data
  private static final String MULTIPART_FORM_DATA = "multipart/form-data";
  // The MIME content type for plain text data
  private static final String PLAIN_TEXT = "text/plain";
  private ActionRequest actionRequest;
```

```
  //these are initialized lazily
  private Map<String, List<String>> formParams;
  private Map<String, List<MimeBodyPart>> formBodyParts;
  private boolean mIsInitialized=false;
  private boolean mHasMultipart=false;
  /**
    * Constructor
    */
  public FormDataHelper(ActionRequest request)
  {
  actionRequest = request;
  }
  /**
    * Get the parameter value for the given name
    * @param name
    * @return
    * @throws PortletException
    */
  public String getParameter(String name)
    throws PortletException
  {
     String value = actionRequest.getParameter(name);
     if (value != null){
        return value;
 }
// If we don't find a value, make sure we have parsed in the parameters and body
parts
     initParameters();
     if (!mHasMultipart)
     {
       return null;
     }
     List<String> paramList = (List<String>)formParams.get(name);
     if (paramList != null)
     {
     return (String)paramList.get(0);
     }
    else
    {
       return null;
    }
}
/**
 * Get all the parameter values for the given name
 * @param name
 * @return
 * @throws PortletException
 */
public String[] getParameterValues(String name)
throws PortletException
{
   String[] values = actionRequest.getParameterValues(name);
   if (!mHasMultipart)
   {
      return values;
   }
   // Make sure we have parsed in the parameters and body parts
   initParameters();
     List<String> valueList = formParams.get(name);
     if (valueList == null)
```

```
     {
        return values;
     }
        else
     {
         int size = valueList.size();
         if (values != null)
         {
            List<String> newValueList = new ArrayList<String>(values.length +
            size);
            newValueList.addAll(Arrays.asList(values));
            newValueList.addAll(valueList);
            valueList = newValueList;
        }
        values = new String[size];
        valueList.toArray(values);
        return values;
   }
}
/**
 * Get mime body part
 * @param name
 * @return
 * @throws PortletException
 */
public MimeBodyPart getMimeBodyPart(String name)
throws PortletException
{
    // Make sure we have parsed in the parameters and body parts
    initParameters();
    if (!mHasMultipart)
    {
      return null;
    }
    List<MimeBodyPart> parts = (List<MimeBodyPart>)formBodyParts.get(name);
    return parts == null ? null : (MimeBodyPart)parts.get(0);
}
/**
 * Get all the mime parts for the given name
 * @param name
 * @return
 * @throws PortletException
 */
public MimeBodyPart[] getMimeBodyParts(String name)
throws PortletException
{
// Make sure we have parsed in the parameters and body parts
  initParameters();
  if (!mHasMultipart)
  {
    return null;
  }
  List<MimeBodyPart> parts = formBodyParts.get(name);
  if (parts == null)
  {
    return null;
  }
  MimeBodyPart[] mimeBodyParts = new MimeBodyPart[parts.size()];
parts.toArray(mimeBodyParts);
return mimeBodyParts;
```

```
}
/*
 * Lazy initialization of the parameter map and body parts.
 * This is what processes the form and figures out what is there.
 */
private void initParameters() throws PortletException
{
  if (mIsInitialized)
{
  return;
}
// Strip off any extra attributes and whitespace from the content type,
// so we can compare it
  String contentType = actionRequest.getContentType();
  if (contentType == null)
{
  mIsInitialized = true;
  return;
}
  int sepIndex = contentType.indexOf(';');
  if (sepIndex != -1)
{
    contentType = contentType.substring(0, sepIndex).trim();
}
   if (contentType.equalsIgnoreCase(MULTIPART_FORM_DATA))
 {
     formParams = new HashMap<String, List<String>>(20);
     formBodyParts = new HashMap<String, List<MimeBodyPart>>(20);
     DataSource datasource = new DataSource()
         {
             public InputStream getInputStream() throws IOException
             {
                 return actionRequest.getPortletInputStream();
             }
             public OutputStream getOutputStream() throws IOException
             {
                 throw new IOException("OutputStream not available");
             }
             public String getContentType()
             {
                 return actionRequest.getContentType();
             }
             public String getName()
             {
                 return getClass().getName();
             }
         };
     try
     {
         MimeMultipart multipartMessage = new MimeMultipart(datasource);
         parseMultiPart(multipartMessage, null);
     }
     catch (MessagingException me)
     {
         throw new PortletException(me);
     }
     catch (IOException ioe)
     {
         throw new PortletException(ioe);
     }
```

```
    mHasMultipart = true;
  }
 mIsInitialized = true;
}
private void parseMultiPart(MimeMultipart multipartMessage, String
parentFieldName)
throws MessagingException, IOException
{
  int partCount = multipartMessage.getCount();
  // Go through each body part, decided on its 'type' and add to the
  // parameter map and file list as appropriate
  BodyPart part;
  MimeBodyPart mimePart;
  Object content;
  String[] disps;
  String disp, lcDisp;
  List<MimeBodyPart> partValues;
  List<String> values;
  for (int i = 0; i < partCount; i++)
{
  part = multipartMessage.getBodyPart(i);
  if (!(part instanceof MimeBodyPart))
{
  continue;
}
  mimePart = (MimeBodyPart)part;
  // The Content Disposition header tells us how to treat the body part
  disps = mimePart.getHeader("Content-Disposition");
  if (disps == null || disps.length == 0)
{
  continue;
}
  disp = disps[0];
  lcDisp = disp.toLowerCase();
  // Get the field name out of the disposition header, if present
  int nameStart, nameEnd;
  if ((nameStart = lcDisp.indexOf("name=\"")) != -1 &&
  (nameEnd = lcDisp.indexOf("\"", nameStart + 6)) != -1)
{
   parentFieldName = disp.substring(nameStart + 6, nameEnd);
}
  // If we don't have a field name, there's not much we can do with this body part
  if (parentFieldName == null)
{
  continue;
}
  // If this is a multipart body part, we recurse on its contents using
  // the current field name
  if (mimePart.getContentType().toLowerCase().startsWith(MULTIPART_PREFIX))
{
  content = mimePart.getContent();
  if (content instanceof MimeMultipart)
    {
       parseMultiPart((MimeMultipart)content, parentFieldName);
    }
}
// Decide whether this is a parameter or a file, according to whether
// the filename attribute is present in the disposition header
else if ((nameStart = lcDisp.indexOf("filename=\"")) != -1 &&
(nameEnd = lcDisp.indexOf("\"", nameStart + 10)) != -1)
```

```
{
  partValues = formBodyParts.get(parentFieldName);
  if (partValues == null)
   {
    partValues = new ArrayList<MimeBodyPart>();
    formBodyParts.put(parentFieldName, partValues);
}
partValues.add(mimePart);
//get the Filename from header -- so we can clip the path if it exists.
 String
 filename=getFileNameFromHeader(lcDisp.substring(nameStart+10,lcDisp.length()-1));
   if (!(filename).equals(mimePart.getFileName())){
   mimePart.setFileName(filename);
  }
}
   else if (mimePart.getContentType().toLowerCase().startsWith(PLAIN_TEXT))
{
  content = mimePart.getContent();
  if (content instanceof String)
   {
   values = formParams.get(parentFieldName);
   if (values == null)
           {
             values = new ArrayList<String>();
             formParams.put(parentFieldName, values);
           }
     values.add((String)content);
       }
    }
  } // end for each body part
}
private String getFileNameFromHeader(String filename){
// The filename may contain a full path.  Cut to just the filename.
   int slash =
    Math.max(filename.lastIndexOf('/'), filename.lastIndexOf('\\'));
    if (slash > -1) {
    filename = filename.substring(slash + 1);  // past last slash
      }
     return filename;
   }
}
```

3. Create the Java portlet class that extends `GenericJavaPortlet`, as shown in Example 8–24. Use this class when generating your Java Portlet (`.portlet` file) through the IDE. Notice that the `processAction` method instantiates the `FormDataHelper` class and processes the file data out of the mime parts.

Example 8–24 JavaFileUploadPortlet

```
package oracle.samples.wlp.portlets;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.mail.MessagingException;
import javax.mail.internet.MimeBodyPart;
import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
```

```
import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import oracle.samples.wlp.FormDataHelper;
/**
 * A JSR286 portlet that uploads content from a multipart form.
 * This was intended to use with JSF for the form,
 * but JSF does not have a component equivalent to the <input type="file" />,
 * so instead it uses a .jsp file for the html for this portlet.
 */
public class JavaFileUploadPortlet extends GenericPortlet {
private static final String DISPATCH_JSP = "/demo/fileupload/fileupload.jsp";
/**
 * Dispatches to a jsp to display the form.
 */
public void doView(RenderRequest renderRequest, RenderResponse renderResponse)
    throws PortletException, IOException
    {
  PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher(DISPATCH_
JSP);
 rd.include(renderRequest, renderResponse);
}
/**
 * Takes the action request which creates a FormDataHelper so that the multipart
 * form data can be accessed.
 */
@Override
public void processAction(ActionRequest actionRequest, ActionResponse
actionResponse)
{
        FormDataHelper formdata = new FormDataHelper(actionRequest);
        try {
        MimeBodyPart[] mimeparts = formdata.getMimeBodyParts("file");
        int uploaded =0;
        if (mimeparts == null)
        {
        actionResponse.setRenderParameter("renderMessage", "Didn't find files to
upload.");
        return;
        }
        for (MimeBodyPart filedata: mimeparts){
        // Just to demonstrate what information you can get from the uploaded
file.
        System.out.println("File type: " + filedata.getContentType());
        System.out.println("File name: " + filedata.getFileName());
        System.out.println("File size: " + filedata.getSize() + " bytes");

        /************  your business logic here  ************
         * This is writing the image to the "uploadedcontent"
 * directory under WebContent.
         * This directory needs read/write permissions by the 'user
 * running the webserver'.
         */
        InputStream inputstream = filedata.getInputStream();
        if (filedata.getSize() > 0){
        byte [] imageBytes = new byte[inputstream.available()];
        inputstream.read(imageBytes);
        String realpath = getPortletContext().getRealPath("/uploadedcontent/");
```

```
        File file =new File(realpath + "/" + filedata.getFileName());
        FileOutputStream fop=new FileOutputStream(file);
        fop.write(imageBytes);
        fop.flush();
        fop.close();
        uploaded++;
            }
        }
        actionResponse.setRenderParameter("renderMessage", "Uploaded " + uploaded
+ " files.");
       }
        catch (PortletException e) {
        // Show error message.
            FacesContext.getCurrentInstance().addMessage("uploadForm", new
FacesMessage(
                FacesMessage.SEVERITY_ERROR, "File upload failed with
PortletException error.", null));
e.printStackTrace();
        }
catch (MessagingException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
catch (IOException e) {
FacesContext.getCurrentInstance().addMessage("uploadForm", new FacesMessage(
                FacesMessage.SEVERITY_ERROR, "File upload failed with IOException
error.", null));
e.printStackTrace();
}

    }
}
```

From here, you could expand your HTML to create a separate portlet to display the content you just uploaded. This is just an example of how to upload content. Typically, the location of the uploaded content is configurable.

8.14.1.2 File Upload with Tomahawk tags

To avoid writing your own parsing mechanism, another option is to install third party libraries like Tomahawk. Tomahawk provides the component called `inputFileUpload` that provides the capability to upload content. Although WebLogic Portal does not explicitly support it, you can use this component by adding third party libraries. See Third-Party Libraries for more details on how to configure this library.

8.14.2 Login/Logout Example

Another common example that nearly every web application needs is the capability to authenticate/unauthenticate the user. This section provides some code examples to help you develop login/logout functionality within Java-JSF portlets. There are multiple ways to develop this functionality, this section focuses on some of them.

This section includes the following:

- Section 8.14.2.1, "Login Portlet Design"
- Section 8.14.2.2, "Handling Redirects with JSR-286/JSR-329"
- Section 8.14.2.3, "Invalidating the Session with the JSR-329 Bridge"

8.14.2.1 Login Portlet Design

The login portlet discussed in this section is implemented in JSF. It uses a single view that toggles the visibility of the login/logout controls based on the authenticated state of the user.

This login portlet offers the following features:

- Implemented as a JSF portlet
- A single view with two forms - one for login and one for logout
- Uses a JSF managed bean and dispatches to a servlet to perform the authentication logic
- Employs the JSF localization facility

Figure 8–2 JSF Login

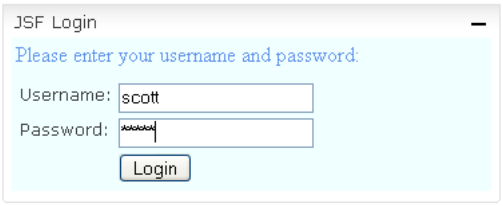
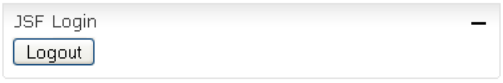


Figure 8–3 Logout



This is different from a typical web application since in a web application the login/logout page is normally the first screen the user reaches and thus all future pages have an authenticated session. In the portal paradigm, the user may access portlets within a portal without logging in and seeing a non-customized view of the application. Then, once he/she does login, via a login portlet like shown here, then all other portlets within the portal must re-render with this new authenticated session and show the user's customized view. The opposite is true once a user logs out. This need to re-render all the components in the portal tree adds a little more complexity to managing login/logout functionality in a portal vs a regular web application.

The two biggest considerations for the developer are triggering redirects to manage the re-rendering of the component tree, and authenticating/unauthenticating the user session within the middle of a JSF lifecycle.

8.14.2.2 Handling Redirects with JSR-286/JSR-329

By the time a user is authenticated, the portal framework has also started processing the rendered page. It is too late for the framework to re-compute the page. Therefore, the solution is to force a redirect after the user logs in or logs out via the login portlet. In the native bridge architecture, this is accomplished via a backing file. But with JSR-286/JSR-329, this can be simplified because JSR-329 allows for backing bean action methods to trigger events to do redirects.

8.14.2.3 Invalidating the Session with the JSR-329 Bridge

The WLP JSF portlet native bridge's limitation is that a user's HttpSession cannot be invalidated when the JSF lifecycles are processing. This is no longer an issue with the JSR-329 bridge.

8.14.3 Login Portlet Implementation

This section includes the following:

8.14.3.1 JSF Login View

The `login.jsp` page contains two forms - one for login and one with a button to logout. Only one form is visible at a time, and the visibility is controlled by a flag that indicates whether the user is authenticated.

Example 8–25 The login.jsp for the JSF 1.2 Login Portlet

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
<f:view>
    <f:loadBundle basename="oracle.samples.wlp.jsf.portlets.login.loginportlet"
var="i18n" />
      <h:form id="loginBeanForm"
rendered="#{!JSFLoginPortletRequestBean.authenticated}">
      <h:panelGrid id="outerLayout" columns="1" width="100%"
style="background-color: azure;">
      <h:panelGroup id="titleLine">
      <h:outputText value="#{i18n.login_intro}:" style="color: cornflowerblue;
font-size: medium"/>
   </h:panelGroup>
       <h:messages  layout="table" showDetail="false" showSummary="true"
style="color: red;" />
   <h:panelGroup id="formFields">
   <h:panelGrid columns="2" width="60%"
      style="background-color: azure">
   <h:panelGroup style="text-align: right">
   <h:outputText value="#{i18n.login_username}:"/>
   </h:panelGroup>
      <h:panelGroup style="text-align: left">
      <h:inputText id="username" required="true"
   value="#{JSFLoginPortletRequestBean.username}" />
   </h:panelGroup>
      <h:panelGroup style="text-align: right">
      <h:outputText value="#{i18n.login_password}:"/>
   </h:panelGroup>
      <h:panelGroup style="text-align: left">
      <h:inputSecret id="password" required="true"
      value="#{JSFLoginPortletRequestBean.password}" />
  </h:panelGroup>
   <h:panelGroup/>
   <h:panelGroup style="text-align: left">
   <h:commandButton id="loginButton" immediate="false"
action="#{JSFLoginPortletRequestBean.authenticate_generic}" value="#{i18n.login_
button}"/>
    </h:panelGroup>
```

```
    </h:panelGrid>
    </h:panelGroup>
   </h:panelGrid>
</h:form>
   <h:form id="logoutForm" rendered="#{JSFLoginPortletRequestBean.authenticated}">
   <h:panelGrid id="outerLayout" columns="1" width="100%" style="background-color:
azure;">
   <h:commandButton action="#{JSFLoginPortletRequestBean.userLogout_generic}"
id="logoutButton" value="#{i18n.logout_button}"/>
   </h:panelGrid>
</h:form>
</f:view>
```

8.14.3.2 JSF Managed Backing Bean

The `login.jsp` in Example 8–25 uses a JSF managed bean, which is shown in Example 8–26. Notice the JSF managed backing bean calls the `JSFJavaPortletHelper` class which contains the core logic to log the user in.

Example 8–26 JSF Managed Bean Calls the JSFJavaPortletHelper Class

```
package oracle.samples.wlp.jsf.portlets.login;
import java.io.Serializable;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import oracle.samples.wlp.jsf.JSFJavaPortletHelper;
public class JSFLoginPortletRequestBean implements Serializable {
private static final long serialVersionUID = 1L;
private static final String BUNDLE_NAME =
"oracle.samples.wlp.jsf.portlets.login.loginportlet";
private String username;
private String password;
private String action;
// ACTION METHODS
/**
     * Action method for the login CommandButton
     * Uses non-wlp specific way of authentication
     *
     * @return
     */
    public String authenticate_generic() {
        // perform the actual authentication call
        setAction("login");
        boolean success = JSFJavaPortletHelper.authenticate_generic(username,
password);
        if (success) {
            // Wipe out the given username & password, just in case someone
            // is tempted to make this bean Session scoped (should be
            // Request) By keeping this password around in the bean,
            // it is open to temptation for abuse
            password = "invalidated";
            username = "invalidated";
        }
        else {
            //Login Failed
            String errorText = JSFJavaPortletHelper.getBundleMessage(BUNDLE_NAME,
"login_error");
            // Add the message to the context
            FacesContext fc = FacesContext.getCurrentInstance();
```

```
            FacesMessage msg = new FacesMessage(
                  FacesMessage.SEVERITY_ERROR, errorText, errorText);
            fc.addMessage(null, msg);

        }
        return null;
    }
    /**
     * Action method called during logout for the servlet to do the logout
     */
    public String userLogout_generic() {
        setAction("logout");
        JSFJavaPortletHelper.logout_generic();
        return null;
    }
// GETTERS AND SETTERS

**
* @return true if the user is authenticated
*/
public boolean isAuthenticated() {
    return JSFJavaPortletHelper.isAuthenticated();
}
/**
* @return the user name
*/
public String getUsername() {
    return this.username;
}
public String getUsernameForDisplay(){
    return JSFJavaPortletHelper.getUsernameForDisplay("");

}
/**
* @param username the user name to be authenticated
*/
public void setUsername(String username) {
this.username = username;
}
/**
* Retrieves a placeholder for the password. We never
* want to display back the actual password to the user,
* so just return an empty string
* @return an empty String
*/
public String getPassword() {
  return this.password;
}
/**
* @param password the password to be used for authentication
*/
public void setPassword(String password) {
this.password = password;
}
public String getAction() {
   return this.action;
}
public void setAction(String action) {
   this.action = action;
}
```

```
}
```

The managed bean needs to be wired into the application. XML element shown in Example 8–27 must be added to `faces-config.xml`.

Example 8–27 Registering the Login Managed Bean in faces-config.xml

```
<managed-bean>
   <description>Handles authentication for the Login portlet.</description>
   <managed-bean-name>JSFLoginPortletRequestBean</managed-bean-name>
    <managed-bean-class>
       oracle.samples.wlp.jsf.JSFLoginPortletRequestBean
   </managed-bean-class>
<managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

8.14.3.3 Resource Bundle

A good practice is to create a resource bundle for the portlet. This should be located in a file that matches the basename of the `<f:loadBundle>` tag. In this example the resource bundle is in:
`oracle/samples/wlp/jsf/portlets/login/loginportlet.properties` in the Java resources `/src` folder.

Example 8–28 Login.jsp

```
#login.jsp
login_titlle=Login Page
login_imageAlt=Login
login_intro=Please enter your username and password
login_username=Username
login_password=Password
login_button=Login
logout_button=Logout
login_error=The username or password are invalid.
welcome_intro=Welcome
```

8.14.3.4 Portlet Definition File

Example 8–29 contains the pieces that tie together all the pieces described in this section. The `.portlet` file defines the portlet name and that it is a Java portlet.

Example 8–29 The Portlet Definition File for the Login Portlet (JSF 1.2 Java Portlet)

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root
    xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0"
    xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0
portal-support-1_0_0.xsd">
    <netuix:javaPortlet
        definitionLabel="login_generic" portletName="DemoLoginGeneric"
title="Generic Login"/>
</portal:root>
```

Example 8–30 portlet.xml File

```
<portlet>
     <portlet-name>DemoLoginGeneric</portlet-name>
     <display-name>DemoLoginGeneric</display-name>
     <portlet-class>javax.portlet.faces.GenericFacesPortlet</portlet-class>
     <init-param>
         <name>javax.portlet.faces.defaultViewId.view</name>
         <value>/demo/generic/login.jsp</value>
     </init-param>
     <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>view</portlet-mode>
     </supports>
     <portlet-info>
        <title>Login</title>
        <short-title>Login</short-title>
     </portlet-info>
</portlet>
```

The `portlet.xml` file in Example 8–30 shows that the `DemoLogin` portlet from the `.portlet` XML file is a `GenericFacesPortlet`, and contains `defaultViewId.view` as an `init-param`. If you subclassed `GenericFacesPortlet`, the subclass name should be in the `<portlet-class>` element. The `<init-param>` element of `javax.portlet.faces.defaultViewId.view` points to the `login.jsp` shown in Example 8–28. When the developer is creating a portlet via the portlet editor in workshop, and saves the portlet, then the `.portlet` file gets created as does the entry in the `portlet.xml` file. This is how the pieces are tied together.

Let's review the pieces we have and how they work together. The browser requests to view this portlet, the portlet container determines that it is a Java portlet, so it looks in the `portlet.xml` file for the class name, the class is instantiated and calls the `login.jsp` because that is what the `defaultViewId` is set to. Now the `login.jsp` uses the managed bean. This managed bean is invoked when a user submits the form. The web container recognizes the managed bean since it is configured in the `faces-config.xml` file.

Notice that the managed bean uses a class called `JSFJavaPortletHelper` (JSFJavaPortletHelper Class). In this code, you will notice that the actual authentication happens in a servlet named `LoginServlet`.

In the servlet, there are some important things to note. First, the `HttpServletRequest` that the LoginServlet receives from the dispatch is a wrapped `HttpServletRequest`. The `WLS ServletAuthentication` class requires a non-wrapped request in its method calls. So, the `unwrap` method in this LoginServlet example displays how the given wrapped `HttpServletRequest` can be unwrapped to get the actual `HttpServletRequest` the `ServletAuthentication` methods require.

Second, after authenticating, an attribute is associated with the request so the managed bean can use it to determine if the login/logout was successful. This is a design decision at this point, and you may have other security checks to do before a user can log in. Moreover, you should namespace this attribute name carefully in real-world implementations.

Example 8–31 LoginServlet

```
package oracle.samples.wlp.jsf.portlets.login;
import java.io.IOException;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;
import javax.servlet.http.HttpServletResponse;
import oracle.samples.wlp.jsf.portlets.login.JSFLoginPortletRequestBean;
import weblogic.servlet.security.ServletAuthentication;
/**
 * Login servlet that authenticates/unathenticates the user.
 *
 */
public class LoginServlet extends HttpServlet {
    public static final String AUTHENTICATED="LoginServlet.authenticated";
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doLoginLogout(request,response);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        doLoginLogout(request,response);
    }
    /**
     * Login in the user.
     * Instead of using ServletAuthentication, use your company's login logic in
this servlet.
     *
     * The request is updated with an attribute of "authenticated" with a boolean
value of
     * true = user is authenticated, false = the user is NOT authenticated;
     *
     * Note:  The given request needs to be unwrapped, so that Weblogic Server
gets an unwrapped request in order
     * to do authentication.  The given request needs to be kept, so that an
attribute can be added to it and be
     * seen on the JSFLoginPortletRequestBean.
     *
     * @param request
     * @param response
     */
    public void doLoginLogout(HttpServletRequest givenRequest, HttpServletResponse
givenResponse){

        JSFLoginPortletRequestBean loginPortletRequestBean =
(JSFLoginPortletRequestBean)
givenRequest.getAttribute("JSFLoginPortletRequestBean");
        if (loginPortletRequestBean != null) {
            if (loginPortletRequestBean.getAction().equals("login")) {
                //Note:  wls specific authentication here.  Implement your own
here as necessary.
                HttpServletRequest unwrappedRequest = unwrap(givenRequest);
                int result =
ServletAuthentication.weak(loginPortletRequestBean.getUsername(),
loginPortletRequestBean.getPassword(),unwrappedRequest,givenResponse);
                givenRequest.setAttribute(AUTHENTICATED, result ==
ServletAuthentication.AUTHENTICATED);
            }
            else {
                if (loginPortletRequestBean.getAction().equals("logout")) {
```

```
                    givenRequest.getSession().invalidate();
                    givenRequest.setAttribute(AUTHENTICATED, false);
                }
            }
        }
    }
    /**
     * Need to unwrap the request, so we can send the real request to the
ServletAuthentication
     * @param request
     * @return HttpServletRequest
     */
    private HttpServletRequest unwrap(HttpServletRequest request){
        while (request instanceof HttpServletRequestWrapper) {
            HttpServletRequestWrapper wrequest = (HttpServletRequestWrapper)
request;
            request =(HttpServletRequest) wrequest.getRequest();
        }
        return request;
    }
}
```

Remember you must add your servlet class to the `web.xml` file.

```
<servlet>
<!-- should only be accessible thru a named request dispatcher -->
<servlet-name>LoginServlet</servlet-name>
<servlet-class>oracle.samples.wlp.jsf.portlets.login.LoginServlet</servlet-class>
</servlet>
```

8.14.3.5 Redirect

The next important topic is *redirect*. Notice that both the `authenticate()` and `logout()` methods do a redirect. The page must be redirected so that any other portlets on the portal re-retrieve their markup to the newly authenticated/unauthenticated user. There are other ways of doing a redirect, but in WLP 10.3.2, this is the recommended way, which is called *send an event*. The code snippet is:

```
//redirect the page, so any customization for this user is picked up after
successful login
      obj = fc.getExternalContext().getResponse();
      ActionResponse ar = (ActionResponse) obj;
      r.setEvent(new QName("urn:com:oracle:wlp:netuix:event:portal",
"framework.redirectBeforeRender"),null);
```

8.14.4 Putting Login Portlet Into A Portal environment

The login portlet is ready, now let's see it working in a portal environment. To do this, you need to create a second portlet, called `WelcomePortlet` in this example. It will change the display to show the user name once login has occurred and no name once logout has occurred. The expected behavior is to see the `WelcomePortlet` refresh to show the authenticated user's version.

Example 8–32 shows the `welcome.portlet` file.

Example 8–32 welcome.portlet File

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root
   xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0"
   xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0
   portal-support-1_0_0.xsd">
    <netuix:javaPortlet
       definitionLabel="welcome" portletName="welcome" title="Welcome"/>
</portal:root>
```

Example 8–33 Entry in the portlet.xml file

```
<portlet>
    <portlet-name>welcome</portlet-name>
    <display-name>welcome</display-name>
<portlet-class>javax.portlet.faces.GenericFacesPortlet</portlet-class>
   <init-param>
       <name>javax.portlet.faces.defaultViewId.view</name>
       <value>/demo/welcome.jsp</value>
   </init-param>
 <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>view</portlet-mode>
  </supports>
  <portlet-info>
    <title>Welcome</title>
    <short-title>Welcome</short-title>
  </portlet-info>
</portlet>
```

Notice that the `defaultViewId` is the `welcome.jsp`. The `welcome.jsp` then uses the `JSFLoginPortletRequestBean` to determine if authentication has occurred and to show the user's display name.

Example 8–34 welcome.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f"  uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h"  uri="http://java.sun.com/jsf/html"%>
<f:view>
    <f:loadBundle basename="oracle.samples.wlp.jsf.portlets.login.loginportlet"
var="i18n" />
<h:form id="welcomeGenericForm"
rendered="#{!JSFLoginPortletRequestBean.authenticated}">
        <h:outputText value="#{i18n.welcome_intro}" />
    </h:form>
<h:form id="welcomeUserForm"
rendered="#{JSFLoginPortletRequestBean.authenticated}">
        <h:outputText value="#{i18n.welcome_intro}" />
<h:outputText  value="#{JSFLoginPortletRequestBean.usernameForDisplay}"/>
    </h:form>
</f:view>
```

Now add the WelcomePortlet to a portal using the workshop IDE. The resulting `.portal` file should look something like Example 8–35.

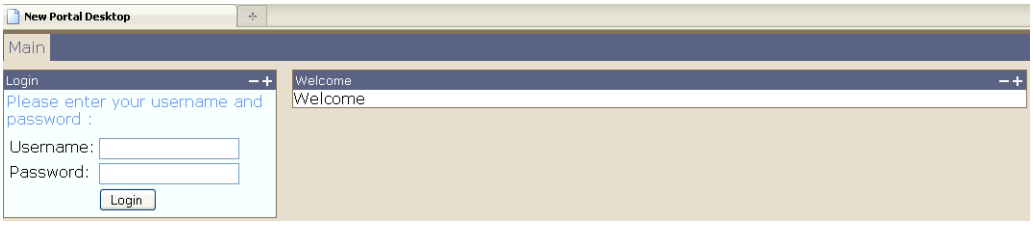
Example 8–35 demo.portal

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root xmlns:html="http://www.w3.org/1999/xhtml-netuix-modified/1.0.0"
    xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0"
    xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0
portal-support-1_0_0.xsd">
    <netuix:desktop definitionLabel="demo_demo_portal" discEnabled="false"
dvtEnabled="false"
        encoding="UTF-8" markupName="desktop" markupType="Desktop"
scrollToWindow="true"
        title="New Portal Desktop" treeOptimizationEnabled="true">
        <netuix:lookAndFeel definitionLabel="bighornLookAndFeel"
            description="A look and feel using the bighorn skin and skeleton"
            markupName="bighornLookAndFeel" markupType="LookAndFeel"
skeleton="bighorn"
            skin="bighorn" title="Bighorn">
            <netuix:titlebarButtonOrder>
                <netuix:otherButtons/>
                <netuix:namedButton name="float"/>
                <netuix:namedButton name="edit"/>
                <netuix:namedButton name="help"/>
                <netuix:namedButton name="minimized"/>
                <netuix:namedButton name="maximized"/>
                <netuix:namedButton name="delete"/>
            </netuix:titlebarButtonOrder>
        </netuix:lookAndFeel>
        <netuix:shell description="A header and footer is included in this shell."
            markupName="headerFooter" markupType="Shell" title="Header-Footer
Shell">
            <netuix:head/>
            <netuix:body>
                <netuix:header/>
                <netuix:book defaultPage="demo_demo_portal_page_1"
                    definitionLabel="demo_demo_portal_book_1" markupName="book"
                    markupType="Book" title="Main Page Book">
                    <netuix:singleLevelMenu
                        description="This menu provides a single level of tabs
used to navigate across pages."
                        markupName="singleLevelMenu" markupType="Menu"
                        title="Single Level Menu"/>
                    <netuix:content>
                        <netuix:page definitionLabel="demo_demo_portal_page_1"
                          markupName="page"
                            markupType="Page" title="Main">
                            <netuix:content>
                                <netuix:flowLayout
                                    description="This layout uses the flowLayout
                                    control to create two columns in which
                                    placeables flow vertically."

htmlLayoutUri="/framework/markup/layout/twocolumn.html.txt"

iconUri="/framework/markup/layout/twocolumn.gif"
                                    markupName="twoColumnFlowLayout"
                                    markupType="Layout"
                                    orientation="horizontal" title="Two Column
                                    Flow Layout">
                                    <netuix:placeholder
```

```
                                    description="The left most placeholder in
                                    this layout."
                                    markupName="twoColumnFlow_left"
                                    markupType="Placeholder"
                                    title="left" usingFlow="false"
                                    width="25%">
                                    <netuix:portletInstance
                                     contentUri="/demo/login.portlet"
                                     instanceLabel="login_1"
                                      markupType="Portlet" title="Login"/>
                                </netuix:placeholder>
                                <netuix:placeholder
                                    description="The right most placeholder in
                                    this layout."
                                    markupName="twoColumnFlow_right"
                                    markupType="Placeholder"
                                    title="right" usingFlow="false"
                                    width="65%">
                                    <netuix:portletInstance
                                        contentUri="/demo/welcome.portlet"
                                        instanceLabel="welcome_1"
                                        markupType="Portlet" title="Welcome"/>
                                </netuix:placeholder>
                            </netuix:flowLayout>
                        </netuix:content>
                    </netuix:page>
                </netuix:content>
            </netuix:book>
            <netuix:footer/>
        </netuix:body>
    </netuix:shell>
</netuix:desktop>
</portal:root>
```

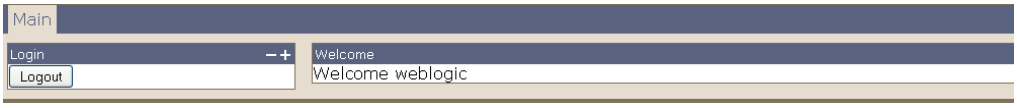
Now when running the portal, the first time rendered, the user isn't authenticated.

Figure 8–4 First Time/ Not Authenticated



Then, after logging in, it should look like Figure 8–5.

Figure 8–5 Authenticated



The redirect should automatically happen, which causes the `WelcomePortlet` to refresh itself and welcomes the new user.

8.15 Preparing JSF Portlets for Production

This section discusses best practices to follow before deploying a JSF portlet into a production environment.

- Section 8.15.1, "Configuration Tasks"
- Section 8.15.2, "Handling Errors"
- Section 8.15.3, "Performance and Scalability"
- Section 8.15.4, "Securing JSF Portlets"
- Section 8.15.5, "Localizing JSF Portlets"

8.15.1 Configuration Tasks

This section discusses configuration tasks to perform before deploying a JSF portlet to a production environment.

8.15.1.1 Configuring URL Templates for Proxy Servers

WLP is responsible for generating URLs that properly stay within the context of the portal. WLP does this for page tabs and also for portlet links. Section 8.11, "Understanding Navigation Within a JSF Portlet" describes how WLP rewrites links for JSF Command Button and Link components automatically.

In production, the URL that a user enters to navigate to a WebLogic Portal instance must not target the machine hosting that instance. The user should instead be routed through a proxy server or load balancer. This is a best practice for scalability and security. WLP provides a configuration feature for configuring URLs properly in such an environment. The JSF link rewriting mechanism honors this non-JSF configuration feature.

For more information on the WLP URL template feature, see the section "Working with URLs" in the Oracle Fusion Middleware Portlet Development Guide for Oracle WebLogic Portal at `http://download.oracle.com/docs/cd/E13155_01/wlp/docs103/portals/develop_portals.html#wp1006790`. Even though Apache Beehive dependency has been removed in 10.3.2 release, the following template configuration is still valid for proxy server configuration. You do not need the Apache Beehive facet for this configuration file to work.

The steps for configuring URLs to make use of a proxy server are:

1. In Eclipse, in your Portal Web Project, go to the **Merged Resources** view.
2. Right-click `WEB-INF/beehive-url-template-config.xml`, then choose **Copy to Project**.
3. Go to the **Project Navigation** view, then double-click the copied file.
4. Add the URL template entry shown in Example 8–36, replacing the IP address with the IP address of the proxy server or load balancer.
5. Configure the proxy server to forward to WebLogic Portal. (See your vendor's proxy server configuration documentation for instructions on how to do this.)

Example 8–36 Configure URL Generation That Refers to a Proxy Server

```
<url-template>
    <name>default</name>
    <value>http://192.168.0.5:{url:port}/{url:path}?{url:queryString}
{url:currentPage}
```

```
    </value>
</url-template>
```

8.15.1.2 JSF Portlets with WSRP

When developing portlets with JSF, the WSRP capabilities of WLP work correctly. JSF portlets can be exposed as WSRP portlets, just as with any other portlet type. For detailed information on WSRP portlets, see *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal* available at http://download.oracle.com/docs/cd/E15919_01/wlp.1032/e14235.pdf.

Caution: WSRP-1.0 and WSRP-2.0 do not support portlets with Ajax features. However, WLP includes proprietary extensions in WSRP-2.0 to provide limited support for client rewriteable resource URLs. Some Ajax functionality can be supported if the container in use is the WebLogic Portal container.

8.15.2 Handling Errors

When moving to production, you want to make sure all possible errors are handled gracefully and the user will not see a stack trace. See Section 8.10, "Using Common WLP Features With JSF Portlets" for details.

8.15.3 Performance and Scalability

This section discusses best practices for developing efficient and scalable JSF portlets.

- Section 8.15.3.1, "JSF Portlets in a Clustered Environment"
- Section 8.15.3.2, "Portlet Render Caching"

8.15.3.1 JSF Portlets in a Clustered Environment

WebLogic Portal runs on WebLogic Server, which includes industry-leading clustering technology. Clustering provides for both load balancing and failover capabilities. For the most part, WebLogic Server achieves both transparently. You do not need to know about the underlying clustering capabilities.

However, for failover to work properly, WebLogic Server replicates the user's `HttpSession` to a secondary node in the cluster. This enables the user to have a seamless experience if one node were to fail. But, in order for an `HttpSession` to be replicated without loss of data, the objects set as attributes within it must be serializable. For more information, see the section "Failover and Replication in a Cluster" in *Oracle Fusion Middleware Using Clusters for Oracle WebLogic Server* available at http://download.oracle.com/docs/cd/E12839_01/web.1111/e13709.pdf.

For JSF portlets to support failover, the JSF portlet state is written to the `HttpSession` object. Therefore, all JSF state objects must be `serializable`. Even request-scoped managed beans are written into the `HttpSession` in a portal environment. Therefore, it is a best practice for all managed beans, even those that are request scoped, to be serializable.

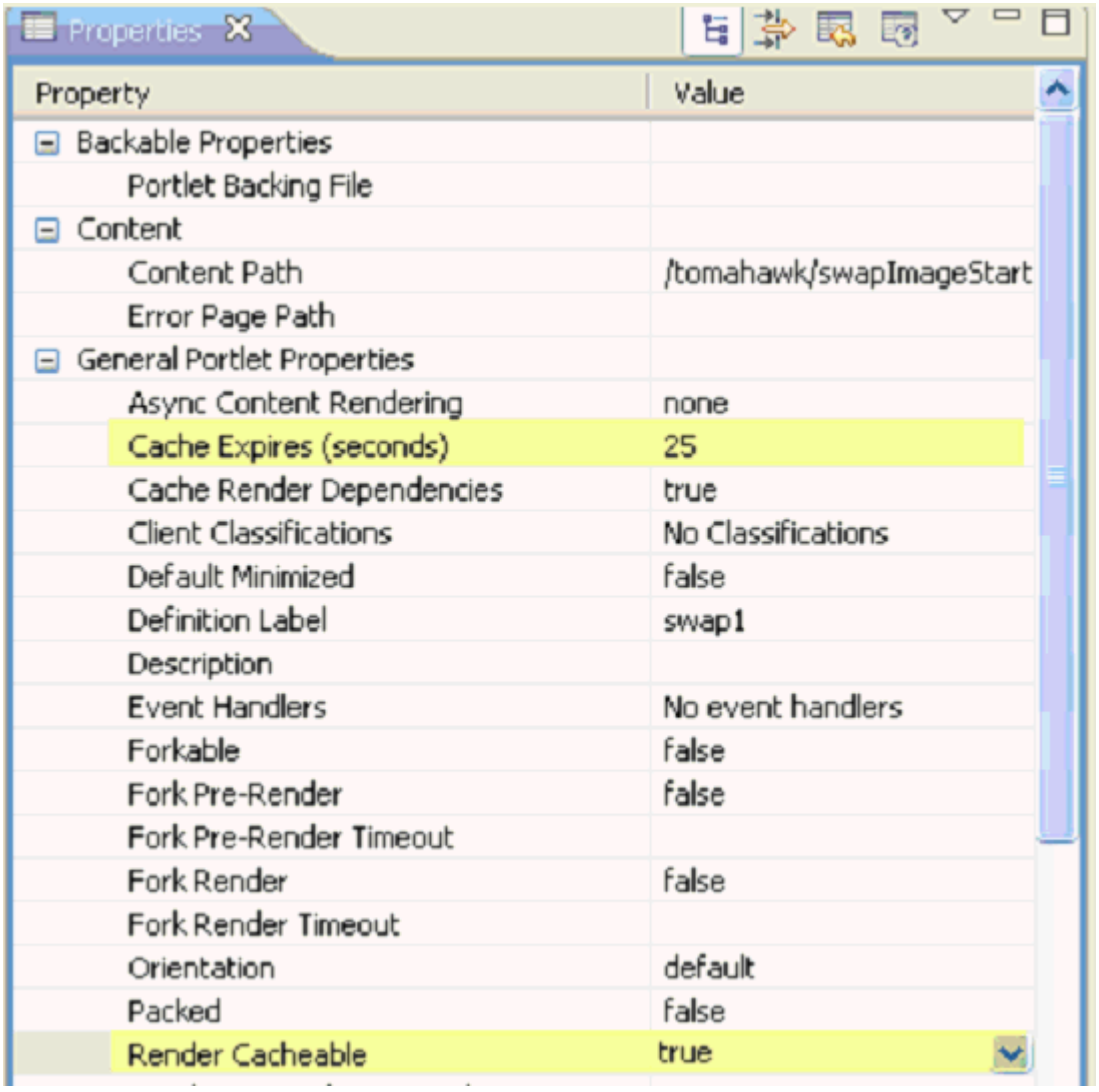
8.15.3.2 Portlet Render Caching

The WLP portal framework provides an easy solution for improving portlet rendering performance. The WLP Portlet Caching mechanism is the preferred way of handling caching, and not the JSR-286 suggested caching mechanism. When a portlet is not designed for user interaction or does not handle an interportlet communication event during the request, the portlet's markup can be served from cache. By default, a portlet's rendered markup is not cached. To enable portlet caching, set the following values in the Portlet Properties view: (see Figure 8–6).

- Set the Render Cacheable property to `true`.
- Set the Caches Expires property to the number of seconds to cache the portlet markup.

> **Note:** When a JSF portlet's markup is rendered from cache, none of the JSF life cycle methods are invoked during the request.

Figure 8–6 Portlet Cache Configuration



8.15.4 Securing JSF Portlets

This section discusses best practices for securing JSF portlets.

- Section 8.15.4.1, "Deny Direct Access to the Portlet Views"
- Section 8.15.4.2, "Session Timeouts"

8.15.4.1 Deny Direct Access to the Portlet Views

To develop JSF portlets as standalone JSF applications, you must map the Faces servlet in `web.xml`, as shown in Example 8–37.

Example 8–37 Prefix Servlet Mapping in web.xml

```
<servlet-mapping>
    <servlet-name>faces</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>faces</servlet-name>
    <url-pattern>*.faces</url-pattern>
</servlet-mapping>
```

Note that these mappings exist in `web.xml` only to support the direct access use case. Generally, in production it is not recommended to allow users to target the JSF views directly. It is better to force users to navigate to those views through the portal user interface. This can be enforced by removing the prefix and suffix mappings to the Faces servlet in `web.xml`.

8.15.4.2 Session Timeouts

It is a good practice to test the behavior of a web application for the request that immediately follows an `HttpSession` timeout. JSF portlets behave the same as traditional JSF applications during a session timeout in that the previous state of the portlet is not retained. Thus the user is taken back to the initial state of the JSF portlet just like it is in a JSF application. It is therefore important to ensure that your portal components re-render with the unauthenticated user in the session timeout use case.

8.15.5 Localizing JSF Portlets

This section discusses best practices and techniques for localizing JSF portlets prior to moving to production.

8.15.5.1 Configuring the Localization

When implementing localization, keep in mind that there are three different localization layers in the portal architecture. The first layer is of localization at the WLP portal level for books and pages. The second layer is of localization at the Java portlet level for title and description. The third layer is of localization at the JSF component level.

WLP provides portal localization for the `.book` and `.page` files. For detailed information, see the section "Localizing Titles for File-Based Books, Pages, and Portlets" in *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

The Java Portlet specification provides the localization for the title and description of the portlet. To ensure that your portlet works across different third-party portlet containers, the Java standard locale support should be used. This is accomplished by

adding <supported-locale> to the portlet's entry in the portlet.xml file. For further details, see the JSR-286 specification at http://jcp.org/aboutJava/communityprocess/final/jsr286/index.html.

In addition, JSF provides localization for the JSF components within a JSF view using a locale pluggable interface. The ViewHandler interface has a method calculateLocale() that performs the localization functionality of the JSF display components. When developing JSF portlets, it is a best practice to create a bundle for each individual portlet definition. Or, if several portlets are always to be used together, a resource bundle can be shared between a group of portlets. This enables portlets to be reused more easily across portal projects.

Another best practice is to make sure WLP, JSF, and Java portlets are all configured with the same set of locales. This ensures a consistent localized experience for a rendered page.

8.16 Third-Party Libraries

This section includes the following:

- Section 8.16.1, "Facelets"
- Section 8.16.2, "Tomahawk"

8.16.1 Facelets

WLP does not support the use of Facelets with JSF 1.2. If you require adding Facelet support, the portlets must use the WLP native Faces bridge. See the section "Using Facelets" available at http://download.oracle.com/docs/cd/E13155_01/wlp/docs103/portlets/jsf_portlet_development.html#wp1040473.

8.16.2 Tomahawk

WLP does not officially support the use of Tomahawk components, so they are not included in the WebLogic Portal installation. This also means there are no assurances that all Tomahawk tags will work flawlessly within a WebLogic Portal environment. However, you can configure these components in the portal application and test them. It is recommended to use the latest Tomahawk version for JSF 1.2.

Follow these instructions to configure it:

1. Download the Tomahawk JAR file for JSF 1.2: at least tomahawk12-1.1.10.jar or later.
2. Copy the JAR file into WEB-INF/lib.
3. In web.xml, add the following entries in the appropriate places.

Example 8–38 web.xml Settings For Tomahawk

```
<!-- Make entries that explicitly enable the portal-friendly changes
introduced with latest versions of Tomahawk ->
<context-param>
     <param-name>org.apache.myfaces.CHECK_EXTENSIONS_FILTER</param-name>
     <param-value>false</param-value>
</context-param>
<context-param>
     <param-name>
       org.apache.myfaces.DISABLE_TOMAHAWK_FACES_CONTEXT_WRAPPER
```

```
     </param-name>
     <param-value>false</param-value>
</context-param>
<!-- Configure the mechanism for bringing in resources (.js, .css). ->
<context-param>
     <param-name>org.apache.myfaces.ADD_RESOURCE_CLASS</param-name>
     <param-value>
       org.apache.myfaces.renderkit.html.util.NonBufferingAddResource
     </param-value>
</context-param>
<!-- Map the resource loading capability of Tomahawk ->
<!-- myfaces configuration -->
<filter>
     <filter-name>MyFacesExtensionsFilter</filter-name>
<filter-class>org.apache.myfaces.webapp.filter.ExtensionsFilter</filter-class>
    <init-param>
        <description>Set the size limit for uploaded files.
            Format: 10 - 10 bytes
                    10k - 10 KB
                    10m - 10 MB
                    1g - 1 GB
        </description>
        <param-name>uploadMaxFileSize</param-name>
        <param-value>20m</param-value>
    </init-param>
</filter>
<!-- extension mapping for adding <script/>, <link/>, and other resource tags to
JSF-pages  -->
<filter-mapping>
    <filter-name>MyFacesExtensionsFilter</filter-name>
    <!-- servlet-name must match the name of your javax.faces.webapp.FacesServlet
entry -->
    <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
<!-- extension mapping for serving page-independent resources (javascript,
stylesheets, images, etc.)  -->
<filter-mapping>
    <filter-name>MyFacesExtensionsFilter</filter-name>
    <url-pattern>/faces/myFacesExtensionResource/*</url-pattern>
</filter-mapping>
```

8.17 Tips for Logging, Iterative Development, and Debugging of JSF Portlets

For WebLogic Portal Web Projects, the integration of logging is pre-configured. The WebLogic Integrated Commons Logging facet is a required facet for any Portal Web Project.

Follow these steps to increase the log sensitivity level to enable Debug messages to be output to the console:

1. Log on to the WebLogic Console, usually at a URL like: `http://localhost:7001/console`.
2. Navigate to **Servers** > **AdminServer** > **Logging** tab.
3. Click the **Advanced** link to get the full set of controls.
4. In the Log file section, set the `Severity Level` to **Debug**.

5. In the Standard out section, set the **Severity Level** to **Debug**.
6. Click **Save**.
7. On the next request to a portal page that contains JSF portlets, you will see debug output on the server console window.

8.17.1 Using Iterative Development for JSF Portlets

This section discusses techniques for iterative development of JSF portlets.

- Section 8.17.1.1, "Testing Outside of the Portlet Container"
- Section 8.17.1.2, "Using Application Republish"
- Section 8.17.1.3, "HttpSession Caching"
- Section 8.17.1.4, "Handling OutOfMemory Errors"

8.17.1.1 Testing Outside of the Portlet Container

Sometimes it is desirable to test the JSF portlet view outside of the portlet container. In other words, it is possible to target the view URL directly like a standard JSF application does. If the JSF view can execute without the portal, then this is a good way to debug issues with the JSF code. Targeting the view file directly isolates the JSF container from the portlet container, so it helps in analyzing and fixing problems. See Section 8.15.4, "Securing JSF Portlets" for more details on how to map the Faces servlet in `web.xml`.

8.17.1.2 Using Application Republish

WebLogic Server automatically republishes changes to JSP files. However, if auto publishing is disabled, you must explicitly republish the changes to source code or deployment descriptors from the Server pane in Eclipse (among other ways). This process is not specific to Portal Web Projects.

8.17.1.3 HttpSession Caching

JSF caches the user's views in the `HttpSession`. Therefore, while WebLogic Server detects that a JSP has been updated, any existing session contains an outdated copy of the view. It is therefore necessary to begin a new `HttpSession` after modifying a JSF JSP. This issue is not specific to Portal Web Projects.

8.17.1.4 Handling OutOfMemory Errors

Unfortunately, certain implementations of JSF have memory leaks that occur during a web application redeploy. During long development sessions across many redeployments, the server may fail with an `OutOfMemoryException`. Therefore, you must restart the server when this exception occurs. This issue is not specific to Portal Web Projects.

8.17.2 Debugging

This section discusses how to use the Eclipse debugger to troubleshoot JSF portlets. See the Eclipse documentation for information on enabling the Eclipse debugger.

When debugging a portlet, usually the best place to start is to set breakpoints in the entire code, including managed bean methods as well as subclassed Java portlet code. But sometimes you need to look into the portal framework. This is because the JSF bridge invokes the JSF implementation, which in turn processes and renders the portlet. For this reason, you should set break points in the JSF implementation and

within the JSF bridge as well. Source code is available for the open source JSF implementations and JSR-329 bridge implementation.

This section includes the following:

- Section 8.17.2.1, "Attaching Source (Step 1)"
- Section 8.17.2.2, "Suggested JSF Framework Break Points (Step 2)"

8.17.2.1 Attaching Source (Step 1)

You can investigate resources on the web that explain how to attach source code to the JAR files in your project (for example, you can search for "eclipse attach source"). The main issue for JSF development is locating the proper source files for the JSF implementation used in your web project.

You can download the Sun reference implementation (RI) code from the Mojarra project site at `https://javaserverfaces.dev.java.net`.

8.17.2.2 Suggested JSF Framework Break Points (Step 2)

The following list provides some JSF implementation break points to get started (assumes Sun RI):

- `*com.sun.faces.lifecycle.LifecycleImpl.execute()` - The front door to all JSF processing, a good place to start.
- `*com.sun.faces.lifecycle.RestoreViewPhase.execute()` – Restores the correct view; useful if the portlet is rendering the wrong view.
- `*com.sun.faces.lifecycle.InvokeApplicationPhase.execute()`- Invokes an action; useful when diagnosing issues invoking action methods.

8.18 Appendix: JSFJavaPortletHelper

You may find it helpful to create a helper class to develop the common functionality that you can reuse in your portlets. Example 8–39 shows how you can implement a helper class. This class is a concrete code example of many of the concepts discussed in this chapter. Use at your own discretion.

Example 8–39 JSFJavaPortletHelper Class

```
package oracle.samples.wlp.jsf;
import java.security.Principal;
import java.util.ResourceBundle;
import javax.faces.context.FacesContext;
import javax.portlet.ActionRequest;
import javax.portlet.ActionResponse;
import javax.portlet.PortletContext;
import javax.portlet.PortletRequest;
import javax.portlet.PortletPreferences;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.PortletResponse;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.xml.namespace.QName;
import oracle.samples.wlp.jsf.portlets.login.LoginServlet;
import weblogic.servlet.security.ServletAuthentication;
import com.bea.netuix.servlets.controls.portlet.PortletPresentationContext;
```

```
import com.bea.netuix.servlets.controls.portlet.backing.PortletBackingContext;
import com.bea.netuix.servlets.manager.AppContext;
/**
 * A helper class with examples of many useful methods for JSF portlets. These
 * methods are expected to be called from JSF managed beans.
 * This is to aid in the development of JSF portlets.
 */
public class JSFJavaPortletHelper {
    // STANDARD CONTEXT
    /**
     * Gets the HttpServletRequest from the FacesContext.
     * Must only be called from a JSF managed bean.
     * @return an HttpServletRequest implementation
     */
    static public HttpServletRequest getServletRequest() {
        HttpServletRequest httpServletRequest = null;
        FacesContext fc = FacesContext.getCurrentInstance();
        Object obj = fc.getExternalContext().getRequest();
        //how jsr329 bridge finds the request, another way is to dispatch to a
servlet or jsp
        if (obj instanceof PortletRequest){
            PortletRequest pr = (PortletRequest) obj;
            httpServletRequest = (HttpServletRequest)
 pr.getAttribute("javax.servlet.request");
        }
        //how native bridge finds the request
        else if (obj instanceof HttpServletRequest){
            httpServletRequest = (HttpServletRequest) obj ;
        }
        return httpServletRequest;
    }
    /**
     * Gets the HttpSession from the FacesContext.
     * Must only be called from a JSF managed bean.
     *
     * @return a HttpSession implementation
     */
    static public HttpSession getSession() {
        HttpServletRequest request = getServletRequest();
        return request.getSession();
    }
    /**
     * Gets the HttpServletResponse from the FacesContext.
     * Must only be called from a JSF managed bean.
     *
     * @return a HttpServletResponse implementation
     *
     */
    static public HttpServletResponse getServletResponse() {
        HttpServletResponse httpServletResponse = null;
        FacesContext fc = FacesContext.getCurrentInstance();
        Object obj = fc.getExternalContext().getRequest();
        //how jsr329 bridge finds the response, another way is to dispatch to a
servlet or jsp
        if (obj instanceof PortletRequest){
            PortletRequest pr = (PortletRequest) obj;
            httpServletResponse = (HttpServletResponse)
pr.getAttribute("javax.servlet.response");
        }
        //how native bridge finds the request
```

```
        else {
        obj = fc.getExternalContext().getResponse();
            httpServletResponse = (HttpServletResponse) obj ;
        }
        return httpServletResponse;
    }

    /**
     * Gets the PortletResponse from the FacesContext. Only valid when using the
JSR329-bridge.
     *
     * @return PortletResponse
     *
     */
    static public PortletResponse getPortletResponse() {

    FacesContext fc = FacesContext.getCurrentInstance();
        Object obj = fc.getExternalContext().getResponse();
        if (obj instanceof PortletResponse){
        return (PortletResponse)obj;
        }
        else {
        return null;
        }
    }

    /**
     * Gets the localized resource bundle using the passed bundle name
     * Must only be called from a JSF managed bean.
     *
     * @param bundleName the String name of the bundle
     * @return the ResourceBundle containing the localized messages for the
     */
    static public ResourceBundle getBundle(String bundleName) {
        FacesContext context = FacesContext.getCurrentInstance();
        ResourceBundle bundle =
ResourceBundle.getBundle(bundleName,context.getViewRoot().getLocale());
        return bundle;
    }
    /**
     * Gets the localized message using the passed bundle name and message
     * Must only be called from a JSF managed bean.
     *
     * @param bundleName the String name of the bundle
     * @param messageKey the String key to be found in the bundle properties
     * @return the String containing the localized message
     */
    static public String getBundleMessage(String bundleName, String
            messageKey) {
        String message = "";
        ResourceBundle bundle = getBundle(bundleName);
        if (bundle != null) {
            message = bundle.getString(messageKey);
        }
        return message;
    }
    // *************** PORTAL ENVIRONMENT **************//
    //Note: Although these methods show getting the actual Servlet request from
the PortletRequest, another way to do this
    //would be to dispatch to an actual Servlet that then gets an instance of the
```

```
PortletBackingContext and
    //PortletPresentationContext.  This latter way may be more portable and
isolate your WLP specific code better.

    /**
     * Gets the PortletBackingContext object. This method will return null
     * if called during the RENDER_RESPONSE JSF lifecycle.
     * Must only be called from a JSF managed bean.
     *
     * @return the active PortletBackingContext, or null
     */
    static public PortletBackingContext getPortletBackingContext() {
        return
 PortletBackingContext.getPortletBackingContext(getServletRequest());
    }
    /**
     * Gets the PortletPresentationContext object. This method will return
     * if NOT called during the RENDER_RESPONSE JSF lifecycle.
     * Must only be called from a JSF managed bean.
     *
     * @return the active PortletPresentationContext, or null
     */
    static public PortletPresentationContext getPortletPresentationContext() {
        HttpServletRequest request = getServletRequest();
        return PortletPresentationContext.getPortletPresentationContext(request);
    }
    /**
     * Returns true if the user can make customizations
     * (preferences, add/move/remove portlets, add pages) to the portal.
     * This is based on factors such as: is the user authenticated,
     * is it a streaming portal (not a .portal file),
     * and customization is enabled in netuix-config.xml.
     * Can be called from any web application class.
     *
     * @return a boolean, true if it is possible for the user to make
customizations
     */
    static public boolean isCustomizable() {
        return AppContext.isCustomizationAllowed(getServletRequest());
    }
    // *****  AUTHENTICATION ******///
    //The following are authentication methods that are used in the Login/Logout
Example
    /**
     * Is the current user authenticated?
     * Must only be called from a JSF managed bean.
     *
     * @return true if the user is authenticated, false if not
     */
    static public boolean isAuthenticated() {
        Principal principal =
FacesContext.getCurrentInstance().getExternalContext().getUserPrincipal();
        return principal != null;
    }

    /**
     * Is the current user authenticated?
     * This checks an attribute set on the PortletRequest by the LoginServlet,
whether
     * the user is authenticated.
```

```
     *
     * @return true if the user is authenticated, false if not
     */
    static public boolean isAuthenticatedViaAttribute() {
        PortletRequest pr =
(PortletRequest)(FacesContext.getCurrentInstance().getExternalContext().getRequest
());
        Boolean authenticated= ((Boolean)
pr.getAttribute(LoginServlet.AUTHENTICATED));
        return authenticated.booleanValue();
    }
    /**
     * Get the current user's username from the container.
     * Must only be called from a JSF managed bean.
     *
     * @return the user name, null if the user is not authenticated
     */
    static public String getUsername() {
        String username = null;
        Principal principal =

FacesContext.getCurrentInstance().getExternalContext().getUserPrincipal();
        if (principal != null) {
            username = principal.getName();
        }
        return username;
    }
    /**
     * Get the current user's username for display. If the user is not
authenticated, it
     * will return the name passed as the anonymousUsername parameter.
     * DO NOT use this method
     * for anything other than display (e.g. access control, auditing, business
logic),
     * as the passed anonymous name may conflict with an actual username in the
system.
     * Must only be called from a JSF managed bean.
     *
     * @param anonymousUsername a String localized name to use for an anonymous
user, like "Guest"
     * @return the user name
     */
    static public String getUsernameForDisplay(String anonymousUsername) {
        String username = anonymousUsername;
        Principal principal =
FacesContext.getCurrentInstance().getExternalContext().getUserPrincipal();
        if (principal != null) {
            username = principal.getName();
        }
        return username;
    }
    // USER AUTHENTICATION METHODS
    /**
     * Authenticate the user with WebLogic Server
     * See <code> authenticate_generic </code> for using a serlvet to do
authentication.
     *
     * @param username the String username
     * @param password the String password as provided by the user
     * @return true if the login was successful, false if not
```

```
     */
    static public boolean authenticate(String username, String password) {
        HttpServletRequest httpServletRequest = getServletRequest();
        HttpServletResponse httpServletResponse = getServletResponse();
        //wlp specific way of authentication
        int result = ServletAuthentication.weak(username, password,
httpServletRequest, httpServletResponse);

       //redirect the page, so any customization for this user is picked up after
successful login
        if (result != ServletAuthentication.FAILED_AUTHENTICATION) {
            Object obj =
 FacesContext.getCurrentInstance().getExternalContext().getResponse();
            if (obj instanceof ActionResponse){
                ActionResponse ar = (ActionResponse) obj;
                ar.setEvent(new QName("urn:com:oracle:wlp:netuix:event:portal",
"framework.redirectBeforeRender"),null);
            }
        }
        return result != ServletAuthentication.FAILED_AUTHENTICATION;
    }

    /**
     * This will authenticate the user putting the WLS specific authentication
process in a servlet.
     * Thus isolating container specific code better.
     * It calls to a LoginManager servlet to do the authentication.
     * If the environment is not using the standard JSR329-bridge, it will default
to WLP specific authentication.
     * If authentication is successful, it will then set an event using JSR286
style eventing, to request a redirect before render
     * so that any personal specific components will be re-rendered.
     * @param username of user to authenticate
     * @param password of user to authenticate
     * @return true to indicate username/password has been authenticated, false if
it failed.
     *
     */
    static public boolean authenticate_generic(String username, String password) {
        HttpServletRequest httpServletRequest = null;
        HttpServletResponse httpServletResponse = null;
        FacesContext fc = FacesContext.getCurrentInstance();
        Object obj = fc.getExternalContext().getContext();
        //how jsr329 bridge finds the request
        if (obj instanceof PortletContext){
            PortletContext pc = (PortletContext) obj;
            PortletRequestDispatcher prdispatcher =
pc.getNamedDispatcher("LoginServlet");
            try{

prdispatcher.include((PortletRequest)fc.getExternalContext().getRequest(),
                    (PortletResponse)fc.getExternalContext().getResponse());
            }
            catch(Exception e) {
                //MODIFY :  user proper logging methods
                System.out.println("Exception on dispatch:" + e.toString() );
                e.printStackTrace();
                return false;
            }
        }
```

```
        //how native bridge finds the request & response
        else if (obj instanceof ServletContext){
            httpServletRequest = (HttpServletRequest)
fc.getExternalContext().getRequest() ;
            httpServletResponse = (HttpServletResponse)
 fc.getExternalContext().getResponse();
            //wlp specific way of authentication
            ServletAuthentication.weak(username, password, httpServletRequest,
httpServletResponse);
        }
       //redirect the page, so any customization for this user is picked up after
successful login
        if (isAuthenticated()) {
            obj = fc.getExternalContext().getResponse();
            ActionResponse ar = (ActionResponse) obj;
            ar.setEvent(new QName("urn:com:oracle:wlp:netuix:event:portal",
"framework.redirectBeforeRender"),null);
            return true;
        }
        else {
            return false;
        }

    }

    /**
     * Logout the user invalidating the session object directly and then send
event to redirect.
     * See <code> logout_generic </code> for using a serlvet to do
un-authentication.
     * Call this if using the jsr329 bridge, otherwise use a backing file
     */
    static public boolean logout() {
        FacesContext fc = FacesContext.getCurrentInstance();
        Object obj = fc.getExternalContext().getResponse();
        if (obj instanceof ActionResponse){
            HttpSession session = getSession();
            session.invalidate();
            ActionResponse ar = (ActionResponse) obj;
            ar.setEvent(new QName("urn:com:oracle:wlp:netuix:event:portal",
"framework.redirectBeforeRender"),null);
            return true;
        }
        else {
            return false;
        }

    }

    /**
     * Logout the user invalidating the session object through the use of a
servlet.
     * Send event to redirect after un-authenticating the user.
     * Call this if using the jsr329 bridge, otherwise use a backing file
     *
     **/
    static public boolean logout_generic() {
        FacesContext fc = FacesContext.getCurrentInstance();
        Object obj = fc.getExternalContext().getContext();
        //how jsr329 bridge finds the request, wlp specific implementation
```

```
        if (obj instanceof PortletContext){
            PortletContext pc = (PortletContext) obj;
            PortletRequestDispatcher prdispatcher =
pc.getNamedDispatcher("LoginServlet");
            try{

prdispatcher.include((ActionRequest)fc.getExternalContext().getRequest(),
                    (ActionResponse)fc.getExternalContext().getResponse());
            }
            catch(Exception e) {
                //MODIFY :  user proper logging methods
                System.out.println("Exception on dispatch:" + e.toString() );
                e.printStackTrace();
                return false ;
            }
        } else {
            return false;
        }
        //check the attribute the servlet set
        boolean authorized = isAuthenticatedViaAttribute();
        if (!authorized) { //logout was successful
            obj = fc.getExternalContext().getResponse();
            ActionResponse ar = (ActionResponse) obj;
            ar.setEvent(new QName("urn:com:oracle:wlp:netuix:event:portal",
"framework.redirectBeforeRender"),null);
        }
        return authorized;
    }

    // NAMESPACES AND LABELS
    /**
     * Gets the current portlet's instance label.
     * Must only be called from a JSF managed bean.
     *
     * @return the String instance label
     */
    static public String getInstanceLabel() {
        return getInstanceLabel(getServletRequest());
    }
    /**
     * Gets the current portlet's instance label.
     * Can be called from any web application class.
     *
     * @param the HttpServletRequest object
     * @return the String instance label
     */
   static public String getInstanceLabel(HttpServletRequest request) {
        String label = "_global";
        PortletBackingContext pbc =
PortletBackingContext.getPortletBackingContext(request);
        if (pbc != null) {
            label = pbc.getInstanceLabel();
        }
        else {
            PortletPresentationContext ppc =
PortletPresentationContext.getPortletPresentationContext(request);
            if (ppc != null) {
                label = ppc.getInstanceLabel();
            }
        }
```

```
        return label;
    }
    /**
     * Gets the current portlet's definition label.
     * Must only be called from a JSF managed bean.
     *
     * @return the String definition label
     */
    static public String getDefinitionLabel() {
        return getDefinitionLabel(getServletRequest());
    }
    /**
     * Gets the current portlet's definition label.
     * Can be called from any web application class.
     *
     * @param the HttpServletRequest object
     * @return the String definition label
     */
    static public String getDefinitionLabel(HttpServletRequest request) { String
label = "_global";
    PortletBackingContext pbc =
PortletBackingContext.getPortletBackingContext(request);
    if (pbc != null) {
        label = pbc.getDefinitionLabel();
    }
    else {
        PortletPresentationContext ppc =
            PortletPresentationContext.getPortletPresentationContext(request);
        if (ppc != null) {
            label = ppc.getDefinitionLabel();
        }
    }
    return label;
    }
    // ****   PORTLET PREFERENCE METHODS  **** //
    /**
     * Gets an instantiated preferences object for the portlet;
     * it must be obtained once per request.
     * Must only be called from a JSF managed bean.
     * Only works with JSR-329 bridge
     *
     * @return the PortletPreferences object for the request
     */
    static public PortletPreferences getPreferencesObject() {
    FacesContext fc = FacesContext.getCurrentInstance();
        PortletRequest pr = (PortletRequest) fc.getExternalContext().getRequest();
        return pr.getPreferences();
    }
    /**
     * Gets the single value preference.
     *
     * @param name the String name of the preference
     * @param value the String default value to use if the preference isn't
     * @return the String value
     */
    static public String getPreference(PortletPreferences prefs, String name,
            String value) {
        if (prefs != null) {
            value = prefs.getValue(name, value);
        }
```

```
        return value;
    }
    /**
     * Sets a single value preference into the preferences object.
     * storePreferences() must be called subsequently to persist the change.
     *
     * @param name the String name of the preference
     * @param value the String value of the preference
     */
    static public void setPreference(PortletPreferences prefs, String name, String
value) {
        if (prefs != null) {
            try {
                prefs.setValue(name, value);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    /**
     * After setting updated values into the preferences object, call this
     * method so they can be stored in the persistent store in a single atomic
     * operation.
     *
     * @param prefs the PortletPreferences to be persisted
     * @return a boolean, true if the store succeeded
     */
    static public boolean storePreferences(PortletPreferences prefs) {
        if (!isCustomizable()) {
            return false;
        }
        try {
            prefs.store();
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
}
```

9 Developing Portlets

This chapter discusses features for developing and configuring portlets. This chapter contains the following sections:

9.1 Portlet Properties

Portlet properties are named attributes of the portlet that uniquely identify it and define its characteristics. Some properties—such as title, definition label, and content URI—are required; many optional properties allow you to enable specific functions for the portlet such as scrolling, presentation properties, pre-processing (such as for authorization) and multi-threaded rendering. The specific properties that you use for a portlet vary depending on your expected use for that portlet.

During the development phase of the portal life cycle, you generally edit portlet properties using the Oracle Enterprise Pack for Eclipse interface; this section describes properties that you can edit using Oracle Enterprise Pack for Eclipse.

During staging and production phases, you typically use the WebLogic Portal Administration Console to edit portlet properties; only a subset of properties are editable at that point. For instructions on editing portlet properties from the WebLogic Portal Administration Console, refer to Section 17.2.2, "Modifying Library Portlet Properties" and Section 17.2.3, "Modifying Desktop Portlet Properties."

For a detailed description of all portlet properties, refer to Section 9.1.3, "Portlet Properties in the Portal Properties View" and Section 9.1.4, "Portlet Properties in the Portlet Properties View."

This section contains the following topics:

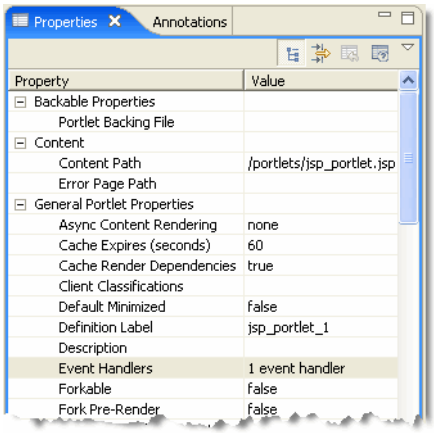
9.1.1 Editing Portlet Properties

To edit portlet properties, follow these steps:

1. Navigate to the location of the portlet whose properties you want to edit, and double-click the `.portlet` file to open it in the workbench editor.
2. Click the border of the desired portlet component to display the properties for that component in the Properties view.

   The displayed properties vary according to the active area that you select. If you click the outer border, properties for the entire portlet appear; if you click the inner border, properties for the content of the portlet appear, and so on.
3. Navigate to the Properties view to view the current values for the portlet properties. Figure 9–1 shows a segment of a JSP portlet's Properties view:

Figure 9–1 Editing Portlet Properties - JSP Portlet Properties View Example



4. Double-click the field that you want to change.

   If you click on a property field, a description of that field displays in the status bar.

   Values for some portlet properties are not editable after you create the portlet.

   In some cases, from the property field you can view associated information pertaining to that portlet property; for example, the Java portlet Class Name property contains a read-only value with an **Open** button to view the associated Java file. For more information about options available in the Properties view, refer to Section 9.1.2, "Tips for Using the Properties View."

9.1.2 Tips for Using the Properties View

The behavior of the Properties view varies depending on the type of field you are editing. The following tips might help you as you manipulate the content of the data fields in the Properties view.

- If a file is associated with a portlet property, the Properties view includes an **Open** button in addition to a **Browse** button; you can click **Open** to display the appropriate Eclipse editor/view for the file type.
- If you want to edit the XML source for a portlet, you can right-click the `.portlet` file in the Package Explorer view and choose **Edit with** > **XML Editor** to open the file using the basic XML editor that Eclipse provides.

> **Caution:** The Eclipse XML editor has limited validation capabilities. Oracle recommends the use of a robust validation tool to ensure that your hand-edited XML is valid.

- The book, page, and portlet actions in the palette display properties in the Properties view when you select them in the palette. The cell editor for the content file property is read only, and includes an **Open** button; clicking **Open** displays the Eclipse editor/view for the applicable file type.
- For page flow portlets, a property editor is available for page flow content paths when displaying a page flow portlet in the editor. The property editor is a dialog cell editor that allows you to type in the URI of the page flow directly, or you can click the ellipses button to launch the page flow class picker dialog. If you open the dialog, the page flow class name is converted to a URI when you leave the dialog. WebLogic Portal stores the URI in the `.portlet` file when you save the portlet. The property editor validates the page flow URI specified and warns you if you choose a URI that has no corresponding page flow class. You can choose to proceed anyway and store an invalid URI; you should create a valid class later so that the portlet works correctly.
- For page flow portlets, while in the portlet editor you can double-click the portlet content view to launch the corresponding Java element specified in the portlet content path. This consists of the page flow source if the source is available in the project or attached to the JAR containing the class. If the source cannot be located, then the disassembled class browser is displayed showing the contents of the class.
- Due to a limitation in Eclipse, some long property descriptions are truncated in the Status bar. To display the entire description, while the property is highlighted click the Show Property Description button in the menu. A popup window displays the full text of the property's description. Click outside the window to close it.

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

9.1.3 Portlet Properties in the Portal Properties View

The properties described in this section are contained within the `.portal` file and are editable using Oracle Enterprise Pack for Eclipse. The values you enter here override the corresponding value in the `.portlet` file, if a value exists there.

To display the portlet properties that display in the Properties view for a portal, follow these steps:

Note: These steps assume that you have an existing portal that contains portlets.

1. Double-click the `.portal` file of the portal for which you want to view portlet instance properties.

   A WYSIWYG view of the portal appears in the editor.

2. Click a portlet to highlight it.

   An orange border appears around the outside edge of the portlet.

The Properties view displays the properties of the portlet instance; Figure 9–2 shows an example.

Figure 9–2 Portlet Instance Properties in the Portal Properties View

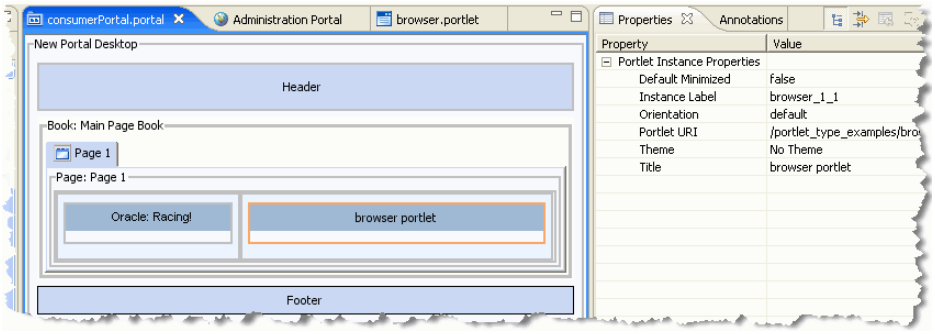


Table 9–1 describes these properties and their values.

Table 9–1 Portlet Instance Properties in the Properties View

Property	Value
Default Minimized	Optional. Select `true` for the portlet to be minimized when it is rendered. The default value is `false`. Change the value for this property only if you want to override the default value provided by the `.portlet` file.
Instance Label	Required. A single portlet, represented by a `.portlet` file, can be used multiple times in a portal. Each use of that portlet is a portlet instance, and each portlet instance must have a unique ID, or Instance Label. A default value is entered automatically, but you can change the value. Instance labels help WebLogic Portal manage the runtime state of multiple instances of portlets independently of each other on the server. WebLogic Portal also uses instance labels during URL rewriting and scoping of various HTML controls such as names of forms, and ID attributes.
Orientation	Optional. Hint to the skeleton to position the portlet title bar on the top, bottom, left, or right side of the portlet. You must build your own skeleton to support this property. The allowable values are: default, top=0, left,=1 right=2, bottom=3. Enter a value for this property only if you want to override the orientation specified in the `.portlet` file. Selecting `default` removes the orientation attribute from the portlet, book, and/or portlet instance; use this value if you want to revert to the framework default setting for this attribute.

Table 9–1 (Cont.) Portlet Instance Properties in the Properties View

Property	Value
Portlet URI	Required. The path (relative to the project) of the parent `.portlet` file. For example, if the file is stored in `Project\myportlets\my.portlet`, the Portlet URI is `/myportlets/my.portlet`.
Theme	Optional. Select a theme to give the portlet a different Look & Feel than the rest of the desktop.
Title	Enter a title if you want to override the default title specified in the `.portlet` file. The title is used in the portlet title bar.

9.1.4 Portlet Properties in the Portlet Properties View

The properties described in this section are contained within the `.portlet` file and are editable using Oracle Enterprise Pack for Eclipse. The values you enter here override the corresponding value in the `.portlet` file, if a value exists there.

When you select the *outer border* of a portlet in the portlet editor, a related set of properties appears in the Properties view. The displayed properties vary according to the type of portlet that you are viewing. Figure 9–3 shows a portion of the Properties view for a portlet.

Figure 9–3 Properties View Example Showing Portlet Properties

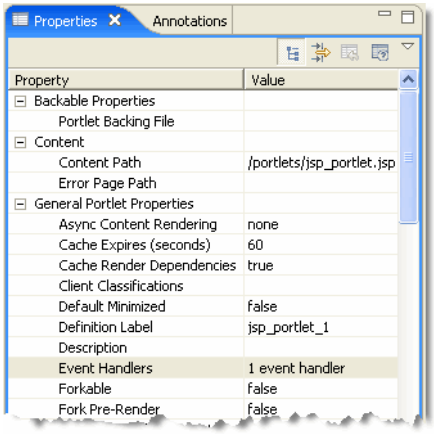


Table 9–2 describes these properties and their values.

Table 9–2 Properties in the Portlet Properties View

Property	Value
Portlet Backing File	Optional. If you want to use a class for preprocessing (for example, authentication) prior to rendering the portlet, enter the fully qualified name of that class. That class should implement the interface com.bea.netuix.servlets.controls.content.backing.JspBacking or extend com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking. From the data field you can choose to browse to a class or open the currently displayed class.
Singleton Backing Instance	Optional. Performance setting for books, pages, and portlets that use backing files. When Singleton Backing Instance is set to `true`, an instance of a backing file is shared among all books, pages, or portlets that request the backing file. You must synchronize any instance variables that are not thread safe. When Singleton Backing Instance is set to `false`, a new instance of a backing file is created each time the backing file is requested by a different book, page, or portlet.
Content Path	Required. The path (relative to the project) to the file/class to be used for the portlet's content. From the data field you can choose to browse to a file (or class for page flow portlets) or open the currently displayed file/class. For example, if the content is stored in `Project/myportlets/my.jsp`, the Content URI is `/myportlets/my.jsp`.
Error Page Path	Optional. The path (relative to the project) to the JSP or HTML file to be used for the portlet's error message if the main content cannot be rendered. For example, if the error page is stored in `Project/myportlets/error.jsp`, the Content URI is `/myportlets/error.jsp`.
Async Content Rendering	Allows you to specify whether to use asynchronous content for a given portlet and the implementation to use. An editable dropdown menu provides the selections `none`, `ajax`, `iframe`, and `iframe_unwrapped`. Portlet files that do not contain the `asyncContent` attribute appear with the initial value `none` displayed. For more information, refer to Section 10.5, "Asynchronous Portlet Content Rendering." **Note:** The `iframe_unwrapped` value is used for interoperability with WebCenter 10g ADF Faces portlets. You must use the `iframe_unwrapped` value if you are consuming (through WSRP) a WebCenter 10g ADF Faces portlet in a WebLogic Portal. Using this value prevents potential rendering problems by wrapping the ADF Faces portlet in an IFrame, while explicitly excluding WebLogic Portal-specific markup from rendering within the IFrame. For more information on WSRP interoperability between WebCenter and WebLogic Portal, see the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*. **Tip:** You can also enable asynchronous rendering for an entire portal desktop by setting a portal property in either Oracle Enterprise Pack for Eclipse or the WebLogic Portal Administration Console. For more information on asynchronous desktop rendering, see the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Cache Expires (seconds)	Optional. When the `Render Cacheable` property is set to `true`, enter the number of seconds after which the portlet cache expires.
Cache Render Dependencies	This instance-scoped boolean property appears in the Properties view whenever a window portlet or proxy portlet is loaded, allowing render dependencies to be cached. See also Section 9.5.1, "Portlet Dependencies." The value defaults to `true` if the attribute is not already included in the `.portlet` file. The value is read-only for proxy portlets and editable for all other portlet types. For proxy portlets, the value is initialized from the producer whenever a proxy portlet is generated from the portlet wizard. This property does not affect requests targeted to the portlet.

Table 9–2 (Cont.) Properties in the Portlet Properties View

Property	Value
Client Classifications	Optional. Select the multichannel devices on which the portlet can be viewed. The list of displayed devices is obtained from the file `Project_Path\WEB-INF\client-classifications.xml`. You must create this file to map clients to classifications in your portal web project. For more information about this task, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*. In the Manage Portlet Classifications dialog: Select whether you want to enable or disable classifications for the portlet. Move the client classifications you want to enable or disable from the Available Classifications to the Selected Classifications. Click **OK**. When you disable classifications for a portlet, the portlet is automatically enabled for the classifications that you did not select for disabling.
Default Minimized	Required. Select `true` if you want the portlet to be minimized when it is rendered. The default value is `false`.
Definition Label	Required. Each portlet must have a unique value within the web project. For Java portlets, you type the desired value when creating the portlet; for the remaining portlet types, a value is generated automatically when you create the portlet. Definition labels can be used to navigate to portlets. Also, components must have Definition Labels for entitlements and delegated administration. As a best practice, you should edit this value in Oracle Enterprise Pack for Eclipse to create a meaningful value. This is especially true when offering portlets remotely, as it makes it easier to identify them from the producer list.
Description	Optional. A short text description of the portlet. The description is displayed in the Administration Console and Visitor Tools areas, and is sent from a WSRP producer where applicable.
Event Handlers	Optional. Use this value to configure interportlet communication using portlet events. The default is `No event handlers`. To select or add an event handler, click **Browse** in the Properties view. You an also click the Event Handlers link in the portlet editor. Both of these methods bring up the Portlet Event Handlers dialog box. For details, see Chapter 12, "Configuring Local Interportlet Communication." For information on event handling in remote portlets, see the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.
Forkable	For details on this property, refer to Section 10.4, "Portlet Forking."
Fork Pre-Render	For details on this property, refer to Section 10.4, "Portlet Forking."
Fork Pre-RenderTimeout (seconds)	For details on this property, refer to Section 10.4, "Portlet Forking."
Fork Render	For details on this property, refer to Section 10.4, "Portlet Forking."
Fork Render Timeout (seconds)	For details on this property, refer to Section 10.4, "Portlet Forking."
Orientation	Optional. Hint to the skeleton to position the portlet title bar on the top, bottom, left, or right side of the portlet. You must build your own skeleton to support this property in the `.portal` file. Following are the numbers used in the `.portal` file for each orientation value: top=0, left=1, right=2, bottom=3. You can override the orientation in each instance of the portlet (in the Properties view).

Table 9–2 (Cont.) Properties in the Portlet Properties View

Property	Value
Packed	Optional. Rendering hint that can be used by the skeleton to render the portlet in either expanded or packed mode. You must build your own skeleton to support this property. When packed="false" (the default), the portlet takes up as much horizontal space as it can. When packed="true," the portlet takes up as little horizontal space as possible. From an HTML perspective, this property is most useful when the window is rendered using a table. When packed="false," the table's relative width would likely be set to "100%." When packed="true," the table width would likely remain unset.
Render Cacheable	Optional. To enhance performance, set to `true` to cache the portlet. For example, portlets that call web services perform frequent, expensive processing. Caching web service portlets greatly enhances performance. Do not set this to true if you are doing your own caching. For more information, refer to Section 10.2, "Portlet Caching."
Render Dependencies Path	Optional. Lets you specify the path to a render dependency file (also called a portlet dependency file). Render dependencies are resources that are required for rendering a portlet. For details, see Section 9.5.1, "Portlet Dependencies."
Shared Parameters	Optional. Portlet shared parameters are similar to JSR 286-based public render parameters. The primary difference is that shared public render parameters handle complex data types while shared parameters only handle String data. For details, see Section 9.3, "Using Shared Parameters."
Required User Properties Mode	For remote portlets only. Optional. Possible values are `none`, `all`, or `specified`. If the value is `specified`, then you must enter a list of property names in the field Required User Properties Names field.
Required User Properties Names	For remote portlets only. Optional. Use this field if you entered a value of `specified` in the Required User Properties Mode field; enter a comma-delimited list of property names.
Title	Required. Enter the title for the portlet's title bar. You can override this title in each instance of the portlet (in the portal editor, as described in Section 9.1.3, "Portlet Properties in the Portal Properties View").
Page Flow Action	Optional. The initial action to be executed in a page flow. If not specified, the `begin` action is used. Only available for Page Flow portlets. **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Page Flow Refresh Action	Optional. The action to be executed in the page flow when the page is refreshed but the portlet is not targeted. This is equivalent to using portlet event handlers configured on the `onRefresh` portal event to invoke the page flow action. Only available for Page Flow portlets.

Table 9–2 (Cont.) Properties in the Portlet Properties View

Property	Value
Request Attribute Persistence	Optional. Possible values are `none`, `session`, and `transient-session`. This attribute controls attribute persistence for Page Flow, JSF, and Struts portlets. The default is `session`, where request attributes populated by an action are persisted into a collection class that is placed into a session attribute so that the portal framework can safely include the forwarded JSP on subsequent requests without re-running the action. Using the value `session` can cause session memory consumption and replication that would not otherwise occur in a standalone Page Flow, JSF, or Struts application. The value `transient-session` places a serializable wrapper class around a HashMap into the session. The value `none` performs no persistence operation. JPF or Struts portlets that have the `transient-session` value applied generally have the same behavior as existing portlets; however, in failover cases, the persisted request attributes disappear on the failed-over-to server. In the failover case, you must write forward JSPs to handle this contingency gracefully by, at a minimum, not expecting any particular request attribute to be populated; ideally you should include the ability to either repopulate automatically or present the user with a link to re-run the last action to repopulate the request attributes. For non-failover cases, request attributes are persisted, providing a performance advantage for non-postback portlets identical to default `session` persistence portlets. Portlets that have the `none` value applied will never have request attributes available on refresh requests; you must write forward JSPs to assume that they will not be available. You can use this option to completely remove the framework-induced session memory loading for persisted request attributes. **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Faces Events	(Optional) Lets you add name/action pairs to a JSF portlet. The name field is simply an alias. Event handlers (and the Event Handler dialog) can simply reference this name. The action is a reference to a JSF view ID, such as `myfaces/foo.face`. Only available for JSF portlets. For more information on adding event handlers, see Section 12.4, "Portlet Event Handling."
Content Presentation Class	A CSS class that overrides any default CSS class used by the component's skeleton. For proper rendering, the class must exist in a cascading style sheet (CSS) file in the Look and Feel's selected skin, and the skin's skin.xml file must reference the CSS file. **Sample:** If you enter "my-custom-class", the rendered HTML from the default skeletons looks like this: `<div class="my-custom-class">` The properties you enter are added to the component's parent tag. This property also applies to books and pages. For more information, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Content Presentation Style	Optional. The primary uses are to allow content scrolling and content height-setting. For scrolling, enter the following attributes: overflow:auto – Enables vertical and horizontal scrolling For setting height, enter the following attribute: height:200px where 200px is any valid HTML height setting. You can also set other style properties for the content as you would using the Presentation Style property. The properties are applied to the component's content/child tag.

Table 9–2 (Cont.) Properties in the Portlet Properties View

Property	Value
Offer as Remote	Optional. Defines whether the portlet is accessible using the WSRP producer. The default is `true`, which allows the portlet to be accessed. For more information about entitling remote portlets, refer to the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.
Content Backing File	Optional. If you want to use a backing file for content prior to rendering the portlet, enter the fully qualified name of the appropriate class. That class should implement the interface com.bea.netuix.servlets.controls.content.backing.JspBacking or extend com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking.
Can Delete	Optional. If set to `true` the portlet can be deleted from a page.
Can Float	Optional. If set to `true` the portlet can be floated into a separate window. For instructions on creating a floatable Java portlet, which requires editing the weblogic-portlet.xml file, in Section 6.20, "Adding an Icon to a Java Portlet."
Can Maximize	Optional. If set to `true` the portlet can be maximized.
Can Minimize	Optional. If set to `true` the portlet can be minimized.
Edit Path	Optional. The path (relative to the project) to the portlet's edit page.
Help Path	Optional. The path (relative to the project) to the portlet's help file.
Icon Path	Optional. The path (relative to the project) to the graphic to be used in the portlet title bar. You must create a skeleton to support this property.
Content Path	Required. The path (relative to the project) to the JSP, HTML, or `.java` file to be used for portlet's mode content, such as the edit page. For example, if the content is stored in Project/myportlets/editPortlet.jsp, the Content URI is /myportlets/editPortlet.jsp. Although a Browse button appears for this property, if you want to point to a page flow you must manually enter the path to the `.java`.
Error Path	Optional. The path (relative to the project) to the JSP, HTML, or `.java` file to be used for the error message if the portlet's mode page cannot be rendered. For example, if the error page is stored in Project/myportlets/errorPortletEdit.jsp, the Content URI is /myportlets/errorPortletEdit.jsp. Although a Browse button appears for this property, if you want to point to a page flow you must manually enter the path to the `.java`.
Visible	Optional. Makes the mode icon (such as the edit icon) in the title bar or menu invisible (`false`) or visible (`true`). Set Visible to `false` when, for example, you want to provide an edit URL in a desktop header.
Name	Optional. Displayed when you select an individual mode. An optional name for the mode, such as `Edit`.
Presentation Class	This property is described in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Presentation ID	This property is described in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Presentation Style	This property is described in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Properties	Optional. A comma-delimited list of name-value pairs to associate with the object. This information can be used by skeletons to affect rendering.
Skeleton URI	This property is described in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Connection Establishment Timeout	Optional. The number of milliseconds after which this portlet will time out when establishing an initial connection with its producer.

Table 9–2 (Cont.) Properties in the Portlet Properties View

Property	Value
Connection Timeout	Optional. The number of milliseconds after which this portlet will time out when communicating with its producer, after the physical connection has been established. If not specified here, the default value contained in the file `WEB-INF/wsrp-producer-registry.xml` is used.
Group ID	Optional. This value is assigned by the producer and is not editable. Portlets with the same Group ID from the same producer can share sessions. The Group ID value is meaningful only to the producer and not manipulated by WebLogic Portal.
Invoke Render Dependencies	This boolean property allows the consumer to obtain render dependencies from the producer during the pre-render life cycle of a proxy portlet. When a portlet on a producer has a `lafDependenciesUri` value, the producer exposes the `invokeRenderDependencies` boolean in the portlet description. For more information on this attribute, refer to Section 9.5.1, "Portlet Dependencies." **Note:** Provide an absolute path for the `lafDependenciesUri` attribute, rather than a relative path. The value defaults to `false` if the attribute is not included in the `.portlet` file. The value is read-only, and is initialized from the producer whenever a proxy portlet is generated from the portlet wizard.
Portlet Handle	Required. The producer's unique identifier for the portlet that this proxy references. The value is not editable.
Producer Handle	Required. The producer's unique identifier.
Templates Stored in Session	Indicates whether the producer stores URL templates in the user's session on the producer side. This boolean is meaningful only when URL Template Processing boolean is set to `true`.
URL Template Processing	Indicates whether the producer uses URL templates to create URLs. If true, the consumer supplies URL templates. If false, the producer rewrites URLs using special rewrite tokens.
User Context Stored In Session	Required. The purpose of this value is to cut down on network traffic by sending the user's context (including the profile) only once per session. For WebLogic Portal producers it will always be `true`. For third party producers it can be `true` or `false`, depending on the response from GetServiceDescription. If it is `false`, the entire user context will be sent on every getMarkup and performBlockingInteraction request. If `true` it will be sent only once per producer session. This boolean value defaults to `false` if the attribute is not included in the `.portlet` file. The value is read-only, and is initialized from the producer whenever a proxy portlet is generated from the portlet wizard.
Listen To	(Deprecated) Allows this portlet to invoke an action when another portlet invokes the same action. This functionality has been replaced with the more complete interportlet communication mechanism. For more information on interportlet communication, refer to Chapter 12, "Configuring Local Interportlet Communication."
Struts Action	The begin action that this struts portlet should invoke on the first request to the portlet. **Note:** Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Table 9–2 (Cont.) Properties in the Portlet Properties View

Property	Value
Struts Module	The struts module that is associated with this struts portlet. A "struts module" is a means of scoping a particular set of struts actions to a group called a module, which generally maps to a single subdirectory of web resources and a separate `struts-config.xml file`.
Struts Refresh Action	Optional. The action to be performed in the struts module when the page is refreshed but the portlet itself is not targeted.
Content Url	Required. The content control takes a URI that is expected to be a URL for a standalone application or web page, and embeds the URL as portlet content.

9.2 Portlet Preferences

Portlet preferences provide the primary means of associating application data with portlets. This feature is key to personalizing portlets based on their usage. This section describes portlet preferences in detail.

After you create a portlet, you can instantiate it several times. Because you can create several instances of a portlet, it is natural to expect each instance to behave differently yet use the same code and user interface. For instance, consider a typical portlet to display a Stock Portfolio. Given a list of stock symbols, this portlet retrieves quotes from a stock quote web service periodically, and displays the quotes in the portlet window. By letting each user change the list of stock symbols and a time interval to reload the quote data, you can let each user customize this portlet.

The portlet needs to be able to store the list of stock symbols and the retrieval interval persistently, and update these values whenever a user customizes these values. In particular, the following data must be persistently managed:

- Default Values – Your portlet may specify a default list of stock symbols and a reasonable retrieval interval. These values are applicable to all usages of the portlet no matter who the user is. The user could even be anonymous.
- Customized Values – Your portlet also needs to be able to store these values when a user updates the values for a given portlet instance. Note that your portlet should also scope this data to an instance, such that other instances of this portlet are not affected by this customization.

Technically, a portlet preference is a named piece of string data. For example, a Stock Portfolio portlet could have the following portlet preferences:

- A preference with name "stockSymbols" and value "ORCL, MSFT"
- Another preference with name "refreshInterval" and value "600" (in seconds).

You can associate several such preferences with a portlet. WebLogic Portal provides the following means to manage portlet preferences:

- Specify portlet preferences during the development phase

  When you are building a portlet using the Oracle Enterprise Pack for Eclipse workbench, you can specify the names and default values of preferences for each portlet. All portlet instances derived from this portlet will, by default, assume the values specified during development.

- Let administrators modify portlet preferences

  WebLogic Portal allows portal administrators to modify preferences for a given portlet instance.This task occurs during the staging phase and uses the WebLogic Portal Administration Console.

- Let portlets access and modify preferences at request time

  At request time, your portlets can programmatically access and update preferences using a `javax.portlet.PortletPreferences` object. You can create an edit page for your portlet to let users update preferences, or you can automatically update preferences as part of your normal portlet application flow.

This section contains the following topics:

- Section 9.2.1, "Specifying Portlet Preferences"
- Section 9.2.2, "Using the Preferences API to Access or Modify Preferences"
- Section 9.2.3, "Portlet Preferences SPI"
- Section 9.2.4, "Best Practices in Using Portlet Preferences"

9.2.1 Specifying Portlet Preferences

The steps to associate preferences with a portlet depend on the type of portlet you are building. If you are using the Java Portlet API, described in Section 9.2.2.3, "Getting and Setting Preferences for Java Portlets Using the Preferences API," the steps follow those specified in the Java Portlet Specification. For other kinds of portlets, you can use the Oracle Enterprise Pack for Eclipse workbench to add preferences to a portlet.

You can also allow the administrator to create new preferences using the Administration Console. However, because the portlet developer is more likely to be aware of how portlet preferences are used by the portlet, it is generally better to create portlet preferences during the development phase.

9.2.1.1 Specifying Preferences for Java Portlets in the Deployment Descriptor

For portlets using Java Portlet API, you can specify preferences in the portlet deployment descriptor according to the specification. For all portlets in a web project, the deployment descriptor is `portlet.xml`, found in the `WEB-INF` directory of the web project. Example 9–1 provides an example.

Example 9–1 Specifying Portlet Preferences in portlet.xml with a Single Value

```
<portlet>
   <description>This portlet displays a stock portfolio.</description>
   <portlet-name>portfolioPortlet</portlet-name>
   <portlet-class>portlets.stock.PortfolioPortlet </portlet-class>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>edit</portlet-mode>
      </supports>
      <portlet-info>
         <title>My Portfolio</title>
      </portlet-info>
      <portlet-preferences>
         <preference>
            <name>stockSymbols</name>
               <value>ORCL, MSFT</value>
         </preference>
         <preference>
            <name>refreshInterval</name>
            <value>600</value>
         </preference>
   </portlet-preferences>
</portlet>
```

This snippet deploys the portfolio portlet with two preferences: a preference with name stockSymbols and value ORCL, MSFT, and another preference refreshInterval with value 600.

Instead of specifying a single value for the stockSymbols preference, you can declare each symbol as a separate value as shown in Example 9–2 below, with the value elements shown in bold.

Example 9–2 Specifying Portlet Preferences with Values Specified Separately

```
<portlet>
   <description>
      This portlet displays a stock portfolio.
   </description>
   <portlet-name>portfolioPortlet</portlet-name>
   <portlet-class>portlets.stock.PortfolioPortlet </portlet-class>
   <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>edit</portlet-mode>
   </supports>
   <portlet-info>
      <title>My Portfolio</title>
   </portlet-info>
   <portlet-preferences>
      <preference>
         <name>stockSymbols</name>
         <value>ORCL</value>
         <value>MSFT</value>
         </preference>
      <preference>
         <name>refreshInterval</name>
         <value>600</value>
      </preference>
   /portlet-preferences>
</portlet>
```

If you prefer that portlets should not be allowed to programmatically update any given preference, you can mark the preference as read-only. Example 9–3 shows an example of preventing a portlet from changing the refreshInterval.

Example 9–3 Specifying a Read-Only Portlet Preference Value

```
<portlet>
   <description>
      This portlet displays a stock portfolio.
   </description>
   <portlet-name>portfolioPortlet
   <portlet-class>portlets.stock.PortfolioPortlet
   <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>edit</portlet-mode>
   </supports>
   <portlet-info>
      <title>My Portfolio</title>
   </portlet-info>
   <portlet-preferences>
      <preference>
         <name>stockSymbols</name>
         <value>ORCL</value>
         <value>MSFT</value>
```

```
      /preference>
      <preference>
         <name>refreshInterval</name>
         <value>600</value>
         <read-only>true</read-only>
      </preference>
   </portlet-preferences>
</portlet>
```

Note that by marking a preference read-only, you are preventing the portlet from changing the current value only at request time. Portal administrators can always change the value(s) of a preference using the Administration Console.

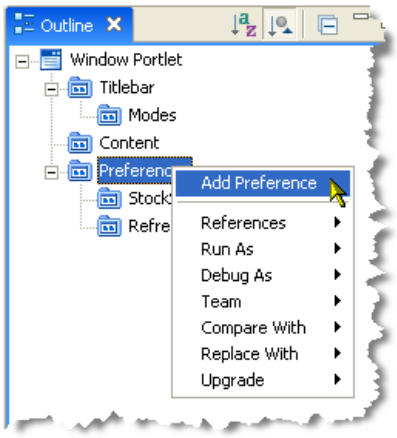
9.2.1.2 Specifying Preferences for Other Types of Portlets using Oracle Enterprise Pack for Eclipse

If you are building other kinds of portlets besides Java portlets, you can add preferences using Oracle Enterprise Pack for Eclipse.

To add a preference, follow these steps:

1. Click to select the portlet for which you want to add a preference.
2. In the Outline view for the portlet, right-click **Preferences** and in the context menu click **Add Preference**. Figure 9–4 shows an example of the preferences context menu.

Figure 9–4 Portlet Preferences Context Menu



A new preference is added to the tree hierarchy with the name **New Preference** Preference.

3. Click the new item to display its properties in the Properties view.
4. Edit the values in the Properties view. Figure 9–5 shows an example of the fields in the Properties view.

Figure 9–5 Portlet Preferences Properties View

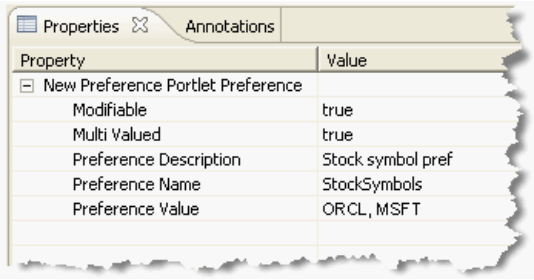


Table 9–3 describes the attributes for portlet preferences as shown in the Properties view.

Table 9–3 Portlet Preference Properties

Field	Value
Modifiable	Indicates whether the preference is read-only or can be modified by the user. The default is `true`.
Multi Valued	Indicates whether the preference can have multiple values. The default is `true`. To specify multiple values for a preference, create multiple preferences with the same name.
Description	A brief description of the preference.
Name	Name of the preference.
Value	Each preference can have one or more values. Each value is of type java.lang.String.

9.2.1.3 Configuring Portlet Preference Deployment Options

By default, changes to portlet preferences made in a `.portlet` or `portlet.xml` are not propagated to the WebLogic Portal Administration Console. For example, if you change the name of a preference in a `.portlet` file and republish the portlet, the name change does not appear in the Administration Console. Instead, the renamed preference appears in the Administration Console as a new preference. For example, if you change the name of preference **Pref_1** to **Pref_2** in a `.portlet` and republish the portlet, in the Administration Console you will not see this name change; instead, you will see two separate preferences show up: **Pref_1** and **Pref_2**.

To change this default behavior, add the following sub-element to the customization element of the `netuix-config.xml` file in your portal web application:

```
<propagate-preferences-on-deploy propagate-to-instances="true" master="file"/>
```

For example:

```
<customization>
    <enable>true</enable>
    <exclude-dir dir="/portlets_excluded" />
    <propagate-preferences-on-deploy propagate-to-instances="true" master="file"/>
</customization>
```

With this setting, changes you make to a portlet preference in a `.portal` or `portlet.xml` file will be propagated to the Administration Console. Changes made to the attributes Name, Modifiable, Multi-valued, and Value are propagated.

9.2.2 Using the Preferences API to Access or Modify Preferences

At request time, portlet preferences for a given portlet are represented as instances of the `javax.portlet.PortletPreferences` interface. This interface is part of the Java Portlet API. This interface specifies methods to access and modify portlet preferences.

9.2.2.1 Getting Preferences Using the Preferences API

Table 9–4 describes methods that allow a portlet to access its preferences.

Table 9–4 Methods that Allow a Portlet to Access its Preference Values

Method	Purpose
String getValue(String name, String default)	Use this method to get the first value of a preference.
String[] getValues(String name, String[] defaults)	Use this method to get all the values of a preference.
boolean isReadOnly(String name)	Use this method to determine whether a given preference is read-only.
Enumeration getNames()	Use this method to get an enumeration of the names of all preferences.
Map getMap()	Use this method to get a map of preferences. The keys in this map are the names of all the preferences, and the values are the same as those returned by getValues(String name, String[] defaults)

9.2.2.2 Setting Preferences Using the Preferences API

Table 9–5 describes methods that allow a portlet to change preference values.

Table 9–5 Methods that Allow a Portlet to Change Preference Values

Method	Purpose
void setValue(String name, String value)	Use this method to set the value of a preference
void setValues(String name, String[] values)	Use this method to set several values for a preference
void store()	Use this method to commit the changes made to preferences for a portlet.
void reset(String name)	Use this method to reset the value of a preference to its default, or remove the preference if there is no default

After modifying preferences by calling setValue(), setValues() and reset() methods, you must call store() explicitly to make the changes permanent; otherwise, changes will not be made permanent.

9.2.2.3 Getting and Setting Preferences for Java Portlets Using the Preferences API

For portlets written using the Java Portlet API, you can obtain an instance of `javax.portlet.PortletPreferences` object from the incoming portlet request – `javax.portlet.RenderRequest` within the `processAction()` method, or `javax.portlet.ActionRequest` within the `render()` method.

In Example 9–4, the portlet displays a form to edit the current values of portlet preferences in a JSP page included from the `doEdit()` method of the portfolio portlet.

Example 9–4 Portlet Displays a Form to Edit Preferences

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<%@ page import="javax.portlet.PortletPreferences" %>

<portlet:defineObjects/>

<%
   PortletPreferences prefs = renderRequest.getPreferences();
   String refreshInterval = prefs.getValue("refreshInterval", "600");
   String symbols = prefs.getValue("stockSymbols", "ORCL, MSFT");
%>

<form method="POST" action="">
   <table>
      <tr>
         <td>Symbols</td><td><input name="symbols"
             value="<%=symbols>"/></td>
      </tr>
      <tr>
         <td>Refresh Interval</td><td><input name="refreshInterval"
         value="<%=refreshInterval>"/></td>
      </tr>
      <tr>
         <td></td>
         <td><input type="submit" value="Submit"/></td>
      </tr>
   </table>
</form>
```

The portlet updates the preferences in its `processAction()` method, as shown in Example 9–5.

Example 9–5 Portlet Updates the Preferences in the processAction() Method

```
public class PortfolioPortlet extends GenericPortlet
{
   {
      public void doEdit(RenderRequest renderRequest, RenderResponse
      renderResponse)
      throws IOException, PortletException
      {
         ...
      }
      public void processAction(ActionRequest actionRequest, ActionResponse
      actionResponse)
      throws PortletException
      {
         String refreshInterval =
         actionRequest.getParameter("refreshInterval");
         String symbols = actionRequest.getParameter("stockSymbols");

         PortletPreferences prefs = actionRequest.getPreferences();
         prefs.setValue("refreshInterval", refreshInterval);
         prefs.setValue("stockSymbols", symbols);
         try
         {
            prefs.store();
         }
         catch(SecurityException se) {
         // Thrown when the user does not have enough privileges to store
```

```
            // preferences. Make sure that the user logged into the portal.
            ...
            }
         catch(catch(IOException ioe) {
         // There is an error storing preferences
         ...
         }
      }
}
```

During `processAction()`, this portlet uses the `javax.portlet.ActionRequest` object to obtain preferences.

9.2.2.4 Getting and Setting Portlet Preferences Using the API for Other Portlet Types

Portlet preferences can be accessed and updated from other kinds of portlets too. The main difference is in the way your portlets obtain an instance of the `javax.portlet.PortletPreferences` object.

- Before rendering, portlets can use `com.bea.netuix.servlets.controls.portlet.PortletBackingContext` to obtain portlet preferences; for example, in a page flow action, or in the `handlePostbackData()` method of the backing file associated with the portlet.
- During the render phase portlets can use `com.bea.netuix.servlets.controls.portlet.PortletPresentationContext` to obtain portlet preferences; for example, in a JSP associated with a page flow.

Both these classes provide a method getPortletPreferences(HttpServletRequest req) that takes `javax.servlet.HttpServletRequest` as an argument and return an object of type `javax.portlet.PortletPreferences`. For more information, see *Oracle Fusion Middleware Java API Reference for Oracle WebLogic Portal* .

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

9.2.2.5 JSP Tags for Getting Portlet Preferences

WebLogic Portal provides a JSP tag library for setting up portlet preferences. Table 9–6 describes the applicable JSP tags.

Table 9–6 JSP Tags for Getting Portlet Preferences

Method	Purpose
getPreference	Use this tag to get the value of a portlet preference.
getPreferences	Use this tag to get all the values of a portlet preference. This tag can also used to write multiple values to the output separated by a separator.

Table 9–6 (Cont.) JSP Tags for Getting Portlet Preferences

Method	Purpose
forEachPreference	Use this tag to iterate through all the preferences of a portlet. You can nest other tags (getPreference, getPreferences, ifModifiable and Else) inside this tag.
ifModifible	Use this tag to include the body of this tag if the given portlet preference is not read-only.
else	Use this tag in conjunction with the ifModifiable tag to include the body of this tag if the given portlet preference is read-only

For more information on the Java classes associated with these tags, refer to the *Oracle Fusion Middleware Java API Reference for Oracle WebLogic Portal*

9.2.3 Portlet Preferences SPI

In WebLogic Portal, the framework includes a default implementation that manages portlet preferences in the built-in PF_PORTLET_PREFERENCE and PF_PORTLET_PREFERENCE_VALUE database tables. If desired, you can replace this implementation with your own.

You can use the Portlet Preferences SPI to allow portal applications to manage portlet preferences outside framework-managed database tables. For example, you can store preferences along with other application data in another back-end system or a different set of database tables.

When propagating a portal, the preferences SPI participates in the propagation process. When you exporting data for the propagation, the SPI is called to obtain the preferences, and when you are importing data, the SPI is called to store the preferences.

The following sections describe how to use the Portlet Preferences SPI.

9.2.3.1 Implement the SPI

You specify the SPI using the interface `com.bea.portlet.prefs.IPreferenceAppStore`. An implementation of this class must be deployed as a EJB jar file.

Example 9–6 provides an example.

Example 9–6 Implementing the SPI Using the Interface IPreferencesAppStore

```
public interface IPreferenceAppStore extends EJBObject
{
   /**
   * Returns preferences for a portlet entity with the given uniqueId.
   *
   * The returned java.util.Map contains
   * com.bea.netuix.application.prefs.Preference
   * objects keyed against their names.</p>
   *
   * @param uniqueId unique ID
   * @return preferences
   */
   public Map getPreferences(PortletPreferencesId uniqueId) throws
   RemoteException, PreferenceAppStoreException;

   /**
```

```
   * Writes the preferences to the underlying persistence.
   *
   * This method should be implemented to be atomic. That is, the
   * implemenation should guarantee that either all preference
   * values are persisted or none at all.
   *
   * The java.util.Map argument should contain
   * com.bea.netuix.application.prefs.Preference
   * objects keyed against their names.
   *
   * @param uniqueId unique ID
   * @param preferences preferences
   */
   public void storePreferences(PortletPreferencesId uniqueId,
   Map preferences) throws RemoteException, PreferenceAppStoreException;

   /**
   * Clear all preferences for the given unique ID from the
   * underlying persistence store.
   *
   * @param uniqueIds unique IDs
   */
   public void removePreferences(PortletPreferencesId[] uniqueIds) throws
   RemoteException, PreferenceAppStoreException;
}
```

9.2.3.2 Using the SPI

To cause the framework to use a new SPI in place of the default SPI, you must update the EJB named `PreferencePersistenceManager` in the `ejb-jar.xml` file within `netuix.jar`. The value `BEA_netuix.DefaultStore` must be changed to the name of the SPI EJB as specified in its deployment descriptor (`ejb-jar.xml`). The value `com.bea.portlet.prefs.provider.DefaultStoreHome` must be changed to the home interface of the SPI implementation.

> **Caution:** To edit the ejb-jar.xml file you need to copy the J2EE library resources into your project. Keep in mind that with future updates to the WebLogic Portal product, you might have to perform manual steps in order to incorporate product changes that affect those resources.

The code segment in Example 9–7 shows the default entries, which you must change to use the SPI.

Example 9–7 Example Code Showing Default Entries that Must be Changed

```
<session>
   <ejb-name>PreferencePersistenceManager</ejb-name>
   <home>com.bea.portlet.prefs.PreferencePersistenceManagerHome</home>
   <remote>com.bea.portlet.prefs.PreferencePersistenceManager</remote>
   <ejb-class>com.bea.portlet.prefs.PreferencePersistenceManagerImpl
   </ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
   <env-entry>
      <env-entry-name>prefs-spi-jndi-name</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
```

```
      <env-entry-value>BEA_netuix.DefaultStore</env-entry-value>
   </env-entry>
   <env-entry>
      <env-entry-name>prefs-spi-home-class-name</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>com.bea.portlet.prefs.provider.DefaultStoreHome
      </env-entry-value>
   </env-entry>
<!-- Snip -->
</session>
```

9.2.4 Best Practices in Using Portlet Preferences

This section discusses best practices in using portlet preferences.

9.2.4.1 Desktop Testing of Portlet Preferences

In order to view and test the preferences that you have created, you must use a desktop view from the WebLogic Portal Administration Console rather than Oracle Enterprise Pack for Eclipse's **Open on Server** view.

Portlets accessed from `.portal` files cannot store preferences. If you update a preference using a `.portal` file, your portlet encounters a `java.lang.UnsupportedOperationException` error.

9.2.4.2 Users Must be Authenticated

You must provide a means for users to log in before they can update preferences; users who are updating portlet preferences must first be authenticated. If an anonymous user attempts to update a portlet, a `java.lang.SecurityException` error occurs.

Note that portlets can always get portlet preferences whether or not the user is anonymous or whether the portlet is accessed via a `.portal` file.

9.2.4.3 Do Not Store Arbitrary Data as Preferences

It is tempting to store arbitrary application data as portlet preferences. For example, if you have a portlet that allows users to upload and store documents on the server, it might seem appropriate to store those documents as portlet preferences. This is not a good practice. The purpose of portlet preferences is to associate some *properties* for a portlet instance without having to be aware of any implementation-specific portlet instance IDs. These properties allow customization of the portlet's behavior. The underlying implementation of portlet preferences is not designed for storing arbitrary application data.

The following steps outline an alternative implementation that can meet the needs of the portlet:

Perform setup steps:

1. Add a preference to your portlet. This preference acts as the primary key to your portlet's application data. Assign a default value for this preference.
2. Create tables in your database to store application data with the value of the preference as the primary key.

Set up preferences in your portlet:

1. When you want to associate application data with the current portlet instance, check the value of the preference. If the value is the default, generate a new value

(for example, using a sequence number generator), and set this as the value of the preference, and store the preference.

2. If the value of the preference is not the default, you do not need to generate a new value.
3. Store your application data using the value of the preference as the primary key.

This procedure ensures that your application data is always scoped to portlet instances.

9.2.4.4 Do Not Use Instance IDs Instead of Preferences

The portal framework maintains instance identity using internally generated instance IDs. Portlets can access their instance IDs using `getInstanceId()` methods on `com.bea.netuix.servlets.controls.portlet.PortletPresentationContext` and `com.bea.netuix.servlets.controls.portlet.PortletBackingContext`.

Storing data directly in the database using portlet instance IDs does not work, for the following reasons:

- The portal framework generates instance IDs, and portlets have no control over when and how those instance IDs are generated.
- Instance IDs might change at any time without the portlet's knowledge. For example, as the user or administrator customizes a desktop using Visitor Tools or the Administration Console, the framework can create new instances or change the instance ID of a portlet. If the instance ID changes, your portlet cannot load the data from your database; the primary key has changed without your portlet being aware of it.

9.3 Using Shared Parameters

Shared parameters permit portlets to share parameter values with other portlets, allowing a simple form of interportlet communication. For instance, if portlet A and portlet B are both configured to use a particular shared parameter, any changes in the parameter's value made by portlet A will be seen by portlet B.

Note: Shared parameters provide the same functionality as, and interoperability with, JSR 286 portlet public render parameters (see also Section 6.8, "Public Render Parameters"). Shared parameters have the same semantics as JSR 286 public render parameters.

Shared parameters can only contain string values. Like HTTP parameters, shared parameter can have zero or more values. A shared parameter value remains unchanged until the portlet (or another portlet) explicitly changes it. The values of shared parameters can be shared within a single portlet application or across multiple portlet applications. For instance, if Portlet A sets a shared parameter, Portlet B can read the value of that parameter, if both portlets are properly configured.

9.3.1 Setting Shared Parameters

You can set shared parameter values during the handlePostback and event-handling portlet life cycles. You can also set shared parameters on a portlet postback URL.

To set the value of a shared parameter, the portlet must use the PortletBackingContext.setSharedParameterValue() or

PortletBackingContext.setSharedParameterValues() methods during handlePostback or event-handling, or you can set a value for the parameter on any postback URL.

If the value is set using the PortletBackingContext methods, the change in value will take effect immediately, and all other portlets using the same shared parameter will see the new value. If set on a postback URL, the new value will take effect when the URL is clicked on, and all portlets sharing that parameter will see the new value.

9.3.2 Accessing Shared Parameters

You can access (read) shared parameter values during the handlePostback, event handling, preRender, and render portlet life cycles.

The current values of shared parameters can be read using the getSharedParameterValue() and getSharedParameterValues() methods of the PortletBackingContext class during handlePostback and event-handling. The same methods can be called on the PortletPresentationContext class during preRender and render portlet life cycle phases. Using these methods will ensure that the most current value of the shared parameter is returned.

If a shared parameter is set on a postback URL, the portlet will be able to access the value of the parameter that was set on the URL using the HttpServletRequest.getParameter() method, but this may not be the most current value of the parameter.

For example, assume Portlet A and Portlet B are both configured to have a shared parameter X. Portlet A sets a value of "1" for X on one of its postback URLs, which then gets clicked on. Portlet B has a backing file with a handlePostbackData() method, which gets called and reads the value of X as 1, then sets the value of X to "2". When rendering, in Portlet A a call to HttpServletRequest.getParameter("X") will return "1" (as this is the value for parameter "X" in the request), but a call to the method PortletPresentationContext.getSharedParameterValue("X") will return "2" (as this is the current value for the shared parameter).

9.3.3 Persistence of Shared Parameters

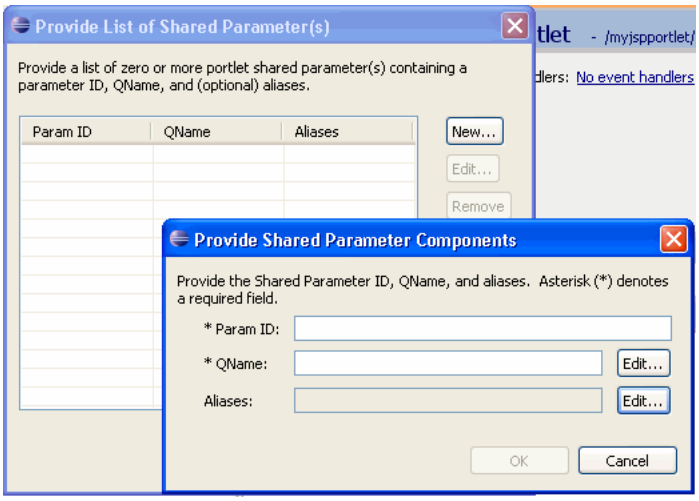
Unlike "normal" portlet parameters, shared parameter values are retained until they are explicitly set to a new value. For example, for a "normal" portlet parameter Y, if the portlet doesn't include a value for Y in a portlet URL, when the URL is clicked on, the portlet will see no value for Y. If the portlet sets a value for a shared parameter X in its backing file during a handlePostback operation, the value for X will remain accessible and constant no matter how many interactions happen with the portlet, until the portlet (or another portlet) explicitly changes the value of the shared parameter.

9.3.4 Creating Shared Parameters

To create a shared parameter:

1. In the Portlet Properties view, select the **Shared Parameter**s button.
2. In the Provide List of Shared Parameter(s) dialog, select **New**. The Provide Shared Parameter Components dialog appears, as shown in Figure 9–6.

Figure 9–6 Provide Shared Parameter Components Dialog



3. In the Provide Shared Parameter Components dialog, enter the following values:
   - **Param ID** – The identifier is a string that identifies the shared parameter within the `.portlet` file. This identifier is the name of the parameter that this particular portlet will use to read and set the shared parameter's value. Other portlets may access the same shared parameter using a different name (identifier) as long as the QName is identical.
   - **QName** – The QName, or qualified name, uniquely identifies the shared parameter. The QName consists of a required local part and a namespace URI. If you do not provide the namespace URI, the default namespace URI for the portal application is used. Click **Edit** to bring up a dialog for providing QName components. For information on QNames, see Section 12.12, "About QNames and Aliases."
   - **Aliases** – You can optionally specify one or more (comma separated) alias names. For more information about aliases, see Section 12.12, "About QNames and Aliases."
4. Click **OK** in both dialogs.

Example 9–8 illustrates how to get and set shared parameters in a JSP portlet's backing file.

Example 9–8 Getting and Setting Shared Parameter Values

```
public boolean handlePostbackData(HttpServletRequest request, HttpServletResponse response)
    {
        PortletBackingContext ctx = PortletBackingContext.getPortletBackingContext(request);
        // Read the shared parameter configured with an identifier of "distanceKM"
        String kilometers = ctx.getSharedParameterValue("distanceKM");
        int km = Integer.parseInt(kilometers);

        // Set a shared parameter for the distance in meters
        ctx.setSharedParameterValue("distanceMeters", Integer.toString(km * 1000));

        return(true);
    }
```

9.4 Backing Files

The most common means of influencing portlet behavior within the control life cycle is to use a portlet backing file. A portlet backing file is a Java class that can contain methods corresponding to portal control life cycle stages, such as init() and preRender(). A portlet's backing context, an abstraction of the portlet control itself, can be used to query and alter the portlet's characteristics. For example, in the init() life cycle method, a request parameter might be evaluated, and depending on the parameter's value, the portlet backing context can be used to specify whether the portlet is visible or hidden. For more information about backing contexts, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Backing files can be attached to portlets either by using Oracle Enterprise Pack for Eclipse or by referencing them directly in a `.portlet` file.

Backing files are simple Java classes that implement the `com.bea.netuix.servlets.controls.content.backing.JspBacking` interface or extend the `com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking interface` abstract class. The methods on the interface are callbacks corresponding to the control's lifecycle methods (refer to Section 9.4.1, "How Backing Files are Executed") and are invoked at the same time the control's life cycle methods are invoked.

The following portal controls support backing files:

- Desktops
- Books
- Pages
- Portlets
- JspContent controls (generic view controls)

The interportlet communication example in Chapter 12, "Configuring Local Interportlet Communication" uses backing files.

This section contains the following topics:

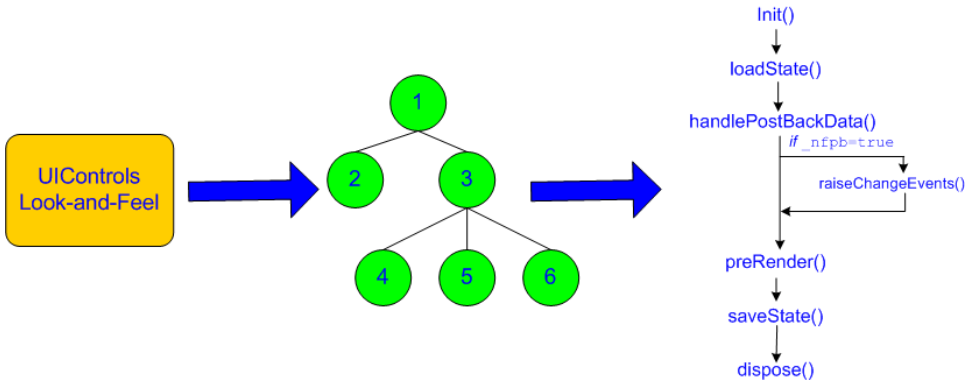
9.4.1 How Backing Files are Executed

All backing files are executed before and after the portal control is called. In its life cycle, each backing file calls these methods:

- `init()`
- `handlePostBackData()`
- `preRender()`
- `dispose()`

Figure 9–7 illustrates the life cycle of a backing file.

Figure 9–7 Backing File Life Cycle



On every request, the following sequence occurs:

> **Note:** In the following steps, the methods are called unless items on inactive pages have been "optimized away" if tree optimization is enabled. For example, if tree optimization is enabled and items on an inactive page are not included on the resulting partial control tree, then the method is not called.

1. All `init()` methods are called on all backing files in depth-first order (that is, in the order they appear in the tree). This method is called whether or not the control (the portal, page, book, or desktop) is on an active page.
2. If the `_nfpb` parameter is set to true, all `handlePostbackData()` methods are called.
   - If the backing file's `handlePostbackData()` method returns true, the `raiseChangeEvents()` method is called. This method causes events to fire, which is necessary if the backing file tries to make any state or mode changes.

     Tip: You can use the method AbstractJspBacking.isRequestTargeted(request) to determine if a request is for a particular portlet.
3. All `preRender()` methods are called for all portal framework controls on an active (visible) page.
4. The controls are called and rendered on the active page.
5. The `dispose()` method is called on each backing file.

9.4.2 Thread Safety and Backing Files

A new instance of a backing file is created per request, so you do not have to worry about thread safety issues. New Java VMs are specially tuned for short-lived objects, so this is not the performance issue it was in the past. Also, `JspContent` controls support an attribute that allows you to specify whether or not the backing file is thread safe, called Singleton Backing Instance. If this value is set to `true`, only one instance of the backing file is created and shared across all requests.

9.4.3 Scoping and Backing Files

The difference between having a backing file as part of `<netuix: portlet backingfile =some_value>` or part of `<netuix: jspContent backingfile=some_value>` is related to scoping.

For example, if you have the backing file on the portlet itself, you can actually stop the portlet from rendering. If the backing file is at the jspContent level, the portlet portion of the control tree has already run; you use this implementation to run processes that are specifically for the JSP in the portlet.

9.4.4 Backing File Guidelines

Follow these guidelines when creating a backing file:

- Ensure `netuix_servlet.jar` is included in the in the project classpath; otherwise, compilation errors occur.
- When implementing the `init()` method, avoid any heavy processing, because the `init()` method is called on all controls, whether they are visible or not.

Example 9–9 shows an example backing file.In this example, the `AbstractJspBacking` class is extended to provide the backing functionality required by the portlet. The example uses a session attribute because of the volatility of the HTTPRequest object; Oracle recommends that you pass data between life cycle methods using the session rather than the request object.

Example 9–9 Backing File Example

```
package backing;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import com.bea.netuix.events.Event;
import com.bea.netuix.events.CustomEvent;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;
public class ListenCustomerName extends AbstractJspBacking
{
   public void listenCustomerName(HttpServletRequest request,
   HttpServletResponse response, Event event)
   {
      CustomEvent customEvent = (CustomEvent) event;
      String message = (String) customEvent.getPayload();
      HttpSession mySession = request.getSession();
      mySession.setAttribute("customerName", message);
   }
}
```

9.4.4.1 Adding a Backing File Using Oracle Enterprise Pack for Eclipse

You can add a backing file to a portlet either from within Oracle Enterprise Pack for Eclipse or by coding it directly into the file to which you are attaching it. In Eclipse, you can specify the backing file in the **Backing File** field of the Properties view, as shown in Figure 9–8. You need to specify the backing directory and, following a dot-separator, *only* the backing file name. Do not include the backing file extension; for example enter this:

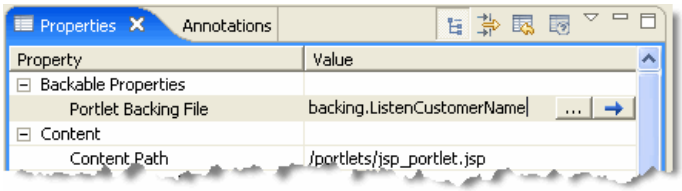
```
backing.ListenCustomerName
```

Not this:

```
backing.ListenCustomerName.java
```

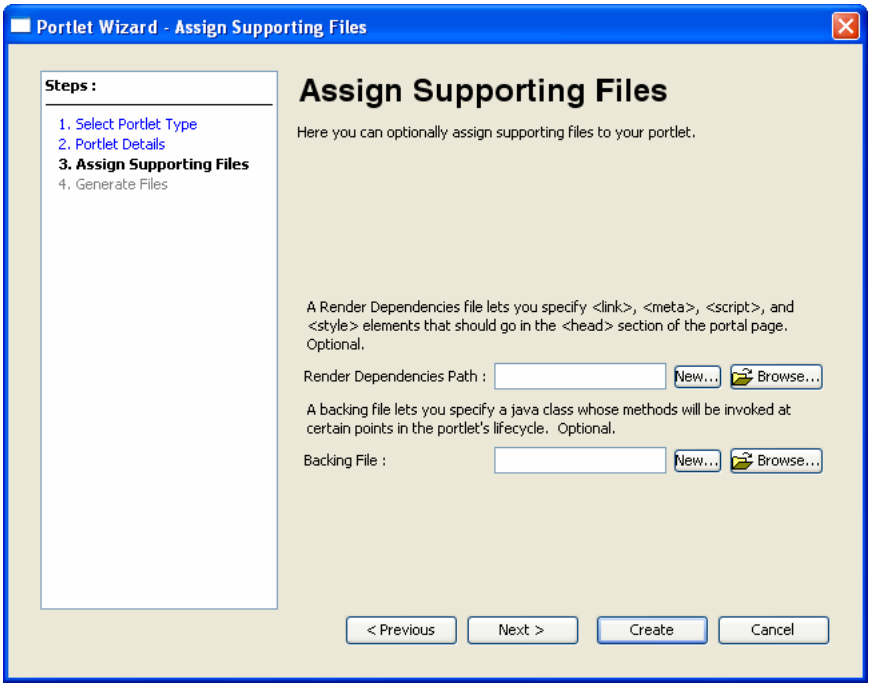
For the preceding example, if you include the file extension, the application interprets it as the file name—because the file path is specified by a dot-separator—and looks for a non-existent file called `java` in a non-existent directory called `ListenCustomerName`.

Figure 9–8 Adding a Backing File Using Oracle Enterprise Pack for Eclipse



The Portlet Wizard provides an optional Assign Supporting Files step, shown in Figure 9–9. This step lets you add a backing file and/or a render dependencies file to your portlet when you create it.

Figure 9–9 Assign Supporting Files Dialog



9.4.4.2 Adding the Backing File Directly to the .portlet File

To add the backing file by coding it into a `.portlet` file, use the `backingFile` parameter within the `<netuix:jspContent>` element, as shown in Example 9–10.

Example 9–10 Adding a Backing File to a .portlet File

```
<netuix:content>
   <netuix:jspContent
   backingFile="portletToPortlet.pageFlowSelectionDisplayOnly.menu.
      backing.MenuBacking"
   contentUri="/portletToPortlet/pageFlowSelectionDisplayOnly/menu/
```

```
      menu.jsp"/>
</netuix:content>
```

9.5 Portlet Appearance and Features

Some aspects of portlet appearance are controlled by default at the portal level, such as colors, layouts, and themes. Appearance/rendering characteristics and portlet-specific features include the use of title bars and associated states (minimize, maximize, float, and delete) and modes that affect portlet content (edit mode, help mode, and custom modes).

The following sections describe how to work with portlet-specific appearance/content features and modes:

- Section 9.5.1, "Portlet Dependencies"
- Section 9.5.2, "Portlet Modes"
- Section 9.5.3, "Creating Custom Modes"
- Section 9.5.5, "Portlet States"
- Section 9.5.6, "Portlet Title Bar Icons"
- Section 9.5.7, "Portlet Height and Scrolling"

9.5.1 Portlet Dependencies

This section explains discusses portlet dependency files (also called render dependency files), a feature that lets you specify resources that are required for rendering a portlet.

> **Note:** You can also add render dependency files to books and pages. For details, see "Adding Render Dependencies to Books and Pages" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

This section contains the following topics:

- Section 9.5.1.1, "Introduction"
- Section 9.5.1.2, "Identifying Portlet Dependencies"
- Section 9.5.1.3, "Creating, Editing, and Adding a Dependency File"
- Section 9.5.1.4, "Example Dependency Files"
- Section 9.5.1.5, "Considerations and Limitations"
- Section 9.5.1.6, "Scoping JavaScript Variables and CSS Styles"
- Section 9.5.1.7, "Rewriting Resource URLs"

9.5.1.1 Introduction

In a rendered HTML page, the proper place to include most types of resources, such as script files or style sheet references, is in the header of the document. Portlets sometimes need to specify resources that are required for rendering the portlet in the page. In the past, methods for making required elements available on the page included placing elements into the skeleton, which is not recommended because this

creates a coupling between the skeleton and the portlet; or putting references directly in the portlet content, leading to the possibility of creating invalid HTML.

The problem was exacerbated in a federated (WSRP) environment because remote portlets are potentially included in several places and there was no way for one of these portlets to indicate that it relies on, for example, a piece of a CSS that resides in an external file.

WebLogic Portal now provides an explicit way to handle this requirement, using the portlet dependencies feature.

The concepts related to skin and skeleton resource dependencies are more formally known as *render dependencies* and *script dependencies*. Typical examples of such dependencies are CSS files and JavaScript files.

Both skins and skeletons can now specify such dependencies as well as associated search paths to be used for resolving these dependencies. Additionally, mechanisms exist to eliminate redundancy and to provide a reliable ordering for dependencies related to skins, skeletons, and theme skin and skeletons. These same capabilities are available for portlets, as well as books and pages, so that a portlet can specify necessary dependencies in a standards-compliant way; you identify these dependencies using appropriate elements located in the head section of the rendered page. The other advantages of the Look & Feel dependencies framework are also realized at a portlet level, such as reliable ordering and redundancy elimination.

9.5.1.2 Identifying Portlet Dependencies

The configuration of portlet dependencies shares the same mechanisms as the standard Look & Feel—you use an XML configuration document conforming to a standard Look & Feel schema. This XML document is referenced from a `.portlet` file using an attribute on the portlet element.

As with a Look & Feel's render dependencies, you can resolve a portlet's render dependencies utilizing a set of application search paths. Additionally, the search paths of the Look & Feel skin, or any appropriate Theme skin, are used before the portlet's own search paths to resolve a portlet's render dependencies.

You can specify a portlet's dependencies configuration file in the Oracle Enterprise Pack for Eclipse Properties view by entering the value in LAF Dependencies Path field. Alternatively, you can add the attribute `lafDependenciesUri` to the portlet element in a `.portlet` file, as shown in the following example:

```
<netuix:portlet definitionLabel="myPortlet" title="My Portlet"
lafDependenciesUri="/portlets/example/myPortlet.dependencies">
```

> **Tip:** You can add multiple dependencies files to a portlet by specifying a comma-separated list of paths in the `lafDependenciesUri` attribute.

By convention, you should adhere to the following guidelines when setting up a portlet's dependencies configuration file:

- Give the file the same name as the `.portlet` file.
- Assign the file a `.dependencies` extension.
- Locate the file at the same level in the file hierarchy as the `.portlet` file.

Although the guidelines listed here are not required, deviating from them can lead to unexpected behavior. For more information, refer to Section 9.5.1.5, "Considerations and Limitations."

The portlet dependencies configuration file uses standard types from the standard Look & Feel schemas and looks similar to the example shown in Example 9–11.

Example 9–11 Portlet Dependencies Configuration File Example

```
<?xml version="1.0" encoding="UTF-8"?>
<p:window
xmlns:p="http://www.bea.com/servers/portal/framework/laf/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/portal/framework/laf/1.0.0
laf-window-1_0_0.xsd ">
   <p:render-dependencies>
     <p:html>
       <p:links>
         <p:search-path>
           <p:path-element>.</p:path-element>
         </p:search-path>
         <p:link rel="stylesheet" type="text/css" href="my.css"/>
       </p:links>
     </p:html>
   </p:render-dependencies>
</p:window>
```

The configuration file shown in Example 9–11 causes a CSS file to be included in the rendered page output (as a link element in the HTML head section). First, the search occurs for the CSS file relative to the Look & Feel or Theme skin search paths for the links element. If the CSS file is not found, then the search path in the configuration file is used. Relative search paths use the directory of the configuration file as a base.

The default behavior is to look first in the Look & Feel or Theme–specified search paths. This behavior allows a Look & Feel/Theme the ability to properly skin portlet resources. However, portlet-level resources should not be placed in the Look & Feel/Theme directories. If a situation arises when you do not want to use this behavior, you can disable it by specifying a value of `false` for the `use-skin-paths` attribute on the `render-dependencies` element.

9.5.1.3 Creating, Editing, and Adding a Dependency File

You can use the New Render Dependencies dialog box in Oracle Enterprise Pack for Eclipse to create a valid dependency file that you can then complete using Oracle Enterprise Pack for Eclipse's XML editor.

> **Tip:** For example dependency files, see Example 9–11, Example 9–12, and Example 9–13.

WLP provides several ways to access the New Render Dependencies dialog box. This dialog lets you create a Render Dependencies file (a `.dependencies` file) in the web project.

You can attach multiple dependencies files to a portlet, book, or page. You can set the Render Dependencies Path for a portlet, book, or page to be a comma-delimited list of paths. All of the dependencies files in those multiple paths will be used to resolve dependencies. This feature includes filtering to avoid duplicates so that the same artifacts are not injected multiple times in the markup.

Note: The `.dependencies` file must reside in a WebLogic Portal framework project, within the web content folder (typically named WebContent).

The following actions bring up the New Render Dependencies dialog. This dialog lets you create a render dependency file. You can then associate the file with a portlet with the portlet's Render Dependencies Path property.

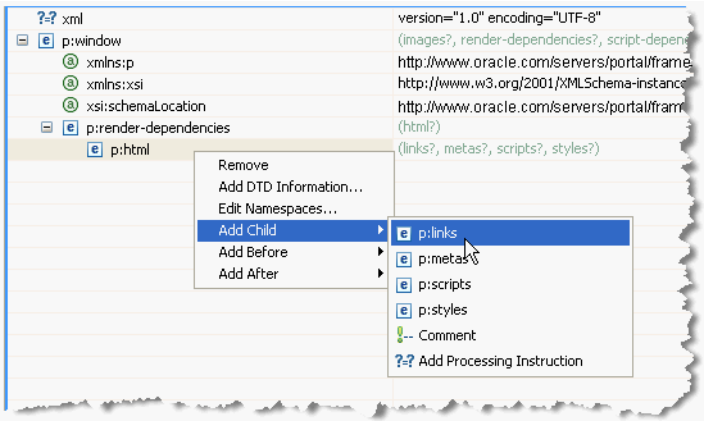
- Right-click in the main body of a portlet in the portlet editor and select Create **Render Dependency File**. This brings up the New Render Dependencies Dialog, which lets you create the file.
- Select **File** > **New** > **Other** > **WebLogic Portal > Markup Files** > **Render Dependencies**. This brings up the New Render Dependencies Dialog, which lets you create the file.
- In the portlet Properties view, edit the Render Dependencies Path property. The Properties view provides a button that brings up the New Render Dependencies dialog.
- Use the Assign Supporting Files page of the New Portlet Wizard to create a dependencies file. See Section 5.5, "Assigning Supporting Files."

You can also create a dependency file from scratch as follows:

1. Select **File** > **New** > **Other**.
2. In the New dialog, open the XML folder and select **XML**. The New XML File wizard opens.
3. Choose **Create XML From XML Schema File** and click **Next**.
4. Enter a name for the XML file in the XML File Name dialog and click **Next**.
5. In the Select XML Schema File dialog, choose **Select XML Catalog Entry** and in the Key column select `laf-window-1_0_0.xsd` as the schema. Click **Next**.
6. In the Select Root Element dialog, choose the root element **window**.
7. Optionally check the boxes that add optional attributes/elements to your new XML file.
8. Click **Finish**.
9. Rename the generated file's extension from `.xml` to `.dependencies`.

You can use the Oracle Enterprise Pack for Eclipse XML editor to add elements and attributes to the dependency file. Right-click on an element and use the menu to select child elements and add attributes. As shown in Figure 9–10, valid choices based on the schema file are automatically populated in the menu.

Figure 9–10 Editing a Dependencies File



Tip: Source view of the XML editor. Simply hover the mouse pointer over the element and a help pop-up appears. Also, in the Source view, you can click in an element and press F2 to display the help pop-up.

9.5.1.4 Example Dependency Files

This section includes the following examples:

- Section 9.5.1.4.1, "Including JavaScript in a Render Dependencies File"
- Section 9.5.1.4.2, "Including Meta and Style Elements in a Render Dependencies File"

9.5.1.4.1 Including JavaScript in a Render Dependencies File

Example 9–12 illustrates how to include both an external JavaScript file as well as an embedded script.

Example 9–12 Including JavaScript

```
<p:window
    xmlns:p='http://www.bea.com/servers/portal/framework/laf/1.0.0'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.bea.com/servers/portal/framework/laf/1.0.0
    laf-window-1_0_0.xsd '>
   <p:render-dependencies>
     <p:html>
       <p:scripts>
         <p:search-path>
           <p:path-element>.</p:path-element>
         </p:search-path>
         <p:script type='text/javascript' src='my.js'/>
         <p:script type='text/javascript'>
             alert('hello world');
         </p:script>
       </p:scripts>
     </p:html>
   </p:render-dependencies>
</p:window>
```

9.5.1.4.2 Including Meta and Style Elements in a Render Dependencies File

Example 9–13 shows the use of both the metas and styles elements. The `metas` element lets you specify HTML meta tags, and the `styles` element lets you embed HTML style tags.

Example 9–13 Use of Meta and Styling Elements

```
<p:window
    xmlns:p='http://www.bea.com/servers/portal/framework/laf/1.0.0'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.bea.com/servers/portal/framework/laf/1.0.0
    laf-window-1_0_0.xsd '>
   <p:render-dependencies>
     <p:html>
       <p:metas>
         <p:meta name='keywords' content='pirate, ninja'/>
       </p:metas>
       <p:styles>
         <p:style type='text/css'>
           div.myClass {
             background-color: red;
           }
         </p:style>
       </p:styles>
     </p:html>
   </p:render-dependencies>
</p:window>
```

9.5.1.5 Considerations and Limitations

At this time, Oracle Enterprise Pack for Eclipse does not provide editing capabilities for portlet render dependencies configuration files; you can use the included Eclipse-based XML file editor for this purpose.

Oracle recommends that you not share a single `.dependencies` file across several portlets. Although WebLogic Portal does not prevent this usage, sharing a single file might lead to confusion when coordinating updates to the file later.

9.5.1.6 Scoping JavaScript Variables and CSS Styles

Whenever you place multiple instances of a portlet on a page, you can encounter scoping problems with JavaScript variables and CSS styles. For example, if a portlet includes inlined JavaScript and you place two instances of that portlet on a page, it is possible that changing a JavaScript variable in one portlet will affect the other portlet.

To ensure that JavaScript and CSS styles are scoped to a specific portlet instance, add the token `wlp_rewrite_` to the front of the variable or style class name. When the portlet is rendered, this token is replaced by the portlet instance label, which is unique for each portlet instance.

For example, to ensure portlet instance-level scoping of a JavaScript variable called `stockQuote` that is defined in a `.js` file that is referenced from a `.dependencies` file, you need to append `wlp_rewrite_` to the front of the variable name:

```
var wlp_rewrite_stockQuote
```

To ensure portlet instance-level scoping of a CSS class name called `portlet_bg` that is defined in a `.css` file that is referenced from a `.dependencies` file, you need to append `wlp_rewrite_` to the front of the class name. For example:

```
.wlp_rewrite_portlet_bg { background_color:white; }
```

In both of these cases, the wlp_rewrite_ token is replaced by the portlet's instance label, which is a unique identifier.

Note: The scoping mechanism described in this section only works for .css and .js files that are referenced with the content-uri dependency file attribute. Files linked with the src attribute or the link tag will not be rewritten.

9.5.1.7 Rewriting Resource URLs

Some portals have several look and feels that include resources that are named the same. For example, look and feel ABC and XYZ might both have a graphic called `logo.gif`. As a portlet developer, you do not know which look and feel a portal administrator or user might choose. To avoid hard coding pathnames to resources in your portlets, you can enclose the resource name with the `wlp_rewrite?` and `/wlp_rewrite` tokens. For example, the following image source is hard coded:

```
<IMG SRC="/framework/skins/bighorn/images/titlebar-button-help.gif">
```

To avoid associating the resource path with a particular look and feel (for example, `/bighorn`), you can do this:

```
<IMG SRC="wlp_rewrite?titlebar-button-help.gif/wlp_rewrite">
```

When you use these tokens, WebLogic Portal searches for the named resource using the same mechanism it uses to search for resources associated with the currently specified look and feel. This means that whichever look and feel is selected, the correct graphics will be retrieved (assuming that the named graphic exists for that look and feel). If the resource cannot be found in the scope of the current look and feel, the original value specified will be used as-is (for example, `titlebar-button-help.gif`).

9.5.2 Portlet Modes

All portlets created with WebLogic Portal support the use of modes. Modes allow you to affect the end user's ability to edit the portlet or display Help for the portlet. You add icon buttons to a portlet's title bar to indicate the availability of a mode.

The following pre-defined modes exist for WebLogic Portal:

- **Edit** – Lets you specify a custom file that lets users modify the portlet's content when they click the Edit button.
- **Help** – Lets you specify a custom file that shows users help content for the portlet when they click the Help button.

You can also create your own custom portlet modes using WebLogic Portal.

Buttons for the selected modes appear in the portlet's title bar. Figure 9–11 shows an example of the default buttons for the portlet modes when displayed in the editor; Figure 9–12 shows the appearance of the mode icons in a running portlet.

Figure 9–11 Portlet Mode and State Buttons in Editor

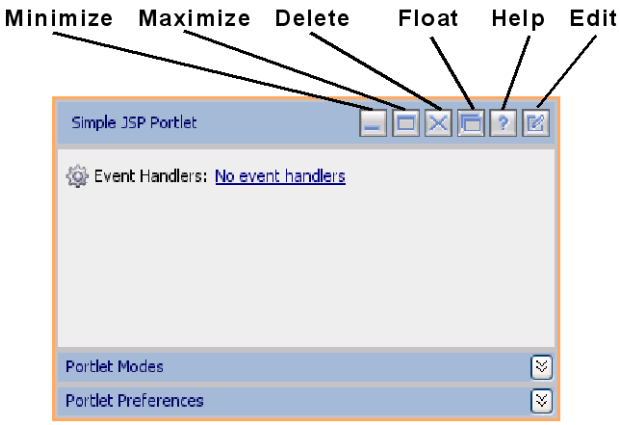
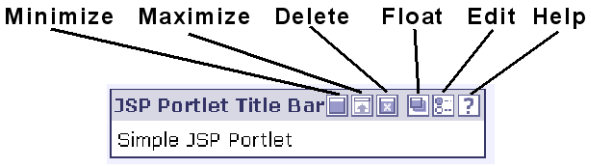


Figure 9–12 Portlet Mode and State Buttons in a Running Portlet



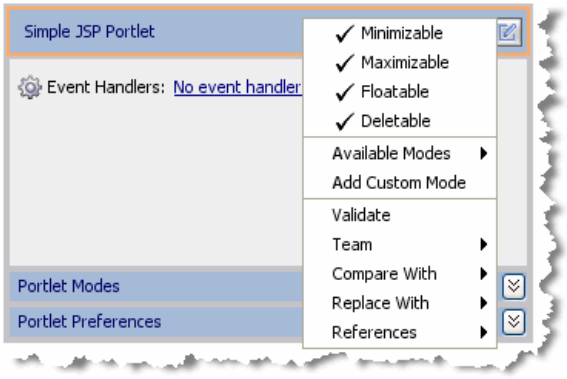
When you use the Portlet Wizard to create a portlet, mode and state settings are available on the Portlet Details dialog. These settings can also be edited in the portlet's Properties view: The following sections describe possible methods of performing these tasks.

9.5.2.1 Adding or Removing a Mode for an Existing Portlet

To add or remove the Help or Edit mode from the title bar, follow these steps:

1. Display the portlet for which you want to add or remove a mode.
2. Right-click the title bar of the displayed portlet to display the context menu. Figure 9–13 shows an example of the title bar context menu.

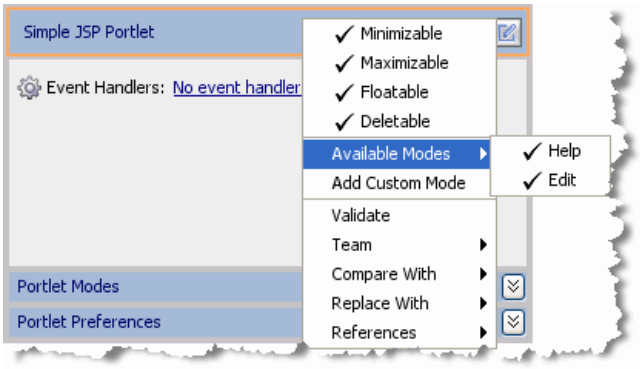
Figure 9–13 Available Portlet Modes - Title Bar Context Menu



3. Click **Available Modes**.

Checkmarks on the submenu indicate the available modes for this portlet, which were determined when you created it. Figure 9–14 shows an example of the submenu.

Figure 9–14 Portlet Mode - Available Modes Submenu



4. Click the mode for which you want to change the availability status. For example, in Figure 9–14, the Help mode is checked (available); when you click Help, the Help button disappears from the title bar.
5. Select **File** > **Save** to save your changes.

9.5.2.2 Properties Related to Portlet Modes

You can view and edit the mode's property details in the Properties view. For example, you can edit the Portlet Backing File property if you want to perform preprocessing before rendering the portlet's mode page (such as the edit page).

To display the mode properties for the portlet, click the expand/contract toggle button in the Portlet Mode area of the portlet. Edit mode properties and Help mode properties display in the Properties view.

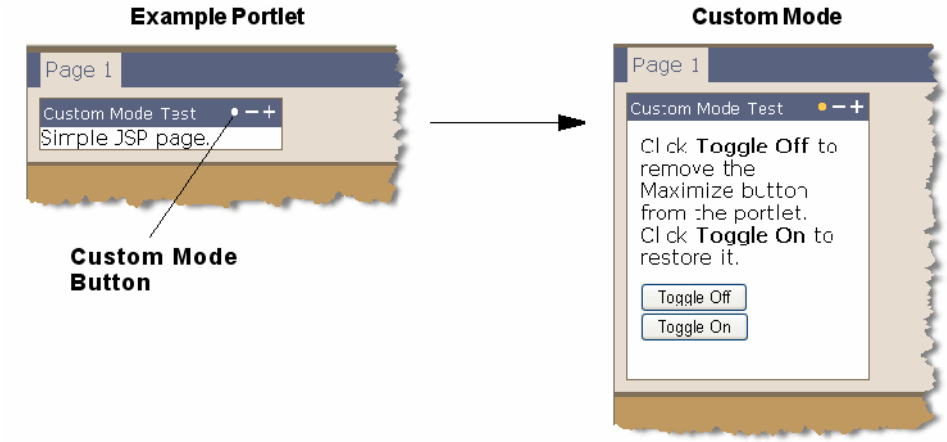
For descriptions of the mode properties, refer to Table 9–7.

9.5.3 Creating Custom Modes

A custom mode is a portlet mode that you implement. Like with the help and edit modes, a custom mode is activated with a button that appears in the portlet's title bar. To implement a custom mode, you need to supply a display part, typically a JSP, and a backing file. This section includes an example that explains how to create a simple custom mode that lets a user add or remove the Maximize button from a portlet. Once you understand the basic principles involved in writing a custom mode, you can create a custom mode to perform the specific tasks you want.

Figure 9–15 shows the example portlet and the portlet's custom mode view. When the user clicks the custom mode button in the example portlet on the left, the portlet display changes to the custom mode view on the right. In this example, the custom mode offers a way for the user to add or remove the portlet's Maximize button.

Figure 9–15 Selecting a Custom Mode



1. Create a JSP portlet in which to embed the custom mode. For information on JSP portlets, see Section 5.4.1, "Building JSP and HTML Portlets." For this example, any JSP portlet will suffice.
2. Create a JSP page to display the custom mode view when a user clicks the custom mode button. For example, Example 9–14 shows a JSP for a custom mode that lets a user add or remove the Maximize button from a portlet. The code to execute this action is in a backing file, which is discussed next. In this example, the JSP is called `togglebutton.jsp`.

Example 9–14 Sample Custom Mode JSP

```
<%@ page import="com.bea.portlet.PostbackURL"%>
<%
    PostbackURL url = PostbackURL.createPostbackURL(request, response);
%>
<TABLE CELLSPACING="10" ID="toggleButtonsTable">
    <TH>Using a Button and Backing File</TH>
    <TR>
      <TD>
        Click <b>Toggle</b> Off to remove the Maximize button from the
portlet.<br>
        Click <b>Toggle On</b> to restore it.
      </TD>
    </TR>
    <TR>
      <TD>
        <FORM method="post" name="Toggle" action="<%=url.toString()%>">
        <INPUT ID="toggle_off" TYPE="SUBMIT" NAME="toggle_off" VALUE="Toggle
Off">
        <INPUT ID="do_nothing" TYPE="SUBMIT" NAME="do_nothing" VALUE="Toggle
On">
        </FORM>
      </TD>
    </TR>
</TABLE>
```

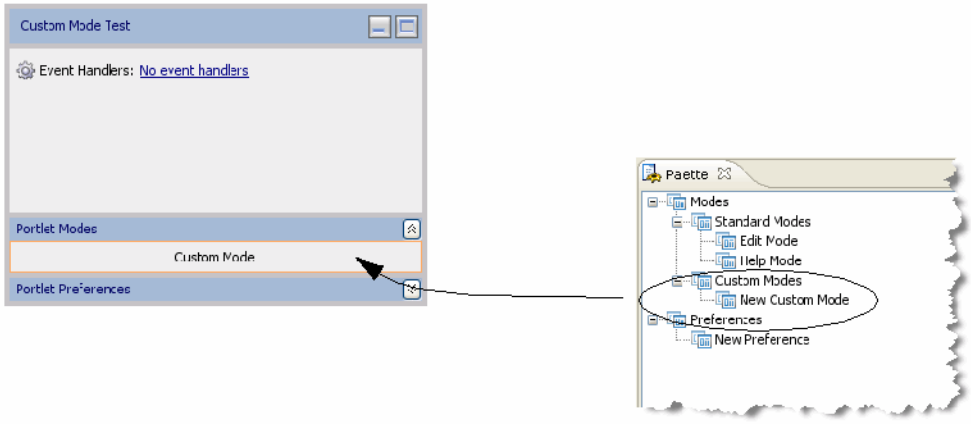
3. Create a backing file for the custom mode. Example 9–15 implements the JspBacking interface and implements the preRender() method of that interface. In this example, the preRender() method removes the Maximize button from the portlet in response to a request. Refer to *Oracle Fusion Middleware Java API Reference for Oracle WebLogic Portal* for details on the API used in this example.

Example 9–15 Sample Backing File

```
package modes;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.bea.netuix.servlets.controls.content.backing.JspBacking;
import com.bea.netuix.servlets.controls.portlet.backing.PortletBackingContext;
import com.bea.netuix.servlets.controls.window.WindowCapabilities;
import com.bea.p13n.util.debug.Debug;

public class MyMode implements JspBacking {

    public void dispose() {
    }

    public boolean handlePostbackData(HttpServletRequest arg0,
    HttpServletResponse arg1) {
        return true;
    }

    public void init(HttpServletRequest arg0, HttpServletResponse arg1) {
    }

    public boolean preRender(HttpServletRequest request, HttpServletResponse
response) {
        PortletBackingContext pbc =
        PortletBackingContext.getPortletBackingContext(request);
        if (request.getParameter("toggle_off") != null)
        {
             try
             {
                pbc.setCapabilityVisible(WindowCapabilities.MAXIMIZED.getName(),
                false);
             }
             catch (NullPointerException npe)
             {
                  //
             }
        }
        return true;
    }
}
```

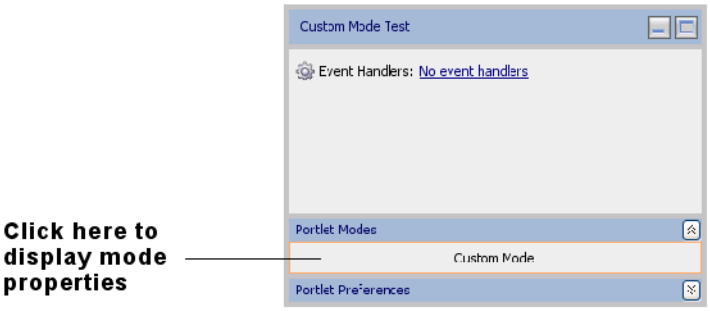
4. Add a new custom mode to the portlet by dragging the New Custom Mode icon from the Design Palette to the portlet, as shown in Figure 9–16. You will be prompted to enter a name for the mode. You can enter a name now, or accept the default and change the name later.

Figure 9–16 Adding a New Custom Mode



5. Open the Properties view for the custom mode. To do this, click in the Custom Mode region of the portlet in the portlet editor, as shown in Figure 9–17. The properties for the custom mode appear in the Properties view.

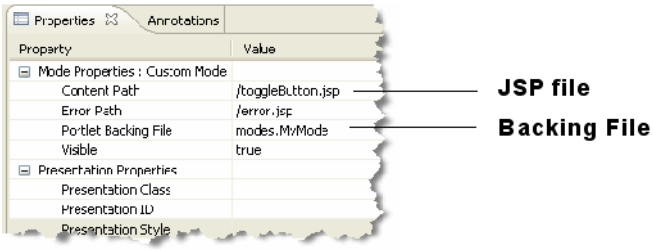
Figure 9–17 Displaying Mode Properties



6. In the Properties view, enter the path of the custom mode JSP in the Content Path field. This is the JSP that is displayed when the mode is activated. You can find the Content Path field in the Mode Properties section of the Properties view, as shown in Figure 9–18.

7. In the Properties view, enter the name of the backing file class, including the full package name. You can find the Portlet Backing File field in the Mode Properties section of the Properties view, as shown in Figure 9–18.

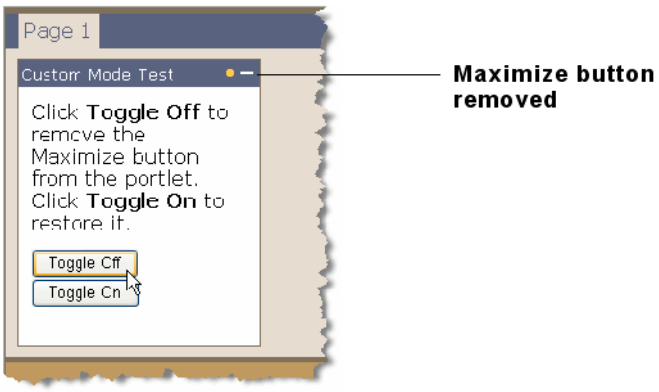
Figure 9–18 Specifying a Content File and a Backing File



Tip: The Properties view lets you set many other custom mode properties, such as an image for the custom mode button, a rollover image, button text, alternate text, and others. Refer to Table 9–7 at the end of this section for information on each of the custom mode properties.

8. Test the custom mode by placing the example portlet in a portal and running it on the server. Select the portlet's custom mode button, as shown previously in Figure 9–15, to display the custom mode view. Click **Toggle Off** to remove the Maximize button, as shown in Figure 9–19.

Figure 9–19 Testing the Example



Section 9.5.4, "Custom Mode Properties" briefly describes each of the custom mode properties.

9.5.4 Custom Mode Properties

Table 9–7, Table 9–8, and Table 9–9 describe the mode, presentation, and toggle button properties.

Table 9–7 Mode Properties

Property	Value
Content Path	Required. The path (relative to the project) to the file/class to be used for the custom mode portlet's content. From the data field you can choose to browse to a file (or class for page flow portlets) or open the currently displayed file/class. For example, if the content is stored in `Project/myportlets/my.jsp`, the Content URI is `/myportlets/my.jsp`.
Error Path	Optional. The path (relative to the project) to the JSP, HTML, or page flow file to be used for the error message if the portlet's mode page cannot be rendered. For example, if the error page is in `project/myportlets/errorPortletEdit.jsp`, the Content URI is `/myportlets/errorPortletEdit.jsp`.
Portlet Backing File	Optional. If you want to use a class for preprocessing (for example, authentication) prior to rendering the portlet, enter the fully qualified name of that class. That class should implement the JspBacking interface or extend AbstractJspBacking. From the data field you can choose to browse to a class or open the currently displayed class.
Visible	Optional. Makes the mode icon in the title bar or menu invisible (`false`) or visible (`true`). Set Visible to `false` when, for example, you want to provide an custom mode URL in a desktop header.

Table 9–8 Presentation Properties

Property	Value
Presentation Class	This property is described in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Presentation ID	This property is described in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Presentation Style	This property is described in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Properties	Optional. A comma-delimited list of name-value pairs to associate with the object. This information can be used by skeletons to affect rendering.
Skeleton URI	This property is described in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Table 9–9 Toggle Button Properties

Property	Value
Activate Alternate Text	Popup text that appears when the mouse pointer hovers over the custom mode button.
Activate Image	An image for the button that activates the custom mode. Place the image in the images directory of the skin that your portal uses.
Activate Rollover Image URI	Provides a rollover image for the custom mode button. Place the image in the images directory of the skin that your portal uses.
Active	Not generally used, but available for use by custom skeletons.
Alternate Text	Not generally used, but available for use by custom skeletons.
Deactivate Alternate Text	Popup text that appears when the mouse pointer hovers over the custom mode button.
Deactivate Image URI	An image for the button that deactivates the custom mode. Place the image in the images directory of the skin that your portal uses.
Deactivate Rollover Image UI	Provides a rollover image for the button that deactivates the custom mode. Place the image in the images directory of the skin that your portal uses.
Image	Not generally used, but available for use by custom skeletons.
Name	The name of the custom mode. If specified, the name appears in the Portlet editor view, Outline view, and Properties view. If no name is supplied, a default name is used.
Rollover Image	Not generally used, but available for use by custom skeletons.

9.5.5 Portlet States

States determine the end user's ability to affect the rendering of a portlet. WebLogic Portal supports these portlet states:

Normal – the typical rendered appearance of the portlet.

- **Minimize** – Collapses the portlet, leaving only the title bar, when the user clicks the **Minimize** button.
- **Maximize** – Makes the portlet take up the entire desktop area (not including the desktop header and footer) when the user clicks the **Maximize** button.
- **Float** – Displays the portlet in a popup window when the user clicks the Float button.

- **Delete** – Removes the portlet from the desktop when the user clicks the **Delete** button.

When you use the Portlet Wizard to create a portlet, state and mode settings are available on the Portlet Details dialog. These settings can also be edited in the portlet's Properties view: The following sections describe possible methods of performing these tasks.

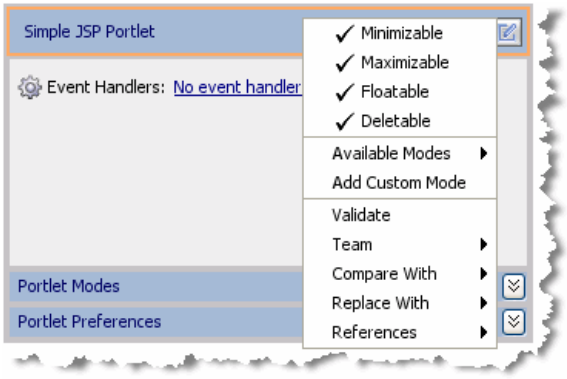
9.5.5.1 Modifying Portlet States in Oracle Enterprise Pack for Eclipse

You can select which of the states you want to include with the portlet by following these steps:

1. Right-click the portlet title bar.

   A context menu showing applicable states appears. Figure 9–20 shows an example of the title bar context menu showing all states as available.

Figure 9–20 Portlet State - Title Bar Context Menu



2. Click to select the state that you want to change.

   Selecting a state adds it to the portlet, while deselecting the state removes it from the portlet. For example, in Figure 9–20, all four states are selected, and appear in the title bar. If you click to deselect **Deletable**, the Delete button on the portlet disappears.

3. Select **File** > **Save** to save your changes.

9.5.5.2 Minimizing or Maximizing a Portlet Programmatically

You can minimize or maximize a portlet either in the portlet file or in a portlet's backing file. The actual code is the same for both. Here is an example of maximizing a (Java page flow) portlet:

```
PortletBackingContext context =
PortletBackingContext.getPortletBackingContext(request);

context.setupStateChangeEvent(WindowCapabilities.MAXIMIZED.getName());
```

You can put this code in an `action` method of the Java page flow or in the `handlePostbackData` method of the backing file. When using the backing file, in order to get the handlePostbackData method to be called, you must have `'_nfpb=true'` in the URL.

These mechanisms do not work if asynchronous content rendering is enabled for the portlet.

Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

9.5.6 Portlet Title Bar Icons

The default state and mode icons used in portlet title bars are stored in the wlp-lookandfeel-web-lib J2EE Shared Library; you can view them in Merged Projects view in the various subdirectories of `framework/skins`.

9.5.7 Portlet Height and Scrolling

All portlets created with WebLogic Portal support height and scrolling.

- **Height** affects the portlet's displayed height on the portlet page.
- **Scrolling** affects whether or not the portlet is scrollable.

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: 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 workbench editor.

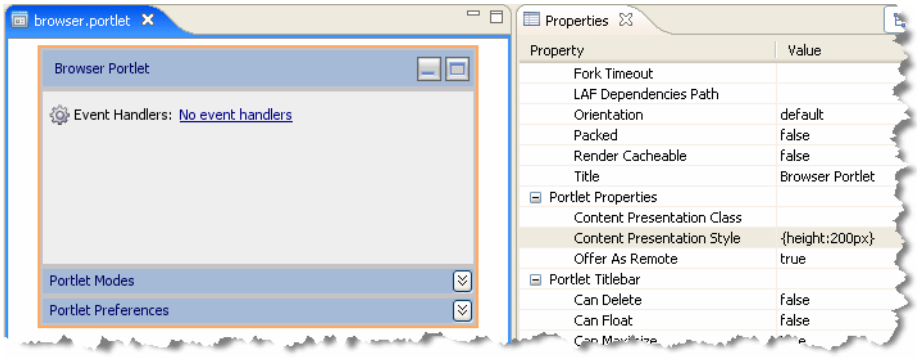
To set these properties, follow these steps:

1. Open a portlet in the workbench editor.
2. Click the outer border of the portlet to display the portlet properties in the Properties view.
3. In the Properties view, set one of the following properties:
   - **Presentation Style** - Enter any of the previously listed attributes for this property. You can use overflow and height. Separate the values with a semicolon.
   - **Presentation Class** - Enter the name of a style sheet class that contains the height or scrolling attributes that you want to use.
   - **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 that you want to use.

   Note: The distinction between Presentation Style and Content Presentation Style, for example, is the location where the styling is applied (portlet or content). The use of one or the other depends on the specifics of what the specific styling is trying to accomplish.

   Figure 9–21 shows an example of a height property, set using Content Presentation Style.

Figure 9–21 Portlet Height and Scrolling Presentation Properties Example



Based on the entries shown in Figure 9–21, the result looks similar to the example in Figure 9–22.

Figure 9–22 Portlet Height and Scrolling—Portlet Appearance Results

If you use the Presentation Class property instead of the Presentation Style property, you must have the corresponding style class defined in a CSS file.

For example, if you use the value .**portlet-scroll** in the **Content Presentation Class** field, you must have the following style class definition already set up in your CSS file:

```
.portlet-scroll
{
   overflow:auto;
   height:250px;
}
```

4. Select **File** > **Save** to save your changes.

9.5.7.1 Making All Portlets Scroll

To provide portlet height and scrolling automatically, you can specify an additional rule for the standard portlet content CSS class. For example, you can do one of the following:

- Add a `<style>` element to the `skin.xml` file for your Look & Feel containing this rule:

```
.bea-portal-window-content
{
     height: 250px;
```

```
        overflow: auto;
    }
```

- Alternatively, you can place the above rule in a custom CSS file and create a `<style>` or `<link>` element in the `skin.xml` file that references the custom CSS file.

For more information on portal skins, themes, and skeletons, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

9.6 Getting Request Data in Page Flow Portlets

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

A page flow stores information in the requests. If you have a portal page with multiple page flow portlets, you need a way for each page flow to individually store and retrieve that information. For example, the request object for a page might have a variable *car_type*, with a value of `x`. When the page flow runs, it obtains this value and uses it in some way. If you have another page flow portlet with a *car_type* value of `z`, and if only one request exists for the whole page, the two page flow portlets might interfere with each other. To prevent this problem, WebLogic Portal essentially makes a copy of the outer (portal) request to make separate *scoped* requests, one for each portlet. This gives each page flow portlet its own unique request to use to store its information.

In some cases, you might want to use information that is stored at the outer request rather than within the scoped request.

For example, if you use regular HTML tags within Netui form tags, you might have something similar to this:

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

Based on the tags used above, you might typically use a regular `getParameter` request like this:

<%request.getParameter(`"test"`)%>

However, to get that HTML input value from the outer request, use the following:

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

9.7 JSP Tags and Controls in Portlets

WebLogic Portal provides JSP tags that you can use within JSPs. You can view available JSP tags in the Design Palette and then drag them into the Source View of your JSP, and use the Properties view to edit elements of the code.

WebLogic Portal also provides custom Apache Beehive controls that make it easy for you to quickly add pre-built modules to your portal; custom Java controls exist for event management, Visitor Tools, Community management, and so on. For example, most user management functionality can be easily exposed with a User Manager Control on a page flow.

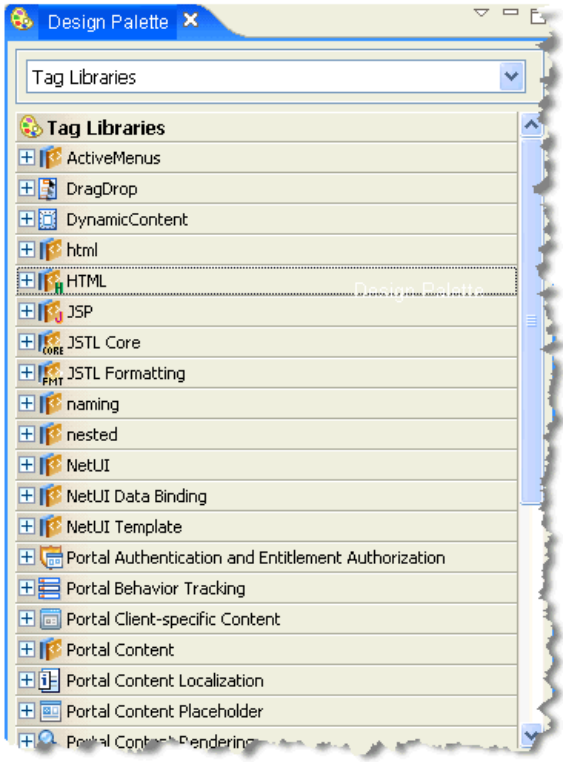
Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Note: The term control is also used to refer to the portal (netuix) framework controls, such as desktop, book, page, and so on. These controls are referred to in the text as *portal framework controls*.

9.7.1 Viewing Available JSP Tags

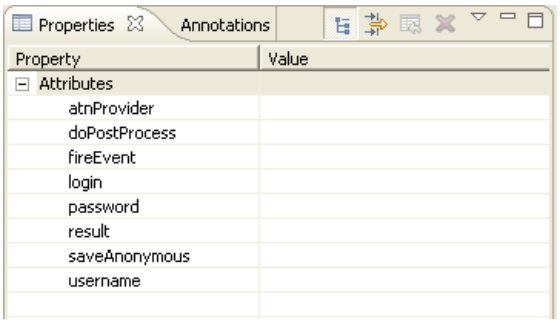
When you open a JSP in Oracle Enterprise Pack for Eclipse, you can use the Design Palette to display all the JSP tags currently loaded and available; Figure 9–23 shows a portion of the display.

Figure 9–23 Design Palette Showing Available JSP Tags



To use a tag, drag it into the editor, use the Source View to edit the code directly, and use the Properties view to set properties, as shown in Figure 9–24:

Figure 9–24 Dragging a JSP Tag into the Design View – Properties for Add User JSP Tag



For information about the Java class associated with each JSP tag, refer to *Oracle Fusion Middleware Java API Reference for Oracle WebLogic Portal*.

9.7.2 Viewing Available Controls

To view the available custom controls provided by WebLogic Portal when viewing a page flow:

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

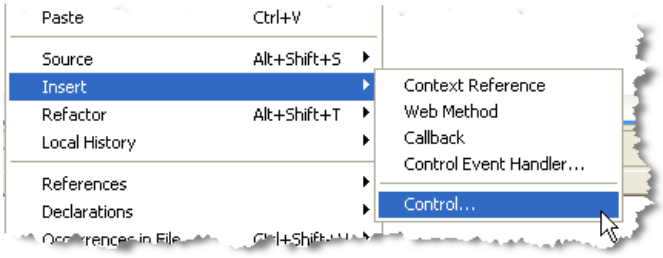
1. Open an existing page flow (`.java` file) or create a new page flow.

   For information about creating page flows using Oracle Enterprise Pack for Eclipse, refer to the *Oracle Enterprise Pack for Eclipse User's Guide*.

2. If you are not already using the Page Flow Perspective, Oracle Enterprise Pack for Eclipse asks if you want to switch to it. Do so.

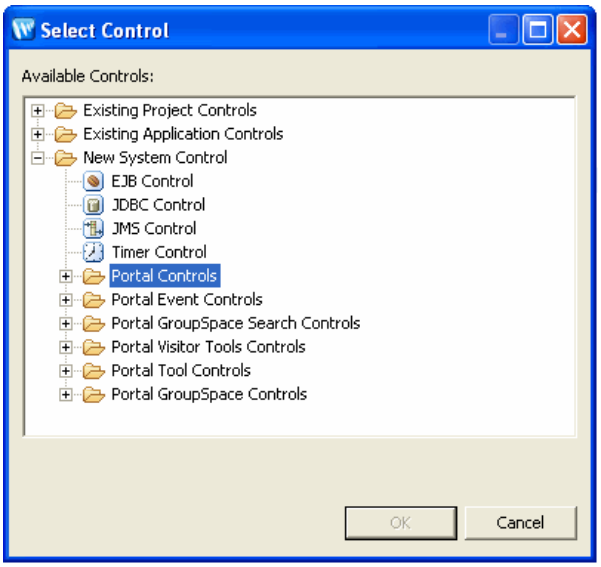
3. Right-click in the source view for the Page Flow and select **Insert** > **Control**, as shown in Figure 9–25.

Figure 9–25 Insert > Control Menu Selection



The Select Control dialog box displays, as shown in Figure 9–26.

Figure 9–26 Select Control Dialog



4. Expand the desired folder to view the custom Java controls for WebLogic Portal that you can choose from.

After you add a custom WebLogic Portal control, all the methods in the control become available to your Page Flow.

For more information about the custom controls provided by WebLogic Portal, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*. For details about each control, refer to the *Oracle Fusion Middleware Java API Reference for Oracle WebLogic Portal*. (Links to the Javadoc for each of the controls packages are conveniently listed in the Javadoc Overview frame.)

9.8 Portlet State Persistence

You can control portlet state persistence using the `persistence-enabled` attribute in the `netuix-config.xml` file, which is located by default in the WEB-INF directory. Using this attribute causes the state to be saved in the WebLogic Portal database. The attribute is set to `false` by default.

The following code segment shows an example of the attribute syntax:

```
<control-state-location>
<session persistence-enabled="true"/>
</control-state-location>
```

WebLogic Portal places an entry for the control tree state in the PROPERTY_KEY table, with the following PROPERTY_SET_NAME value:

- BEA_PORTAL_FRAMEWORK_CONTROL_TREE_STATE

9.9 Advanced Portlet Development with Tag Libraries

During the Development phase, you can use tag libraries to add features to a custom Community or a portal web application. This section discusses the following tag libraries:

- The `ActiveMenus` JSP tag library

- The `DragDrop` JSP tag library
- The `DynamicContent` JSP tag library
- The `UserPicker` JSP tag library

See the *Oracle Fusion Middleware Communities Guide for Oracle WebLogic Portal* for additional information.

9.9.1 Adding ActiveMenus

You can add the `ActiveMenus` JSP tag library to a custom Community or a portal web application.

> **Note:** You do not need to follow the procedures in this section for GroupSpace communities. ActiveMenus are enabled by default for GroupSpace communities. *Note that the GroupSpace application is deprecated.* Oracle recommends that you consider using Oracle WebCenter Collaboration as a foundation for collaboration.

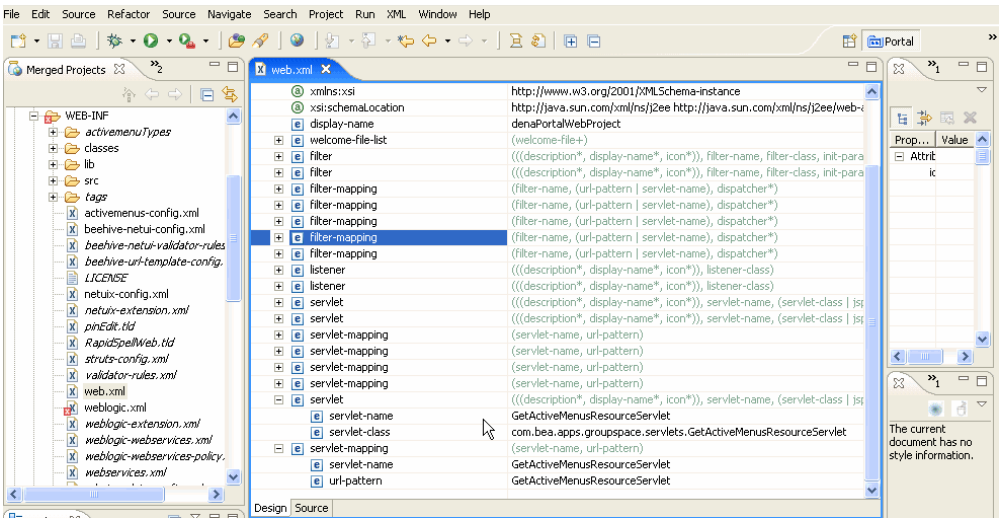
The ActiveMenus JSP tag library lets you set up a popup menu that displays when the mouse hovers over specific text. An `activemenus-config.xml` file controls the contents of each menu. The `activemenus_taglib.jar` file contains the ActiveMenus tag library.

You can tie a user's capability to the ActiveMenu that you see when you hover your mouse over an item (an Issue, for example) and hover over the arrow that appears. In this example, if your assigned capabilities include the ability to delete items, you will see the **Delete** choice.

Perform the following steps to enable ActiveMenus in a custom Community:

1. In Oracle Enterprise Pack for Eclipse, make the `activemenus_taglib.jar` file available to your portal web project. When you create your portal web project, you must enable the **WebLogic Portal Collaboration** check boxes.
2. Add the `activemenus-config.xml` file to your `/WEB-INF` directory in your portal web project. Add the file by right-clicking the `activemenus-config.xml` file and choosing **Copy To Project**. Configure the file by follow the instructions in Section 9.9.1.1, "Configuring the ActiveMenus Tag" to edit the `activemenus-config.xml` file.
3. Register the `GetActiveMenusResourceServlet` by adding the servlet and servlet-mapping to the `web.xml` file in the `/WEB-INF` directory in your portal web project. You can edit the file in Oracle Enterprise Pack for Eclipse by double-clicking the `web.xml` file. Right-click the **web-app** line in the file and choose **Add Child** > **message-destination - welcome-file-list** > **servlet**. Add `GetActiveMenusResourceServlet` to the `servlet-name` line. Add `com.bea.apps.groupspace.servlets.GetActiveMenusResourceServlet` to the `servlet-class` line. See Figure 9–27 to view the edited file in Oracle Enterprise Pack for Eclipse.

Figure 9–27 Editing the web.xml File in Oracle Enterprise Pack for Eclipse



The code sample in Example 9–16 shows the new information you added.

Example 9–16 Code Sample of GetActiveMenusResourceServlet

```
<!-- ActiveMenus Servlet Mappings -->
<servlet>
     <servlet-name>GetActiveMenusResourceServlet</servlet-name>
     <servlet-class>
          com.bea.apps.groupspace.servlets.GetActiveMenusResourceServlet
     </servlet-class>
</servlet>
<servlet-mapping>
     <servlet-name>GetActiveMenusResourceServlet</servlet-name>
     <url-pattern>GetActiveMenusResourceServlet</url-pattern>
</servlet-mapping>
```

4. Redeploy the application for the changes to take effect.

After you enable the ActiveMenus, you must configure the `ActiveMenus` tag.

9.9.1.1 Configuring the ActiveMenus Tag

To use the ActiveMenus tag, you must set up the `activemenus-config.xml` file (the `XSD` that defines this config file is located in the `activemenus_taglib.jar` file as `activemenus-config.xsd`). This `activemenus-config.xml` file file must exist in your web application's `/WEB-INF` directory. Multiple menus can be set up that consist of completely different items, styles, and icons.

Use the following sections to configure the `activemenus-config.xml` file:

- Section 9.9.1.1.1, "Using The TypeInclude tag"
- Section 9.9.1.1.2, "Using The Type Tag"
- Section 9.9.1.1.3, "Using The TypeDefault Tag"
- Section 9.9.1.1.4, "Using The menuItem Tag"

9.9.1.1.1 Using The TypeInclude tag Use the `typeInclude` tag to keep your configuration file clean. Rather than adding the `type` tag (see Section 9.9.1.1.2, "Using The Type Tag") you can add this tag and point its `href` attribute to an XML file

(relative to the web application) that contains all of the `type` information. An example of the `typeInclude` tag is:

```
<typeInclude xhref="/WEB-INF/activemenuTypes/username.xml"/>.
```

You can also use the `type` tag with the `typeInclude` tag in the configuration file. See the code sample in Example 9–17.

Example 9–17 You Can Use the typeInclude Tag with the Type Tag in the activemenus-config.xml File

```
<typeInclude xhref="/WEB-INF/activemenuTypes/username.xml"/>
<type>
     <menuItem>
          <param name="linkId"/>
          <action action="editLink">
               <i18nNamebundleName="com.bea.apps.groupspace.links.
                    LinksPopupMenu" key="edit.link"/>
          </action>
          <img xsrc="wlpAppsCollaborationCore/images/wlp-edit-16.gif"/>
     </menuItem>
</type>
```

When you point to another XML file, ensure that you namespace it correctly, as shown in Example 9–18.

Example 9–18 Pointing to Another XML File Called username.xml

```
<type name="username"
     xmlns="http://www.bea.com/servers/apps/groupspace/ui/
          activemenus-config/9.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.bea.com/servers/apps/groupspace/ui/
          activemenus-config/9.0">
...
</type>
```

9.9.1.1.2 Using The Type Tag The `type` tag defines the individual menus to use within the web application. The `name` attribute must be unique for each menu, because the name is how the menu is referenced when you use the ActiveMenus tag. Following is an example of the `type` tag:

```
<type name="foo">
</type>
```

> **Note:** The TypeDefault and MenuItem tags must be contained within the type tag.

9.9.1.1.3 Using The TypeDefault Tag The `typeDefault` tag defines what displays in the browser where the `ActiveMenus` tag is used. You can control the text that displays, the style of the text, and the image that appears on the mouseover of that text (which denotes the menu itself).

The following items display within the browser where you used the `ActiveMenus` tag:

- The `displayText` Attribute – Defines the actual text that displays. If the `displayText` is not defined, whatever text is placed in the `display` attribute of the `ActiveMenus` tag appears. However, if you want to display other text, you can specify a class and a method within that class that returns a String to display. The following example shows how to display other text.

  GetUserNameFromProfile.java

  ```
  public class GetUserNameFromProfile
  {
       public static String getName(String userName)
       {
            return "XXX-" + username + "-XXX";
       }
  }
  ```

  If you use this code, the configuration defined above, and the following `ActiveMenus` tag: `<activemenus display="UserName" type="foo"/>`, the following displays in the browser: `XXX-UserName-XXX`.

  This example allows you to use the information entered in the body of the `ActiveMenus` tag to look up other information to display. For instance, a username can be used to look up a user's full name to display. The only rules surrounding this action is that the method used for the display text is public, static, takes in a String, and returns a String. No other information can be passed into that method.

- The `displayTextStyle` Attribute – Defines the CSS style or class that stylizes the display text. In order for the `class` attribute to work correctly, the class must be defined on the page (or the CSS file that defines the class must be imported).
- The `displayMenuImage` Attribute – Defines the image that appears when the display text is passed over with the mouse. If this tag is not defined, the default image is used. This image is in the `activemenus_taglib.jar` file and is called `menu_default.gif`.
- The `menuStyle` Attribute – Defines the CSS style or class that stylizes the menu itself, which can include the border or background color. For the `class` attribute to work correctly, the class must be defined on the page (or the CSS file that defines the class must be imported).

> **Note:** The TypeDefault and MenuItem tags must be contained within the type tag.

9.9.1.1.4 Using The menuItem Tag

The `menuItem` tag defines the individual items within the popup menu. Example 9–19 shows a code sample using the `menuItem` tag.

Example 9–19 The menuItem Tag

```
<menuItem>
     <param name="userId"/>
     <xmlHttp url="GetFirstNameServlet"/>
     <row class="menuRow" style="backround-color:red"/>
     <text class="menuText" style="color:#000000"/>
     <rowRollover class="menuRowRollover" style="background-color:green"/>
     <textRollover class="menuTextRollover" style="color:#FFFFFF"/>
</menuItem>
<menuItem>
     <javascript>
```

```
     <name>Testing</name>
     <script>testing(this);</script>
     </javascript>
</menuItem>
<menuItem default="true" showMenuItem="false">
     <param name="q" value="foo"/>
     <link url="http://www.google.com">
          <name>Google</name>
     </link>
</menuItem>
<menuItem>
     <showMenuItem className="com.foo.CheckUserRights" methodName=
          "doesUserHaveRights">
          <rights name="can_view"/>
          <rights name="can_edit"/>
     </showMenuItem>
     <allParams/>
     <action action="addEditLink" disableAsync="true">
          <i18nName bundleName="com.foo.LinksPopupMenu" key="edit.link"/>
     </action>
</menuItem>
<menuItem>
     <allParams/>
     <dcAction action="showFeedData" dcContainerId="feedDataContainer">
          <i18nName bundleName="com.foo.LinksPopupMenu" key="show.
               feedData"/>
     </dcAction>
</menuItem>
```

The `menuItem` tag defines the individual items within the popup menu with the following four types:

- The `javascript` Element – This element can be any JavaScript that you want to run when the user clicks this menu item. To make this more useful, you can retrieve the values that you specify in the `param` tag (see the code sample below) through custom parameters that are added to the menu item. Following is a basic example of how to implement JavaScript.

  ```
  ...
       <activeMenus:activemenus display="Foo Link" type="link">
            <param name="linkId" value="${fooLink.id}"/>
            <param name="linkParent" value="${fooLink.parent}"/>
       </activeMenus:activemenus>
  ...
  ```

  The next step is to define the custom JavaScript in your configuration file. The JavaScript must pass in the code shown in the following sample.

```
...
     <type name="link">
          <menuItem>
               <allParams/>
               <javascript>
                    <name>Testing</name>
                    <script>fooTest(this);</script>
               </javascript>
          </menuItem>
     </type>
...
```

The last step in implementing the JavaScript element is to access the values in your JavaScript function, as shown in the following code sample.

```
...
     <script>
     function fooTest(object)
     {
          var linkId = object.getAttribute("linkId");
          var linkParentName = object.getAttribute("linkParent");
     }
     </script>
 ...
```

- The `xmlHttp` Element – The `xmlHttp` references a servlet (which must follow all standard servlet configuration). Whatever the servlet outputs is shown in that row of the menu. If `""` or `null` is returned from the `xmlHttp` servlet, the menu item row does not appear in the menu. The information is retrieved through an `xmlHttp` request, which allows the information to be updated without refreshing the page. For example, you could show a user's online status that would update without having to make a full post. The two rules that surround writing your servlet for this is that all the processing must happen in the servlet's `doPost()` method. The second rule is that the defined parameters are passed in as request parameters. Following is an example of getting the query parameters:

  ```
  String userName = request.getHeader("linkId");
  ```

- The `link` Element – This static URL opens a new browser window pointed to the defined URL. This tag can take in either a `name` tag or an `i18nName` tag (defined below) that is displayed within the menu itself. Any defined parameters are added to the end of the link as regular request parameters.
- The `action` Element – This `action` name must be available to the page or portlet that contains the `ActiveMenus` tag. This element runs the action within the current browser, so you can use forwards to control your page flow. This tag can take in a `name` tag or an `i18nName` tag (defined below) that will appear within the menu itself. Any defined parameters passed in are available on the request as parameters. Following is an example of retrieving these values from a page flow:
- `String linkId = getRequest().getParameter("linkId");`

  You can also use an attribute called `disableAsync` within AJAX-enabled portlets. If you want your menu item action to submit outside of the AJAX framework (so the page makes a full post), set this attribute to `true`. By default, the attribute is set to `false`.
- The `dcAction` Element – If you have a Dynamic Content container set up within your page, you can set up a menu item to call an action and have it update the Dynamic Content container. This works the same as an `action` menu item, and takes in the action name to execute. The only difference is you must specify the `dcContainerId` and it must correspond to a `dcContainerId` that is defined within a `<dc:executeContainerAction>` tag on the page.
- Other attributes and elements that you might use include the following:
  - The `showMenuItem` Element – Add this element if you need to conditionally show the menu item (for example, based on a set of rights for the current user). You define a class name and a method name that determines if the menu item should be shown. You can use multiple `showMenuItem` tags, each using different classes, methods, or rights. If you use more than one tag, all cases must be satisfied in order for the menu item to appear. For example, if

the user passes nine of 10 cases, the menu item does not appear because all cases were not passed. Example 9–20 shows how you can use the `showMenuItem` tag.

Example 9–20 The CheckUserRights.java Class with the showMenuItem Tag

```
public class CheckUserRights
{
     public static boolean doesUserHaveRights(HttpServletRequest request,
          String[] rights)
     {
          for(int i=0;i<rights.length;i++)
          {
               if(!checkAccess(request, rights[i]))
               {
                    return false;
               }
          }
          return true;
          }
     }
```

- The `default` Attribute – When this attribute is used in a `menuItem` tag and set to `true`, the display text anchor's `href` will be the link or action. Use this attribute when you want a default action to occur when clicking the main link, and you also want to display the action for consistency purposes. The default value for this attribute is `false`.
- The `showMenuItem` Attribute – When this attribute is used in a `menuItem` tag and set to `false`, the menu item does not appear in the ActiveMenu. Use this attribute when you want a default action to occur when you click the main link, but you do not want to display the action. The default value for this attribute is `true`.

> **Note:** Do not wrap an ActiveMenus tag in an anchor tag because you can get undesired results. Instead, use the default and showMenuItem attributes to control the ActiveMenu display text link.

- The `allParams` Element – This element specifies that all of the parameters defined on the tag (see Section 9.9.1.2, "Using the ActiveMenus Tag") are set up on this menu item. If this element is not used (and the `param` element is not used), then parameters are not set up on the menu item.
- The `param` Element – This element sets the specified parameters on the menu item. The `param` element has a `name` attribute that must match the `name` attribute on a `param` element that is set within the `ActiveMenu` tag (see Section 9.9.1.2, "Using the ActiveMenus Tag"). This also has a `value` attribute that can be used to hard code a value at configuration time. If this `value` attribute has been set, but a `value` was also specified at run-time (for example, using the `param` tag within the `ActiveMenu` tag), the run-time value takes precedence over the hard-coded value. Also, if just the hard-coded value is to be used, the `param` tag does not have to be specified when you use the `ActiveMenus` tag.
- The `name` Element – This element displays only the static name defined within the tag as the menu item.
- The `i18nName` Element – This element has both a `bundleName` attribute, which must map to an available `.properties` file, and a key attribute. The

bundleName attribute uses the standard Java ResourceBundle convention. The key attribute defines the key to grab within the specified bundle. The text that relates to this key within this bundle is what appears in the menu item.

- The img Element – This element adds the specified image to the left column as an icon. You must specify the path to the image file in relation to your web application.
- The bgImg Element – This element replaces the background image used in the left column with the specified image. You must specify the path to the image file in relation to your web application.
- The row Element – This element defines the CSS style or class that stylizes the row of the menu item. For the class attribute to work correctly, the class must be defined on the page (or the CSS file that defines the class must be imported).
- The text Element – This element defines the CSS style or class that stylizes the text of the menu item. For the class attribute to work correctly, the class must be defined on the page (or the CSS file that defines the class must be imported).
- The rowRollover Element – This element defines the CSS style or class that stylizes the row of the menu item when it is rolled over. For the class attribute to work correctly, you must define the class on the page (or the CSS file that defines the class must be imported).
- The textRollover Element – This element defines the CSS style or class that stylizes the text of the menu item when it is rolled over. For the class attribute to work correctly, you must define the class on the page (or the CSS file that defines the class must be imported).

Note: The TypeDefault and MenuItem tags must be contained within the type tag.

9.9.1.2 Using the ActiveMenus Tag

The taglib.tld file is located in the activemenus_taglib.jar file.

You can use the following attributes and elements with the ActiveMenus tag:

- The display Attribute – This attribute defines what appears in place of the tag itself. If you use the displayText attribute, this is the value that is passed to the method defined in the displayText tag.
- The type Attribute – This required attribute defines what is in the menu and must match a type defined in the activemenus-config.xml file.
- The href Attribute – This optional attribute can override the default anchor href for the display text of the tag.
- The newWindow Attribute – This optional href attribute specifies the link to open in a new browser window. This is a Boolean attribute, and you set it to true or false.
- The class Attribute – This optional attribute defines a CSS class for the display text.
- The style Attribute – This optional attribute defines a CSS style to place on the display text.
- The rightClick Attribute – This Boolean attribute turns the menu into a right-click menu, rather than a rollover menu. The default is false. If this

attribute is set to `true`, you right-click the display text to bring up the menu. The menu appears under the mouse.

- The `escapeXml` Attribute – This attribute is the same as `escapeXml` within the JSTL tags. If you set it to `true`, characters are converted to their corresponding character entity codes.
- The `param` Element – This element sets up parameters that can be passed in and used for the different menu items. The following two attributes are both required:
  - The `name` Attribute – This is the parameter name and must match the `name` attribute (if used) when defining a menu item in the `activemenus-config.xml` file. The `name` attribute also references the parameter within your menu item code. You can use a runtime expression.
  - The `value` Attribute – This is the parameter value, and you can use a runtime expression.

Note: If a class is specified on the tag, the default class specified in the activemenus-config.xml file is overridden and the default style is not placed on the activename. If a style is specified on the tag, the default class is placed on the activename. If a class="" is specified on the tag, the default class is not placed on the activename.

9.9.2 Enabling Placeable Movement

You can use the DragDrop JSP tag library to enable placeable movement functionality in a custom Community or a portal web application. You must identify moveable objects that are displayed on a JSP, and identify drop zones that are configured to react to a dropped moveable object. The drop zones react by triggering Page Flow actions, calling JavaScript functions, or posting data to a servlet.

Perform the following actions before you use the DragDrop tag library:

- Include the `dragdrop_taglib.jar` file in the web application's CLASSPATH
- Place the code shown in Example 9–21 into your `web.xml` file

Example 9–21 Code Entry in the web.xml File

```
<servlet>
     <servlet-name>DragDropResourceServlet</servlet-name>
     <servlet-class>com.bea.apps.communities.servlets.
          GetDragDropResourceServlet
</servlet-class>
</servlet>
<servlet-mapping>
     <servlet-name>DragDropResourceServlet</servlet-name>
     <url-pattern>DragDropResourceServlet</url-pattern>
</servlet-mapping>
```

9.9.2.1 Using the DragDrop Tags

Three tags are defined in the DragDrop tag library. Following are descriptions of how each tag is used, along with sample JSP code:

- The `dragDropScript` Tag – This tag includes the necessary DragDrop JavaScript libraries in the page. The logic embedded into the tag ensures that these libraries are included only once per request.

- The `draggableResource` Tag – This tag identifies a moveable resource on the page.
- The `resourceDropZone` Tag – This tag identifies an area on the page that reacts when a moveable resource is dropped.

9.9.2.1.1 Using the dragDropScript Tag You must include the `dragDropScript` tag before you use any other DragDrop tags on the page. This tag ensures that the appropriate JavaScript libraries are included. The `dragDropScript` tag does not take any attributes.

The following example shows how to use the `dragDropScript` tag: `<dragdrop:dragDropScript/>`.

9.9.2.1.2 Using the draggableResource Tag The `draggableResource` tag specifies a moveable resource on the page. The tag takes the following attributes:

- The `resourceId` Attribute – The unique identifier of the resource that is being moved. This identifier should be an ID that can be used by the underlying business logic to uniquely identify the resource.
- The `resourceName` Attribute – The representative name of the resource being moved.

The `draggableResource` tag performs a search for a child `img` tag that has a `dragdrop:image` attribute. This image becomes the image that is displayed while performing the placeable movement operation. The image must have an absolute height and width attribute.

The `resourceId` value is accessible through the JavaScript function `getSourceId()`, when the value is dropped onto a `resourceDropZone`. The `resourceId` value is also available as a parameter in the request named `sourceId`, when it is dropped onto a `resourceDropZone` that triggers a `POST` action. See Example 9–22.

Example 9–22 The sourceId Request Dropped onto a resourceDropZone

```
<dragdrop:draggableResource imageId="0" resourceId="${id}"resourceName=
    "${name}">
     <img src="/image.gif" width="16px" height="16px"dragdrop:image="true"/>
         ${name}
</dragdrop:draggableResource>
```

9.9.2.1.3 Using the resourceDropZone Tag The `resourceDropZone` tag identifies an area where moveable resources can be placed.

The tag takes the following attributes:

- The `targetId` Attribute – The unique identifier of the drop zone object. This identifier can be an ID that can be used by the underlying business logic to uniquely identify which object received the drop action.
- The `jsFunctionCall` Attribute – A JavaScript function that executes when a `draggableResource` is dropped on this `resourceDropZone`.
- The `pageFlowAction` Attribute – A valid Page Flow action that is initiated when a `draggableResource` is dropped on this `resourceDropZone`.
- The `formAction` Attribute – A valid JSP or servlet that receives a `POST` action when a `draggableResource` is dropped on this `resourceDropZone`.

Only one of the following attributes is required: `jsFunctionCall`, `pageFlowAction`, or `formAction`. The `jsFunctionCall` takes precedence, then `pageFlowAction`, and finally `formAction`.

The `targetId` value is accessible through the JavaScript function `getTargetId()` when a moveable resource is placed. It is also available as a parameter in the `targetId` request when a moveable resource is placed that triggers a POST action. The following code shows how this works:

```
<dragdrop:resourceDropZone targetId="${id}" pageFlowAction="moveIssue">
     <img src="/folder.gif"/>Issues Folder
</dragdrop:resourceDropZone>
```

Example 9–23 demonstrates how the `moveIssue` action can be coded in a file called `IssuesPageFlowController.java`.

Example 9–23 Coding the moveIssue Action

```
@Jpf.Action(forwards={ @Jpf.Forward(name = "success", path =
     "displayIssuesTree.do")})
     protected Forward moveIssue() {
          Forward forward = new Forward("success");
          String sourceId = getRequest().getParameter("sourceId");
          String targetId = getRequest().getParameter("targetId");
          move(sourceId, targetId);
          return forward;
     }
```

9.9.3 Enabling Dynamic Content

You can use the `DynamicContent` tag library to quickly update parts of a JSP page in a custom Community or a portal web application.

> **Tip:** See also the *Oracle Fusion Middleware Client-Side Developer's Guide for Oracle WebLogic Portal* for information on developing rich, interactive applications.

The `DynamicContent` tags let you use an AJAX request to update part of a JSP page within a Page Flow-based portlet. The tags allow parts of the page to be updated without performing a full portal request. These AJAX requests are smaller and faster than full portal requests, and therefore provide a more responsive user experience when interacting with a portal application.

> **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

These tags are easy to incorporate into standard Page Flow-based portlet development and can help create advanced user interface features that improve a user's portal experience.

Note: The DynamicContent tags are not related to Asynchronous Portlet Content Rendering. Asynchronous portlets allow for the entire portlet content to be rendered independently of the portal. The DynamicContent tags are designed to affect small parts of a JSP page within a portlet.

9.9.3.1 Understanding the DynamicContent Tags

This section describes the main tags in the `DynamicContent` tag library.

9.9.3.1.1 The Container Tag The `Container` tag designates a place on the JSP page that contains the HTML output from the execution of a Page Flow action. The only required attribute for this tag is a container id. This id is referenced by other `DynamicContent` tags to identify the container. The following code shows how this tag is used: `<dc:container dcContainerId="outputContainer"/>`.

9.9.3.1.2 The Container Action Script Tag This tag is a child of the `Container` tag and identifies a Page Flow action that can be executed and whose HTML output is placed inside the parent container. The `containerActionScript` tag takes the following attributes:

- The `action` attribute – The Page Flow action name.
- The `initial` attribute – Designates an action in the container as the initial action. This is the action that initially populates the container.
- The `async` attribute – Specifies if the action is performed synchronously or asynchronously. The default is synchronous.
- The `onErrorCallback` Attribute – A user-defined JavaScript function that is called if a client-side error occurs during the AJAX request creation and processing.

Only the `action` attribute is required. The following code sample shows how this tag is used in the parent `Container` tag:

```
<dc:container dcContainerId="outputContainer">
     <dc:containerActionScript action="resetDynamicContentContainer"
          initial="true"/>
     <dc:containerActionScript action="showServerTime"/>
<dc:container/>
```

9.9.3.1.3 The Execute Container Action Tag The Execute Container Action tag is used to create a call to a specific action inside a container. This tag takes the following attributes:

- The `dcContainerId` attribute – The id of the container in which the action is defined.
- The `action` attribute – The Page Flow action name.
- The `async` attribute – This specifies if the action is performed synchronously or asynchronously. The default is synchronous.
- The `var` attribute – A request attribute variable that holds a reference to the action JavaScript call.

The `dcContainerId` and `action` attributes are required. Following is a sample of how this tag is used:

```
<dc:executeContainerAction action="showServerTime" dcContainerId=
```

```
     "outputContainer"
          var="showServerTimeVar"/>
```

In the previous example, the call to the specified action is stored in the variable `showServerTimeVar`. This variable can then be referenced, as shown in the following HTML code:

```
<form>
     <input type="button" onclick="${showServerTimeVar}" value="Show Server
          Time"/>
</form>
```

When the user clicks a button, an AJAX request is created that executes the `showServerTime` action and places the HTML output generated by that action into the container with the id of `outputContainer`.

9.9.3.1.4 The Parameter Tags The `DynamicContent` tags also include tags for parameters that are passed into the action through the request. You can define parameters within the `executeContainerAction` tag or the `containerActionScript` tag. These parameters are then accessible in the Page Flow action by calling the `request.getParameter()` method.

9.9.3.2 Using the DynamicContent Tags

Some critical limitations are associated with the `DynamicContent` tags. The AJAX requests used to trigger the Page Flow actions are not processed through the main portal servlet. These requests go through a special servlet that performs some processing to ensure that the proper Page Flow instance is used. Many key elements that are normally available in the request are not accessible from these AJAX requests. For example, in Community-based portal applications, the `CommunityContext` object is not accessible from the AJAX request. The lack of access to some of these framework elements could have an impact on things like entitlements and security.

Because of these limitations, the `DynamicContent` tags are best suited for specific uses that involve small amounts of processing, with few dependencies on larger framework services. The following use cases could benefit from the `DynamicContent` tags:

- Update a small location on a JSP page to display frequently updated data obtained through periodic client-side polling. For example, you could notify users of unread mail or display the number of users logged onto a system.
- Use the tags as a pagination mechanism for tabled data presented across multiple pages.
- Send multiple requests to the server to obtain successive images to navigate through a series of images in a photo gallery. The `DynamicContent` tags provide a tool to avoid an expensive portal request to view each photo.
- Obtain remote data, such as stock quotes or weather information from remote sites. The obtained data can be displayed in a designated area on the page without updating other parts of the page.

9.9.4 Using the User Picker

During the Development phase, you can use the `UserPicker` tag library to add a form button to a JSP page in a custom Community or a portal web application.

The `UserPicker:popupButton` tag provides the developer with the ability to add a form button to a JSP page which opens a popup window that displays a list of current

users. You can select a user from this list. The name of the selected user is populated into a specified form field on the parent window.

9.9.4.1 Using the UserPicker Tags

This section describes the `UserPicker:popupButton` tag in a custom Community and how to use the following attributes:

- The `inputId` Tag – The id of the HTML form input element that is populated with the selected user's name. This tag is optional.
- The `inputTagId` Tag – The `tagId` of the netui-based form input element that is populated with the selected user's name. If the `inputId` tag is provided, the `inputTagId` tag is ignored. This tag is optional.
- The `buttonImage` Tag – The src path to the image for the popup button. This tag is required.
- The `atnProviderName` Tag – The Authentication Provider name. If an `atnProviderName` is supplied, there is no provider drop-down box in the popup window. If an `atnProviderName` is not supplied, the default provider is used. If you have configured multiple Authentication Providers, a drop-down box appears in the popup window to allow you to specify a provider. This tag is optional.

> **Tip:** When the UserPicker:popupButton tag is used in a Community, the Community members are listed, rather than users.

9.10 Detached Portlets

WebLogic Portal supports the use of detached portlets, which provide popup-style behavior. Technically, a detached portlet is defined as anything outside of the calling portal context. Any portlet type supported by WebLogic Portal can be rendered as a detached portlet.

Note: Opening the same portal desktop in multiple browser windows that share the same process (and, therefore, the same session) is not supported. For example, using the `render:pageURL` tag or the JavaScript `window.open` function to open a portal desktop in a new window is not supported and can result in unwanted side effects. For instance, if a portlet is deleted in the new window, and then the user tries to interact with that portlet in the main window, the interaction will fail.

9.10.1 Considerations for Using Detached Portlets

Keep the following considerations in mind as you implement detached portlets:

- Detached portlets are never referenced from within a portal; there is no portlet instance in the portal associated with a detached portlet.
- The detached or "pop-up" portlet feature is not supported for remote (WSRP) portlets.
- Detached portlets can be streamed but generally cannot be entitled or customized; the library instance can be entitled, but portlet instances that are de-coupled from the portlet library cannot. For more information about library portlet instances and de-coupling, refer to the *Oracle Fusion Middleware Production Operations Guide for Oracle WebLogic Portal*.

- Detached portlet are not visible or accessible from the WebLogic Portal Administration Console portlet library.
- In a streamed portal, the primary instance of the portal is used. In some cases, the primary instance cannot be determined; for example, you might have set entitlements on the primary instance to make it not viewable, or you could have set up a configuration that excludes portlets from the scanner and poller so that they are not streamed into the database. If the primary instance cannot be determined, a static version of the portlet is used (the portlet will be served in file mode). In these cases, some features related to a streamed portal (such as a community context) will not be available, and applications might be required to implement workarounds.
- Although technically a detached portlet can be implemented to use asynchronous rendering, this is not a best practice and is not recommended.
- No presentation mechanism is provided as part of the detached portlet feature; the application must define how to actually present the portlet. For example, a floated portlet will automatically be popped up in a separate window; detached portlets have no such mechanism, so your application must handle popping up the window.
- When developing detached portlets, you can place them anywhere in the hierarchy of your portal web application; the portal references the absolute path to the portlet.
- The framework for standalone portlets creates a "dummy" control tree above the portlet, including desktop, book, and page controls. The context objects associated with such controls reflect the state of the dummy controls, and not of the main control tree; for example, if a portlet tries to get information about its current book or page, the Book/Page Presentation/Backing Context objects will not reflect the actual structure of the portal. There might also be cases where the dummy control tree does not support certain backing context APIs. When developing your portal, you need to keep this artificial control tree structure in mind.

9.10.2 Building Detached Portlets

You use the `standalonePortletUrl` class or associated JSP tag to create URLs to detached portlets.

To create a detached portlet URL from a JSP page, you use the render:standalonePortletUrl JSP tag or class; the following example shows the syntax of the JSP tag:

```
<render:standalonePortletUrl portletUri="/absolute_path/detached_portlet_
name.portlet" .../>
```

To create a detached portlet URL from Java code, use the following example as a guide:

```
StandalonePortletURL detachedURL =
StandalonePortletURL.createStandalonePortletURL(request, response);

detachedURL.setPortletUri("/path/to/detached.portlet");
```

9.11 Working with Inlined Portlets

A file-based portlet can exist either as a stand-alone `.portlet` file or as an inlined portlet. Typically, within the Oracle Enterprise Pack for Eclipse portal editing

framework, `.portlet` files are included in portals by reference. For instance, when you drag a `.portlet` file onto a portal, page or book, a reference is created to the portlet file inside the portal, page, or book. On the other hand, an inlined portlet's entire XML definition is embedded directly in a page or book.

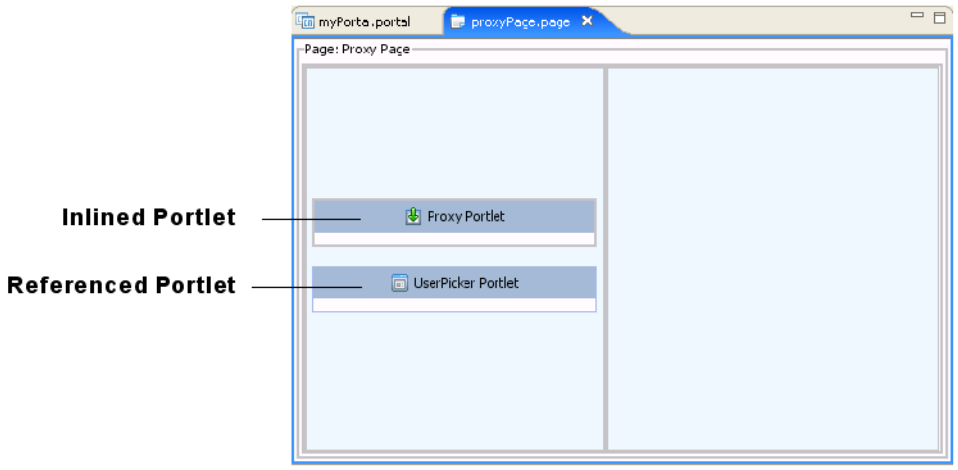
Inlined portlets are created under the following circumstances:

- If you create a remote book or page that contains portlets, those portlets will be inlined in the `.book` or `.page` file. For detailed information on creating remote books and pages, see the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.
- If you use the Export/Import Utility to extract a `.book` or `.page` file that contains portlets, those portlets will be inlined if they were originally inlined. If the original page contained referenced portlets, they will be referenced when the page is extracted. For detailed information on the Export/Import Utility, see the *Oracle Fusion Middleware Production Operations Guide for Oracle WebLogic Portal*.

You can drag and drop, cut, copy, and paste inline portlets from one page or book to another from within the portal editor.

Figure 9–28 shows a remote page that contains an inlined portlet and a referenced portlet. Note that the icon used in an inlined portlet is distinct from a referenced portlet.

Figure 9–28 Inlined Portlet in the Portlet Editor



Tip: You can edit the properties of inlined portlets exactly like file-based portlets; however, portlet states and modes are not editable for inlined portlets.

9.11.1 Extracting Inlined Portlets

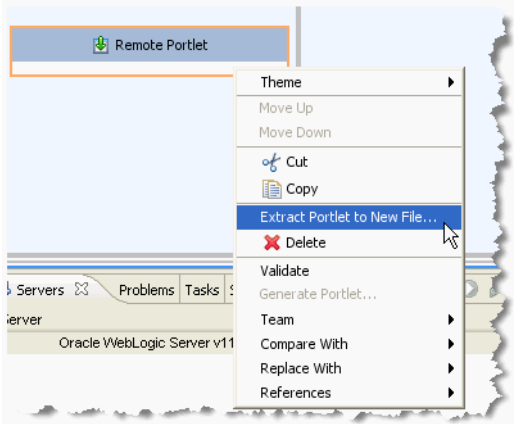
You can export an inlined portlet to a `.portlet` file. When you do this, the resulting `.portlet` file is functionally equivalent to any other `.portlet` file. When you extract an inlined portlet, the inlined portlet XML code is automatically removed from the source file (a page or book) and replaced with a reference to the newly created `.portlet` file.

Note: After you extract an inlined portlet, you can undo the operation (re-inline the portlet). However, note that the .portlet file that was created during the extraction will not be deleted from your system. The source document will simply not reference the .portlet file any longer.

To extract an inlined portlet, do the following:

1. Right-click the inlined portlet in the Book or Page Editor and select **Extract Portlet to New File**, as shown in Figure 9–29.

Figure 9–29 Extract Portlet to New File



2. In the Save As dialog, enter a name for the new portlet.

Tip: **Project** > **Clean** command.

9.11.2 Setting the Theme of an Inlined Portlet

You can set the theme of an inlined portlet exactly as you would for a referenced portlet. To set the theme, right-click the inlined portlet in the book or page editor, and pick a theme from the Theme menu. The theme is retained for that portlet as long as it remains referenced in the page or book.

9.12 Extracting Books and Pages

You can extract any book or page in a portal to a `.book` or `.page` file. Once a book or page is extracted, you are free to use it in another portal within the same portal web application if you wish.

The procedure for extracting books and pages is similar to the procedure for extracting inlined portlets, described in Section 9.11.1, "Extracting Inlined Portlets." To extract a book or page, do the following:

1. Right-click border of the book or page in the Portal Editor and select **Extract Book (or Page) to New File**.
2. In the Save As dialog, enter a name for the new book or page file.

Tip: Any theme applied to a book or page is retained for an extracted book or page as long as the book or page remains referenced in the portal.

9.13 Avoiding Committing Responses

The WebLogic Portal framework generally uses a non-buffered model for rendering portlets, meaning that each portlet renders directly to the underlying portal response. This generally scales and performs better than a fully buffered response.

Because individual portlets write to the underlying portal HttpServletResponse, the actions of individual portlets or portlet components may commit the response. For example, simply flushing an outputstream or writer, as is frequently automatically done for JSPs and simple println() calls, will automatically commit the response. Once the response is committed, certain operations such as setting cookies become impossible for other portlets when they render.

In general, the best practice for portlet developers in WebLogic Portal is to perform such operations during the pre-render life cycle (such as in a backing file's preRender() method), but this may not always be possible, for example if JSR168 or JSR286 portlets are being used, or if the portlet is accessed over WSRP.

To allow portlets to set cookies and headers during the render life cycle, you can use a setting in the `WEB-INF/wlp-framework-common-config.xml` file. If you set the `<avoid-response-commit>` element to `true`, a response wrapper will be put on the base portal response to avoid committing the response as long as possible.

This response wrapper is used when rendering portal pages and ignores calls to flushBuffer(). The wrapper also ensures that flushes of the response output stream or response writer do not automatically commit the response. The response may still be automatically committed when the output exceeds the response buffer size; the optional bufferSize attribute can be used to set a response buffer size other than the servlet container's default buffer size.

When using this configuration option for non-Java portlets, use the PortletPresentationContext object's addCookie() and addHeader() methods to add cookies and headers on the response from portlets during the render life cycle. You can do this as long as the response buffer has not overflowed and forced the response to commit. Cookies or headers set on the HttpServletResponse (instead of the PortletPresentationContext) are generally ignored during the render phase of the portlet life cycle.

For Java portlets' (JSR168/JSR286), the PortletResponse addCookie() and addProperty() methods automatically set the cookies or headers on the underlying response, as long as the response buffer has not overflowed and forced the response to commit.

For WSRP portlets returning cookies or headers during a getMarkup operation, the framework will also automatically set the cookies or headers on the underlying response. If using a WSRP interceptor on a getMarkup operation, the PortletPresentationContext methods must be used to set cookies or headers on the consumer's response.

10 Optimizing Portlet Performance

The process of optimizing your portlets for the best possible performance spans all phases of development. You should continually monitor performance and make appropriate adjustments.

This chapter describes performance optimizations that you can incorporate as you develop portlets.

This chapter contains the following sections:

10.1 Performance-Related Portlet Properties

Customizing performance-related portlet properties can help you improve performance. For example, you can set process-expensive portlets to pre-render or render in a multi-threaded (forkable) environment. If a portlet has been designed as forkable (multi-threaded) you have the option of adjusting that setting when building your portal.

The following portlet properties are performance related:

- Render Cacheable/Cache Expires
- Forkable/Fork Render/Fork Render Timeout
- Fork Pre-Render/Fork Pre-Render Timeout
- AsyncContent

Section 9.1, "Portlet Properties" includes descriptions of these properties. If you design your portlets to allow portal administrators to adjust cache settings and rendering options, you can modify those properties in the Administration Console.

10.2 Portlet Caching

You can cache the portlet within a session instead of retrieving it each time it recurs during a session (on different pages, for example). Portlets that call web services perform frequent, expensive processing; caching web service portlets greatly enhances performance. Portlet caching is well-suited to caching personalized content; however,

it is not well suited to caching static content that is presented identically to all users and that rarely expires.

The ideal use case of the portlet cache is for content that is personalized for each user and expires often. In other situations, it might be more beneficial to use other caching alternatives. For more information and a detailed examination of the *Render Cacheable* property and a discussion of when you should or should not use it, refer to *"Portlet Caching,"* at http://www.oracle.com/technology/pub/articles/dev2arch/2005/01/portlet_caching.html. See also the *Oracle Fusion Middleware Cache Management Guide for Oracle WebLogic Portal*.

10.3 Remote Portlets

Remote portlets are made possible by Web Services for Remote Portlets (WSRP), a web services standard that allows you to "plug-and-play" visual, user-facing web services with portals or other intermediary web applications. WSRP allows you to consume applications from WSRP-compliant Producers, even those far removed from your enterprise, and surface them in your portal.

While there might be a performance boost related to the use of remote portlets, it is unlikely that you would implement them for this reason. The major performance benefit of remote portlets is that any portal framework controls within the application (portlet) that you are retrieving are rendered by the producer and not by your portal. The expense of calling the control life cycle methods is borne by resources not associated with your portal.

Note: Fetching data from the producer can be expensive. You need to decide if that expense is within reason given the resources locally available.

If the expense of portal rendering sufficiently offsets the expense of transport and the other limitations described above are inconsequential to your application, using remote portlets can provide some performance boost to your portal.

For more information on implementing remote portlets with WSRP, refer to the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

10.4 Portlet Forking

Portlet forking allows portlets to be processed on multiple threads. Depending on the available server resources, this means that the portal page will refresh more quickly than if all portlets were processed sequentially. Forking is supported for JSP, Page Flow, Java, and WSRP portlets (consumer portlets only).

Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Note: Although using this feature might reduce the response time to the user in most situations, on a heavily loaded system it can actually decrease overall throughput as more threads are being used on the server/JVM for each request—adding to contention for shared resources.

This section includes these topics:

- Section 10.4.1, "Configuring Portlets for Forking"
- Section 10.4.2, "Architectural Details of Forked Portlets"
- Section 10.4.3, "Best Practices for Developing Forked Portlets"

10.4.1 Configuring Portlets for Forking

Forking is easy to enable – you just set properties using the portlet Properties editor in Oracle Enterprise Pack for Eclipse, as shown in Figure 10–1. The available forking properties are described in this section. For detailed information on the Portlet Properties editor, see Section 9.1, "Portlet Properties."

Figure 10–1 Forking Properties

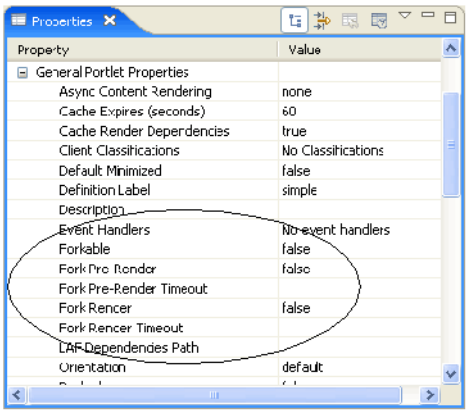


Table 10–1 Portlet Forking Properties

Property	Value
Forkable	This property must be set to true if you want the portlet to be forked. This property identifies the portlet as safe to run forked. If this attribute is false (the default), the portlet will not be forked regardless of the settings of the other two forking properties. See Section 10.4.3, "Best Practices for Developing Forked Portlets" for tips on developing forked portlets. When set to true, a portal administrator can use the **Run the Portlet in a Separate Thread** property. If set to false, that property is not available to administrators. See the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal* for information on using the Administration Console to edit portlet properties.
Fork Pre-Render	Enables forking (multi-threading) in the pre-render life cycle phase. For an overview of the portal life cycle, see Section 10.4.2, "Architectural Details of Forked Portlets." See also "How the Control Tree Affects Performance" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal* for more information about the control tree life cycle. Setting Fork Pre-Render to `true` indicates that the portlet's pre-render phase should be forked. See Section 10.4.2.2.1, "Dispatching Pre-Render Forked Portlets to Threads" for more information on the pre-render phase.

Fork Pre-Render Timeout (seconds)	If Fork Pre-Render is set to `true`, you can set an integer timeout value, in seconds, to indicate that the portal framework should wait only as long as the timeout value for each fork pre-render phase. The default value is -1 (no timeout). If the time to execute the forked pre-render phase exceeds the timeout value, the portlet itself times out (that is, the remaining life cycle phases for this portlet are cancelled), the portlet is removed from the page where it was to be displayed, and an error level message is logged that looks something like the following example.
	
	`<May 26, 2005 2:04:05 PM MDT> <Error> <netuix>`
	
	`<BEA-423350> <Forked render timed out for portlet`
	
	`with id [t_portlet_1_1]. Portlet will not be included in response.>`

Fork Render	Setting to `true` tells the framework that it should attempt to multi-thread render the portlet. This property can be set to true only if the `Forkable` property is set to `true`. See Section 10.4.2.2.2, "Dispatching Render Forked Portlets to Threads" for more information on the render phase.
Fork Render Timeout (seconds)	If Fork Render is set to `true`, you can set an integer timeout value, in seconds, to indicate that the portal framework should wait only as long as the timeout value for each fork render portlet. The default value is -1 (no timeout). When a portlet rendering times out, an error is logged, but no markup is inserted into the response for the timed-out portlet. Selecting a value of `0` or `-1` removes the timeout attribute from the portlet; use this value if you want to revert to the framework default setting for this attribute.

The forking properties, if set, appear as XML elements a `.portlet` file. Example 10–1 shows a sample of a portlet configured for both pre-render and render forking:

Example 10–1 Forking Properties Set in a .portlet File

```
<netuix:portlet title="Forked Portlet"
    definitionLabel="forkedPortlet1"
    forkable="true"
    forkPreRender="true"
    forkRender="true">
    <netuix:content>
        <netuix:jspContent contentUri="/portlets/forked.jsp"
        backingFile="backing.PreRenderBacking"/>
    </netuix:content>
```

```
</netuix:portlet>
```

10.4.2 Architectural Details of Forked Portlets

Generally, forking is easy to understand and to enable. However, with a deeper understanding of how forking works, you can avoid potential problems and unwanted side effects. This section discusses the architectural design of forked portlets. For specific implementation tips, see Section 10.4.3, "Best Practices for Developing Forked Portlets."

This section includes these topics:

- Section 10.4.2.1, "Understanding Request Latency and the Portal Life Cycle"
- Section 10.4.2.2, "Queuing and Dispatching Forked Portlets for Processing"
- Section 10.4.2.3, "Threading Details and Coordination"
- Section 10.4.2.4, "Forking Versus Asynchronous Rendering"

10.4.2.1 Understanding Request Latency and the Portal Life Cycle

For most requests to the portal, the total time to process the request, or *request latency*, is roughly related to the time needed to run through the portal life cycle phases successively for all the portlets. Each life cycle phase is performed by walking through a tree of objects, called the *control tree*, that make up the portal. Each phase is essentially a depth-first walk over the tree, where the root of the tree is the desktop, and the leaves of the tree are the books, pages, portlets, and other so-called controls. Figure 10–2 illustrates the general structure of a portal control tree.

Figure 10–2 Simple Portal Schematic Example

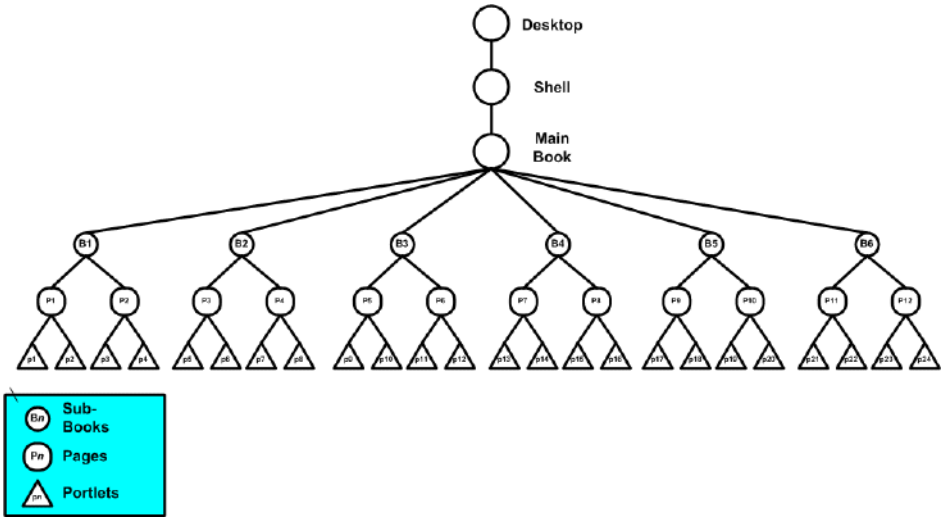
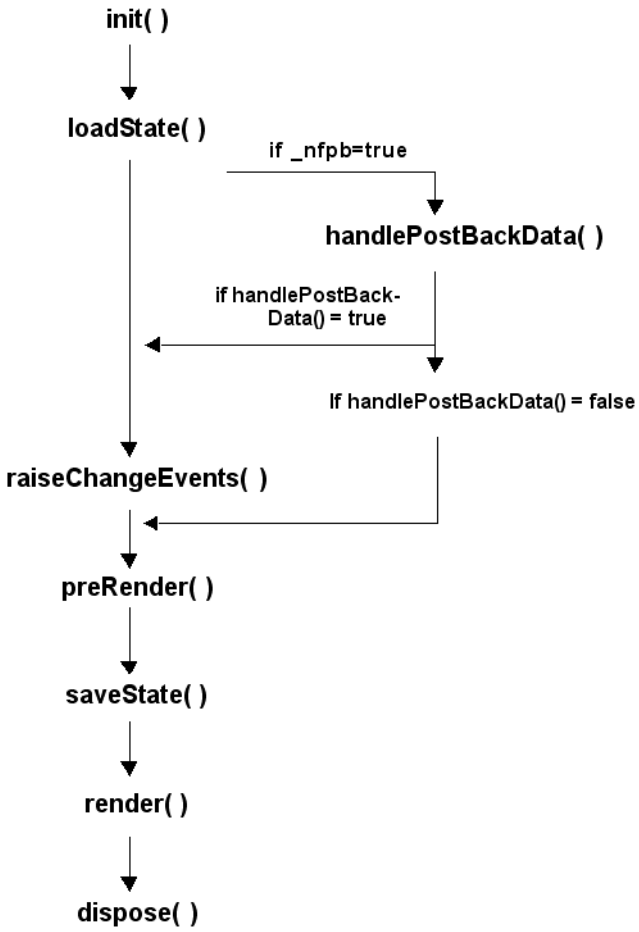


Figure 10–3 illustrates the successive phases of the portal rendering life cycle. During the first traversal of the control tree, the init() method is called on each control. On the second traversal, loadState() is called, and so on, until every control is processed.

Typically, portlet processing time is dominated by the execution of business logic, especially if the portlets must access remote resources such as databases or web services, or if they are computationally intensive. Forking allows you to parallelize some of these longer running portlet operations to decrease the overall request latency. If forking is enabled, these operations are collected in a queue and dispatched

to multiple threads for processing. Depending on your server's resource availability, forking can theoretically reduce request latency to the maximum latency of any of the forked portlets.

Figure 10–3 Flow of Portal Life Cycle Methods



10.4.2.2 Queuing and Dispatching Forked Portlets for Processing

During the *pre-render phase* of the portal life cycle, all portal controls are iterated and pre-rendering operations are executed. Any portlets that are marked for *either* pre-render or render forking are identified during this pass and, if they are marked for forking, they are placed in separate queues: a pre-render queue and a render queue. (See Section 10.4.1, "Configuring Portlets for Forking" for details on how to mark portlets for pre-render and render forking.)

At the appropriate times, these queues are dispatched to threads and processed, as explained in the following sections. See also Section 10.4.2.3, "Threading Details and Coordination."

10.4.2.2.1 Dispatching Pre-Render Forked Portlets to Threads In the pre-render phase of the portal life cycle, portlets typically perform business logic, typically by handling postback data or by calling a backing file method, such as the AbstractJSPBacking.preRender() method.

During normal pre-render processing of the portal, any portlet that is marked for pre-render forking is placed into a queue and the pre-render processing is skipped.

After the entire pre-render phase has been performed, the queue is inspected. If it is not empty, the queue is dispatched and the portlets in the queue are assigned to a worker thread. After the queue is fully dispatched, the main portal thread waits until either all the worker threads are completed or timed out.

10.4.2.2.2 Dispatching Render Forked Portlets to Threads In some cases, business logic is performed during the render phase of the portal life cycle, typically when JSP scriptlets are used.

Before running through the render life cycle, the render queue is examined. If it is not empty, the queue is dispatched and any portlets in the queue are assigned to worker threads. As with pre-render forking, the main portal thread waits until all of the render threads are either completed or timed out. The resulting buffered response from each thread is saved for each completed forked portlet. At this point, the actual render life cycle phase is run. When a portlet is encountered that was marked for forking, the render processing is skipped and the saved buffered response data for the portlet is written to into the response.

Some types of portlets, notably Struts or Page Flow portlets, provide a mapping between the underlying application technology and the portal life cycle model. Usually in these cases, actions are provided to handle business logic during the handle postback or pre-render phases of the life cycle.

Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

10.4.2.3 Threading Details and Coordination

The worker threads used by the forking feature are implemented as WLS WorkManager classes (commonj.work.WorkManager). WebLogic Portal does not directly allocate any threads; rather, a WorkManager is identified by its JNDI name. If found, the WorkManager is used to dispatch the worker threads (Work instances). The default WorkManager for dispatching forked portlets is called wm/forkedRenderQueueWorkManager, with a default called wm/Default. If you need to customize the WorkManager for any reason, you can specify an alternate instance through the weblogic.xml or weblogic-config.xml file by associating the alternate instance with the JNDI name wm/forkedRenderQueueWorkManager. See also Section 10.4.3.1, "Consider Thread Safety."

The framework uses a ForkedLifecycleContext object to coordinate between the mainline life cycle thread and the forked Worker instances. During initialization of a Worker, the ForkedLifecycleContext is created and registered with the forking dispatch queue. When the Work instance has completed, the ForkedLifecycleContext is set to completed and the waiting mainline thread is notified. Alternately, if the waiting mainline thread determines that the forked Work instance is taking too long and should be timed out, the ForkedLifecycleContext is marked as timed out and the Work instance is removed from the dispatch queue. Note that in this case, the Work item is not aborted, and will keep running until the portlet code being run for either the pre-render or render phase is completed. You can obtain the current ForkedPreRenderContext or ForkedRenderContext using a utility method on those classes from the request. You can then check if a timeout has been set to detect cases

where the Worker thread was timed out by the portal framework and should be aborted.

10.4.2.4 Forking Versus Asynchronous Rendering

Regardless of whether or not you use render forking, the portal does not render until all portlets complete rendering. If you want portlets to render individually, you can use asynchronous portlet rendering.

Asynchronous portlet content rendering refers to page processing that occurs on the client browser; multiple threads are spawned, using AJAX or IFRAME technology. Asynchronous portlet rendering allows the contents of a portlet to render independently from the surrounding portal page. This can provide a significant performance boost; for example, when a portal visitor works within a portlet, only that individual portlet needs to be redrawn.

Caution: Using forked rendering with asynchronous portlet content rendering is unnecessary, is not recommended, and could result in unexpected behavior.

For details on asynchronous rendering, see Section 10.5, "Asynchronous Portlet Content Rendering." For a comparison of portlet forking and asynchronous rendering, see Section 10.5.6, "Comparison of Asynchronous and Conventional or Forked Rendering."

10.4.3 Best Practices for Developing Forked Portlets

This section discusses three primary issues you need to consider when developing forked portlets: thread safety, runtime environment, and interportlet communication issues.

10.4.3.1 Consider Thread Safety

Although the portal framework handles thread safety issues that affect the framework itself, any code you write that is intended to be used in forked portlets should be written in a threadsafe manner.

- Only mark thread-safe portlets as forkable. This helps to ensure that administrators do not incorrectly enable forking for portlets that were not written with thread safety in mind.
- Cautiously evaluate interactions between your code and portal framework constructs. For example, do not unwrap the request and response objects. They are used specifically to isolate the request and response. For certain types of portlets, particularly Page Flow and Struts portlets, an additional wrapper is put in place, so one level of unwrap may work, but unwrapping to the root request or response will cause threading issues.
- Avoid using portal-managed objects, such as the request and response, for your own code synchronization. These objects are used by the portal framework for synchronization. If you use them for that purpose, out of order lock acquisition and deadlocks can occur.

Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

10.4.3.2 Consider the Runtime Environment for Forked Portlets

When designing forked portlets, try to maximize their independence from other constructs in the portal (such as BackingContext) and from other portlets. Such dependencies create problems for forked portlets because forked portlets are inherently isolated from the runtime environment.

10.4.3.2.1 Isolation of Forked Portlets from the Runtime Environment The primary difference between the runtime environment for forked portlets and non-forked portlets is in their level of isolation. This difference occurs because of the way that forked portlets are collected and dispatched outside of the life cycle execution for the main portal control tree.

Each life cycle iteration of the control tree results in a life cycle method being called for that control. In this way, each control has the opportunity to perform life cycle specific business logic. Additionally, each life cycle method invocation involves both a begin and end operation, which enables setup and teardown for controls that require such functionality.

Enabling preRender or render forking moves the execution of a portlet's life cycle processing from occurring within the main portal control tree walk to outside of it. The main side effects of this are:

- The forked portlet is essentially isolated from any stateful setup that its placement in the control tree provided.
- Forked portlets are executed out of order, both in terms of other nodes in the control tree and even amongst other sibling portlets. For the preRender phase, controls deeper in depth-first order will be executed ahead of forkPrerender portlets. For the render phase, all forkRender portlets will be executed before any other control in the tree processes its render phase.

As a developer of forked portlets, be aware that code meant to be executed in a forked portlet should be as stand-alone as possible. Avoid relying on interaction with other portlets, other controls higher in the control tree, or state provided by other controls in the control tree.

Do not rely on any processing done during the same life cycle in other portlets, because forking a portlet both takes it out of order with respect to control tree execution and applies an arbitrary ordering among forked portlets in the dispatch queue.

10.4.3.2.2 BackingContext and Pre-Render Forked Portlets For preRender forked portlets, one of the main areas of concern for forked portlets is the BackingContext framework. This framework is managed in part by a stack-based implementation involving the request, which depends on Backable controls in the control tree to push and pop their BackingContext instances onto and off of the request. All of these activities happen during the pre-render life cycle phase. When writing a portlet that expects a particular BackingContext stack environment, problems can occur with Fork Pre-Render mode.

Any access to BackingContexts through the request will result in that BackingContext not being available while forked.

To work around this BackingContext issue, you can use non-contextual methods to obtain BackingContexts for other presentation controls in the control tree, but these generally involve explicit walking of the context tree, and some contexts may be unavailable because the context in question has already been cleaned up by the control that manages it in preRender.

10.4.3.3 Use Caution with Interportlet Communication and Forked Portlets

Interportlet communication (IPC) is another area of concern for forked portlets. Again, the more you can isolate a portlet's logic, the more successfully it will run in a forked environment.

IPC is performed in several different life cycles. When an IPC scenario is enabled that results in an IPC call initiated during preRender, and a portlet is also enabled for forking, that IPC will not be performed, since the actual dispatch of the IPC event queue happens immediately following the main execution of preRender() over the control tree. This is of primary concern to portlets that raise IPC events in a backing file preRender() method, from a Page Flow, a Struts begin action, or from a JSF beginning view root.

Note: Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. Apache Struts is also an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

10.5 Asynchronous Portlet Content Rendering

Asynchronous portlet rendering allows you to render the content of a portlet independently from the surrounding portal page. This can provide a huge performance boost; for example, when a portal visitor works within a portlet, only that individual portlet needs to be redrawn.

> **Tip:** You can also enable asynchronous rendering for an entire portal desktop. Both portlet-specific (as discussed in this section) and desktop asynchronous rendering offer quicker response times than synchronous rendering. Note that the portlet-specific and desktop options for asynchronous rendering are mutually exclusive features. For more information on asynchronous desktop rendering and tips on deciding which method to choose, see the chapter "Designing Portals for Optimal Performance" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

You can use either AJAX technology or IFRAME technology to implement asynchronous rendering for individual portlets. When using asynchronous portlet rendering, a portlet renders in two requests. The normal portal page request process occurs first; during this process, the portlet's non-content areas, such as the title bar, are rendered. Rather than rendering the actual portlet content, a placeholder for the content is rendered. A subsequent request process displays the portlet content.

This section contains the following topics:

- Section 10.5.1, "Implementing Asynchronous Portlet Content Rendering"

- Section 10.5.3, "Considerations for IFRAME-based Asynchronous Rendering"
- Section 10.5.4, "Considerations for AJAX-based Asynchronous Rendering"
- Section 10.5.5, "Comparison of IFRAME- and AJAX-based Asynchronous Rendering"
- Section 10.5.6, "Comparison of Asynchronous and Conventional or Forked Rendering"
- Section 10.5.8, "Asynchronous Content Rendering and IPC"

10.5.1 Implementing Asynchronous Portlet Content Rendering

The portlet property `asyncContent` in the Properties view allows you to specify whether to use asynchronous rendering, and to select which technology to use. An editable drop-down menu provides the selections `none`, `ajax`, `iframe`, and `iframe_unwrapped`. If you want to create a customized implementation of asynchronous rendering, you can do so by editing the `.portlet` file to set this up.

> **Note:** The `iframe_unwrapped` value is used for interoperability with WebCenter 10g ADF Faces portlets. You must use the `iframe_unwrapped` value if you are consuming (through WSRP) a WebCenter 10g ADF Faces portlet in a WebLogic Portal. Using this value prevents potential rendering problems by wrapping the ADF Faces portlet in an IFrame, while explicitly excluding WebLogic Portal-specific markup from rendering within the IFrame. For more information on WSRP interoperability between WebCenter and WebLogic Portal, see the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

Portlet files that do not contain the `asyncContent` attribute appear with the initial value `none` displayed in the Properties view. Any changes to the setting are saved to the `.portlet` file.

> **Note:** Although Browser portlets use an internal implementation that appears similar to that of an asynchronous portlet and both portlet types use IFRAME HTML elements, the actual implementations are quite different. Browser portlets are merely displaying static embedded documents, but asynchronous IFRAME portlets are managed by the framework.

Keep the following global considerations in mind for any asynchronous rendering implementation:

- As a best practice, do not depend on the built-in navigation features (Back and Forward buttons) of a browser. Build navigation into your portlets so that navigation is handled at that level of interaction.

  If navigation is handled by the browser, the behavior of a portlet will vary according to the type of asynchronous rendering technology used, and this inconsistency can be confusing to the end user. For example, with IFRAME technology each portlet interaction is tracked, but this interaction does not update the "view" from the server's perspective; if the user clicks the Back button, the server takes the user to a state preceding the interaction with the portlet.

- The initial (completion of) page load for an asynchronously rendered portlet page will be longer because, for example, loading a page containing five asynchronous portlets entails six requests to the server. However, because the portal page begins to load quickly, the user might perceive a faster load time even if the completion takes more time overall.
- You should pre-define portlet sizes using Look & Feel settings; otherwise, as the page loads, the portlets might resize several times as they adjust to their arrangement on the page.
- Portlet backing files are run twice: once for the outer (portal) request and another for the inner (content) request. You can use the set of framework APIs in the PortletBackingContext class to distinguish between these two requests; for more information, refer to the Javadoc information for these APIs:

  com.bea.netuix.servlets.controls.portlet.PortletPresentationContext.isAsyncContent()

  com.bea.netuix.servlets.controls.portlet.PortletPresentationContext.isContentOnly()

  com.bea.netuix.servlets.controls.portlet.backing.PortletBackingContext.isAsyncContent()

  com.bea.netuix.servlets.controls.portlet.backing.PortletBackingContext.isContentOnly()
- Asynchronous portlet rendering can be used with control tree optimization. Most of the best practices for control tree optimization also apply to the design of asynchronous rendering. For more information on control tree optimization, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
- Interportlet communication is not supported when asynchronous content rendering is enabled; however, you can temporarily disable asynchronous rendering in specific situations if needed. For details, refer to Section 10.5.8, "Asynchronous Content Rendering and IPC." If you require interportlet communication, consider using asynchronous desktop rendering, as described in the chapter "Designing Portals for Optimal Performance" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
- HTTP redirects are not supported when asynchronous content rendering is enabled; however, you can temporarily disable asynchronous rendering using the same mechanisms as those described in Section 10.5.8, "Asynchronous Content Rendering and IPC."
- Using forked pre-rendering or forked rendering in an asynchronous portlet is unnecessary and in any case is not recommended, and although this is not an error condition, it could result in unexpected behavior.
- Using PostbackURLs (not derived types) within an asynchronous portlet (or a floated portlet) causes the portlet to lose various aspects of its state, including the results of render caching. Additionally, multiple instances of such portlets will begin to share state. To avoid this issue, you can use one of these workarounds:
  - Use alternative mechanisms for generating URLs more appropriate to the portlet type, such as `<render:jspContentUrl>` or `<netui:anchor>`.
  - Add `GenericURL.WINDOW_LABEL_PARAM` directly to the PostbackURL with the value returned from `PortletPresentationContext.getLabel()` or `PortletBackingContext.getLabel()`.

- WebLogic Portal allows portlets to change the current window state or mode of a portlet either programmatically, or using parameters added to URLs. When you enable asynchronous rendering for a portlet, these mechanisms will not provide a consistent view to the end user; for example, the title bar rendered above the portlet will not immediately reflect the change in the mode or state.
- In addition to the issues described in Section 10.5.8, "Asynchronous Content Rendering and IPC," you must carefully consider the implications whenever a portlet tries to communicate with the portal (or the portal communicates with the portlet). For example, suppose a portlet or JSP places data in the request for the *doobie* portlet to process; if portlet *doobie* is asynchronous, it is running on a different request and will never see the data. Because of this behavior, there will be cases when you should not use asynchronous portlets in your implementation.

10.5.2 Thread Safety and Asynchronous Rendering

If you use asynchronous portlet content rendering, be sure that your code (for example, in backing files) is thread safe. The portal framework handles the major issues outside of a developer's control, such as concurrent access to the request and response; and it manages coordination of issues such as waiting for all async operations to finish and assembling the results in the correct order. But the portlet developer has the responsibility for ensuring that the user code is thread safe.

This consideration also applies to parallel (forked) portlet processing. See Section 10.4, "Portlet Forking."

10.5.3 Considerations for IFRAME-based Asynchronous Rendering

Some special considerations associated with IFRAME-based asynchronous rendering include:

- IFRAME rendering varies depending on the browser. Making an IFRAME implementation seamless to an end user involves several factors, such as proper skin/skeleton development conventions, cross-browser development, and so on.
- If the content is larger than the IFRAME region, horizontal and/or vertical scrolling will be enabled. Be careful of content which itself contains scrolling regions, as it can be difficult to manipulate all scrolling regions to view all embedded content.
- IFRAME rendering might complicate other aspects of portal development, such as cross-portlet drag and drop.
- IFRAME rendering works whether or not Javascript is enabled.
- You can disable asynchronous portlet content rendering for certain operations by using the `<render:context>` tag or the `AsyncContentContext` class as described in Section 10.5.8.2, "Disabling Asynchronous Rendering for a Single Interaction"; however, these mechanisms do not work correctly when IFRAME-based asynchronous rendering is used. To avoid this issue, turn off asynchronous rendering or use AJAX-based asynchronous rendering.

10.5.4 Considerations for AJAX-based Asynchronous Rendering

Some special considerations associated with Asynchronous JavaScript and XML (AJAX)-based asynchronous rendering include:

- AJAX technology relies on Javascript. If users disable Javascript in their browser, AJAX-based portlets will be broken (the content will never render).

- This mechanism might not be compatible with other AJAX mechanisms, such as those that might typically be used by content authors to dynamically populate forms, for example. Generally speaking, a best practice is to either let WebLogic Portal manage AJAX at the portal level, or turn off AJAX for a portlet if you intend to incorporate AJAX behaviors into your portlet.
- The current AJAX implementation does not support XHTML. The implementation performs DOM operations that are known not to work in some browsers when using an XHTML content type. This problem arises when a Look & Feel skeleton is configured to use an XHTML content type.You can avoid this problem by doing one of two things:
  - Use an HTML content type for the portal
  - Use the IFRAME-based implementation of async portlet rendering

10.5.5 Comparison of IFRAME- and AJAX-based Asynchronous Rendering

Table 10–2 summarizes the advantages and disadvantages of IFRAME- and AJAX-based asynchronous rendering. Oracle recommends AJAX as a more robust implementation.

Table 10–2 IFRAME-based and AJAX-based Asynchronous Portlet Summary Table

Type	Advantages	Disadvantages
IFRAME	Works with Javascript enabled or disabled Support for embedded media (non-HTML) files Supports XHTML.	Generally perceived as providing a less intuitive user experience Can complicate more full-featured portlet development tasks, such as cross-portlet drag and drop
AJAX	Generally perceived as providing a more intuitive user experience Provides a single document for full-featured portlet development tasks, such as cross-portlet drag and drop Provides better Look & Feel integration	Works only with Javascript enabled Does not currently support XHTML

10.5.6 Comparison of Asynchronous and Conventional or Forked Rendering

The following table compares some of the behavior and features of conventional or forked rendering and asynchronous portlet content rendering.

Table 10–3 Comparison of Behaviors - Forked/Conventional Rendering and Asynchronous Rendering

Behavior/Feature	Forked or Conventional Rendering	Asynchronous Rendering
Completed rendering of page	Page does not render until all portlet processing is complete	Page, and portlet frames, render immediately; individual portlet content renders as processing completes
HTML page	No changes between conventional rendering and forked rendering	Page uses AJAX or IFRAME for rendering.
Rendering requests	Requires only one request.	Requires $n + 1$ requests (where n is the number of asynchronous portlets) True only for page requests; when interacting with an individual portlet, only one request is required.

Table 10–3 (Cont.) Comparison of Behaviors - Forked/Conventional Rendering and Asynchronous

Behavior/Feature	Forked or Conventional Rendering	Asynchronous Rendering
Refresh	Entire page refreshes when interaction occurs on any portlet	Refresh required only for an individual portlet.
IPC Support	IPC supported	IPC not supported, although some workarounds exist for AJAX asynchronous portlets.
Page request/response	Server response to page request includes content of page	Portal page does not include portlet content (less information needs to be returned by the server); page starts loading faster

10.5.7 Portal Life Cycle Considerations with Asynchronous Content Rendering

This section provides more information about life cycle and control tree implications associated with using asynchronous content rendering.

For the *initial* request for a portal page, backing files attached to the portlet will run in the context of a full portal control tree. However, portlet content—such as Page Flows, managed beans, JSP pages, and so on—will not run for this initial request.

The values for the above-referenced APIs will be:

```
PortletBackingContext.isAsyncContent() = true

PortletBackingContext.isContentOnly() = false
```

For the *subsequent* content requests, backing files attached to the portlet, and the portlet content itself—such as Page Flows, managed beans, JSP pages, and so on—will run in the context of a "dummy" control tree.

The values for the above-referenced APIs will be:

```
PortletBackingContext.isAsyncContent() = true

PortletBackingContext.isContentOnly() = true

PortletPresentationContext.isAsyncContent() = true

PortletPresentationContext.isContentOnly() = true
```

An important consequence of this model is that when asynchronous content rendering is enabled for portlets, the portlet content will run in isolation from the rest of the portal. Such portlets therefore cannot expect to have direct access to other portal controls such as books, pages, desktops, other portlets, and so on.

10.5.8 Asynchronous Content Rendering and IPC

Although IPC is not supported when asynchronous content rendering is enabled, WebLogic Portal provides some features that allow these two mechanisms to coexist in your portal environment. In addition, you can disable asynchronous rendering for single requests using the mechanisms described in this section.

This section also applies to HTTP redirects.

Note: The techniques described in this section do not currently work with IFRAME portlets.

> **Tip:** If you enable asynchronous rendering at the portal/desktop level, you can use IPC without restrictions. For more information on asynchronous portal/desktop rendering, see the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

10.5.8.1 File Upload Forms

For forms containing file upload mechanisms, the WebLogic Portal framework automatically causes page refreshes without the need for any workarounds.

10.5.8.2 Disabling Asynchronous Rendering for a Single Interaction

Generally, if a portlet needs to disable asynchronous content rendering for a single interaction (such as a form submission, link click, and so on), you should use the mechanism described in this section.

> **Tip:** When you use these mechanisms to disable asynchronous rendering, the portlet's action/rendering will be performed using two requests. The portlet's action is performed in the page request, while the portlet's rendering is performed on a subsequent request. Ensure that your action does not use request attributes to pass information to your JSP page.

From a JSP page, use the `<render:context>` tag as follows. You can find this tag in the Design Palette under Tag Libraries > Portal Framework Rendering > Context.

```
<render:context asyncContentDisabled="true">

Form, anchor, etc. would appear here

(that is, <netui:form action="submit"...)

</render:context>
```

From Java code:

```
try {

AsyncContentContext.push(request).setAsyncContentDisabled(true);

// Code that generates a URL would appear here

} finally {

AsyncContentContext.pop(request)

}
```

10.5.8.3 URL Compression

URL compression interferes with some of the AJAX-specific mechanisms for page refreshes. Because of this, URL compression must also be disabled whenever asynchronous content rendering is disabled to force page refreshes. WebLogic Portal disables URL compression automatically except when file upload forms are used; in this situation, you must explicitly disable it. Use the following examples as a guide:

From a JSP page:

```
<render:controlContext urlCompressionDisabled="true">

Form, anchor, etc. would appear here
```

(that is, <netui:form action="submit"...)

```
</render:controlContext>
```

From Java code:

```
try {

UrlCompressionContext.push(request).setUrlCompressionDisabled(true);

// Code that generates a URL would appear here

} finally {

UrlCompressionContext.pop(request)

}
```

For more information about implementing URL compression, refer to the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

11

Monitoring and Determining Portlet Performance

Oracle WebLogic Portal supports the collection of analytics. You can use these analytics to deliver information to other products such as Oracle WebCenter Analytics or they can be consumed locally via custom code. These analytics are exposed through a public WebLogic Portal API in the com.bea.netuix.servlets.controls.analytics package. For more information, see the *Oracle Fusion Middleware Java API Reference for Oracle WebLogic Portal*. For information on using Oracle WebCenter Analytics, see the *Oracle Fusion Middleware Oracle WebCenter Analytics Administrator's Guide for Oracle WebLogic Portal*.

This chapter focuses on deriving and using these metrics for uses such as monitoring Service Level Agreements (SLAs), triggering a warning if a specific portlet's response time is excessive, and replacing a misbehaving portlet with an alternate portlet.

This appendix contains the following sections:

11.1 Introduction

Portals aggregate content and applications. Portals encapsulate these applications or content into subcomponents called portlets. Portlets are then brought together into a unified view that can be managed in one place. What happens when one of these portlets misbehaves or becomes unavailable? WebLogic Portal has many options for dealing with these scenarios. A few common ones are Web Service for Remote Portlets (WSRP) timeouts, interceptors, caching, threading, and AJAX enabled portlets. Each of these solutions deals with the problem in a different way and each has their own set of advantages and disadvantages. Using the public API `com.bea.netuix.servlets.controls.analytics` package, you can provide a more comprehensive way of dealing with this problem.

11.2 Use Case

Supposed that you have a portal that brings together different applications (portlets) into a single portal. Each of these applications (portlets) is mission critical in their own right and if one goes down it is mandatory that it not disrupt the other applications. Additionally, if one application goes down, an alternate (backup) application must be

displayed in its place. After the original application resumes normal service levels, it should be brought back online to replace the temporary application.

The first challenge is determining when an a portlet is not meeting a particular Service Level Agreement (SLA). The second challenge is providing the ability to bring certain portlets online or offline based on logic applicable the SLA.

> **Note:** The other previously mentioned methods for dealing with these types of issues can be used in conjunction with the method described here. For simplicity, this example does not include any other mechanisms.

The solution to portlet failure and replacement uses two features available in WebLogic Portal. The first feature deals with detecting when a portlet is not meeting a particular SLA. The second feature deals with disabling the poorly behaving portlet and enabling a temporary portlet in its place. Each of these features are useful by themselves but when combined provide a more comprehensive solution to the problem.

11.3 Detecting a Misbehaving Portlet

To capture analytic events, you use the same hook that Oracle WebCenter Analytics uses to capture all of its events. However, rather than using that hook to provide the rich set of reporting capabilities provided by Oracle WebCenter Analytics, you can use the hook to simply determine when a portlet is misbehaving.

The first step is to implement the `AnalyticEventHandler` interface. One ore more implementations of this interface are called by the server whenever a portlet or page completes its processing. This means any implementation must be extremely efficient, as it may get called 50 or more times per request.

Example 11–1 Portlet Rendering Time Detection

```
public class MyAnalyticEventHandlerImpl implements AnalyticEventHandler
{
   private final static long SLA = 5000000000L;  // Five seconds (nano time)

   /**
   * <p>Implementation class may perform any one-time initializations here. If
   * this method fails (throws an exception) the event handler will not be
   * registered and no event handling will take place.</p>
   */
   public void init() {
      System.out.println("My Analytic Event Handler Initialized");
   }

   /**
   * <p>This method is called by the container at the end of each page's and
   * portlet's run. It is invoked for every page and portlet on every request.
   * Since this method is called so frequently the implementation must be
   * extremely efficient, or the entire portal's performace will suffer.</p>
   * @param analyticEvent the event to be logged.
   */
   public void log(AnalyticEvent analyticEvent)
   {
      // Ignore all but portlet events.
      if (analyticEvent.getAnalyticEventObject().equals
```

```
            (AnalyticEvent.AnalyticEventObject.PORTLET))
      {
         if (analyticEvent.getTotalTime() > SLA)
         {
            System.out.println("WARNING: portlet " +
               analyticEvent.getDefinitionLabel() + " is exceeding SLA of " +
               String.valueOf(SLA / 1000000000L) + " seconds.");
         }
      }
   }

   /**
   * <p>Implementation class may perform cleanup operations here.
   * Note: there is no guarantee this method will be called.
   * </p>
   */
   public void dispose() {
   }
}
```

The main method of interest is the `log(AnalyticEvent analyticEvent)` method. This method is invoked for every page and portlet on every request. The `AnalyticEvent` class contains a variety of information including the times for various lifecycle phases of the portlet.

The `AnalyticEventObject` has three attributes or methods that provide the functionality to detect a misbehaving portlet:

- `getAnalyticEventObject()`– Reports what type of object this event is for, such as a portlet or a page.
- `getDefinitionLabel()` or `getInstanceLabel()` – Provides the unique identifiers for the portlet.
- `getTotalTime()` – Provides the total time it takes for the portlet to run through all of its lifecycles.

If the total time for a portlet to render exceeds 5 seconds, an error message is logged to the console. Note that times returned by these methods are in nanoseconds. As illustrated in the next section, you can use the `ServiveLevelManager` service to disable the portlet an enable another portlet.

Generally, you should package the service provider in a JAR file. The JAR should consist of any necessary classes (including the `AnalyticEventHandler`), and file named `META-INF/services/com.bea.netuix.servlets.controls.analytics.AnalyticEventHandler`. This services file must contain one and only one class name and must be a concrete `AnalyticEventHandler` implementation.

The service provider JAR or JARs should then be deployed with the application by including the JAR in the web application's `WEB-INF/lib` directory. Provider JARs may also be included in the application or system class path, although this changes the scoping of the provider class objects, and causes the provider implementations to be shared by multiple web applications.

11.4 Disabling the Bad Portlet and Enabling an Alternative Portlet

The `ServiceLevelManager` service allows you to enable and disable portlets based on a variety of identifiers. You can either disable a specific instance of a portlet or all instances of the portlet definition. In this example, all instances of the portlet definition

are disabled. This choice is based on the assumption that if a portlet definition is poorly behaving, then all instances of that portlet are behaving poorly. You can just as easily disable one offending instance.

Alternatively, you can call the `ServicLevelManger` interface from an administration JSP. If you are using this approach, the administration JSP would list all the poorly behaving portlets and an administrator would have to manually disable the misbehaving portlets and re-enable the good ones.

In the scenario described in Section 11.2, "Use Case," you create an alternate portlet that could display a message saying the service is down, or provide a cached set of results, or get information from an alternate source. Regardless of what the alternate portlet does, it goes in the same placeholder as the misbehaving portlet.

As illustrated in Example 11–2, the alternate portlet is disabled on startup. When a portlet goes amiss, the alternate portlet is enabled and the bad portlet is disabled.

Example 11–2 Enabling an Alternate Portlet and Disabling a Misbehaving Portlet

```
/** Service-provider interface for {@link AnalyticEventHandler} implementations.
 * <p/>
 * An analytic event handler is a concrete subclass of this interface that
 * has a public no-argument constructor and implements the interface methods
 * specified below.
 * <p/>
 * An AnalyticEventHandler implementer should generally package their
 * provider in a jar. That jar should consist of any necessary classes
 *(including, of course, an implementation of AnalyticEventHandler), as well
 * as a file named <tt>META-INF/services/com.bea.netuix.servlets.controls.
 * analytics.AnalyticEventHandler<tt>. That services file must contain one
 * and only one class name which must be a concrete AnalyticEventHandler
 * implementation.
 * <p/>
 * The provider jar(s) should then be deployed with the application by
 * including the jar in the webapp's <tt>WED-INF/lib</tt> directory.
 * Provider jars may also be included in the application or system classpath,
 * although this changes the scoping of the provider class objects, and
 *causes the provider implementations to be shared by multiple web apps.
 * <p/>
 * On initialization, the {@link com.bea.netuix.servlets.controls.
 * analytics.AnalyticEventDispatcher} will load such provider. Id the
 * Dispatcher fails to load or initialize the event handler a error message
 * will be logged and no event handling will take place.
 * <p/>
 * NOTE: Implementations of the interface methods must be safe for use by
 * multiple concurrent threads.
 * <p/>
 */
public class MyAnalyticEventHandlerImpl implements AnalyticEventHandler
{
   private final static long SLA = 5000000000L;  // Five seconds (nano time)
   /**
    * <p>Implementation class may perform any one-time initializations here.
    * If this method fails (throws an exception) the event handler will not
    * be registered and no event handling will take place.</p>
    */
   public void init() {
      System.out.println("My Analytic Event Handler Initialized");
      // Disable alternate portlet
      ServiceLevelManagerFactory serviceLevelManagerFactory =
      ServiceLevelManagerFactory.getInstance();
```

```
      ServiceLevelManager serviceLevelManager =
      serviceLevelManagerFactory.getServiceLevelManager("/portal_1");

serviceLevelManager.setServiceLevelForDefinitionLabel(PortletServiceLevel.suspende
d, "alternate_pdl");
   }
   /**
    * <p>This method is called by the container at the end of each page's
    * and portlet's run. It is invoked for every page and portlet on every
    * request. Since this method is called so frequently the implementation
    * must be extremely efficient, or the entire portal's performace will
    * suffer.</p>
    * @param analyticEvent the event to be logged.
    */
   public void log(AnalyticEvent analyticEvent)
   {
      // Ignore all but portlet events.
      if (analyticEvent.getAnalyticEventObject().equals
         (AnalyticEvent.AnalyticEventObject.PORTLET))
      {
         // This will disable any portlet that has a response time > then
         // the SLA
         if (analyticEvent.getTotalTime() > SLA)
         {
            System.out.println("WARNING: portlet " +
               analyticEvent.getDefinitionLabel() + " is exceeding SLA of "
               + String.valueOf(SLA / 1000000000L) + " seconds.");
            ServiceLevelManagerFactory serviceLevelManagerFactory =
            ServiceLevelManagerFactory.getInstance();
            System.out.println("Servlet context path: " +
            analyticEvent.getServletContextName());
            // Note: service level manager is scoped to the context path
            // (request.getContextPath()). Before 10.2 this has to be known,
            // in 10.2, you can retrieve it from the AnalyticEvent using
            // String getWebappContextPath();
            ServiceLevelManager serviceLevelManager =
            serviceLevelManagerFactory.getServiceLevelManager("/portal_1");
            PortletServiceLevel portletServiceLevel =
            serviceLevelManager.getServiceLevelForDefinitionLabel
            (analyticEvent.getDefinitionLabel());
            System.out.println("Suspending Portlet: " +
            analyticEvent.getDefinitionLabel());
            serviceLevelManager.setServiceLevelForDefinitionLabel
            (PortletServiceLevel.suspended,
            analyticEvent.getDefinitionLabel());
            // Activating alternate portlet
            System.out.println("Activating Alternate Portlet ");

serviceLevelManager.setServiceLevelForDefinitionLabel(PortletServiceLevel.active,
"alternate_pdl");
         }
      }
   }
   /**
    * <p>Implementation class may perform cleanup operations here.
    * Note: there is no guarantee this method will be called.
    * </p>
    */
   public void dispose() {
   }
```

```
}
```

To get a reference to the `ServiceLevelManager`, you need to access it through the service level factory. `ServiceLevelManagers` are scoped to the web application. As such the factory requires the webapp context path. The method of interest is:

```
serviceLevelManagerFactory.getServiceLevelManager("/mywebapp");
```

To disable (or enable) a particular portlet, set the `PortletServiceLevel` on the selected portlet. In Example 11–2, the following code disables the alternate portlet:

```
serviceLevelManager.setServiceLevelForDefinitionLabel(PortletServiceLevel.
   suspended, analyticEvent.getDefinitionLabel());
```

The alternate portlet is enabled by the same method:

```
serviceLevelManager.setServiceLevelForDefinitionLabel(PortletServiceLevel.
   active, "alternate_pdl");
```

In this example, the bad portlet is not disabled until it has actually run because you do not know it is bad until it takes more then 5 seconds to run. In an actual application, you would probably use timeouts for a better implementation. Also, the alternate portlet does not go online until the next request, as this request has already finished. If you want to have the alternate portlet run as part of this request, you could perform a redirect that picks up the new portlet.

In WebLogic Portal 10.0, you had to hard code the web application context path:

```
ServiceLevelManager serviceLevelManager =
      serviceLevelManagerFactory.getServiceLevelManager("/portal_1");
```

In WebLogic Portal 10.2 and later versions, you can get this path from the analytic event using:

```
String getWebappContextPath();
```

12

Configuring Local Interportlet Communication

This chapter discusses IPC and describes the IPC techniques available to portlet developers.

This chapter includes these topics:

12.1 Introduction

Interportlet communication (IPC) describes a handful of technologies and features that allow portlets to share data and react to events. This chapter discusses the IPC techniques supported by WLP.

> **Tip:** A good way to learn about how to use IPC in WLP is to work through the comprehensive example in Chapter 13, "Interportlet Communication Example With Event Handling."

For an overview of the IPC methods supported by WLP, see Section 12.2, "Overview of Interportlet Communication Techniques." The remainder of the chapter discusses each IPC method in greater detail.

12.2 Overview of Interportlet Communication Techniques

WLP supports several types of interportlet communication. You need to determine which type of IPC works best for your application. For instance, if you simply want to make some data from one portlet available to another, public or shared render parameters might be the best, simplest approach. If your application requires a portlet to notify other portlets that it has changed, then event handling is the best approach.

Table 12–1 summarizes the IPC techniques that WLP supports and discusses the advantages or disadvantages of each one.

Table 12–1 Interportlet Communication Features of WLP

IPC Feature	Description	Support	Notes and Recommendations
Events	All WLP portlet types, including remote portlets, support event handling. The event framework allows portlets to fire and receive events. Event handling is the primary focus of this chapter. **Note:** For information on using event handling for remote (WSRP) portlets, see "Interportlet Communication for Remote Portlets" in the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.	All portlet types	In general, event handling is the recommended technique for IPC. Events support complex payloads, permit chaining, support notification, and present a loose coupling between portlets.
Public Render Parameters (JSR 286 Portlets) and Shared Parameters (Other Portlet Types)	Public render parameters and shared parameters allow portlets to share simple String values with other portlets. When any portlet makes a change to a shared parameter, other portlets which share that parameter will always see the latest value set for that parameter. For details, see Section 12.9, "Public Render Parameters" and Section 9.3, "Using Shared Parameters."	All portlet types	An advantage to using public render parameters and shared parameters over event handling is that you avoid the need for an explicit event handler, because the Portal framework automatically provides the latest values of shared parameters to all portlets that subscribe to them. Unlike events, shared parameters do not provide an explicit notification when a value changes.
Custom Data Transfer (remote portlets only)	Custom data transfer provides a simple way to pass data between producers and consumers in a federated portal environment. For detailed information on custom data transfer, see "Transferring Custom Data" in the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.	Remote portlets only	Generally, the recommended best practice is to use event handling or public render parameters / shared render parameters instead of custom data transfer for remote IPC.

12.3 Differences Between Portal Events and Java Portlet Events

The Java portlet specification version 2.0 (JSR 286) provides a model for handling events that is slightly different from the event model WLP uses for other portlet types. This section describes some of the differences between the event models.

In the WLP event model, you can declare event handlers that have a variety of possible actions, such as firing another event, changing the portlet's mode or state, or invoking a method on the backing file. The Java portlet event model supports only one action when an event is received: the processEvent() method (or another method, if properly annotated) is called on the portlet. The event handling code in the Java

portlet can then take whatever actions are necessary, such as firing another event or changing the portlet's mode or state.

The WLP event model allows portlet event handlers to listen to events coming only from certain portlets. The Java portlet event model does not support this behavior directly, but this can be achieved by including information about the sending portlet in the event's payload, and having the receiving portlet take appropriate action based on this information.

The WLP event model allows portlets to receive events only if they were generated for the particular portlet instance (the **Self Instance Only** option). For example, check the **Self Instance Only** checkbox if you want a portlet to receive an event when it is minimized, but not to receive an event when another portlet is minimized. The Java portlet event model does not have any corresponding concept, but by default some event types (refresh, activation, deactivation, mode change and state change events) are only delivered to Java portlets if the event was generated for the particular instance. For details, see Section 12.8, "Subscribing Java Portlets to Portal Framework Events." Note that this model does not prevent functionality available to other portlet types, but it does require a bit more code to achieve.

For example, if Portlet A wants to know when Portlet B has been minimized and both are Java portlets, only Portlet B will receive the event when it is minimized. However, in its event handler method, Portlet B can send another event to which Portlet A can subscribe to let Portlet A know that Portlet B was minimized.

The WLP event model also allows portlets to receive events only if the portlet is currently being displayed; the Java portlet event model has no equivalent concept-events are always sent to Java portlets whether or not they are being displayed.

All events sent from Java portlets are treated as custom events for other portlet types. For example, if in a Java portlet you use the following code:

```
actionResponse.setEvent(new QName("sampleNamespace", "sampleEvent"), null);
```

Other portlet types can receive this event by declaring a custom event handler with a QName consisting of the namespace `sampleNamespace` and local part `sampleEvent`.

12.4 Portlet Event Handling

Events are supported for all of the portlet types supported by WLP and are the recommended mechanism for performing IPC in WebLogic Portal. Events permit chaining, support notification, and present a loose coupling between portlets. Portlet events can optionally carry accompanying data called a payload, which is a serializable Java object.

While events are supported for all portlet types, they are surfaced to Java portlets slightly differently, to comply with the JSR 286 specification. For all portlet types, event handlers may be declared in the `.portlet` files, but for JSR 286 portlets to remain compatible with other Java Portlet containers, they must exclusively use Java Portlet Events. For more information, see Section 12.7, "Events in Java Portlets."

The Portlet Editor provides a Portlet Event Handlers dialog for configuring events on a portlet. Three components that must be configured for all portlet event types declared in `.portlet` files are:

- **Event Handlers** – Handlers specify which type of event the portlet will listen for. Types of event handlers include portal events, custom events, faces events, and others. For a complete list and descriptions, see Table 12–2.

- **Actions** – You specify an action to perform in response to an event. Actions include changing the window mode, changing the window state, firing a custom event, and others. For a complete list and descriptions, see Table 12–3.
- **Event Name** – Each type of event handler requires an event name or action to be specified. The portal event handler is designed to respond to pre-defined events, such as the event fired when a portlet is minimized (onMinimize event). The custom event handler, on the other hand, responds to a named event that is typically fired programmatically from a JSP or Java class. In a similar way, the page flow handler responds to a named Action.

This section contains the following topics:

12.4.1 Event Handlers

Event handlers listen for events raised on subscribed portlets and fire one or more actions when a specific event is detected. An event handler element is a child of the `<portlet>` element, and a portlet can have any number of events associated with it. Table 12–2 lists the event handlers that are available on the Add Handler menu of the Portlet Event Handlers wizard:

Table 12–2 Event Handlers

Event	Description
Handle Generic Event	Generic event handlers respond to all kinds of events, custom events, portal events, Struts events, and so on. A generic event handler just requires the event name to process the event. For details, see Section 12.4.1.1, "Generic Event Handlers" below. **Note:** Generic Events are deprecated. Oracle recommends that you use another event handler.
Handle Portal Event	Portal events are fired in response to certain kinds of actions that occur within the portal. The actions include state and mode changes. An example of a state change is minimizing a portlet. When you click the minimize button, an onMinimize event is fired. Mode change events are fired when a user clicks the edit or help button on a portlet. In addition, a portal event is fired when a portlet is refreshed. Portal event handlers do something in response to a portal event.
Handle Custom Event	Custom events are fired by you, the developer. Custom events can be fired from the handlePostbackData() method of a backing file or from an event handler. See Section 12.6, "Custom Event Handling" for more information.

Table 12–2 (Cont.) Event Handlers

Event	Description
Handle PageFlow Event	Page flow events are sent by the framework automatically when a page flow action is triggered. You can define a page flow event handler (on that portlet) that responds to these events and performs actions, such as to notify other portlets (that is, raise a custom event) or invoke a backing file call-back method, and so on. **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Handle Struts Event	Struts events are fired automatically by the framework when a Struts action is triggered. You can define a Struts event handler (on that portlet) that responds to these events and performs actions, such as to notify other portlets (that is, raise a custom event) or invoke a backing file call-back method, and so on. **Note:** Apache Struts is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Handle Faces Event	Faces events are fired automatically by the framework when a JSF action is triggered. You can define a Faces event handler (on that portlet) that responds to these events and performs actions, such as to notify other portlets (that is, raise a custom event) or invoke a backing file call-back method, and so on.

12.4.1.1 Generic Event Handlers

The generic event handler, with an event attribute value of *myEvent*, will be triggered on the following conditions:

- A custom event with event=*myEvent* is fired within the portal.
- A page flow action with name *myEvent* is raised by a portlet within the portal.
- The same conditions to which the `<handlePortalEvent event=myEvent>` handler would react.
- A generic event (see below) with event=`myEvent` is fired within the portal.

Using a generic event handler allows you to more effectively decouple your portal design, because your application does not need to know the source or type of an event. You can change the portlet type (for example, from a page flow portlet to a JSP portlet, with a backing file firing custom events) without affecting how you events are processed.

> **Note:** Because of the non-specific way generic event handlers are triggered, it is possible that they will be triggered by an event that isn't intended. Because of this, generic events and generic event handlers have been deprecated by Oracle going forward, and it is recommended that you use the event handler for the specific type of event you wish to trap. Previous limitations regarding WSRP changing event types to generic events have been eliminated, so there is no longer any reason to use a generic event handler.

12.4.2 Event Actions

When event handlers are triggered by receiving an appropriate event, they perform a configurable action on the host portlet. Triggering events may come from another portlet in the application, or possibly from the same portlet, depending on the type of event and how the handler is configured. For example, when the user minimizes a portlet, a portal event called `onMinimize` is sent, which causes any event handlers listening for it to perform their actions, such as firing a custom event or calling a method on the portlet's backing file.

Table 12–3 lists the event actions available for portlets.

Table 12–3 Event Actions

This action...	Has this effect...
Change Window Mode	Changes the mode from its current mode to a user-specified mode; for example, from help mode to edit mode.
Change Window State	Changes the state from its current state to a user-specified state; for example, from maximized to delete state.
Activate Page	Opens the page on which the portlet currently resides.
Fire Generic Event	Fires a user-specified generic event. Generic events have been deprecated; Oracle recommends that you use custom events instead.
Fire Custom Event	Fires a user-defined custom event.
Invoke BackingFile Method	Runs a method in the backing file attached to the portlet. For more information on backing files, refer to Section 9.4, "Backing Files."

12.4.3 Event Types

An event action depends upon the type of event being raised. Except for portal events, all other events can be identified in the **Events** field on the Portlet Event Handlers Wizard, as described in Section 12.5, "Using the Portlet Event Handlers Wizard." Events available with the portal event handler are listed in Table 12–4.

Table 12–4 Events Available to a Portal Event Handler

This event...	Fires an action when the portlet...
onActivation	Becomes visible
onDeactivation	Ceases to be visible
onMinimize	Is minimized
onMaximize	Is maximized
onNormal	Returns to its normal state from either a maximized or minimized state
onDelete	Is deleted from the portal
onHelp	Enters the help mode
onEdit	Enters the edit mode
onView	Enters the view mode
onRefresh	Is refreshed
onInit	The onInit event is broadcast once per portal request. Use this event if you want to define an event handler that is always executed on every portal request.

Table 12–4 (Cont.) Events Available to a Portal Event Handler

This event...	Fires an action when the portlet...
onLookAndFeelReinit	This event is fired when a Look & Feel is re-initialized. This happens whenever the Look & Feel is dynamically changed, such as when any of the reinit() methods on a PortalLookAndFeel object are called. See the PortalLookAndFeel class in *Oracle Fusion Middleware Java API Reference for Oracle WebLogic Portal* for more information about the reinit() methods. For instance, if you capture information about the Look & Feel, you need to know when the Look & Feel changes so that you can refresh it with the captured information. This event provides that notification.
onModeChange	Mode changes.
onStateChange	State changes.
onCustomEvent	Receives a custom event. Refer to Section 12.4.1, "Event Handlers."

12.5 Using the Portlet Event Handlers Wizard

The Portlet Event Handlers wizard included in Oracle Enterprise Pack for Eclipse allows you to configure several types of event handlers and actions. The following steps summarize the process of setting up an event handler using the wizard:

1. Select a type of event handler to create.
2. Determine the portlets to which that handler will listen.
3. Select an event for which the handler will listen.
4. Select and configure an action to fire when the event occurs.

The following sections describe the dialogs of the wizard and provide information about the information required in each field of the dialogs.

For a specific procedural example of how to use the event handler wizard, refer to Chapter 13, "Interportlet Communication Example With Event Handling."

12.5.1 Opening the Portlet Event Handlers Wizard

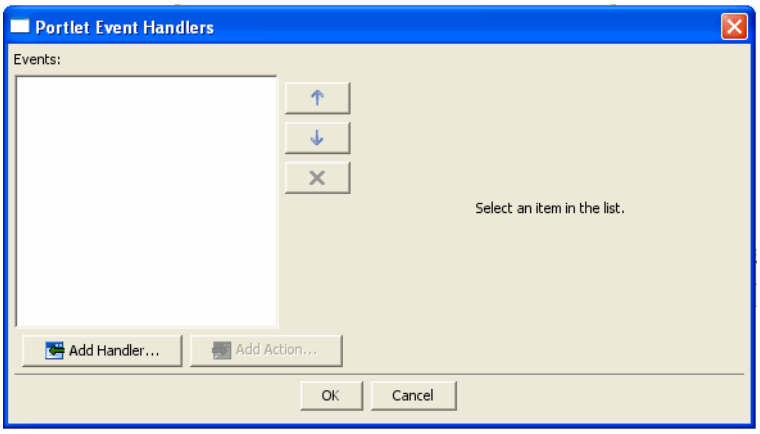
You can open the Portlet Event Handlers wizard in two ways:

- The wizard opens when you open a portlet in Oracle Enterprise Pack for Eclipse and click the ellipsis button next to Event Handlers in the Properties view.
- The wizard opens when you click the Event Handlers link in the portlet editor.

> **Note:** If no event handlers have been added, the Events field indicates that. If any event handlers have been added, the field indicates the number that currently exist.

The wizard appears, as shown in Figure 12–1.

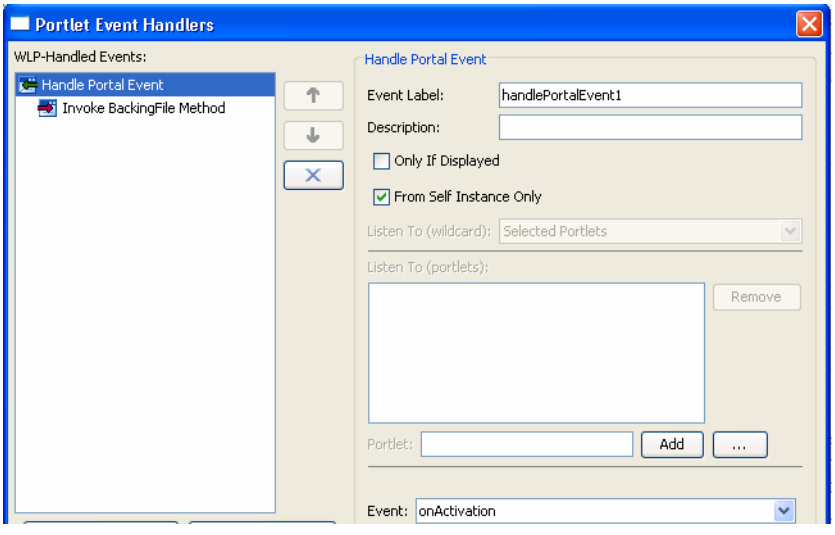
Figure 12–1 Portlet Event Handlers Wizard



When you click **Add Handler**, the event handler drop-down menu allows you to select a handler; to add an action, click **Add Action** to open the event action drop-down menu.

Based on your selection, the dialog box expands, displaying additional fields that you can use to set up the handler or action. Figure 12–2 shows an example of the expanded dialog for adding an event handler. Note that the entry fields in the wizard are validated; the **OK** button is disabled until all errors are corrected for the currently displayed properties.

Figure 12–2 Expanded Event Handlers Dialog



12.5.2 Portlet Event Handlers Wizard - Add Handler Field Descriptions

Table 12–5 explains the fields in the Add Handler dialog and how your selections affect the behavior of the event.

Note: WebLogic Portal attempts to validate the settings of the Handle Event part of the Portlet Event Handlers dialog. You will receive an error message if any problems are detected. For detailed information on the WebLogic Portal validation framework, see the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Table 12–5 Portlet Event Handlers Wizard - Add Handler

Field	Description
Event Label	Required. This identifier can be used by the `<filterEvent>` tag in the portal file to distinguish multiple event handlers in the same portlet.
Description	Optional.
Only If Displayed check box	Optional. Indicates that the portlet to receive the event must be on the current page and not minimized or maximized—the portlet's content must be currently in a rendered state. (Remember that the user must also be entitled to see the portlet.) The default is `true`. **Note:** If the event is <handlePortalEvent event="onMinimize" fromSelfInstanceOnly="true"> then it is logically impossible for this event to fire if onlyIfDisplayed="true".
From Self Instance Only checkbox	Optional. Defines whether the handler for a given portlet instance is invoked only when the source event originates from that instance. The default is `false`. If From Self instance Only is set to true, any Listen To values are ignored.
Listen To (wildcard)	Identifies the portlet(s) that this portlet can listen to. The values include: ■ Any – Listens to events fired from any portlet in the portal. ■ This – Listens to events fired from the currently selected portlet. ■ Selected Portlets – (default) Listens to events fired from selected portlets only. Click the **...** button in the Listen To (portlets) part of the dialog to select portlets. ■ This and Selected Portlets – Listens to events fired from the currently selected portlet and portlets selected in the Listen To (portlets) part of the dialog.
Listen To (portlets)	Optional. Allows you to specify the portlets that this portlet can listen to. You can choose a `.portlet` file from the file system by clicking the **...** button). When you select a `.portlet` file and click **OK**, the portlet is added to the Listen To list. This part of the dialog is only enabled if you chose either the Selected Portlets or This and Selected Portlets option in the Listen To (wildcard) menu. **Caution:** The values that you enter here are not validated. A typo in either an event name or a definition label can be very difficult to resolve later.
Portlet	You can type a portlet name in the field and click **Add**, or click the browse button to navigate to the portlet for which you want to listen.
Event or Action	Depending on the event handler you added, you will choose an event or an action for which the portlet will listen. For example, if you added the `HandlePortalEvent` handler, you can use the **Event** drop-down menu to select portal events, such as the `onRefresh` event. If you choose a handler that exposes actions, type the name of the action in the **Action** field. For example, if you chose `HandlePageFlowEvent`, you could type `submitReport`. The `submitReport` action of the page flow is now visible in the **Action** drop-down menu.

12.5.3 Portlet Event Handlers Wizard - Add Action Field Descriptions

The available fields for the action depend on the type of action that you select. Table 12–6 explains the possible fields in the expanded Portlet Event Handlers dialog and how your selections affect the behavior of the action.

Table 12–6 Portlet Event Handlers Wizard - Add Action

Field	Description
Change Window Mode	Enter the value of the new window mode.
Change Window State	Enter the value of the new window state; possible values are normal, minimized, maximized.
Activate Page	This action activates the page on which the portlet `<portlet_def_id>` currently resides. This action will fire only when triggered during the handlePostBack life cycle. Do not select the Activate Page action if the Only If Displayed check box is selected. Logically, if the portlet is responding to the event only if it is displayed, the page that it is on must be active anyway.
Invoke Struts Action	Use this selection to cause an Apache Struts action to be raised. The value must be an unqualified name of a Struts action defined in the embedded content. This action is only available on the menu for Struts portlets. **Note:** Apache Struts is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Fire Generic Event	Use this selection to cause a generic event to be raised. Enter the name of the generic event.
Fire Custom Event	Use this selection to cause a custom event to be raised. Enter the name of the custom event.
Invoke BackingFile Method	Use this selection to cause a backing file method to run. Enter the name of the method that you want to invoke. This action is only available on the menu if a backing file is configured for the portlet.
Invoke Page Flow Action	Use this selection to cause a page flow action to be raised. This action is only available on the menu for page flow portlets. **Note:** Page Flows are a feature of Apache Beehive, which is an optional framework that you can integrate with WLP. See "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.
Invoke Faces Action	Use this selection to cause a JSF action to be raised. This action is only available on the menu for JSF portlets.

12.5.4 Definition Labels and Interportlet Communication

IPC behavior is based on portlet definition labels; that is, all portlet instances of a given `.portlet` file respond to the same events. You can use the event handler options *Only If Displayed* and *From Self Instance Only* to discriminate among the instances of the same `.portlet` file. For a description of these options, refer to Section 12.5.2, "Portlet Event Handlers Wizard - Add Handler Field Descriptions."

12.6 Custom Event Handling

All WLP portlet types support custom event handling. A custom event is an event that you define, and can contain a developer-defined payload. A custom event handler is triggered when a custom event is received, and the handler can fire any pre-defined action, including calling a user-defined method in a backing file.

The Portlet Event Handlers dialog with the Handle Custom Event handler selected is shown in Figure 12–3. Note that the entry fields in the wizard are validated; the **OK** button is disabled until all errors are corrected for the currently displayed properties.

Figure 12–3 Custom Event Handler Dialog

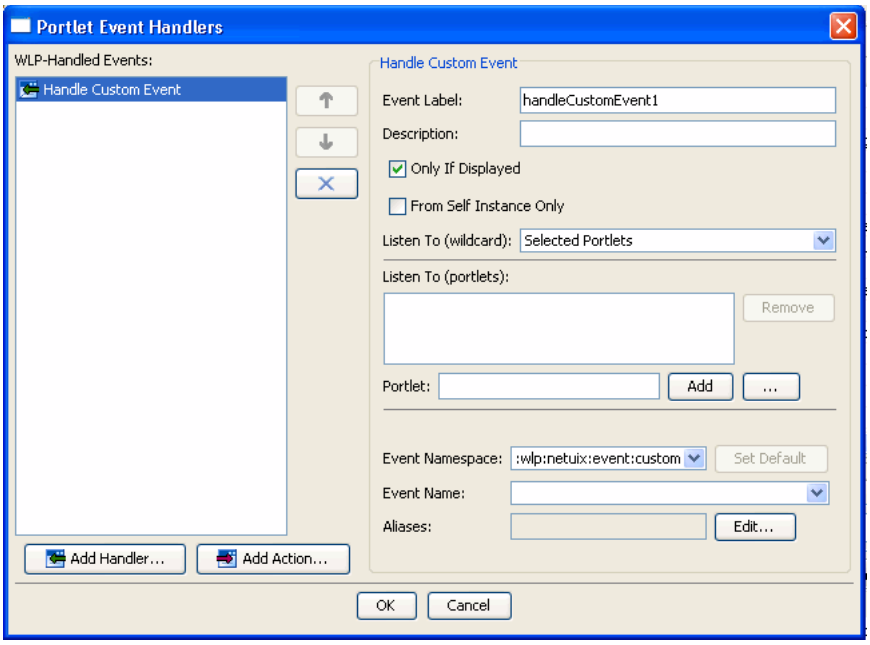


Table 12–7 describes the fields and features of the Custom Event Handler form dialog shown in Figure 12–7.

Table 12–7 Custom Event Dialog Fields

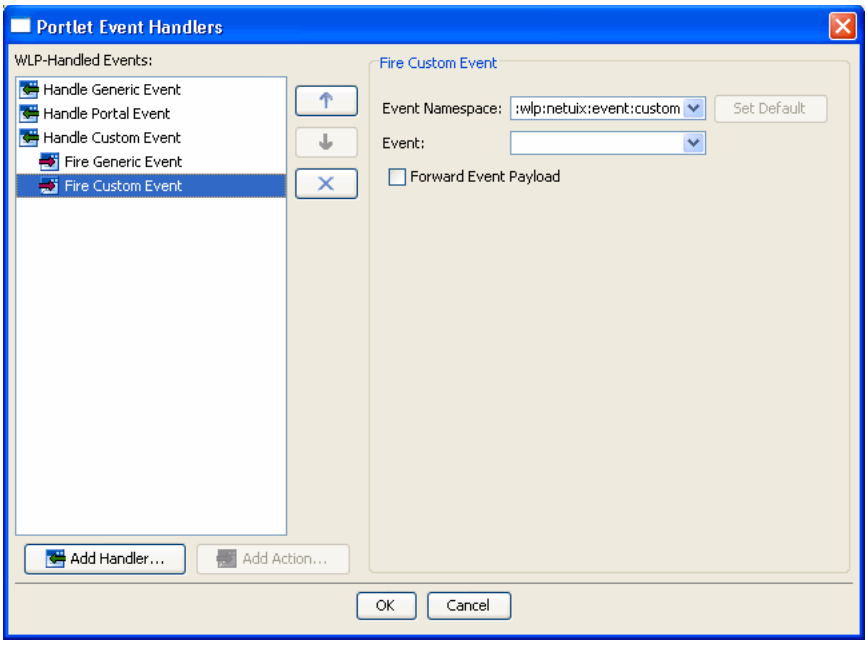
Field	Description
Event Label	Required. This identifier can be used by the `<filterEvent>` tag in the portal file to distinguish multiple event handlers in the same portlet.
Description	Optional.
Only If Displayed checkbox	Optional. Indicates that the portlet to receive the event must be on the current page and not minimized or maximized—the portlet's content must be currently in a rendered state. (Remember that the user must also be entitled to see the portlet.) The default is `true`. **Note:** If the event is <handlePortalEvent event="onMinimize" fromSelfInstanceOnly="true"> then it is logically impossible for this event to fire if onlyIfDisplayed="true".
From Self Instance Only checkbox	Optional. Defines whether the handler for a given portlet instance is invoked only when the source event originates from that instance. The default is `false`. If From Self instance Only is set to true, any Listen To values are ignored.

Table 12–7 (Cont.) Custom Event Dialog Fields

Field	Description
Listen To (wildcard)	Identifies the portlet(s) that this portlet can listen to. The values include: ■ Any – Listens to events fired from any portlet in the portal. ■ This – Listens to events fired from the currently selected portlet. ■ Selected Portlets – (default) Listens to events fired from selected portlets only. Click the **...** button in the Listen To (portlets) part of the dialog to select portlets. ■ This and Selected Portlets – Listens to events fired from the currently selected portlet and portlets selected in the Listen To (portlets) part of the dialog.
Listen To (portlets)	Optional. Allows you to specify the portlets that this portlet can listen to. You can choose a `.portlet` file from the file system by clicking the **...** button). When you select a `.portlet` file and click **OK**, the portlet is added to the Listen To list. This part of the dialog is only enabled if you chose either the Selected Portlets or This and Selected Portlets option in the Listen To (wildcard) menu. **Caution:** The values that you enter here are not validated. A typo in either an event name or a definition label can be very difficult to resolve later.
Portlet	You can type a portlet name in the field and click **Add**, or click the browse button to navigate to the portlet for which you want to listen.
Event Namespace	Enter a QName or NCName to uniquely identify the event. For details, see Section 12.12, "About QNames and Aliases."
Event Name	Enter a name for the event. The custom event handler responds to this named event, which is typically fired programmatically from a JSP or Java class.
Aliases	(Optional) Enter an alias for the event. For details, see Section 12.12, "About QNames and Aliases."

For information on the Add Action button selections, see Section 12.4.2, "Event Actions." Note that the Fire Custom Event action requires an Event Namespace, as shown in Figure 12–4. The Namespace must be a qualified QName namespace. For information on namespaces, see Section 12.12, "About QNames and Aliases."

Figure 12–4 Fire Custom Event Dialog



12.7 Events in Java Portlets

The JSR 286 specification adds support for events to Java portlets. WLP has previously supported events for all portlet types, including JSR 168 portlets, but in a manner slightly different from the JSR 286 specification. While the methods for handling events described in Section 12.4, "Portlet Event Handling" will continue to work for JSR 286 portlets, it is recommended that you use the JSR-286-standard event handling for Java portlets. This will maximize portability of these portlets to other Portal containers.

This section explains how to add events to Java portlets in a JSR-286-compliant way.

12.7.1 Overview

In JSR 286, portlets can declare events that they are interested in receiving, called processing events, and events that they are expected to fire, called publishing events. Only events declared as processing events will be delivered to the portlet, but the portlet is allowed to fire events it does not declare as publishing events. The list of declared published events is used exclusively at development time to give an indication to developers of what types of events a portlet may fire; it is not used at portlet runtime.

The Portlet Event Handlers dialog box for Java portlet types allows you to declare non-JSR-286-compliant event handling in the WLP-Handled Events tab, and to declare JSR-286-compliant events in the Java Portlet Events tab.

WLP-handled events are placed in the `.portlet` file for the portlet and are handled by the WLP framework. These events are not portable to other JSR 286 portlet containers, and therefore are not the recommended technique for handling events for Java portlets. Oracle recommends that you use Java portlet events instead.

Because Java portlet events comply to the JSR 286 standard, they are portable to other JSR 286 portlet containers. In the Java Portlet Events tab, the dialog handles the job of updating the `portlet.xml` file with the correct elements and attributes. Java portlets

receive and can send events from the portlet's Java code. Note that a single portlet can both send and receive events.

Refer to the Java Portlet Specification Version 2 for more information on Java portlet event handling.

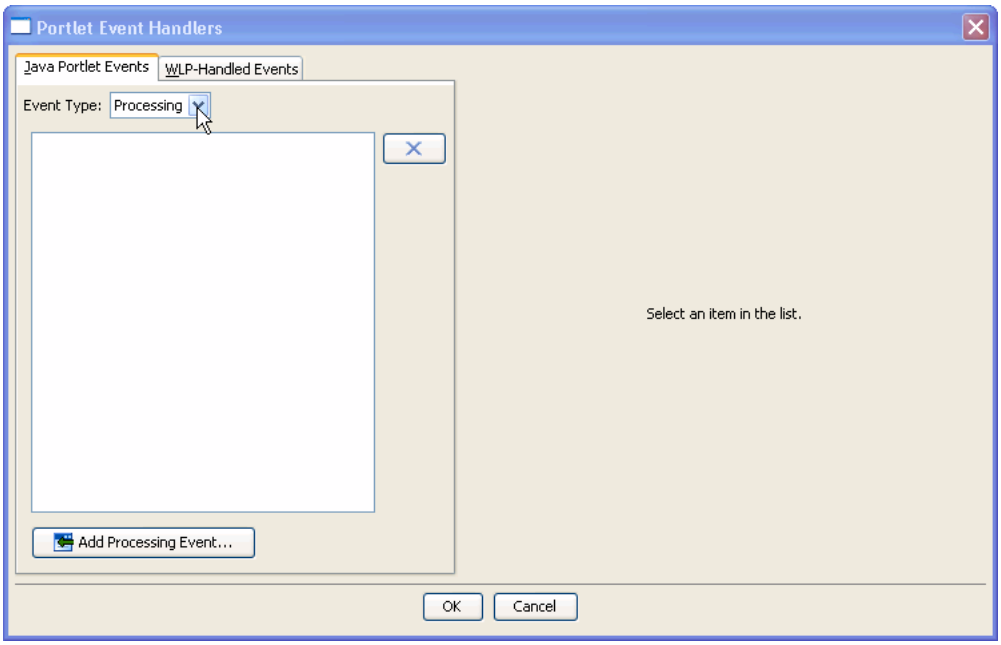
12.7.2 Adding a Processing Event

By declaring a processing event, you are declaring an event that the portlet can receive. Java portlets only receive events that are declared as processing events.

To add a processing event to a Java portlet:

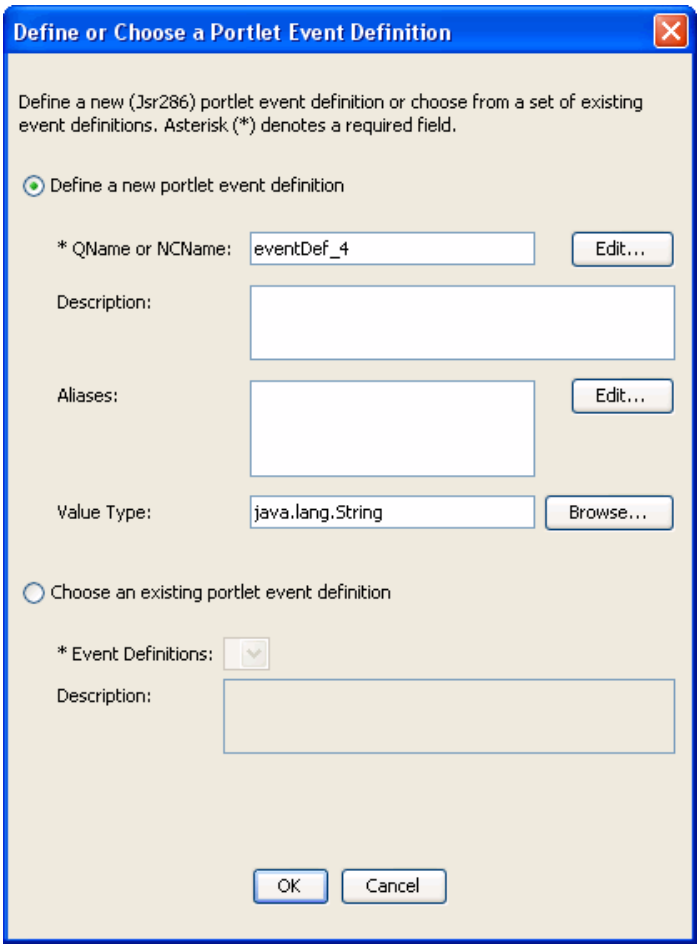
1. Open the Portlet Event Handlers dialog. You can open the dialog by clicking the Event Handlers button in the Properties view or by clicking the Event Handlers link in the portlet editor. Note that the entry fields in the dialog are validated; the **OK** button is disabled until all errors are corrected for the currently displayed properties.
2. In the Portlet Event Handlers dialog, select the Java Portlet Events tab.
3. Select Processing from the Event Type menu, as shown in Figure 12–5.

Figure 12–5 Selecting a Processing Event Type



4. Click **Add Processing Event**. The Define or Choose a Portlet Event Definition dialog opens, as shown in Figure 12–6.

Figure 12–6 Define or Choose a Portlet Event Definition Dialog



5. To create a new event, select **Define a new portlet event definition**, and fill out the form.

   Optionally, you can choose to select an existing event definition. To do this, select the radio button and pick an event from the drop down menu. This menu is only populated if one or more events were previously defined by other portlets in the same portlet application (`portlet.xml`).

6. (Optional) Specify an alias name. For information on aliases, see Section 12.12, "About QNames and Aliases."

   To add an alias, click **Edit** and in the next dialog, click **New**. The Provide QName Components dialog appears (see Figure 12–7). Enter an optional Namespace URI and a Local Part for the QName and click OK. The alias definition appears in the Provide List of QName Alias(es) dialog (Figure 12–8). You can add as many event aliases to the event declaration as you like.

Figure 12–7 Provide QName Components Dialog

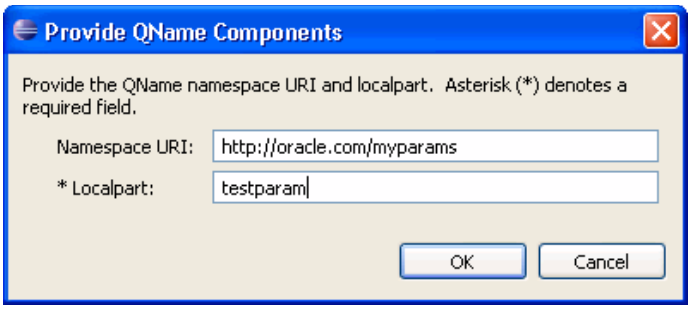
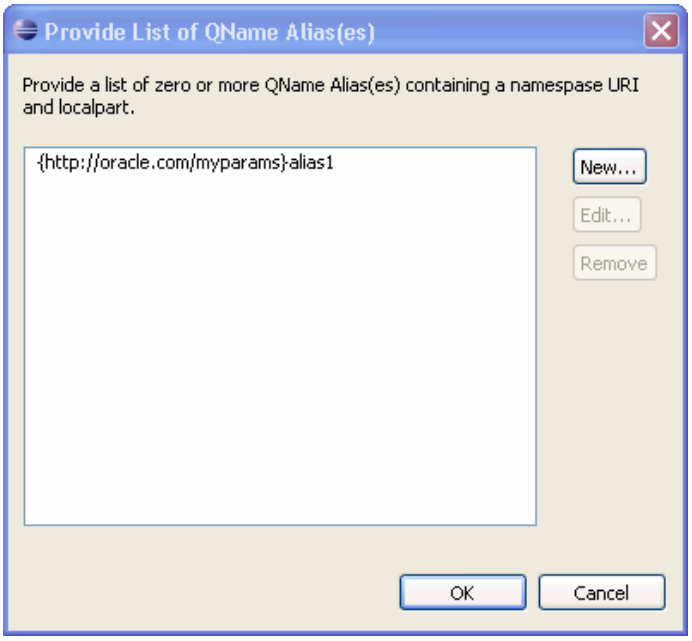


Figure 12–8 Provide List of QName Alias(es) Dialog



7. (Optional) Enter a data type in the Value Type field. The Value Type specifies the Java object type of the event payload. If unspecified, all data types are accepted. If you specify a type, the Java portlet container will enforce the type of the event's payload: incoming events with payloads not matching this type will not be delivered to the portlet, and if the portlet fires an event with a payload type not matching the declared type, an IllegalArgumentException will be thrown. Per the JSR 286 specification, any types declared here must have a valid JAXB binding; most Java object types representing primitives have such bindings, as does java.lang.String.
8. Click **OK** to create the event. The new event appears in the Event Type list in the Portlet Event Handlers dialog box.
9. Click **OK** in the Portlet Event Handlers dialog when you are finished entering processing events. Note that the event definition is not saved in the `portlet.xml` file until you save the `.portlet` file.

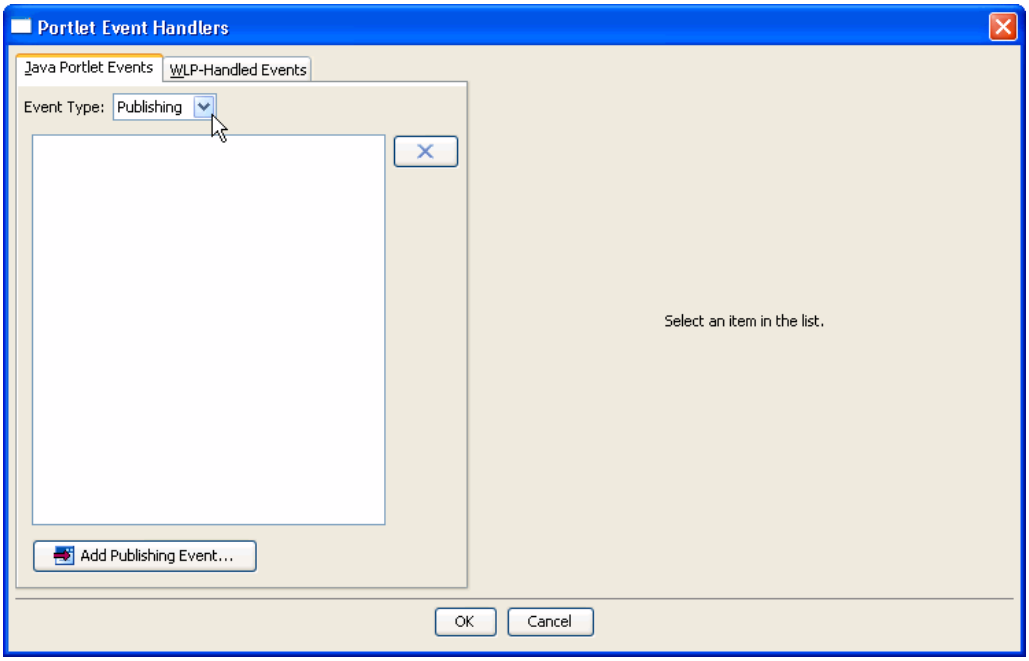
12.7.3 Adding a Publishing Event

By declaring a publishing event, you are giving a hint as to the types of events the portlet can send. There is no enforcement of this as a restriction at runtime.

To add a publishing event to a Java portlet:

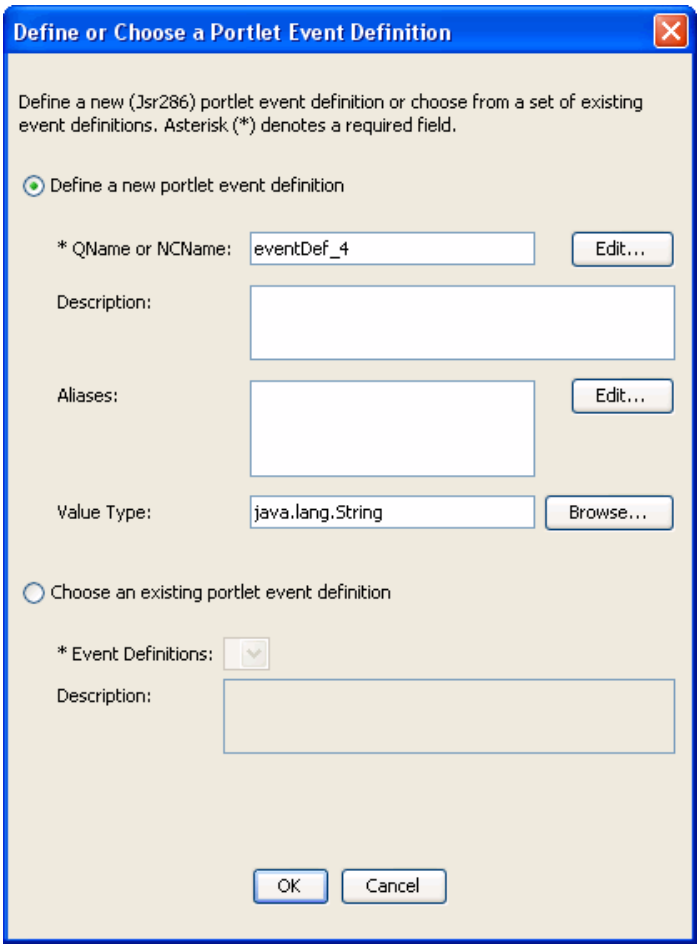
1. Open the Portlet Event Handlers dialog. You can open the dialog by clicking the Event Handlers button in the Properties view or by clicking the Event Handlers link in the portlet editor. Note that the entry fields in the dialog are validated; the **OK** button is disabled until all errors are corrected for the currently displayed properties.
2. In the Portlet Event Handlers dialog, select the Java Portlet Events tab.
3. Select Publishing from the Event Type menu, as shown in Figure 12–9.

Figure 12–9 Selecting a Publishing Event Type



4. Click **Add Publishing Event**. The Define or Choose a Portlet Event Definition dialog opens, as shown in Figure 12–10.

Figure 12–10 Define or Choose a Portlet Event Definition Dialog



5. To create a new event, select **Define a new portlet event definition**, and fill out the form. For the QName or NCName, you can edit the defaults by clicking the Edit button. This name is a unique name to identify the event. For more information, see Section 12.12, "About QNames and Aliases."

   Optionally, you can choose to select an existing event definition. To do this, select the radio button and pick an event from the drop down menu. This menu is only populated if one or more events were previously defined.

6. (Optional) For details on the Aliases section, see Section 12.12, "About QNames and Aliases." You can enter more than one alias.

7. (Optional) Enter a data type in the Value Type field. The Value Type specifies the Java object type of the event payload. If unspecified, all data types are accepted. If you specify a type, the Java portlet container will enforce the type of the event's payload: incoming events with payloads not matching this type will not be delivered to the portlet, and if the portlet fires an event with a payload type not matching the declared type, an IllegalArgumentException will be thrown. Per the JSR 286 specification, any types declared here must have a valid JAXB binding; most Java object types representing primitives have such bindings, as does java.lang.String.

8. Click **OK** to create the event. The new event appears in the Event Type list in the Portlet Event Handlers dialog box.

9. Click **OK** in the Portlet Event Handlers dialog when you are finished entering publishing events. Note that the event definition is not saved in the `portlet.xml` file until you save the `.portlet` file.

12.7.4 Modifying a Java Portlet Event

This section explains how to modify Publishing and Processing Java portlet events.

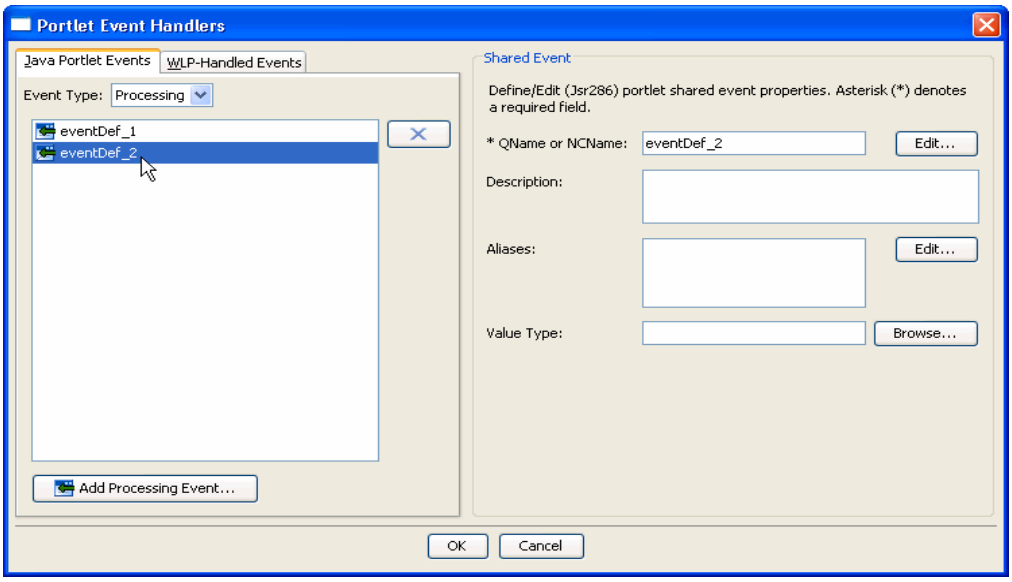
1. Open the Portlet Event Handlers dialog. You can open the dialog by clicking the Event Handlers button in the Properties view or by clicking the Event Handlers link in the portlet editor.
2. In the Portlet Event Handlers dialog, select the Java Portlet Events tab.

   Use the Event Type drop-down menu to select either a **Publishing** or **Processing**. Publishing events are sent by the portlet and Processing events are received by the portlet.

   Edit the selected event in the Java Portlet Event section on right side of the dialog box, as shown in Figure 12–11. After making changes to the editable fields, click **OK**. For more information on the editable fields, see Section 12.7.2, "Adding a Processing Event."

Note: All entry fields in the Portlet Event Handlers dialog are validated. An error dialog appears when the validation fails. You cannot click **OK** or switch events, tabs, or event types until the errors are corrected for the currently displayed event properties.

Figure 12–11 Editing a Java Portlet Event Definition



12.7.5 Deleting a Java Portlet Event

If you are deleting a Java portlet event, a dialog appears, and you must choose one of the following options:

- **Remove (disassociate) the Java portlet event from this portlet only.**

In this case, the event definition remains in the `portlet.xml` file. Only the mapping to the current portlet is removed. Choose this case if other portlets reference the event. For example, if multiple portlets reference the same event, you would want to leave the event definition in the `portlet.xml` file.

- **Delete the Java portlet event from the portlet.xml.**

  In this case, the entire event and all mappings to the event are removed from the `portlet.xml` file. Only pick this option if you know that no other portlets require the event definition.

For example, Example 12–1 shows an excerpt from a `portlet.xml` file where three events are defined in the `<event-definition>` elements: `eventDef_1`, `eventDef_2`, and `eventDef_3`. The definition for the portlet called `myJsf` has all three events added to it, in the `<supported-processing-event>` elements.

If you remove `eventDef_1` from the portlet named `myJsf` using the first option (disassociate), the `<supported-processing-event>` element named `eventDef_1` will be removed from the portlet definition, as shown in Example 12–2. Note that any other portlets will continue to reference the event. If you select the second delete option, the entire `<event-definition>` element and the `<supported-processing-event>` element(s) will be removed. In this case, no portlets will reference the event.

Example 12–1 Event Definitions and Mapping Elements Before a Disassociate Operation

```
<portlet-app ...>
    <portlet>
        <portlet-name>myJsf</portlet-name>
        <portlet-class>JsfPortlet</portlet-class>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>view</portlet-mode>
        </supports>
        <portlet-info>
           <title>JSF Portlet</title>
        </portlet-info>
        <supported-processing-event><name>eventDef_1</name>
        </supported-processing-event>
        <supported-processing-event><name>eventDef_2</name>
        </supported-processing-event>
        <supported-processing-event><name>eventDef_3</name>
        </supported-processing-event>
    </portlet>
    ...
    <event-definition>
    <name>eventDef_1</name></event-definition>
    <event-definition>
    <name>eventDef_2</name></event-definition>
    <event-definition>
    <name>eventDef_3</name></event-definition>
    ....
</portlet-app>
```

Example 12–2 Event Definitions and Mapping Elements After Disassociate Operation

```
<portlet-app ...>
    <portlet>
        <portlet-name>myJsf</portlet-name>
        <portlet-class>JsfPortlet</portlet-class>
```

```
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>view</portlet-mode>
        </supports>
        <portlet-info>
           <title>JSF Portlet</title>
        </portlet-info>
        <supported-processing-event><name>eventDef_2</name>
        </supported-processing-event>
        <supported-processing-event><name>eventDef_3</name>
        </supported-processing-event>
    </portlet>
    ...
    <event-definition>
    <name>eventDef_1</name></event-definition>
    <event-definition>
    <name>eventDef_2</name></event-definition>
    <event-definition>
    <name>eventDef_3</name></event-definition>
    ....
</portlet-app>
```

12.8 Subscribing Java Portlets to Portal Framework Events

Java portlets can receive events generated by the portal framework (as opposed to events fired by other portlets) by simply subscribing to the appropriate event name. The table below gives the namespace and local name for the event QName for Java portlets to subscribe to for portal-generated events.

The column **From Self Instance Only** indicates whether the portlet will only receive the event if the event was fired for the particular portlet instance.

When the local name in the table below is in italics, it indicates that the event's local name should be the name of the specific action or event name the portlet wishes to receive an event about. For example, if the Java portlet wanted to receive an event whenever a JSF portlet triggered a JSF action named `reset`, the Java portlet would subscribe to an event with a QName consisting of the namespace `urn:com:oracle:wlp:netuix:event:faces` and local name `reset`.

Table 12–8 Namespaces and Local Names Portal Framework Events

Event Type	Sent	Namespace	Local Name	From Self Instance Only
Faces	When a JSF action occurs on a JSF portlet.	urn:com:oracle:wlp:netuix:event:faces	*actionName*	No
Generic	When another portlet sends a Generic event.	urn:com:oracle:wlp:netuix:event:generic	*eventName*	No
Custom	When any portlet sends a custom event.	See Section 12.8.1, "Custom Event Namespaces."	*eventName*	No
Struts	When a Struts action occurs on a Struts portlet.	urn:com:oracle:wlp:netuix:event:struts	*actionName*	No
PageFlow	When a page flow action occurs in a JPF portlet.	urn:com:oracle:wlp:netuix:event:pageflow	*actionName*	No
Init	Every time a portal page is rendered.	urn:com:oracle:wlp:netuix:event:init	init	No
LookAndFeelReinit	When a Look and Feel is dynamically changed.	urn:com:oracle:wlp:netuix:event:laf-reinit	reinit	No

Table 12–8 (Cont.) Namespaces and Local Names Portal Framework Events

Event Type	Sent	Namespace	Local Name	From Self Instance Only
Notification	When the user has a notification.	urn:com:oracle:wlp:netuix:event:notification	See Section 12.8.2, "Local Name for Notification Events."	No
Refresh	Whenever the portlet is refreshed.	urn:com:oracle:wlp:netuix:event:refresh	refresh	No
Activation	When a portlet first becomes visible.	urn:com:oracle:wlp:netuix:event:window	activate	Yes
Deactivation	When a portlet ceases to be visible.	urn:com:oracle:wlp:netuix:event:window	deactivate	Yes
Mode Change	When the portlet's mode changes.	urn:oasis:names:tc:wsrp:v2:types	newMode	Yes
State Change	When the portlet's state changes.	urn:oasis:names:tc:wsrp:v2:types	newWindowState	Yes

12.8.1 Custom Event Namespaces

For custom events, the namespace of the event is the namespace the event was sent with. For non-Java portlets, if no namespace was specified for the custom event when it was sent, the default custom event namespace of `urn:com:oracle:wlp:netuix:event:custom` is applied. For example, if in the `.portlet` file for a non-Java portlet there is an event handler with an action of:

```
<netuix:fireCustomEvent event="myCustomEvent"/>
```

A Java portlet could receive this event by subscribing to an event with a QName consisting of the namespace `urn:com:oracle:wlp:netuix:event:custom` and local name `myCustomEvent`.

If an empty namespace is explicitly declared for a custom event, the event will not receive the default custom event namespace. For example, if in the `.portlet` file for a non-Java portlet there is an event handler with an action of:

```
<netuix:fireCustomEvent qname="{}myCustomEvent"/>
```

A Java portlet could receive this event by subscribing to an event with a QName consisting of the namespace "" (the empty string) and local name `myCustomEvent`.

12.8.2 Local Name for Notification Events

For notification events, the local name for the event to subscribe to is generated by appending the namespace of the notification payload, the string "_._", and the notification's payload name. In this way, Java portlets can subscribe to receive events only for the particular notification(s) they wish.

12.9 Public Render Parameters

Public render parameters are a JSR 286 feature that allows portlets to share simple String values with other portlets during all phases of the portlet lifecycle. For more information, see Section 6.8, "Public Render Parameters."

12.10 Shared Parameters

Shared parameters are the same as JSR 286 portlet public render parameters, but for non-Java portlet types. For details, see Section 9.3, "Using Shared Parameters."

12.11 IPC Special Considerations and Limitations

The following sections describe special considerations that you should keep in mind as you implement interportlet communications.

This section contains the following topics:

- Section 12.11.1, "Using Asynchronous Portlet Rendering with IPC"
- Section 12.11.2, "Consistency of the Listen To Field"

12.11.1 Using Asynchronous Portlet Rendering with IPC

Although IPC is not supported when asynchronous content rendering for specific portlets is enabled, WebLogic Portal provides some features that allow these two mechanisms to coexist in your portal environment. In addition, you can disable asynchronous rendering for single requests using the mechanisms described in Section 10.5.8, "Asynchronous Content Rendering and IPC."

> **Tip:** If you enable asynchronous rendering at the portal/desktop level, you can use IPC without restrictions. For more information on asynchronous portal/desktop rendering, see the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

12.11.2 Consistency of the Listen To Field

Pay attention to the **Listen To** field when you set up the listener portlet. The portlet definition you use on the consumer *must match* the WSRP portlet's portlet definition. For example, if you have "portlet_2" listening to "portlet_1", the WSRP portlet corresponding to "portlet_1"—the proxy on the consumer—must also have its portlet definition label set to "portlet_1". For more information on using IPC with WSRP, refer to the *Oracle Fusion Middleware Federated Portals Guide for Oracle WebLogic Portal*.

12.12 About QNames and Aliases

Portlet events and shared parameters (called public render parameters for Java portlets) both make use of QNames and optional aliases to allow interportlet communication. The QName provides a unique identifier for an event or shared parameter that is used internally by the portal framework when distributing events and shared parameter values. QNames allow a degree of isolation for portlets, to ensure that they do not interfere with other portlets' operations. Aliases provide a mechanism for renaming events or shared parameters as they are delivered to individual portlets, allowing communication between portlets that may not have been designed to communicate with each other.

A QName is a qualified name consisting of a Namespace URI and a Local Part, as defined by the Namespaces in XML standard from the W3C. This standard defines exactly what the namespace URI and Local Part can consist of, but to simplify, the namespace is any URI and the local part is an NCName – a string beginning with a letter or underscore followed by any letters, numbers, dashes ('-'), underscores ('_') or periods ('.'). In Java, the string representation of a QName object is

{*namespace*}*localName*, with the *namespace* inside of braces to distinguish it from the local name. For convenience in the rest of this section, this nomenclature will be used.

> **Tip:** The Provide QName Components dialog automatically forms the namespace/local part identifier in the standard syntax, for example: `{http://oracle.com/myevents}testevent`. If you enter the QName directly in the Define or Choose a Portlet Public Render Param dialog, you must use this syntax.

12.12.1 QNames and Aliases in Events

QNames are used as the unique identifying name for events. Having both a namespace and local name for the event allows portlet developers to develop a suite of portlets which communicate with each other, while minimizing the chances of name collisions with other portlets. For example, a set of travel portlets from company XYZ may communicate using an event named `{urn:com:xyz:travel}location.change`. If a set of map portlets from company ABC also makes use of an event with a local name of `location.change`, they can ensure that the event won't cause confusion for the portlets from XYZ by using a unique namespace, for example `{urn:com:abc:map}location.change`.

Aliases make it possible for a single event handler to accept events with multiple different QNames. Aliases let you declare that an event with a particular QName is functionally equivalent to an event with another QName. Aliases are simply a list of QNames. When the portal framework is distributing an event to portlets, if the event's QName matches a declared alias for another event definition, the event will be renamed to match the portlet's expected event QName before it is delivered.

For example, if portlet X has declared an event it wishes to receive with a QName of `{urn:com:xyz:travel}location.change`, with an alias to `{urn:com:abc:map}countrySelected`, when an event with a QName of `{urn:com:abc:map}countrySelected` is fired by any portlet, the event payload will remain the same but the event will be delivered to portlet X using the QName `{urn:com:xyz:travel}location.change`. In this way, portlet X can have a single event-handler which accepts only a single event name, but other events can be declaratively sent to this event handler using aliases.

Aliases apply only to the portlets that explicitly declare them. For example, given the following setup:

Portlet A:

- Handles Event: `{abc}zipCode`

Portlet B:

- Handles Event: `{def}zip`
  - Alias: `{abc}zipCode`
- Handles Event: `{xyz}postalCode`

Portlet C:

- Handles Event: `{xyz}postalCode`
  - Alias: `{abc}zipCode`
  - Alias: `{def}zip`

If an event is sent (by any portlet) with a QName of `{abc}zipCode`, Portlet A will receive the event with a QName of `{abc}zipCode`, portlet B will receive the event

with a QName o `{def}zip`, and portlet C will receive the event with a QName of `{xyz}postalCode`.

If an event is sent (by any portlet) with a QName of `{def}zip`, Portlet A will not receive the event, Portlet B will receive the event with a QName of `{def}zip`, and Portlet C will receive the event with a QName of `{xyz}postalCode`.

12.12.2 QNames and Aliases in Shared Parameters / Public Render Parameters

Shared parameters (called public render parameters in JSR 286 portlets) are accessed by portlets using a simple String name, called the identifier. When a portlet wants to read or set the value of a shared parameter in its code, it does so using this identifier. Since the identifier is used as a parameter name, there is a high probability it could conflict with another parameter of the same name being used by a different portlet.

Note: For detailed information on Shared Parameters, see Section 9.3, "Using Shared Parameters." For detailed information on Public Render Parameters, see Section 12.9, "Public Render Parameters."

For example, a map portlet may use a parameter called `location` to store latitude and longitude coordinates, while a wiki portlet in the same portal may use a parameter called `location` to keep information on what content is currently being displayed. To avoid these name collisions with shared parameters, each shared parameter is declared with both an identifier (the name used by the portlet to access the parameter), and a QName (the unique name used by the portal framework to distribute the value to other portlets).

For example, if there are three portlets on a page:

Portlet A:

- Shared Parameter Identifier: `location`
- Shared Parameter QName: `{urn:abc}map.location`

Portlet B:

- Shared Parameter Identifier: `location`
- Shared Parameter QName: `{urn:xyz}wiki.page.name`

Portlet C:

- Shared Parameter Identifier: `coordinates`
- Shared Parameter QName: `{urn:abc}map.location`

In this scenario, Portlets A and C share the parameter with QName `{urn:abc}map.location`. Portlet A accesses it using the parameter name `location`, while portlet C accesses it using the parameter name `coordinates`. When Portlet B sets a parameter named `location`, even though it has the same identifier as the shared parameter from Portlet A, Portlet A would not see the new value because the QNames are different.

Aliases are used for shared parameters to give an alternate QName for a shared parameter, as a means of "inheriting" the value of other shared parameters. In most cases this is simple and straightforward, but because of how shared parameter values are distributed, this can have some side-effects that are not immediately obvious. For example, given the following setup:

Portlet A:

- Shared Parameter Identifier: `firstName`
- Shared Parameter QName: `{urn:abc}customer.name.first`
  - Alias: `{urn:xyz}customerFirstName`

Portlet B:

- Shared Parameter Identifier: `first`
- Shared Parameter QName: `{urn:xyz}customerFirstName`

Portlet C:

- Shared Parameter Identifier: `name`
- Shared Parameter QName: `{urn:qrs}customer.name`
  - Alias: `{urn:abc}customer.name.first`

In this scenario, if portlet B sets the value of its shared parameter `first` to `Alice`, the portal framework will distribute the shared parameter value using the QName `{urn:xyz}customerFirstName`. Because portlet A has aliased its shared parameter to that QName, portlet A will then see a value of `Alice` for its parameter named `firstName`. Portlet C will not see a new value for its shared parameter `name`.

Continuing this scenario, if portlet A sets its parameter `firstName` to `Bob`, portlet C will then see the value of its parameter named `name` as `Bob`. Portlet B's value of parameter `first` remains `Alice`, as it was not aliased to portlet A's shared parameter.

Furthermore, if portlet C then sets its parameter `name` to `Carl`, there are no aliases for portlet C's shared parameter QName in any of the other shared parameter declarations, so portlet A's parameter `firstName` will have a value of `Bob`, portlet B's parameter `first` will have a value of `Alice`, and portlet C's parameter `name` will have a value `of` Carl.

To avoid this conditional distribution of shared parameter values, you can set up reciprocal aliases. For example:

Portlet A:

- Shared Parameter Identifier: `firstName`
- Shared Parameter QName: `{urn:abc}customer.name.first`
  - Alias: `{urn:xyz}customerFirstName`
  - Alias: `{urn:qrs}customer.name`

Portlet B:

- Shared Parameter Identifier: `first`
- Shared Parameter QName: `{urn:xyz}customerFirstName`
  - Alias: `{urn:abc}customer.name.first`
  - Alias: `{urn:qrs}customer.name`

Portlet C:

- Shared Parameter Identifier: `name`
- Shared Parameter QName: `{urn:qrs}customer.name`
  - Alias: `{urn:abc}customer.name.first`

- Alias: `{urn:xyz}customerFirstName`

Because each shared parameter declaration in this setup has aliases to the other two, setting a shared parameter value in any of the portlets will cause all of the portlets to receive the same value.

13

Interportlet Communication Example With Event Handling

This chapter describes the process of setting up interportlet communications (IPC) between two portlets by using the Portal Event Handlers wizard in Oracle Enterprise Pack for Eclipse. This is a simple example in which minimizing one portlet changes the text string in another portlet in the portal.

Read this chapter to become familiar with the basic concepts of IPC and the IDE features that help you configure interportlet communication. For more detailed information about IPC and the IDE user interface, see Chapter 12, "Configuring Local Interportlet Communication."

This section contains the following topics:

- Section 13.1, "Before You Begin – Environment Setup"
- Section 13.2, "Basic IPC Example"

13.1 Before You Begin – Environment Setup

Before you use the interportlet communication example in this chapter, you must have an existing portal development environment, consisting of a domain, Portal EAR project, Portal Web project, Datasync project, and portal. To complete the pre-requisite tasks, perform the tasks described in *Oracle Fusion Middleware Tutorials for Oracle WebLogic Portal*, using the information in Table 13–1 to enter the necessary values.

1. Create a Portal domain (server).

   Note: For detailed instructions on creating a domain using the Domain Configuration Wizard, see the *Oracle Fusion Middleware Tutorials for Oracle WebLogic Portal*.

2. Create a Portal EAR project.
3. Associate the EAR project with the server.
4. Create a Portal web project.
5. Create a portal.

Table 13–1 IPC Example - Environment Setup Values

Setup Information	Notes/Values
Domain Configuration Wizard - Welcome	Create a new WebLogic domain (the default) **Note:** For detailed instructions on creating a domain using the Domain Configuration Wizard, see the *Oracle Fusion Middleware Tutorials for Oracle WebLogic Portal*.
Domain Configuration Wizard - Select Domain Source	In the **Generate a domain configured automatically to support the following Oracle products** list, select **WebLogic Portal**. When you do this, other components are selected automatically; *keep all of them selected*.
Domain Configuration Wizard - Configure Administrator Username and Password	User name: `weblogic` User password: `webl0gic` Confirm user password: `webl0gic` **Note:** The password must include at least one non-alphabetical character. In this case, the "0" in `webl0gic` is a zero.
Domain Configuration Wizard - Configure Server Start Mode and JDK	Development Mode (the default) JRockit SDK
Domain Configuration Wizard - Customize Environment and Services Settings	No (the default)
Domain Configuration Wizard - Create WebLogic Domain	Domain name: `ipcDomain` Domain location: Accept the default, or specify another directory on your system.
Portal EAR Project Wizard	EAR Project Name: `ipcEAR` Switch to the Portal Perspective if you are not already using it.
Servers view	Right-click the server in the Servers view and select **Add and Remove Projects** Associate the `ipcEAR` project with the portal domain `ipcDomain`.
Portal Web Project Wizard	Web Project Name: `ipcTestWebProject` In the **Add project to an EAR** checkbox: Check the box and add to `ipcEAR`
Portal Wizard	Right-click the `ipcWebProject/WebContent` folder and select **New** > **Portal** Portal Name: `ipcPortal`

With a development environment set up, you can complete the steps described in this section:

- Section 13.2, "Basic IPC Example"

In this exercise, you create individual portlets, JSPs, and backing files to establish interportlet communications within the portal project. You then add these portlets to a portal and test the project to ensure that communication is successful.

13.2 Basic IPC Example

It might be helpful to become familiar with the Portal Event Handlers Wizard and backing files before attempting to replicate this example. For more information about

the wizard, refer to Section 12.5, "Using the Portlet Event Handlers Wizard." For more information on backing files, refer to Section 9.4, "Backing Files."

This exercise includes five main tasks:

1. Section 13.2.1, "Create the Portlets"
2. Section 13.2.2, "Create the Backing File"
3. Section 13.2.3, "Attach the Backing File"
4. Section 13.2.4, "Add the Event Handler to bPortlet"
5. Section 13.2.5, "Test the Project"

13.2.1 Create the Portlets

In this section, you create two JSP files and the JSP portlets that surface these files. You also create a backing file that contains the instructions necessary to complete the communication between the two portlets, and you add an event handler to one of the portlets. After you have created the portlets and attached the backing file, you test the project in your browser.

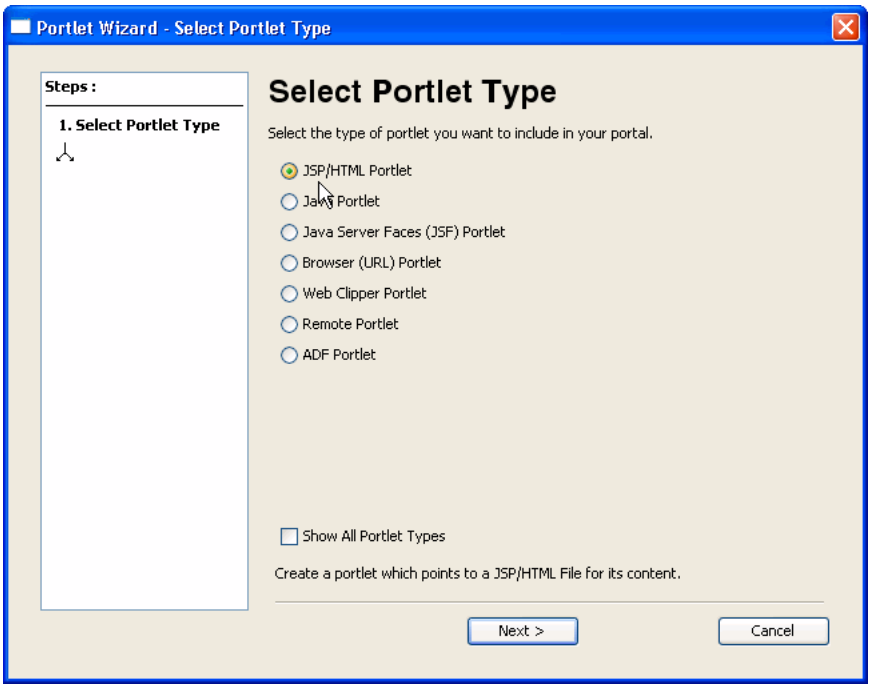
Note: Before continuing with this procedure, ensure that Oracle Enterprise Pack for Eclipse is running and the ipcWebProject node is expanded.

13.2.1.1 Create the JSP Files and Portlets

To create the JSP files that the portlets will surface, do the following:

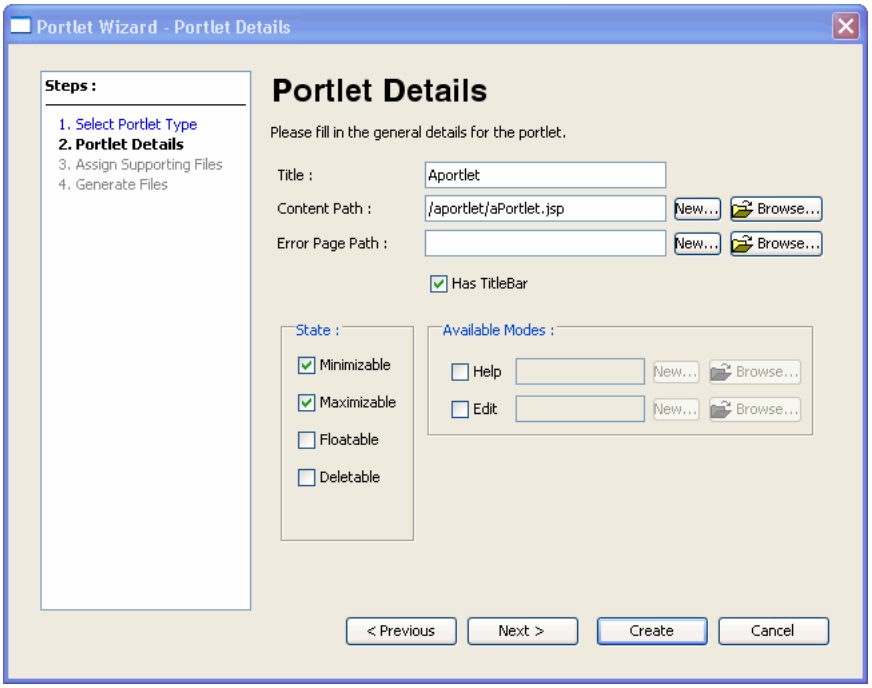
1. In the Project View, right-click the WebContent folder and select **New > Portlet**.
2. In the New Portlet dialog, enter the name `aPortlet.jsp` for the new portlet, and click **Next**.
3. In the Select Portlet Type wizard page, select **JSP/HTML Portlet**, and **click Next**. (See Figure 13–1.)

Figure 13–1 Select Portlet Type



4. In the Portlet Details wizard page, select the **Minimizable** and **Maximizable** states, and click **Create**. (See Figure 13–2.)

Figure 13–2 Portlet Details



5. Locate the `aPortlet.jsp` file. By default, it will be in the `WebContent/aportlet` folder. Double-click the file to open it in the editor.
6. Replace the default text in the file with the text `"Minimize Me!"` as shown in Figure 13–3.

Figure 13–3 JSP File Showing Edited Body Text

7. Save the file as `aPortlet.jsp`
8. In the same directory, make a copy of aPortlet.jsp and give the name `bPortlet.jsp` to the copy.
9. Open `bPortlet.jsp` in the workbench editor if it is not already open.

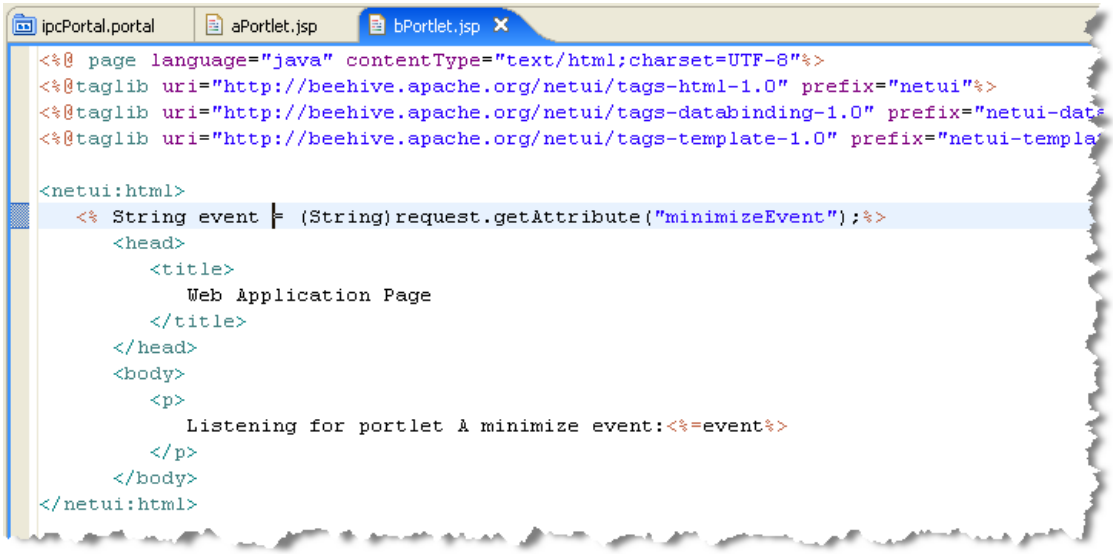
   The XML code for the JSP file appears.
10. Copy the code from Example 13–1 into the JSP, replacing everything from `<netui:html>` through `</netui:html>`. This code displays event handling from the backing file that you will create and attach in a subsequent step.

Example 13–1 New JSP Code for bPortlet.jsp

```
<netui:html>
   <% String event = (String)request.getAttribute("minimizeEvent");%>
      <head>
         <title>
            Web Application Page
         </title>
      </head>
      <body>
         <p>
            Listening for portlet A minimize event:<%=event%>
         </p>
      </body>
</netui:html>
```

The source should look like the example in Figure 13–4.

Figure 13–4 Updated bPortlet JSP Source



11. Save the file.
12. Following the same steps you used previously, generate a portlet from the `bPortlet.jsp` file.

Checkpoint: At this point the `ipcWebProject/WebContent` folder contains these files: `aPortlet.jsp`, `aPortlet.portlet`, `bPortlet.jsp`, and `bPortlet.portlet`.

13.2.2 Create the Backing File

To create the backing file, do the following:

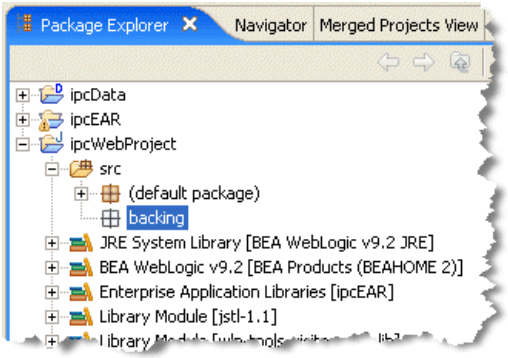
1. In `ipcTestWebProject`, select the `Java Resources/src` folder and select **File** > **New** > **Folder** from the main menu.

   The Create New Folder dialog box appears.

2. Create a folder called `backing`.

   The folder `backing` will appear under `ipcTestWebProject/src`, as shown in Figure 13–5.

Figure 13–5 New Backing File Folder in Package Explorer View

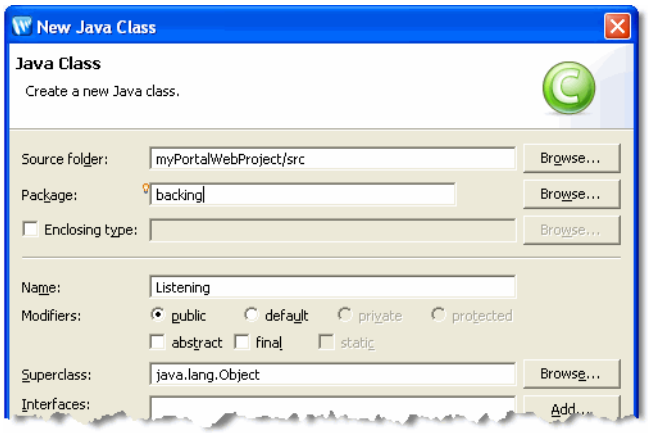


3. Right-click the `backing` folder and select **New** > **Other**.

4. In the **New – Select a wizard** dialog, select **Java** > **Class**, and click **Next**.

   The New Java Class dialog appears, as shown in Figure 13–6. The Source folder field auto-fills with the default path; leave it as is. The Package field auto-fills with `backing`; leave it as is.

Figure 13–6 New Java Class Dialog



5. In the **Name** field, enter `Listening` and click **Finish**.

   The new Java class appears in the editor.

6. Delete the entire default contents of `Listening.java`, and copy the code from Example 13–2 into the file. Figure 13–7 shows the top portion of the `Listening.java` file as it should look after you paste the code into it.

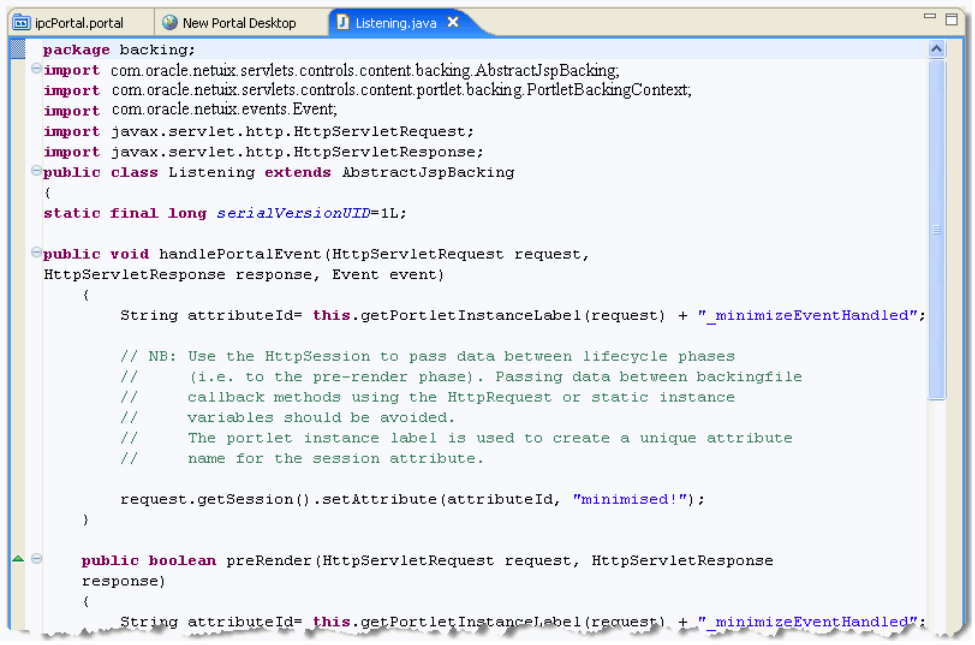
7. When you're finished, save the file.

Example 13–2 Backing File Code for Listening.java

```
package backing;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;
import com.bea.netuix.servlets.controls.portlet.backing.PortletBackingContext;
import com.bea.netuix.events.Event;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Listening extends AbstractJspBacking
{
   static final long serialVersionUID=1L;
   public void handlePortalEvent(HttpServletRequest request,
   HttpServletResponse response, Event event)
   {
      String attributeId= this.getPortletInstanceLabel(request) +       "_
minimizeEventHandled";
       // NB: Use the HttpSession to pass data between lifecycle phases
       //     (that is, to the pre-render phase). Passing data between
       //     backing file callback methods using the HttpRequest or static
       //     instance variables should be avoided.
       //     The portlet instance label is used to create a unique
       //     attribute name for the session attribute.

      request.getSession().setAttribute(attributeId, "minimized!");
   }
   public boolean preRender(HttpServletRequest request, HttpServletResponse
   response)
```

```
    {
       String attributeId= this.getPortletInstanceLabel(request) +
       "_minimizeEventHandled";
          if (request.getSession().getAttribute(attributeId) != null)
          {
             // Reset the session flag
              request.getSession().removeAttribute(attributeId);
             // Pass minimize event notification to the JSP via the request.
             request.setAttribute("minimizeEvent", "Minimize event handled");
          }
          else
          {
             request.setAttribute("minimizeEvent", null);
          }
       return true;
    }
    private String getPortletInstanceLabel(HttpServletRequest request)
    {
       PortletBackingContext context=
       PortletBackingContext.getPortletBackingContext(request);
       return context.getInstanceLabel();
    }
}
```

Figure 13–7 Listening.java with Updated Backing File Code

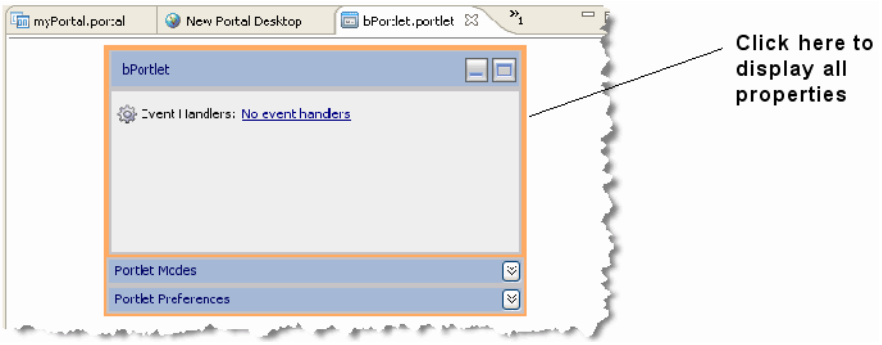


13.2.3 Attach the Backing File

Now you will attach the backing file created in the previous section to `bPortlet.portlet`. Perform the following steps:

1. In the Package Explorer, double-click `bPortlet.portlet` to open it.
2. Click on the portlet in the editor, if needed, to display the portlet's properties. You should see an orange border around the outside of the portlet, as shown in Figure 13–8.

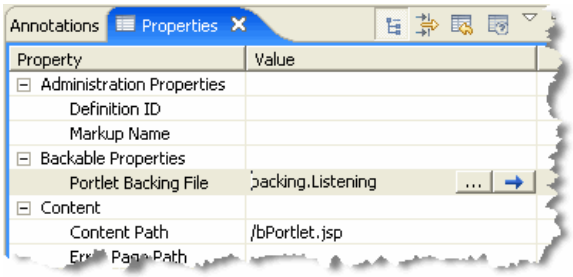
Figure 13–8 bPortlet with Outer Border Selected to Display Properties



Tip: The Properties view is a default view in the Portal perspective. If it is not visible, select Window > Show View > Properties.

3. In the Properties view, enter `backing.Listening` into the **Backable Properties > Portlet Backing File** field, as shown in Figure 13–9.

Figure 13–9 Attaching the Backing File in the Properties View



4. Save the portlet file.

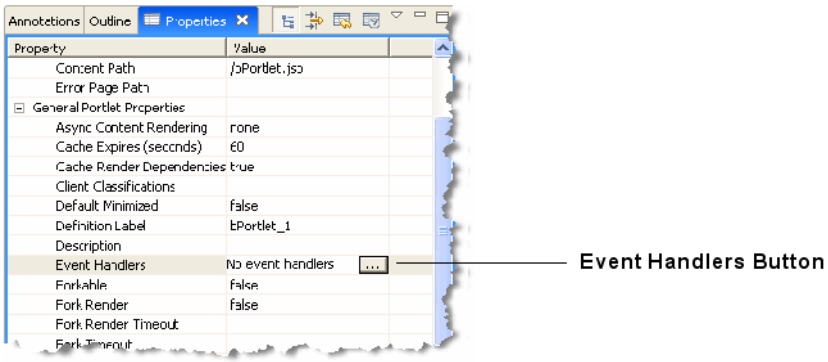
13.2.4 Add the Event Handler to bPortlet

You now add the event handler to `bPortlet.portlet`. This handler will be set up so that it will listen for an event on a specific portlet and fire an action in response to that event. To add the event handler, perform the following steps:

Note: bPortlet.portlet should be displayed in the Oracle Enterprise Pack for Eclipse editor. If it isn't, locate it in the ipcTestWebProject/WebContent folder in the application panel and double-click it.

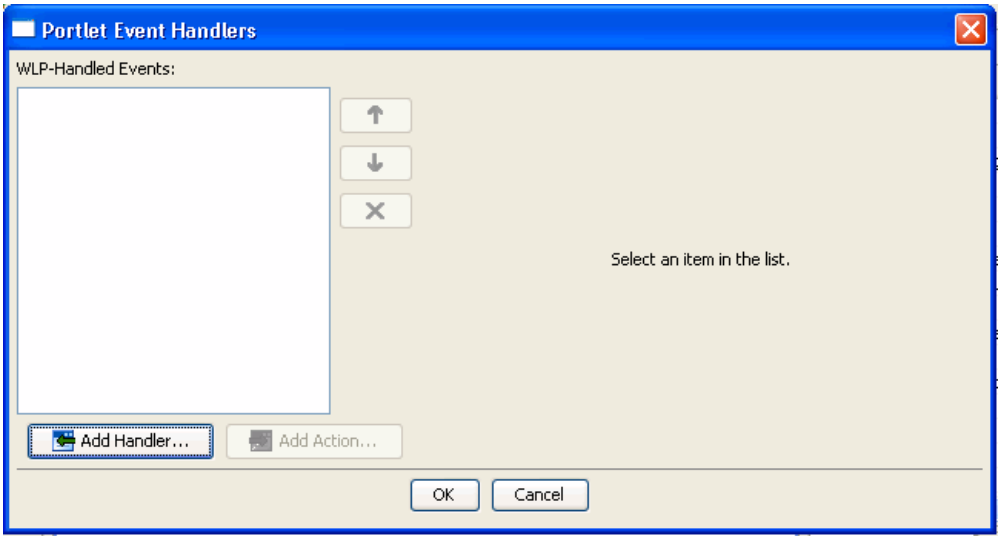
1. Click on the portlet in the editor if needed to display its properties.
2. In the Properties view, click in the Value column of the Event Handlers property. A browse button appears, as shown in Figure 13–10.

Figure 13–10 Event Handlers Button



3. Click the ellipsis button to display the Portlet Event Handlers dialog, as shown in Figure 13–11.

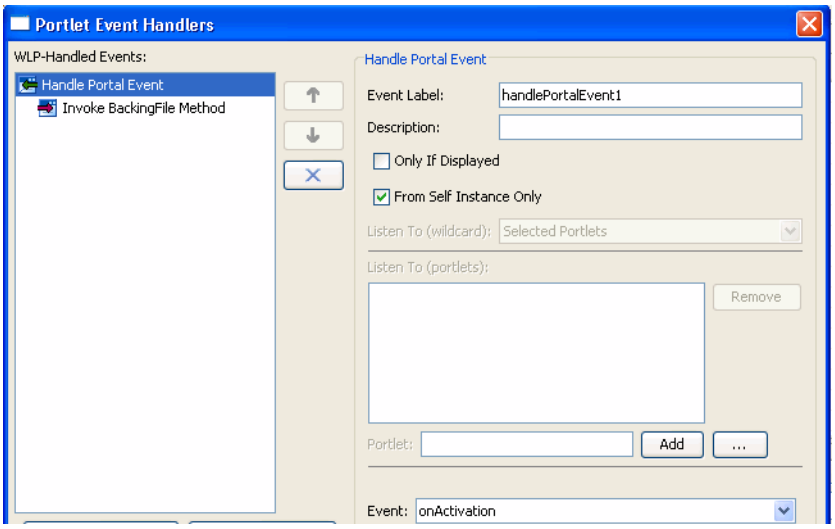
Figure 13–11 Portlet Event Handlers Dialog Box



4. Click **Add Handler** to open the **Event Handler** drop-down list.
5. From the drop down list, select **Handle Portal Event**.

   The Portlet Event Handlers dialog box expands to allow entry of more details, as shown in Figure 13–12.

Figure 13–12 Event Handler Dialog Box Expanded

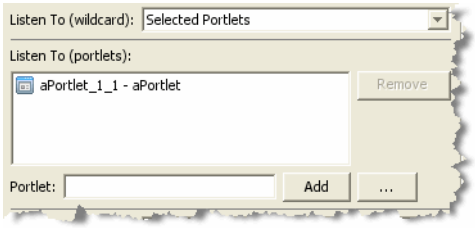


6. Accept the defaults for all fields except **Portlet**.
7. In the **Portlet** field, click the ellipses button.

   The Please Choose a File dialog appears.
8. Click `aPortlet.portlet` and click **OK**.

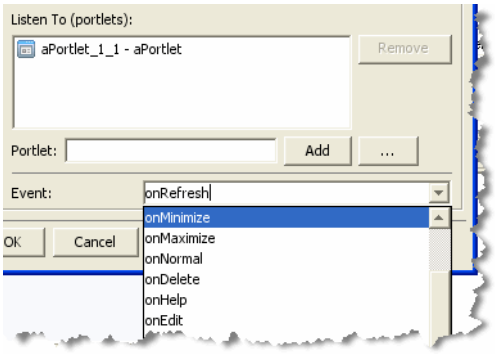
   The dialog box closes and aPortlet_1 appears in the **Listen to (portlets)**: list, as shown in Figure 13–13. The label **aPortlet_1** is the definition label of the portlet to which the event handler will listen.

Figure 13–13 Adding portlet_1



9. Click the **Event** drop-down control to open the list of portal events that the handler can listen for and select **onMinimize**, as shown in Figure 13–14.

Figure 13–14 Event Drop-down List



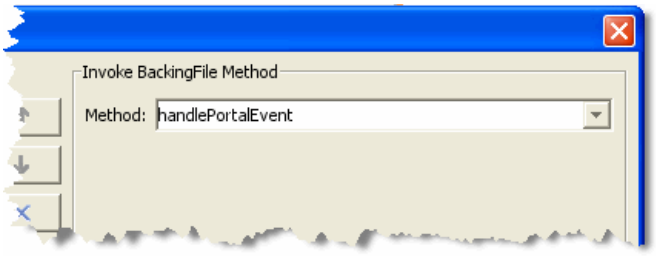
10. Click **Add Action** to open the action drop-down list and select **Invoke BackingFile Method**.

    The Invoke BackingFile selection will not appear unless a backing file is detected by WebLogic Portal.

11. In the **Method** field, enter **handlePortalEvent**, as shown in Figure 13–15.

    The dropdown menu for this field displays the last several values that you entered, if applicable.

Figure 13–15 Adding the Backing File Method



12. Click **OK**.

    The event handler is added. Note that the **Value** field of the **Event Handlers** property now indicates `1 Event Handler`.

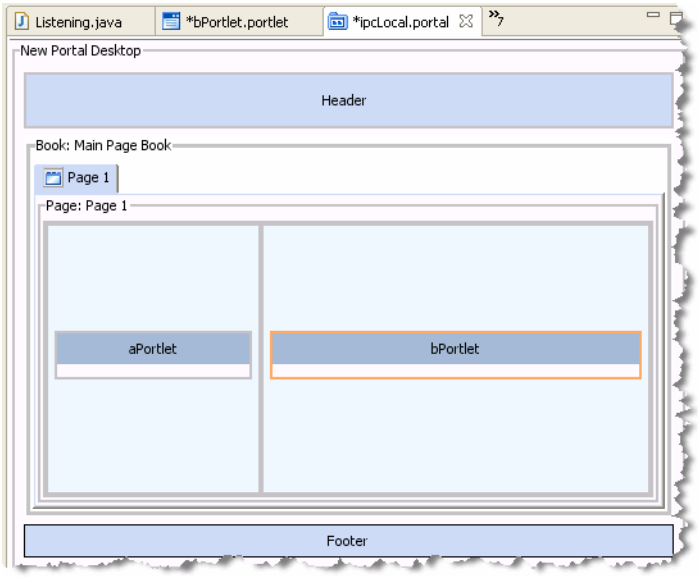
13.2.5 Test the Project

Test the communication between your portlets by following these steps:

> **Note:** Before you begin, ensure that all files are saved.

1. Select `ipcPortal.portal` to display it in the workbench editor.
2. Drag both `aPortlet.portlet` and `bPortlet.portlet` from the Package Explorer view onto the portal layout, as shown in Figure 13–16.

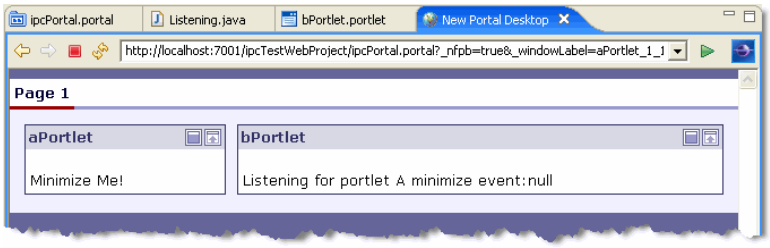
Figure 13–16 Portal Layout with aPortlet and bPortlet Added



3. Save the portal.
4. Run the portal. To do this, right-click `ipcPortal.portal` in the Package Explorer view and select **Run As** > **Run on Server**.
5. At the **Run On Server – Define a New Server** dialog, click **Finish**.

   Wait while the server starts and the application is published to the server. The portal will render in your browser (Figure 13–17).

Figure 13–17 ipcLocal Portal in Browser



6. Click the minimize button to minimize aPortlet.

   Note the content change in bPortlet, as shown in Figure 13–18.

Figure 13–18 ipcPortal Showing the Effect of Minimizing aPortlet

13.2.6 Summary

In this example, you set up your environment and you added two JSP portlets to a local portal. One portlet, aPortlet, was fairly simple, while the second portlet, bPortlet, surfaced a more complex JSP file, used a backing file, and contained a portal event handler. When you tested the communication between the portlets, you observed how the bPortlet changed when an event occurred on aPortlet. This is called local interportlet communication.

14

Adding the Content Presenter Portlet

The Content Presenter portlet allows users to retrieve and display different kinds of content in a portal in real time, without assistance from your IT Department or software developers. For example, you might want to display a list of the most recent Press Releases so users can browse them and click one to read the entire Press Release. You can place images (a photograph or a chart, for example) or add textual content on a portal page. You can also segregate content by subject matter to target different audiences.

> **Note:** Content Presenter requires that the Apache Beehive facets are installed in the web application. For information on installing Apache Beehive facets, see "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

In the Content Presenter Example, you can perform inline editing to quickly change the content that displays in the portlet.

> **Tip:** The Content Presenter portlet works only with streamed portals.

This chapter includes these sections:

- Section 14.1, "Using the Content Presenter Example"
- Section 14.2, "Configuring the Content Presenter Portlet in Your Portal"

14.1 Using the Content Presenter Example

WebLogic Portal includes a Content Presenter Example, and allows you to perform inline editing on a Content Presenter portlet to modify the portlet's content. Editing the portlet's content also changes the content in the content repository. The Example's Content Presenter portlet uses the public Dojo rich text editor.

This section contains the following topics:

- Section 14.1.1, "Starting the Content Presenter Example"
- Section 14.1.2, "Performing Inline Editing in the Content Presenter Example"
- Section 14.1.3, "Enabling Inline Editing in Your Portlets"
- Section 14.2.1, "Configuring the Content Presenter Portlet"

14.1.1 Starting the Content Presenter Example

> **Note:** You must install the Portal Examples before you perform these steps. See "Installing the Portal Examples" in the *Oracle Fusion Middleware Release Notes for Oracle WebLogic Portal* for details.

To start the Content Presenter sample:

1. From the Windows Start Menu, start the WebLogic sample server. (You can also double-click the `startWebLogic.cmd` file located in the `<WLPORTAL_HOME>/samples/domains/portal/bin` directory.)
2. After the server starts, from the Windows Start Menu choose **Oracle Products** > **WebLogic Portal** > **Examples** > **Visit Portal Examples**.
3. On the WebLogic Portal Sample Domain, select **Go to the Content Presenter demo**. (You can also launch the Content Presenter Example in a browser at `http://localhost:7041/contentpresenter/`).
4. Enter your user name and password and click **Login**.

See Section 14.1.2, "Performing Inline Editing in the Content Presenter Example" for instructions on how to perform inline edits to the content in a portlet in the Content Presenter Example.

14.1.2 Performing Inline Editing in the Content Presenter Example

The Content Presenter Example is the only portal where you can perform inline HTML content editing without doing additional setup tasks. By default, the Content Presenter Example lets you immediately edit the content in the Letter from the CEO Portlet, or you can enable the Training Announcement Portlet for inline editing. Inline editing in the Content Presenter Example works only on single-item portlets.

The Letter from the CEO portlet in the Content Presenter Example is already configured for inline editing because it uses the template view with inline editing enabled and has the appropriate entitlement rights. To configure the Training Announcement portlet for inline editing, see Section 14.1.2.2, "Enabling Inline Editing for the Training Announcement Portlet.")

This section contains the following topics:

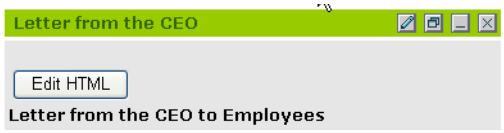
- Section 14.1.2.1, "Entering Inline Edits"
- Section 14.1.2.2, "Enabling Inline Editing for the Training Announcement Portlet"

14.1.2.1 Entering Inline Edits

To enter inline edits to the Letter from the CEO portlet:

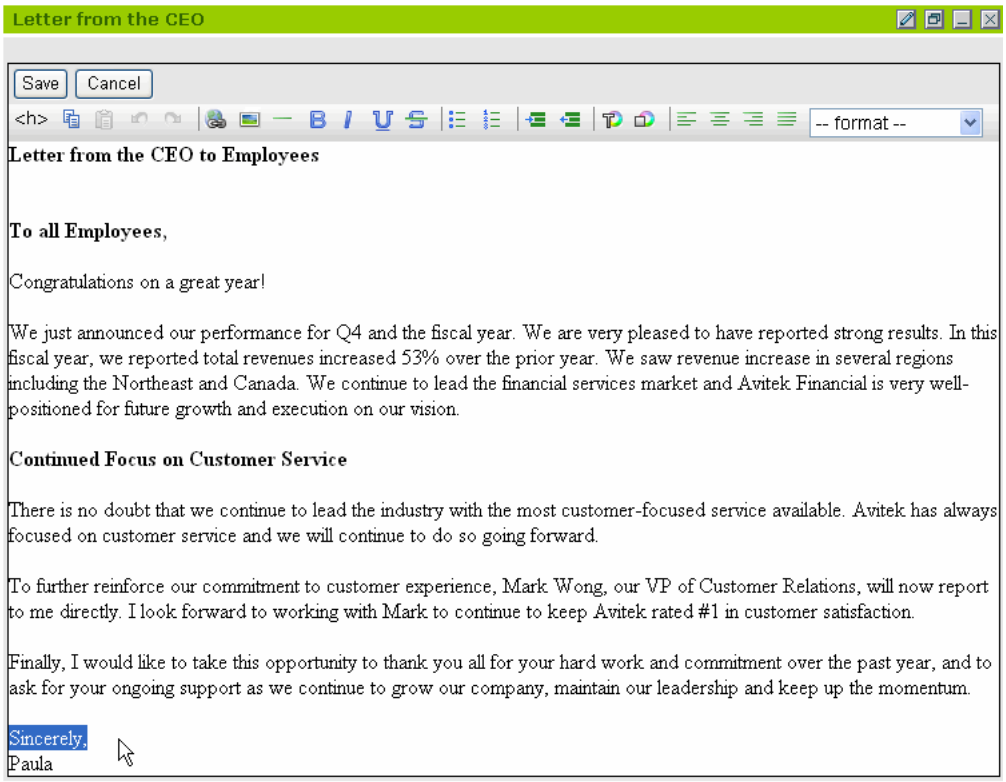
1. Follow the steps in Section 14.1.1, "Starting the Content Presenter Example" to start the Content Presenter Example and log in.
2. After you log in, click **Edit HTML** in the Letter from the CEO portlet, as shown in Figure 14–1. The Content Presenter Configuration Wizard appears.

Figure 14–1 Click the Edit HTML Button that Appears in the Text of the Portlet



3. Enter your edits or insert a link to other content or graphics outside your content management system. For example, before the signature line, type **Sincerely**, as shown in Figure 14–2.

Figure 14–2 Enter Text in the Signature Line



4. Click **Save**. Your changes appear in the portlet and are saved to the CM Repository.

> **Tip:** You can perform inline editing on two portlets in the Content Presenter Example. Inline editing is not available for the Content Presenter portlets in your portal.

14.1.2.2 Enabling Inline Editing for the Training Announcement Portlet

One other portlet in the Content Presenter Example, the Training Announcement portlet, allows inline editing. Inline editing is turned off by default for this portlet, so you must first enable the inline editing capability by choosing a different content display template view.

To enable inline editing for the Training Announcement portlet:

1. Follow the instructions in Section 14.1.1, "Starting the Content Presenter Example" to log into the Content Presenter Example.

2. In the Training Announcement portlet in the Content Presenter Example, click **Edit** in the portlet's title bar, as shown in Figure 14–3.

Figure 14–3 Click Edit to Enable Inline Editing for this Portlet

3. In the Content Presenter Configuration Wizard's Single or Multiple Items window, click **Next**. (If you use the wizard to change this portlet to allow multiple content items, rather than a single item, you cannot enable inline editing. For more information on configuring the portlet, see Section 14.2.1, "Configuring the Content Presenter Portlet.")
4. In the wizard's Select Content window, click **Next**.
5. In the wizard's Select Template & View window, select **Single Item View with Inline Edit** and click **Next**.
6. In the wizard's Portlet Properties window, click **Next**.
7. In the wizard's Finish window, click **Save**. The Edit HTML button appears in the Training Announcement portlet. If you do not see the Edit Content button, you might not have the correct entitlement. See the *Oracle Fusion Middleware Security Guide for Oracle WebLogic Portal* for instructions on setting entitlements.
8. Click **Edit HTML** to change the content in the portlet or insert a link to other content or graphics outside your content management system.
9. Click **Save**. Your changes are saved to the CM Repository.

14.1.3 Enabling Inline Editing in Your Portlets

You can use the three sample JSP files that ship with WLP to enable inline HTML editing in your own Content Presenter portlets. Inline editing does not work with library services enabled, because library services support versioning.

Edit the files in the order listed below. The files are located in the following directories:

1. Display Template (Outer Template) – `<WLPORTAL_HOME>\samples\applications\portalApp\contentPresenterSampleWeb\samplePresenterTemplates\inlineEditExamplePresenterTemplate.jsp`
2. CM Display Template (Inner Template) That Displays the Content – `<WLPORTAL_HOME>\samples\applications\portalApp\contentPresenterSampleWeb\sampleCMTemplates\inlineEditExampleCMTemplate.jsp`
3. JSP File that Performs Other Work – `<WLPORTAL_HOME>\samples\applications\portalApp\contentPresenterSampleWeb\sampleCMTemplates\saveNode.jsp`

The files include detailed comments to help you customize them for your portlets. For example, you might want to replace the DOJO rich text editor with your own rich text

editor. You might want to change the entitlements on the portlets or their look and feel.

14.2 Configuring the Content Presenter Portlet in Your Portal

The Content Presenter portlet ships with WebLogic Portal. You must configure the portlet before you can use it.

The Content Presenter portlet uses a portlet framework that is based on Content Management, metadata, and templates that let business users step through a wizard to quickly retrieve and display content that is appropriate to the audience. The framework allows WebLogic Portal customers to easily publish content in a variety of ways to almost any site.

The Content Presenter portlet can read content from any configured content provider. Content providers can be any repository that implements the WebLogic Portal Content Management Service Provider Interface, including third-party Content Management vendor products, file systems, or other database systems.

When you plan your Content Presenter portlet, determine who will view your content and if you plan to re-use the portlet later for a different audience. Plan entitlements to determine who can choose content to display in the Content Presenter portlet. If a logged-in user has Delegated Administration rights, the user can edit the content in the Content Presenter portlet.

You must be a member of the Portal System Administrators role or the Content Presenter Administrators role to configure the Content Presenter portlet. Portal System Administrators and Content Presenter Administrators have edit and delete capabilities on the Content Presenter portlet itself, and edit capabilities on any page where the portlet is placed.

If you plan to have a group of users (for example, a subset of the Portal System Administrators role) edit the Content Presenter portlet, this subgroup must have edit and delete capabilities for the Content Presenter portlet itself and edit capabilities for the page that contains the portlet. If the group does not have edit and delete capabilities to the portlet, the group's members will not be able to see the Edit icon in the portlet.

See the *Oracle Fusion Middleware Security Guide for Oracle WebLogic Portal* for more information on setting entitlements and creating roles and groups. Portal System Administrators and Content Presenter Administrators can also turn a portlet off by disabling the activation flag in the portlet's preferences in the Administration Console.

Caution: If you add or modify your Content Presenter portlet using the Visitor Tools (rather than the Administration Console), the portlet is no longer configurable in your desktop. You should add or modify your Content Presenter portlet in the Administration Console.

If you use Portlet Publishing to configure a Content Presenter portlet, some features are not available. See Section 14.2.1.2, "Using Portlet Publishing to Expose a Content Presenter Portlet" for more information.

This section contains the following topic:

- Section 14.2.1, "Configuring the Content Presenter Portlet"

14.2.1 Configuring the Content Presenter Portlet

The Content Presenter portlet ships with WebLogic Portal. You must add the Content Presenter facet in Oracle Enterprise Pack for Eclipse when you set up your portal web project. By default, your portal administrator has the ability to administer the Content Presenter portlet (to move portlets, turn a portlet off, and so on) in the Administration Console.

In a desktop, use the Content Presenter's Configuration Wizard to determine the content you want to display, and how to display it (through templates and template views). The Content Presenter portlet stores those choices as portlet preferences for each portlet instance.

Perform the following steps to configure the Content Presenter portlet:

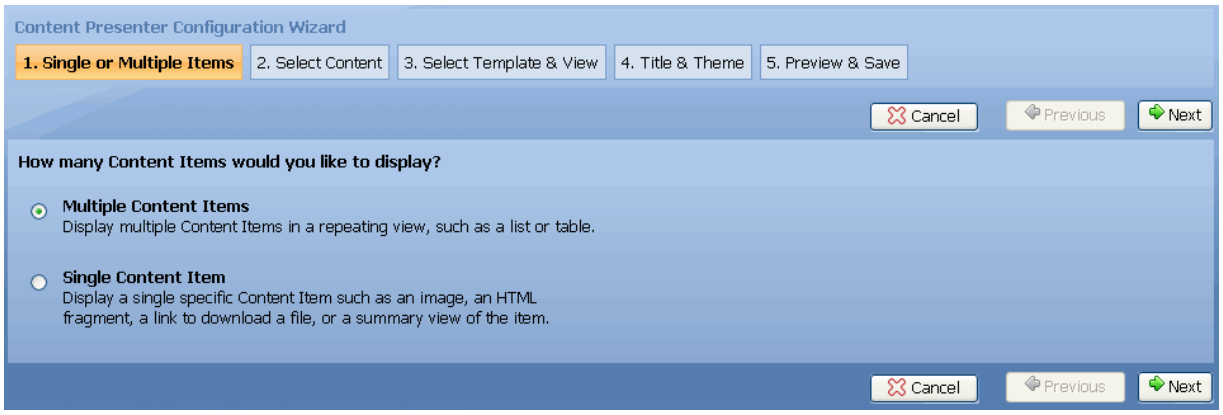
1. In Oracle Enterprise Pack for Eclipse, create and deploy a Portal EAR project, web project, a portal, and a method to authenticate users (such as a login portlet) according to the instructions in the *Oracle Fusion Middleware Tutorials for Oracle WebLogic Portal*. You should also create a Datasync project if you plan to use Content Selectors to display content in the Content Presenter portlet.

   Tip: When you create your Portal Web Project, you must select the **Content Presenter Framework** facet in the WebLogic Portal (Optional) directory in order to view and use the Content Presenter portlet.

2. Start the Administration Console and choose **Portal** > **Portal Management**.
3. Create a page and a desktop (you can choose to create a desktop from a desktop template, library resources, or a `.portal` file) according to the instructions in *Oracle Fusion Middleware Tutorials for Oracle WebLogic Portal*.
4. Select the page you created in Step 3 in the Portal Management tree in the `Library/Pages` directory (or wherever you saved it). Add the Content Presenter portlet to the page by clicking **Add Page Contents**. Click **Add Contents** in the appropriate column, select the check boxes next to the Content Presenter portlet and a login portlet (if that is the method you are using to authenticate users), and click **Save**. (If you do not see a list of portlets in the Add Books and Portlets to Placeholder page, click the drop-down box and select **Portlets**, and click **Show all**.)
5. In the Portal Management tree, select the **Portals** directory, the portal you created, and your desktop.
6. With the desktop selected, click **View Desktop**.
7. In the new browser window, log into the desktop and click **Open Configuration Wizard** in the **Content Presenter** portlet. A new unconfigured Content Presenter portlet does not appear until you log in, and only if you have rights to edit it.
8. In the Content Presenter Configuration Wizard, select one of the following in the **Single or Multiple Items** window:
   - **Multiple Content Items** – Pick more than one content item and display them. You can create a custom list of content, choose all content in a specific folder, use the results of a Content Selector, or run a search to find content. The result is a list of content items in the portlet.
   - **Single Content Item** – Browse or search for one item. The item can be an image, article, link to a file, link to a URL, and so on. The result is the content item displaying inline in the portlet.

See Figure 14–4. Click **Next** after you select an option.

Figure 14–4 Determine How Much Content You Want to Display



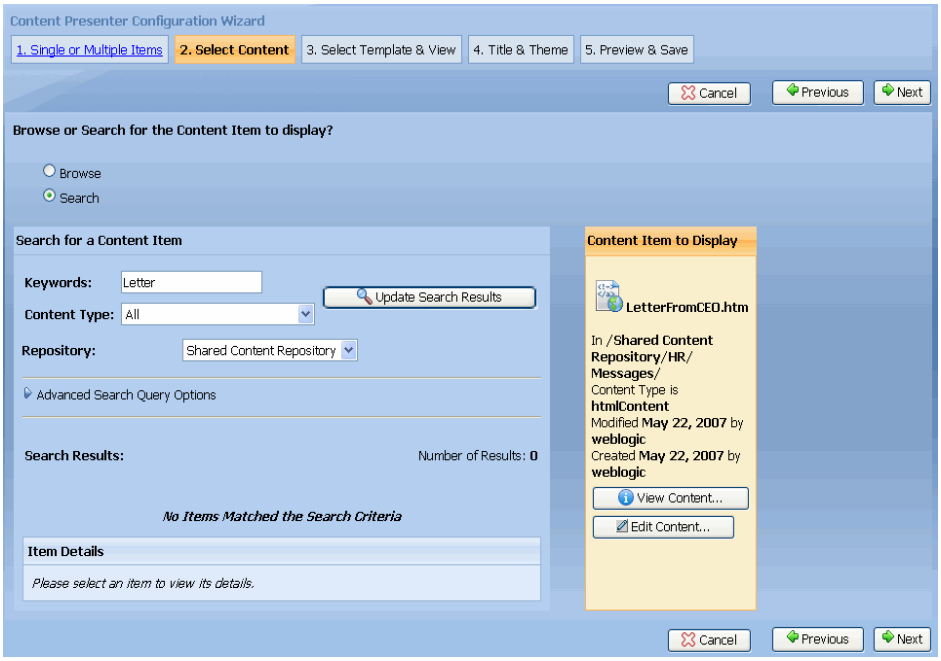
9. The choice you made in Step 8 in this section determines which wizard pages you see and what you choose next.

   a. If you chose **Single Content Item** in Step 8, you can select **Browse** in the wizard's **Select Content** window to navigate through the repository tree to locate content, select the item, and click **Next**.

      You can also select **Search** to find content by keyword or content type. (If you have multiple content repositories configured, you can also search by repository.) By default, all content types are listed in the Content Type field. If you want to control which content types appear in the Content Type field, you can set entitlements by user. In order to have these entitlements by user evaluated, override the `com.bea.content.ui.framework.AllowObjectClassViewRights` context parameter in the `web.xml` file. If you change the setting to **false**, the entitlements set on the content types determine the content types that are listed in the Content Type field. If you change the setting to **true**, or you omit the context parameter altogether, all content types appear for all users.

      See the *Oracle Fusion Middleware Security Guide for Oracle WebLogic Portal* for instructions on setting up users and roles in the Administration Console, and adding view capabilities so users can see the object classes in the Content Type field. Click **Update Search Results** to view the results. Adjust the **Items per Page** field to determine how many search results to display in the wizard. Select an item; see Figure 14–6. Click **Next**.

Figure 14–5 Narrow Your Search with a Keyword



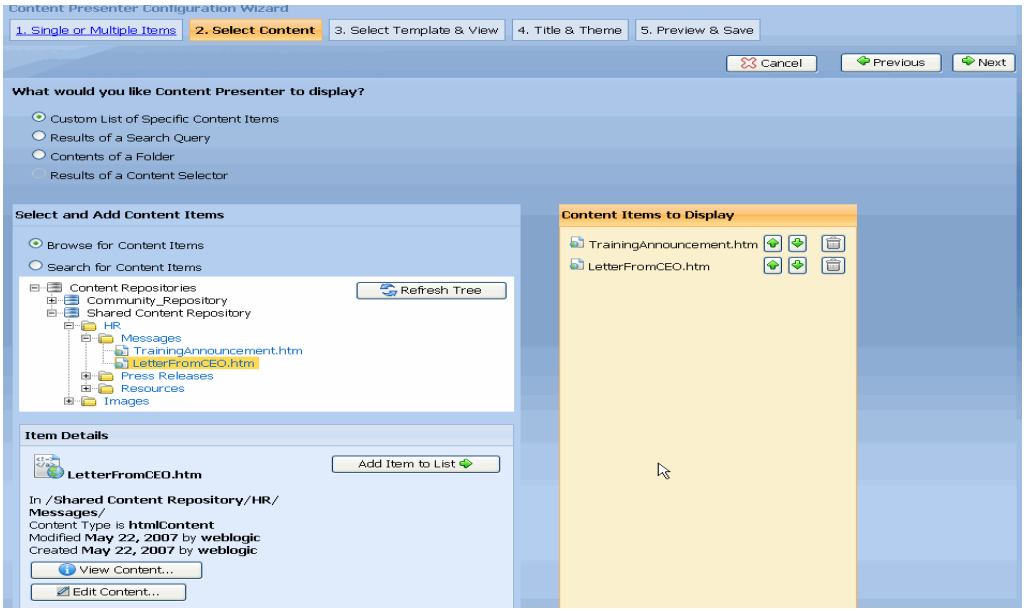
10. If you chose **Multiple Content Items** in Step 8 of this section, select one of the following in the wizard's **Select Content** window to specify your content items:

    – **Custom List of Specific Content Items** – This type of content retrieval is not a live search; it displays content from a list that you create. Click **Browse for Content Items** to navigate through the repository tree to locate content.

      You can also click **Search for Content Items**, enter a keyword, and click **Update Search Results** to retrieve content by keyword. If you have multiple content repositories configured, you can also search within a specific repository. To search your repository, you must configure it and make it searchable. The repository that ships with a WebLogic Portal for a new domain is not automatically indexed for searching. See the *Oracle Fusion Middleware Content Management Guide for Oracle WebLogic Portal* for instructions on indexing your repository.

      Figure 14–6 shows that after you retrieve multiple items and click **Browse for Content Items**, you can select an item and click **Add Item to List**. The custom list lets you control the order in which the content displays by selecting the check box next to the item and clicking **Move Up** or **Move Down**. Moving content to the top of the list ensures that a certain content item displays first. See Figure 14–7.

Figure 14–6 Custom List of Specific Content Items



You can also verify that you selected the correct content by clicking **View Content** to preview the content you selected. The actual content and other details appear in a separate window. If you selected an image, the image also appears. You can also click **Edit Content** to change the property values of the item. If the item is a binary file, you can download the item, upload new values, or change the property values.

After you determine the order of the content, click **Next**.

– **Results of a Search Query** – Locate content by entering a keyword or selecting a content type (such as an image or a book) and clicking **Preview Query Results**. You can enter multiple keywords (separated by a space) and the *or* connector is assumed. For example, if you search for *IRA retirement*, results will include the keyword *IRA* or *retirement*. If you have multiple content repositories configured and you are entitled to view them, you can also search by repository.

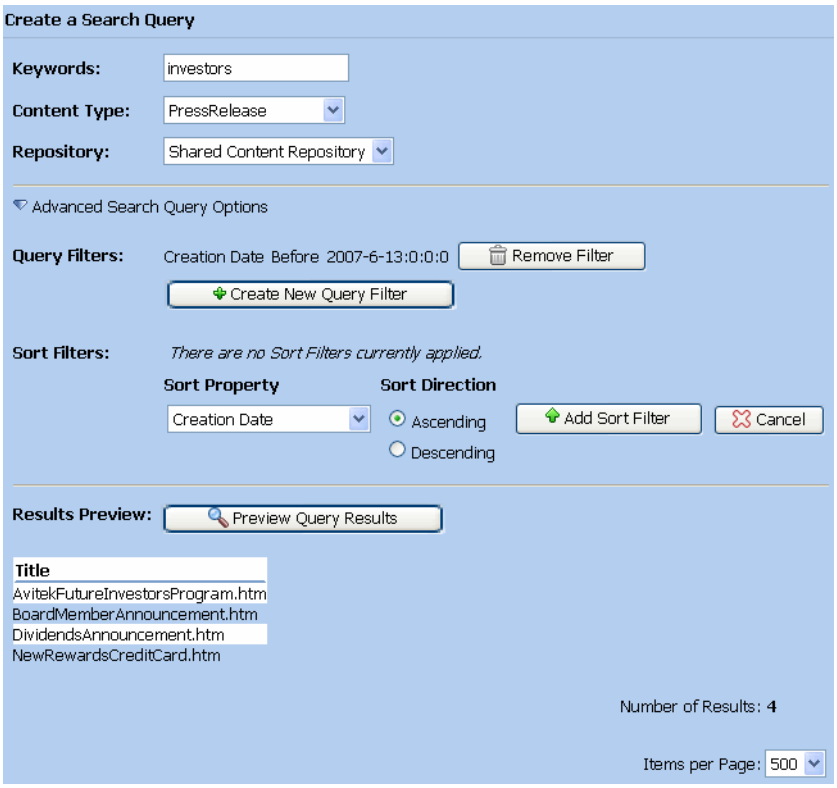
  You can click **Advanced Search Query Options** to create a query filter or a sort filter. Clicking **Create New Query Filter** applies a filter to your query, such as property name, an operator, and a value to narrow down your search results. For example, you can search for content created after 1/1/07 by selecting the **Creation Date** property, selecting **After**, and then entering a date. Click **Add Filter** and the new filter appears in the **Query Filters** section, as shown in Figure 14–7.

Tip: The *Similar* operator retrieves results that contain words that are similar to the keyword (for example, the word might be misspelled).

  You can click **Create New Sort Filter** to determine how to display the search results. For example, you can display the most recently-modified content by selecting the **Last Modified Date** property, selecting **Descending**, and clicking **Add Sort Filter**. The new sort filter appears in the **Sort Filters** section. See Figure 14–7 to see how you can display the most recently-modified content first (descending order). You can change the order in which the results are sorted by clicking the up or down arrow to

move the items in the sort filter up or down in the list (from ascending to descending, for example.)

Figure 14–7 Query Filter and Sort Filter



Click **Preview Query Results** to view the results of the search query. The search query searches both property values and binary content. Adjust the **Items per page** field to determine how many search results to display in the wizard (it does not affect the final display). Each time someone visits the Content Presenter portlet, the search is re-run. Since content in the repository can change, the results might be different the next time the portlet renders. Click **Next**.

Tip: Running a search query is more resource intensive than retrieving content from a specific node or retrieving the contents of a specific folder. When possible, try to retrieve specific content, rather than running a search.

- **Contents of a Folder** – Navigate through the repository tree to locate a content folder, click **Select this Item**, and click **Next**. Each time a user views this portlet, the most current content items in the selected folder are retrieved. Any content can behave like a folder; therefore, you can select any child items under that content.
- **Results of a Content Selector** – Select a Content Selector that you created in Oracle Enterprise Pack for Eclipse, and click **Select this Item**. Content Selectors use rules to target specific groups of people with content items from the WLP Virtual Content Repository. See the *Oracle Fusion Middleware Interaction Management Guide for Oracle WebLogic Portal* for more information. Each time the portlet renders, the most current Content

Selectors are retrieved. Click **Next** after you locate the Content Selector you want to display.

11. In the wizard's **Select Template & View** window, select an item from the **Template Category** field, and then select a template. A template category helps you organize your templates and template views. You can have as many template categories as you need, but you should plan your organization strategy early, so that you do not have to update the preference values of existing Content Presenter portlet instances. A template is similar to a folder and is used to organize content. The default template that appears is based on the single or multiple content choice you made in Step 8 in this section.

    WebLogic Portal ships with the following two default templates:

    – **Default Single Item Template** – Lets you view a single item and a single property. See Figure 14–8. This template appears because you chose **Single Content Item** in Step 8.

    – **Default Multiple Items Template** – Lets you view multiple items and properties in a bulleted list. This template appears because you chose **Multiple Content Items** in Step 8.

    You should create your own custom template based on the content you want to display. The custom templates you create appear in the wizard. See the *Oracle Fusion Middleware Content Management Guide for Oracle WebLogic Portal* for instructions on creating custom templates and views.

    Select an item in the **Template Views** field. A template view controls the layout and formatting of the content in the portlet. For example, you might create a template for your company's Press Releases. You could then create several custom template views that display the content in the following ways: a summary of the Press Release, the full text of the press release, details of the Press Release, and so on.

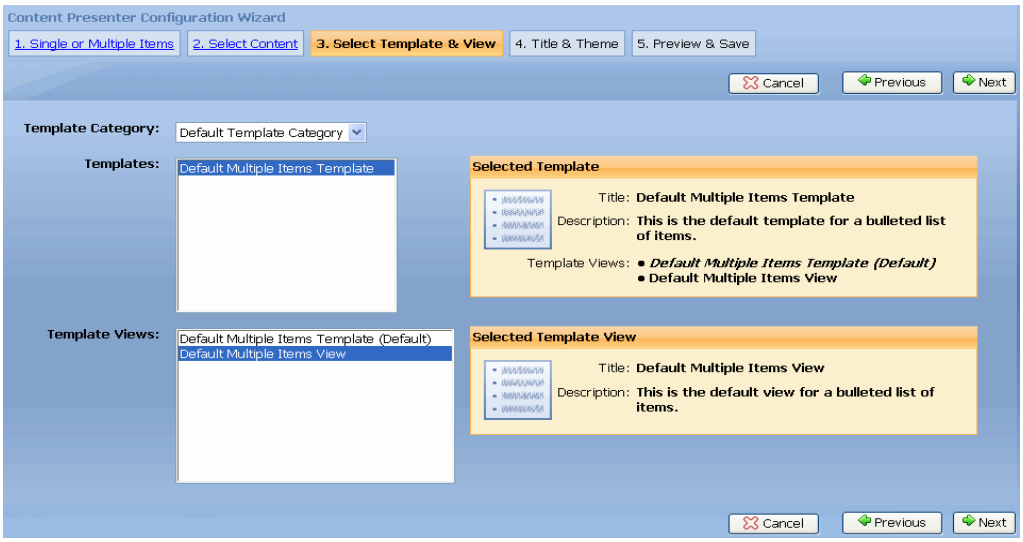
    If you chose the Default Single Item Template in Step 10 in this section (which displays a single piece of content), or a custom template that you created that does not contain views, select **Default Single Item View** for the template view. If you chose the Default Multiple Items Template in Step 10 in this section, select **Default Multiple Items View** for the template view. If you created a custom template view as described in the *Oracle Fusion Middleware Content Management Guide for Oracle WebLogic Portal*, that template view also appears here.

    For templates or views that support pagination of multiple content items, set the **Items per Page** value to the maximum number of items you want to display at one time on the template. With the appropriate pagination JSP tags on the template or view, users can navigate large numbers of items while viewing only a manageable few.

    Note: You can customize how templates and views paginate in the Content Presenter Configuration Wizard. For more information on the pagination tags, see the CM JSP Tag Javadoc in *Oracle Fusion Middleware JSP Tag Java API Reference for Oracle WebLogic Portal*

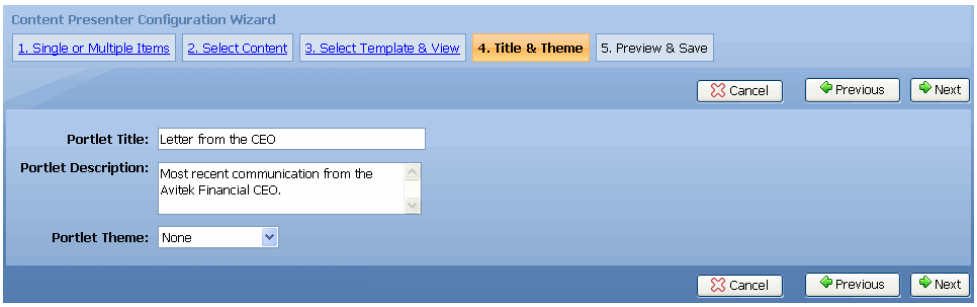
    Click **Next** after you select a template and view. See Figure 14–8.

Figure 14–8 Select a Template and View



12. In the wizard's **Title & Theme** window, enter a title (which appears as the portlet's title in the titlebar and the Administration Console) and description for the portlet. Select a theme from the **Portlet Theme** field and click **Next**. See Figure 14–9. The borderless theme presents your content with no background or border, so the content looks as if it is inline. If you choose the borderless theme, the Edit icon (discussed in Step 12) appears only when you mouse over the top of the portlet.

Figure 14–9 Enter a Description for the Portlet and Pick a Theme



13. In the portlet's **Preview & Save** window, you can click **View Content** or **Edit Content**. Click **Save** to save your changes to this page and to shared portlets in the library. Click **Preview Changes in Portlet** to view the changes in the portlet. Only you can see this version of this portlet. When previewing the portlet, you can click **Edit** to make changes to it. (If someone is authorized to make changes to your portlet and does so while you are previewing the portlet, you can no longer preview your version of the portlet. If this occurs, click **Continue** to see the newer version of the portlet.)

    Tip: Clicking **Preview Changes in Portlet** puts the portlet into a "preview state". The portlet will remain in its preview state until you click **Cancel** or **Save** in the wizard. The preview state is maintained even if you log out or close your browser window. If you make changes in other steps of the wizard, you must click **Preview Changes in Portlet** to update the preview with your changes, or click **Save** to make the changes public.

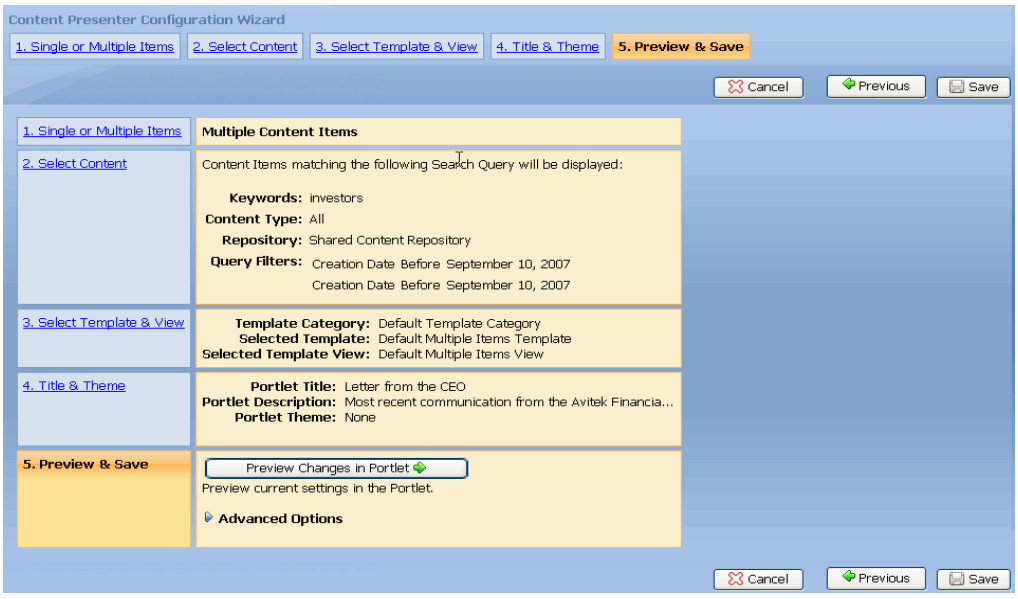
You can also click **Advanced Options** and select **Save to Current Page** (to save your changes to the current page only) or select **Save to Shared Portlet in Library**, to save changes to this shared portlet in the library. Saving changes at the library level globally affects everywhere the shared portlet was placed. You see this option enabled only if you clicked **Create New Shared Portlet**. When you click **Create New Shared Portlet**, you use these settings to create a new portlet in the library. The new portlet is available to all entitled users of this web application and can be placed on any page. The current page is updated to use this new portlet, rather than the current portlet. The new portlet also appears in the Administration Console in the `\Library\Portlets\` directory. Inheritance rules apply to shared portlets in the library. See the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal* for more information on inheritance. Figure 14–10 shows the Finish window.

Note: Content Presenter portlets that are configured on pages in the library will remain on the page in the library, even if the page has been customized on the desktop. The only way to make Content Presenter configurations local to a desktop is to add a configured or unconfigured Content Presenter portlet directly to the desktop, or customize the portlet on the desktop by changing or adding localizations to the portlet on the desktop using the Administration Console.

When you finish previewing the portlet, click **Return to Wizard** or click the **Edit** icon to return to this step in the wizard and make any additional changes. If you want to save your changes, click **Save**.

Tip: To be able to use the Create New Shared Portlet button in the Advanced Options, you must have edit rights on the page and delete rights on the portlet, in addition to the rights needed to configure a Content Presenter portlet.

Figure 14–10 Preview or Save the Portlet



The new content displays in the Content Presenter portlet. Figure 14–11 shows an example of content that is an advertisement for a college savings account.

Figure 14–11 A Single Content Item Displayed in the Content Presenter Portlet

You can use a custom template or template view to change the look of the Content Presenter portlet. See "Content Presenter Content Display Template Views" in the *Oracle Fusion Middleware Content Management Guide for Oracle WebLogic Portal* for instructions.

> **Tip:** The Content Presenter portlet uses error logging to catch exceptions and displays an error message in the portlet if you have rights to configure the portlet. For example, you might receive an error message if content your portlet is referencing was deleted, or a template or view the portlet is using was removed. The error message instructs you to reconfigure the Content Presenter portlet to fix the errors. You must have edit and delete capabilities in order to configure the portlet.

This section also contains the following topics:

- Section 14.2.1.1, "Changing How Much Content Appears in the Portlet"
- Section 14.2.1.2, "Using Portlet Publishing to Expose a Content Presenter Portlet"

14.2.1.1 Changing How Much Content Appears in the Portlet

You can change the amount of content that a portlet displays. Use the Content Presenter Example to edit the Avitek - In the News portlet to display three specific press releases, rather than a list of all press releases.

Perform the following steps to change how much content appears in a portlet:

1. Follow the instructions in Section 14.1.1, "Starting the Content Presenter Example" to log into the Content Presenter Example.
2. In the Avitek - In the News portlet in the Content Presenter Example, click **Edit** in the portlet's title bar.
3. In the Content Presenter Configuration Wizard, click **2. Select Content**.
4. Select **Custom List of Specific Content Items** to display content from a list that you create.
5. Navigate through the repository tree to locate content in the Shared Content Repository folder and select the **HR** > **Press Releases** folder.

6. Select the AvitekFutureInvestorsProgram.htm press release and click **Add Item to List**. Repeat these steps for two additional press releases: DividendsAnnouncement.htm and NewRewardsCreditCard.htm. This step will control what displays in the portlet– rather than all press releases appearing, only these three will appear.
7. If you want user to see the DividendsAnnouncement press release first in the list, select the check box next to it and click **Move Up**.
8. Click **Next** to save your changes.
9. In the Select Template & View window, click **Next**.
10. In the Portlet Properties window, click **Next.**
11. In the Finish window, click **Preview Changes in Portlet**.
12. The three press releases now appear. Click **Return to Wizard**.
13. Click **Save**.

14.2.1.2 Using Portlet Publishing to Expose a Content Presenter Portlet

You must be entitled to use Portlet Publishing to expose a Content Presenter portlet. If you use Portlet Publishing to expose your Content Presenter portlet, some features are not available.

The following list describes features that are not part of a Content Presenter portlet exposed with Portlet Publishing:

- Advanced options are not available when you expose a Content Presenter portlet through Portlet Publishing. You can save the portlet's configuration only at its current location (for example, a desktop or a page in a library).
- There is no Title Bar, so you cannot click **Edit** in the portlet's title bar. You can edit the portlet configuration only by rolling your mouse over the top portion of the portlet, which enables the edit button on the top right of the portlet.
- Themes do not apply in portlets exposed through Portlet Publishing. Therefore, you cannot change the currently selected theme of the Content Presenter portlet.

> **Tip:** When you use Portlet Publishing, work with only one Content Presenter portlet on a page at a time. If you work on more than one Content Presenter portlet on a page, an error appears. Close one of the Content Presenter wizards to continue working.

For more information on Portlet Publishing, see the *Oracle Fusion Middleware Client-Side Developer's Guide for Oracle WebLogic Portal*.

15

Adding a Third-Party Portlet

This chapter discusses special-purpose portlets that are provided by WebLogic Portal and partner companies that you can easily incorporate into your portal.

15.1 Using the Collaboration Portlets

WebLogic Portal provides a set of portlets for adding collaborative features to your portal. For detailed information on how to use the collaboration portlets, see Chapter 16, "Using the Collaboration Portlets."

15.2 Autonomy Portlets

If you want to use Autonomy Corporation's search functionality, you can purchase a license from Autonomy Corporation.

For more information about Autonomy, see *Oracle Fusion Middleware Autonomy Search Integration Sample Guide for Oracle WebLogic Portal*.

15.3 Documentum Portlets

EMC Documentum has partnered with Oracle to offer *EMC Documentum Content Services for Oracle Weblogic Portal*. This product provides a packaged set of Documentum functionality exposed through the Oracle WebLogic Portal infrastructure, allowing users to access and interact with all types of enterprise content including web pages, documents, and rich media such as audio and video.

From a portlet development perspective, a key feature of this product is the inclusion of Documentum portlets—application components that expose standardized, enhanced content management user functions through the portal interface.

Documentum portlets expose four key applications:

- Content management portlets allow users to manage any type of content.
- Web Publisher portlets permit casual users to publish content to web sites and portals.
- eRoom portlets provide dashboard views into EMC Documentum eRooms and allow multiple project management.
- The Enterprise Content Integration (ECI) Services portlet enables continuous access to content in other repositories, databases, and Web sites.

See the Documentum web site for more information on Documentum portlets for WebLogic Portal

15.4 MobileAware Portlets

Oracle Communication and Mobility Server provides a standards-based, non-proprietary environment that extends Oracle WebLogic deployments to offer multichannel mobile services in significantly reduced time frames. Enterprises can broaden the effectiveness of business-critical systems for employees and customers, and mobile carriers can rapidly deploy new, data-centric services, without the need for re-training and re-tooling.

For more information about Oracle Communication and Mobility Server and how to use it with WebLogic Portal, see the product documentation on Oracle Technology Network (OTN) .

16

Using the Collaboration Portlets

WebLogic Portal provides a set of portlets for adding collaborative features to your portal. You can use these collaboration portlets in any WebLogic Portal desktop.

> **Note:** The Collaboration Portlets require that the Apache Beehive facets are installed in the web application. For information on installing Apache Beehive facets, see "Apache Beehive and Apache Struts Supported Configurations" in the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

This chapter includes these topics:

16.1 What Are Collaboration Portlets

WebLogic Portal provides the following collaboration portlets that you can use in any WebLogic Portal desktop.

> **Note:** *User portlets* are portlets that store data on a per-user basis. *Common area portlets* store data in a common location that can be viewed by all users. The Calendar, Address Book, and Tasks portlets are user portlets by default. In some cases, you might want to reconfigure them to be common area portlets. For example, you might want to configure a corporate events calendar where all users see the same data. See Section 16.3, "Configuring Collaboration Portlets for a Shared View" for details.

- **Calendar Portlet** – (User portlet) Lets you create and schedule appointments.
- **Mail Portlet** – (User portlet) Allows you send and receive personal e-mail. This portlet supports IMAP and POP.
- **Address Book Portlet** – (User portlet) Lets you view and manage names, addresses, phone numbers, e-mail addresses, and other information in a personal address book.
- **Tasks Portlet** – (User portlet) Allows you to create and track Community items or personal items on a To Do list.
- **Discussion Portlet** – (Common area portlet) Lets you post and monitor topics of interest.

> **Tip:** The collaboration portlets are also available for use in custom communities. For detailed information on creating communities, see the WebLogic Portal *Oracle Fusion Middleware Communities Guide for Oracle WebLogic Portal*.

16.2 Adding Collaboration Portlets to Your Portal

This section explains how to add collaboration portlets to your portal and configure them properly. The basic steps are:

16.2.1 Step 1: Add Collaboration Facets

You must add the appropriate facets to both the portal EAR projects and the portal Web projects in which the collaboration portlets will be used.

1. Add the relevant collaboration portlet facets to your portal EAR project.
   a. In the Navigator view, right-click your portal EAR project and choose **Properties**.
   b. In the Properties view, select **Project Facets**, and click **Add/Remove Project Facets**.
   c. In the Add/Remove window, expand **WebLogic Portal Collaboration** and select both **Collaboration Portlets Application Libraries** and **Collaboration API**.
   d. Click **Finish**, then **OK**.
2. Add the Collaboration Portlets facet to your portal web project.
   a. Perform the same sub-steps above, selecting the **WebLogic Portal Collaboration** > **Collaboration Portlets** facet.

After you add the collaboration portlet facets, collaboration portlets themselves must be configured properly, as explained in the following steps. After configuration, they are available to add to a portal desktop.

16.2.2 Step 2: Add Collaboration Repository to Your Domain

If you have not done so, you need to create or extend a domain to includes the Collaboration Repository components.

1. If you have an existing server and it is running, stop the server.
2. Start the Configuration Wizard. From the Windows Start menu, choose **Oracle Products** > **WebLogic Server** > **Tools** > **Configuration Wizard**.
3. In the Configuration Wizard, select **Create** for a new domain or **Extend** for an existing domain, and click **Next**.
4. If you selected **Create** in Step 3, select **WebLogic Portal Collaboration Repository c**heck box and click **Next**. If you are extending an existing domain, select the domain root directory, and click **Next**.
5. Complete the remaining wizard windows.
6. Restart the server.

16.2.3 Step 3: Create a Role for Collaboration Portlet Users

Users of the collaboration portlets must be entitled to use the repository in which collaboration data is stored. This section explains how to create an new user role.

1. Start the WebLogic Portal Administration Console and log in.
2. Create a new enterprise application-scoped visitor entitlement. To do this, select **Users, Groups, & Roles** > **Visitor Entitlement** > **Browse Roles**.
3. Set the role scope. In the **Browse Roles from** panel, click **Update** to bring up the Update Role Scope dialog. In the dialog, select **Enterprise Application Scope**, and click **Update**.
4. Select **Visitor Roles** > **Browse Roles** > **Create New Role**. Enter a name for the new role and save it.

16.2.4 Step 4. (Optional) Configure a Repository

Data generated by collaboration portlets is stored in a content repository. By default, collaboration portlets are configured to store data in the repository subfolder /Communities_Repository/Collaboration.

If you wish, you can use any WLP content repository for storing collaboration portlet data. Note that library services must be *disabled* for the repository. Collaboration portlet data is not supported for third party repositories, such as Documentum repositories. See the *Oracle Fusion Middleware Content Management Guide for Oracle WebLogic Portal* for detailed information on content repositories. It is a good practice to create a subfolder in the repository in which to store the data, as explained in this section.

> **Tip:** A general best practice is to create a custom repository for collaboration data. See "Configuring Additional WLP Repositories" in the *Oracle Fusion Middleware Content Management Guide for Oracle WebLogic Portal* for details.

To create a subfolder in which to store collaboration portlet data, do the following:

1. Select **Content** > **Content Management**. In the Repository View, select your repository.
2. Click **Add Folder** and add a new folder to the repository of your choice.

In a later step, you will configure individual collaboration portlets to point to the repository folder of your choice, which is an option if the default location is not desirable.

16.2.5 Step 5. Entitle the Collaboration Data Repository

You must properly entitle the repository folder in which collaboration portlet data will be stored. Only entitled users can use the collaboration portlets.

1. Select the subfolder you created or targeted to store collaboration data.
2. Select the **Entitlements** tab for the subfolder.
3. Click **Add Role** and add the new user role you created for the collaboration portlets. Entitle the role with the capabilities Create, View, Update, and Delete.

16.2.6 Step 6. Add Users to the New Role

You must add any users who will use the collaboration portlets to the new role. Select the role and click **Add Users to Role**. Use the dialog to add users to the role.

> **Tip:** Be sure to add any new users to the role if you want them to use the collaboration portlets. To create new users, select **Users, Groups, & Roles** > **User Management**. After you create a new user, add it to the role.

16.2.7 Step 7. Configure the Collaboration Portlets

Configure the collaboration portlets so that they are aware of the repository that you configured. To do this, you edit certain portlet preferences.

1. Select **Portal** > **Portal Management**.
2. Expand the **Portal Resources** > **Library** > **Portlets** folder.
3. For each collaboration portlet that you wish to use, do the following:
   a. Select a portlet to configure. For example, click **Discussion** to configure the Discussion portlet.
   b. Click **Portlet Preferences**.
   c. Edit **collaboration.personal_repository.path** and set its value to the designated collaboration data folder in your repository. For example, if you created a folder named MyCollaborationData in the repository called MyCollaborationRepository, set the value to: `/MyCollaborationRepository/MyCollaborationData`.
   d. Edit **collaboration.personal_repository.name** and set its value to the name of the repository you are using for collaboration data. For example, if you are using a repository called MyCollaborationRepository, set the value to `MyCollaborationRepository`.

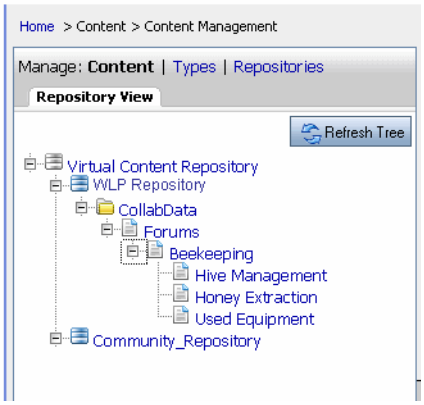
16.2.8 Step 8. Add Collaboration Portlets to Your Desktop

Now that you have configured your collaboration portlets, you can add them to a desktop.

> **Note:** Collaboration portlets only work if the user is authenticated. Your desktop must include a login portlet. For more information on authentication, see the *Oracle Fusion Middleware Security Guide for Oracle WebLogic Portal*.

If you configured everything properly, authorized users can access the collaboration portlets after logging in. Folders will be created in the collaboration repository as they are needed. For example, Figure 16–1 shows the repository structure for an example discussion forum on beekeeping.

Figure 16–1 Repository Structure for a Discussion Forum



16.3 Configuring Collaboration Portlets for a Shared View

This section explains how to reconfigure user portlets to be common area portlets. User portlets restrict the portlet's data to individual users, while common area portlets allow entitled users to share a the same view of the portlet's data.

16.3.1 Overview of User and Common Area Portlets

Collaboration portlets fall into two categories: *common area portlets* and *user portlets*: Typically, common area portlets are recommended for use cases where all users need to share the same view of the portlet's data. For example, you could create a calendar that displays corporate events to all users. In this case, you would need to configure the Calendar portlet (which is a user portlet by default) to be a common area portlet.

- Common area portlets write data to a shared repository that is accessible to all entitled users. By default, the Discussion portlet is a common area portlet.
- User portlets write data to a user-specific repository location that is accessible only to the currently logged in user. A user portlet's repository location is based on the root repository location specified by the portlet preference (see Section 16.2.7, "Step 7. Configure the Collaboration Portlets"). For example, if the Calendar portlet preference is set to be /MyRepository/Collaboration/calendar, and if the Calendar portlet is configured as a user portlet, it will write to the location

/MyRepository/Collaboration/calendar/<username>. The Calendar, Address Book, Tasks, and Mail portlets are user portlets by default.

16.3.2 Configuring a Common Area Portlet

This section explains how to reconfigure user portlets to be common area portlets. Note that by default, the Calendar, Address Book, Tasks, and Mail portlets are user portlets.

> **Tip:** Because mail is usually intended to be used by specific users rather than shared among many users, it is typically not necessary to reconfigure the Mail portlet to be a common area portlet.

1. Copy the portlets you want to configure to your local project. To do this:
   a. Open the Merged Projects View in Oracle Enterprise Pack for Eclipse. (To open the Merged Projects view, select **Window** > **Show View** > **Merged Projects**.)
   b. Right-click each portlet (located in the portlets/collaboration folder) and select **Copy to Project**. See Section 5.2, "Portlets in J2EE Shared Libraries" for information on the Copy to Project feature.
2. Rename each copied portlet. For example, change Tasks.portlet to Tasks-Team.portlet.
3. In a text editor, open each `.portlet` file and change the definition label and title. For example:

   Before:

   ```
   <netuix:portlet definitionLabel="task" title="My Task"
       lafDependenciesUri="/portlets/collaboration/collaboration.dependencies">
   ```

   After:

   ```
   <netuix:portlet definitionLabel="task_team" title="Team Task"
       lafDependenciesUri="/portlets/collaboration/collaboration.dependencies">
   ```

4. Also in the text editor, for each portlet, change the `<netuix:meta>` tag containing the AccountListenerImpl to use the CmAccountListener instead of the PersonalAccountListener. For example:

   Before:

   ```
   <netuix:meta name="collaboration.portlet.AccountListenerImpl"
       content="portlets.collaboration.common.c11n.PersonalAccountListener"/>
   ```

   After:

   ```
   <netuix:meta name="collaboration.portlet.AccountListenerImpl"
       content="portlets.collaboration.common.c11n.CmAccountListener"/>
   ```

5. Save each `.portlet` file.

16.4 Using the Collaboration Portlet Source Code

Source code for the collaboration portlets is available to WebLogic Portal developers, as explained in this section.

16.4.1 Copying the Source Code to Your Project

To use the source code, you must first copy it from a J2EE Shared Library to your workspace.

Source code for the collaboration portlets is located in the J2EE Shared Library `wlp-collab-portlets-web-lib`. To use this source code, you need to copy it from the shared library to your project workspace. See Section 5.2, "Portlets in J2EE Shared Libraries" for information on the Copy to Project feature.

Java source code for the collaboration portlets is copied to `WEB-INF/src/portlets`. Javadoc for the collaboration portlet code is copied to `WEB-INF/src/javadoc.zip`.

16.4.2 Source Code Disclaimers

If you modify any of the source code for the collaboration portlets, be aware of the following disclaimers:

- If you modify the source code and discover a bug, you must either reproduce the problem using the *original* collaboration portlet code or provide a simple code sample that illustrates that a bug exists in the WebLogic Portal API bug.
- If you change the original copy of the collaboration portlet source code and later apply a software patch to WebLogic Portal, be you must copy the updated source code from its library module to your workspace and reapply the changes you made to the original source code.

16.5 Using the Calendar Portlet

The Calendar portlet lets you create and schedule appointments. The Calendar portlet can store personal Calendar information and Community Calendar information.

You can customize the number of Calendar days that display. A Calendar appointment shows up in the Notifications Center when the appointment time is within the amount of time you specify.

16.5.1 Adding a Calendar Appointment

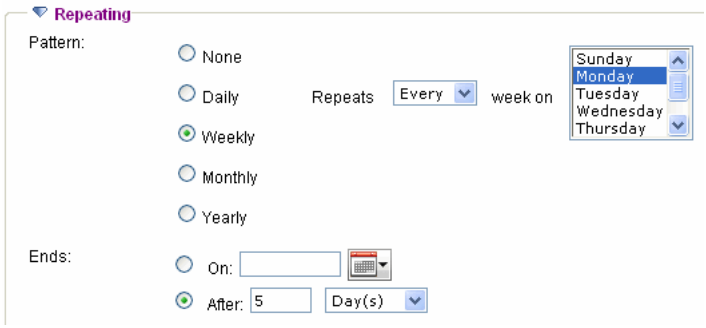
Perform the following steps to add an appointment to your calendar:

1. Select the **Schedule** tab and the **Calendar** tab.
2. Select the **Community** tab or the **Personal** tab to add an item to each of those Calendars.
3. Click **Add Appointment**. You can also click the appropriate time in the calendar and add the appointment.
4. Enter the following information about the appointment:
   - **Primary Information** – Click **Primary Information** and enter the subject, time, date, description, and location of your appointment. You can indicate whether the calendar shows your appointment time as busy or free. You can also indicate the importance of the appointment: `High`, `Medium`, or `Low`. These settings are reflected on the calendar.
   - In the **Description** field, enter text describing the appointment. If your System Administrator and Portal Administrator enabled rich text editing for this portlet, you can click **Rich text** to use the formatting toolbar to format your text, or click **Plain text** to enter text without formatting. If rich text editing is disabled for this portlet, the **Rich text** and **Plain text** links are not available.

Tip: For instructions on enabling rich text editing in a portlet, see Section 16.10, "Setting Up the Rich Text Editor."

- **Repeating** – Click **Repeating** and indicate the frequency of the appointment. For example, if you select `Weekly`, the appointment is scheduled automatically for the same day and time each week. By default, meetings do not repeat. You can also specify an end date for repeating meetings. Figure 16–2 shows a recurring meeting every Monday for five months.

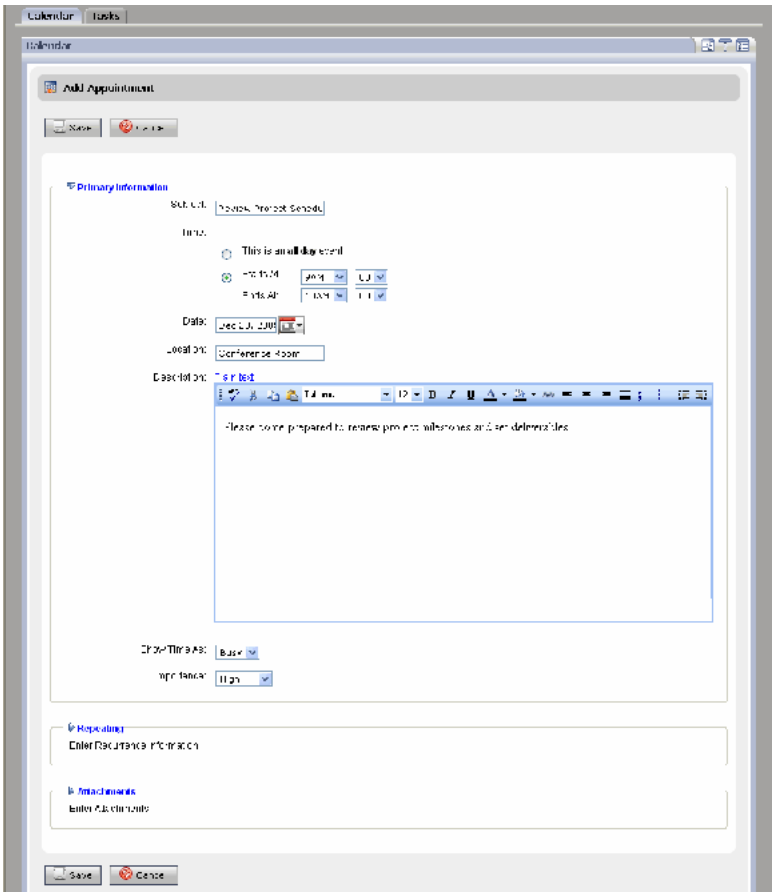
Figure 16–2 Repeating Calendar Appointment



- **Attachments** – Click **Attachments** and add a file to the appointment by clicking **Browse**, choosing a file on your file system, and clicking **Attach**.
- Click **Save**. The new appointment appears in the Calendar portlet.

See Figure 16–3 to view information for a new appointment.

Figure 16–3 New Calendar Appointment



16.5.2 Managing Your Calendar

You can also perform the following tasks:

- **Edit an Appointment** – You can reschedule or change an appointment's time, date, or location by selecting the appointment and clicking **Edit**.
- **Changing the View – Modify what you see in the portlet by clicking Day, Week, Month, or Upcoming.** All day appointments are shown at the top of the calendar page and indicated with a yellow bullet point to clearly distinguish them from timed appointments. Set the default view by clicking the Calendar portlet's **Edit** icon.
- **Editing Preferences** – Click the Calendar portlet's **Edit** icon to specify the following preferences:
  - **Time Zone** – Select the time zone in which you are working.
  - **Default View** – Select Day, Week, or Month to change the scope of your displayed calendar. Select Upcoming Events to display the upcoming events view by default.
  - **Number of days in Upcoming Events** – Select the number of days to display for the Upcoming Events view.
  - **Time Intervals** – Select the interval of time you want your calendar to display by selecting one of the pre-set intervals for Day, Week, or Month views.
  - **Week Starting Day** – Select the day your typical work week begins.

  - **Working Hours** – Select starting and ending hours for your typical work day.
- **Deleting an Appointment** – Select the appointment and click **Delete**.

16.6 Using the Mail Portlet

The Mail portlet allows users to send and receive personal e-mail to other members in your Community and to people outside the Community.

Unread messages are highlighted and display a closed envelope icon. Messages in the Inbox that you have already read are not in bold text and the icon is not highlighted.

16.6.1 Configuring the Mail Portlet

You must configure your mail account and add information about your incoming and outgoing mail server.

Perform the following steps to configure your mail account for incoming and outgoing mail:

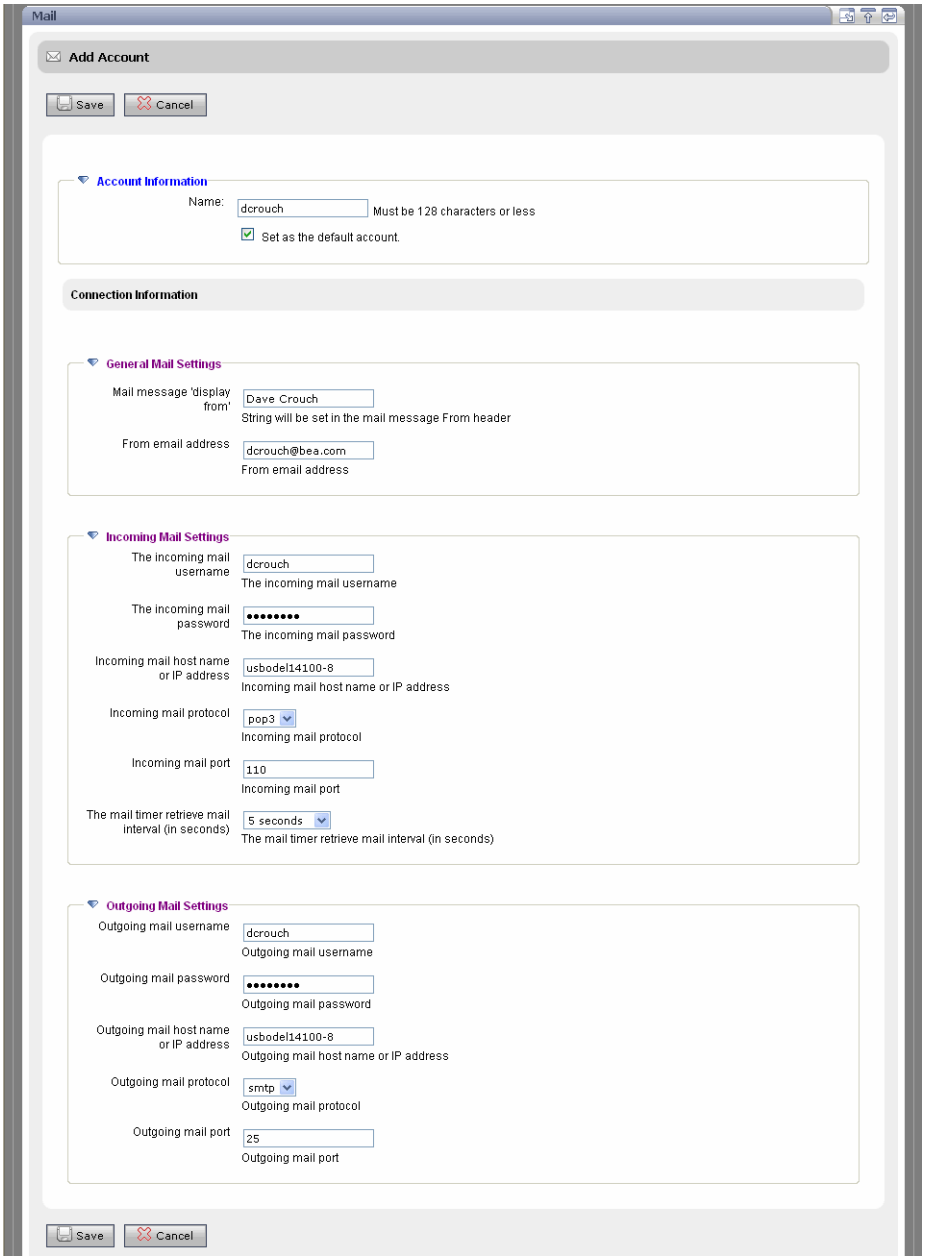
1. Select the **Communicate** tab and the **Mail** tab.
2. If the Add Account page does not display, click the Mail portlet's **Edit** button.
3. Enter information by clicking **Account Information**, **General Mail Settings**, **Incoming Mail Settings**, and **Outgoing Mail Settings**:
   - **Account Information** – Enter the name of your mail account. Do not use special characters. Select the **Set as the default account** to make this your default mail account.
   - **General Mail Settings** – In the **Mail message 'display from'** field, enter the name you want to appear in the From field in your e-mails. In the **From email address** field, enter your e-mail address.
   - **Incoming Mail Messages** – Complete all of the fields to set up your incoming mail server. Each field is required:
     - **The incoming mail username** – The user name to access your incoming mail server.
     - **The incoming mail password** – The password to access your incoming mail server.
     - **Incoming mail host name or IP address** – The server name or IP address of the incoming mail server.
     - **Incoming mail protocol** – The protocol the incoming mail server supports. You can select `POP3` or `IMAP`.
     - **Incoming mail port** – The port to connect to the incoming mail server. This defaults to the value for the selected protocol. Enter port `110` for `POP3` or `143` for `IMAP`.
     - **The mail timer retrieve mail interval** – The time out value in seconds for the incoming mail server to poll for new messages.
   - **Outgoing Mail Messages** – Complete all of the fields to set up your outgoing mail server. Each field is required:
     - **Outgoing mail username** – The user name to access your outgoing mail server.

- **Outgoing mail password** – The password to access your outgoing mail server.
- **Outgoing mail host name or IP address** – The outgoing host name or IP address.
- **Outgoing mail protocol** – The protocol the outgoing mail server supports. You can select `SMTP`.
- **Outgoing mail port** – The port to connect to the outgoing mail server. This defaults to the value for the selected protocol. For SMTP, enter `25`.

See Figure 16–4.

4. Click **Save**.

Figure 16–4 Configuring the Mail Servers



16.6.1.1 Removing a Mail Account

If you entered and saved incorrect information when configuring the Mail portlet, an error appears. You should delete this incorrect setup information by removing the mail account you created.

Perform the following steps to delete the account from the list of mail accounts:

1. In the Mail portlet, click **Edit** (as shown in Figure 16–5) to get to the Mail Preferences window.
2. Click **Mail Accounts** to expand that section.
3. Select the account you want to delete.
4. Click **Delete** to remove the highlighted account.

Figure 16–5 Click the Edit Icon to View the Mail Preferences Window

16.6.2 Sending E-Mail

Perform the following steps to send an e-mail:

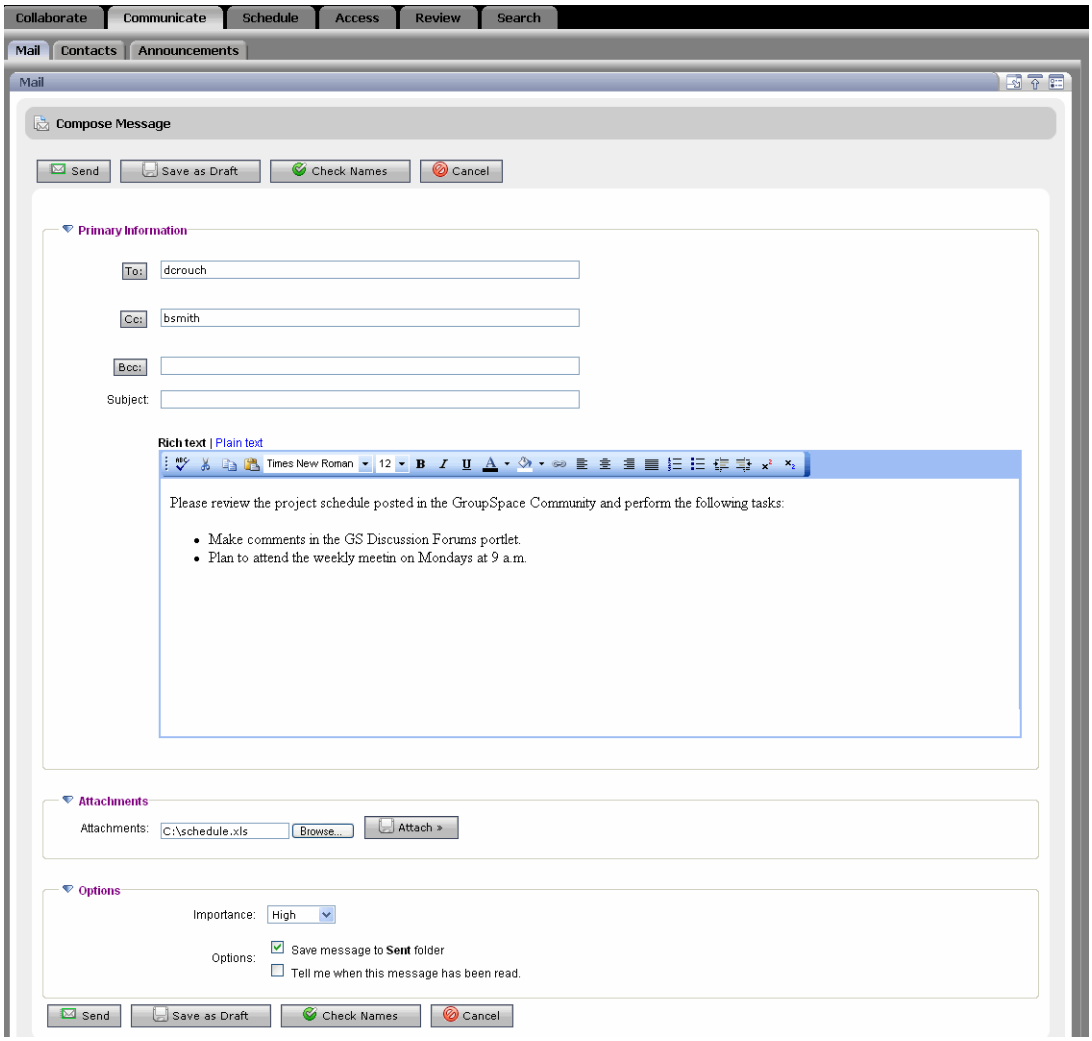
1. Select the **Communicate** tab and the **Mail** tab.
2. Click **New Message** to send a new e-mail.
3. Enter information by clicking **Primary Information**, **Attachments**, or **Options**:
4. **Primary Information** – Click **Primary Information** and enter the recipients, subject, and additional text. The **To** field is the only required field. You can also click **To** to pick a recipient from your list of personal contacts. You can click **Check Names** to view your contacts. In the text area, enter text for the body of the e-mail. If your System Administrator and Portal Administrator enabled rich text editing for this portlet, you can click **Rich text** to use the formatting toolbar to format your text, or click **Plain text** to enter text without formatting. If rich text editing is disabled for this portlet, the **Rich text** and **Plain text** links are not available.

   Tip: For instructions on enabling rich text editing in a portlet, see Section 16.10, "Setting Up the Rich Text Editor."

   - **Attachments** – Click **Attachments** to add a file to the e-mail. Click **Browse**, select the file, click **Open**, and click **Attach**. The file can be a document or a binary file.
   - **Options** – Click **Options** and select a priority for your e-mail. You can also save a copy of the e-mail to your Sent folder and be notified when the recipient reads the mail.

   See Figure 16–6.
5. Click **Send**. (Clicking **Save as Draft** saves the e-mail in your Drafts folder so you can send it later.) You cannot delete your Inbox, Sent Items, or Drafts folders.

Figure 16–6 New E-Mail Message



16.6.3 Viewing Mail

Perform the following steps to send and view your e-mail and attachments:

1. View an e-mail by clicking the item.
2. You can click **Reply** to send a reply to the e-mail, or **Forward** to send the e-mail to someone else.
3. If the e-mail has an attachment, click it and click the attachment name, and **Open** to view it or **Save** to download it.
4. Click **Done** to return to your other mail messages.

> **Tip:** Click **Check Messages** to refresh the page and display new e-mails.

16.6.4 Managing Mail

You can also perform the following tasks in the Mail portlet:

- **Change an E-Mail's Status** – Select the check box next to the e-mail, select **Mark as Unread** or **Mark as Read** from the drop-down list, and click the arrow next to the

Message Actions field. For example, if you selected **Mark as Unread**, the e-mail changes from bold text to normal text and the envelope icon is no longer green.

- **Filter Your E-Mails** – Click Unread or High Priority to view messages with that status. Click **All** to view all e-mails in your Inbox.
- **Create, Rename, Move, and Delete Folders** – Manage your mail folders by clicking **Manage Folders**. Create a new folder by selecting where you want the folder, clicking the **Create Subfolder** link, entering a name, and clicking **OK**. You can also click **Rename**, **Delete**, **Move**, or **Copy** to make changes to folders you created.
- **Change E-Mail Accounts** – Select a different account name from the **Accounts** drop-down list on the Mail portlet.
- **Delete an E-Mail** – Remove an e-mail by selecting the check box next to it and clicking **Delete**. The deleted e-mail is placed in a **Deleted** folder. When you delete items from that folder, the e-mail and any attachments are removed from the database.
- **Create an E-Mail Signature** – Enter text that you can attach to the bottom of outgoing messages by clicking the portlet's **Edit** icon and clicking **Mail Preferences**. Enter the information in the **Signature** field and select the **Add signature to outgoing messages** check box and click **Save**. Click the **Leave Edit** icon to return to return to the Mail portlet.
- **Change the Number of E-Mails Displayed** – Control how many e-mails display in the Mail portlet by clicking the portlet's **Edit** button and clicking **Mail Preferences**. Select `10`, `25`, or `50` from the **Messages Per Page** drop-down list and click **Save**. Click **Leave Edit** to return to the Mail portlet.

16.6.5 Searching Mail

You can search for individual Mail folders and messages for keywords that appear in the **Subject**, **Body**, **From**, or **Date** fields.

16.7 Using the Contacts Portlet

Use the Contacts portlet to view and manage names, addresses, phone numbers, e-mail addresses, and other information in a personal address book. The Contacts portlet can store Personal Contacts and Community Contacts.

The Contacts portlet works with the Mail portlet. If you receive an e-mail from someone, you can open the e-mail in the Mail portlet and click **Add to Contacts** to add the sender to your Contacts list.

16.7.1 Adding a Contact

Perform the following steps to add a new Contact:

1. Select the **Communicate** tab and the **Contacts** tab.
2. Select the **Community** tab to add a Community contact or the **Personal** tab to add a personal contact.
3. Click **Add Contact**. The Add Contact dialog appears.
4. Enter information about the contact by clicking each of the following:

- **Primary Information** – Enter the contact's name, including first, middle, and last name, as well as title and name suffix. You can also enter the contact's e-mail address.
- **Business Information** – Enter the contact's business details in the Business Information section. This section is optional.
- **Personal Information** – Enter the contact's personal information, such as home phone, personal cell phone, birthday, and so on. This section is optional.
- **Other Information** – Enter information about the contact. This section is optional.If your System Administrator and Portal Administrator enabled rich text editing for this portlet, you can click **Rich text** to use the formatting toolbar to format your text, or click **Plain text** to enter text without formatting. If rich text editing is disabled for this portlet, the **Rich text** and **Plain text** links are not available.

  Tip: For instructions on enabling rich text editing in a portlet, see Section 16.10, "Setting Up the Rich Text Editor."

See Figure 16–7 to view the fields for a new business contact.

5. Click **Save**. The contact appears in the Contacts portlet, as shown in Figure 16–8.

Figure 16–7 Adding a New Business Contact

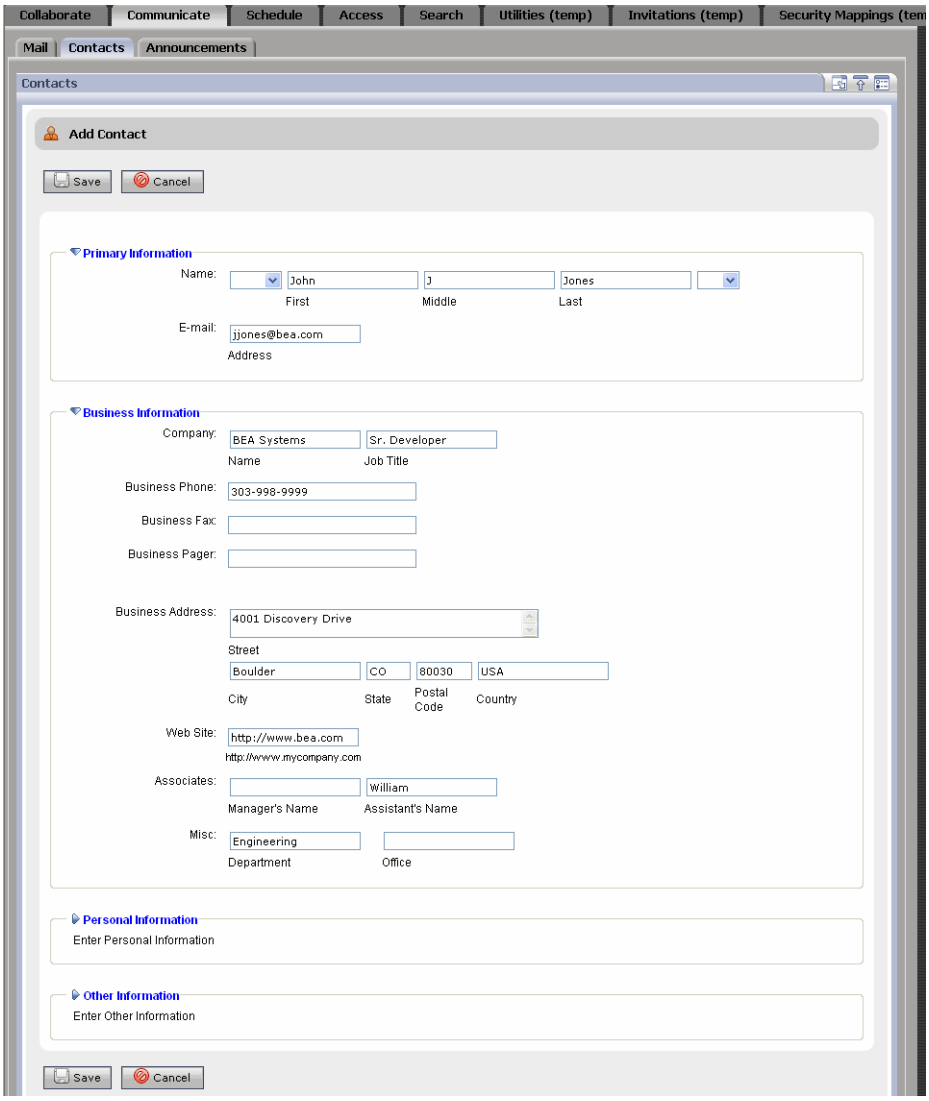
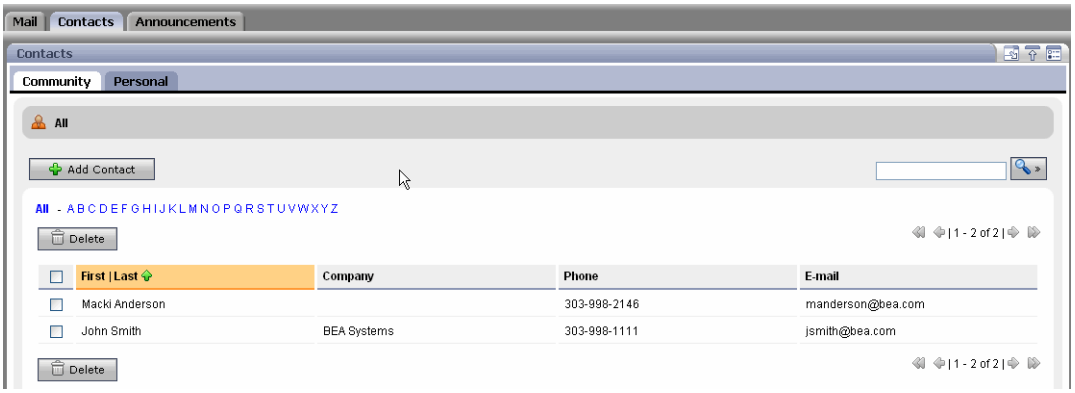


Figure 16–8 New Contact



16.7.2 Filtering and Navigating Contacts

The Contacts portlet provides several ways to filter and navigate through your list of contacts. If the list of contacts is long, the application breaks it into pages. Navigate these pages using the arrow buttons.

To change the default number of contacts per page, click the portlet's **Edit** icon. Click **Addressbook Preferences** and use the **Contacts per Page** drop-down list to change the default number of contacts that appear.

The Contacts portlet lets you filter the contacts that are displayed in the table in several ways, including the following:

- **View All** – Click **All** to list all contacts in the portlet, as shown in Figure 16–9.

Figure 16–9 Viewing All Contacts

- **Alphabetical** – Click a letter link to list the contacts whose last names start with that letter.
- **Sort Contacts** – You can sort the list of contacts by clicking the title of a column. For instance, to sort your contacts by last name in ascending order, click the heading of the Last column. The up arrow indicates that the column is sorted in ascending order. A down arrow indicates descending order.
- **Search for Contacts** – You can search for contacts by entering a search term and clicking **Search**. For more information on searching contacts, see Section 16.7.4, "Searching Contacts."

16.7.3 Managing Contacts

You can also perform the following tasks:

- **Editing a Contact** – Modify a contact's information by clicking the contact's name, clicking **Edit**, and modifying the information.
- **Deleting a Contact** – Remove one or more contacts by selecting the check box next to the contact and clicking **Delete**. To select all check boxes in a column, click the check box in the table's heading row.

16.7.4 Searching Contacts

You can use the following information to search for contacts:

- First name
- Middle name
- Last name
- Company name
- E-mail address

To perform a search, enter the search string in the **Search** field and click **Search**. The results appear in the portlet.

16.8 Using the Tasks Portlet

The Tasks portlet allows you to create and track Community items or personal items on a To Do list. You can view recent tasks and quickly add new tasks. The Tasks portlet can store personal information and Community information.

The Tasks portlet provides summary information about a list of tasks including priority, attachments, subject, status, and due date. Tasks can be sorted by each of these fields, except file attachments and priority.

16.8.1 Adding a Task

Perform the following steps to create a new Task:

1. Select the **Schedule** tab and the **Tasks** tab.
2. Select the **Community** tab to add a Community task or the **Personal** tab to add a personal task.
3. Click **Add Task**
4. Enter the following information:
   - **Primary Information** – Click **Primary Information** and enter the following information:
     - **Subject** – Enter text describing the Task.
     - **Due Date** – Click the calendar icon and select a due date and time.
     - **Start Date** – Click the calendar icon and select a start date and time.
     - **Status** – Click the drop-down list and choose `Not Started`, `Completed`, or `In Progress`.
     - **Importance** – Click the drop-down list and choose `High`, `Low`, or `Medium`. On the Tasks portlet, high priority tasks are denoted with a red exclamation point, while low priority tasks have a blue down arrow.
   - **Other Information** – Click **Other Information** to add categories for this Task (separated by a semicolon), adjust the **Percent Complete**, or add detailed **Comments**. If your System Administrator and Portal Administrator enabled rich text editing for this portlet, you can click **Rich text** to use the formatting toolbar to format your text, or click **Plain text** to enter text without formatting. If rich text editing is disabled for this portlet, the **Rich text** and **Plain text** links are not available.

     Tip: For instructions on enabling rich text editing in a portlet, see Section 16.10, "Setting Up the Rich Text Editor."

   - **Attachments** – Click **Attachments** to add a document to this Task. Click **Browse** to locate the document and click **Attach**. The Task is saved before you add the attachment.

   See Figure 16–10.
5. Click **Save**. This Task appears in the Tasks portlet with a red exclamation point to show it has a high priority, as shown in Figure 16–11.

Figure 16–10 New Task with an Attachment

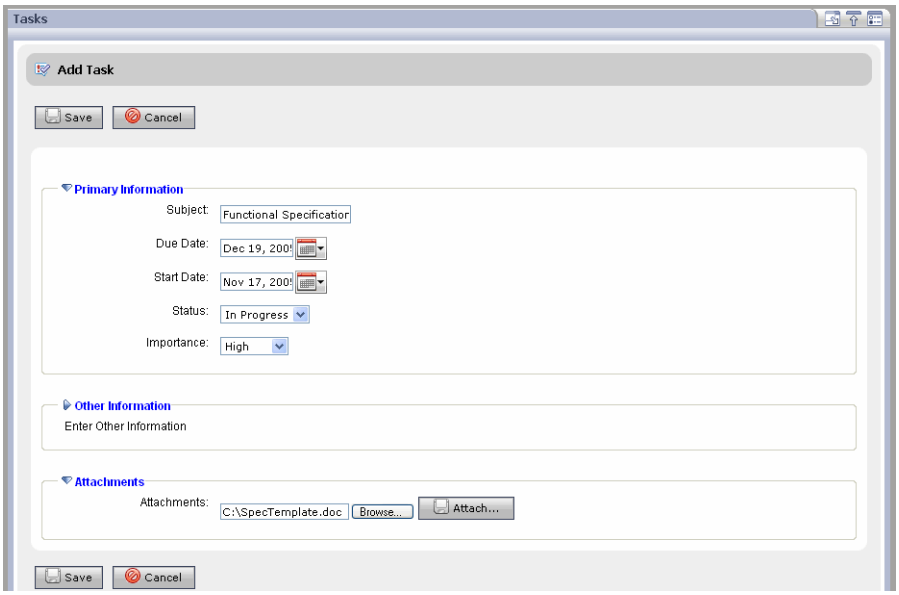
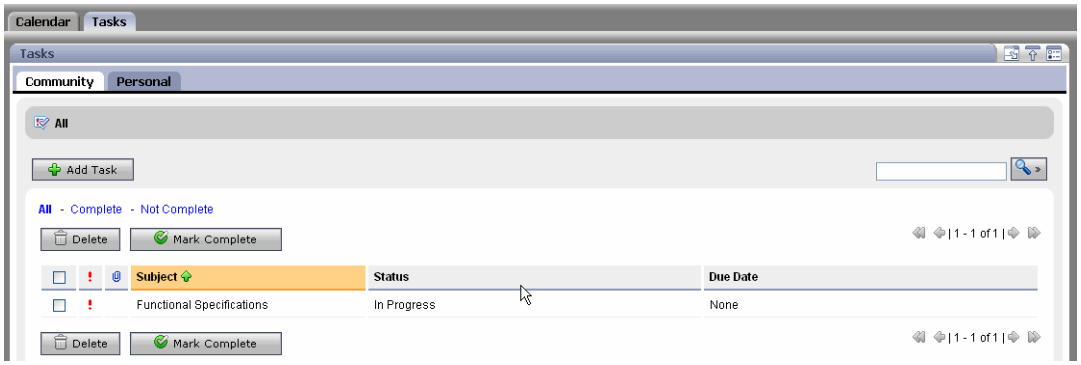


Figure 16–11 High Priority Task



You can view information differently by clicking the columns to reorder them.

16.8.2 Managing Tasks

- **View Detail About a Task** – See more information about the Task by clicking the Task name in the Subject column.
- **Edit a Task** – Modify the Task's details by clicking the Task name in the Subject column and clicking **Edit**.
- **Mark a Task as Completed on Your To Do List** – If the task is completed, you can cross it off your list. Select the check box next to the Task and click **Mark Complete**. The Task now has a line through it. The line remains even if you refresh the page.
- **Find Tasks** – Enter the subject of the Task in the Search bar and click **Search**.
- **Delete a Task** – Remove a Task from your To Do list by selecting the check box next to the Task and clicking **Delete**. Click **OK**.

16.9 Using the Discussion Forums Portlet

The Discussion Forums portlet lets users post and monitor topics of interest. A Community owner or creator creates categories (folders) and Discussion Forums, and other Community members can post topics and reply to the threaded discussions. Community owners and creators can create, edit, and delete categories, Discussion Forums, and topics. Community owners, creators, and contributors have create and edit capabilities, and viewers have read capability only.

A Discussion Forum exists within the Community, rather than outside the Community (such as an RSS channel).

You can organize your Discussion Forums by creating categories to organize Forums with similar topics.

16.9.1 Adding a Category and a Discussion Forum

Perform the following steps to create a category and a Discussion Forum:

1. Select the **Collaborate** tab and the **Discussion Forums** tab.
2. You can create a category to organize Discussion Forums with similar topics. Click **Add Category**.
3. Enter the following information for the new category:
   - **Name** – Enter text describing the category that will contain the Discussion Forums.
   - **Keywords** – Enter searchable words that are contained in this Discussion Forum. Use a comma or a space to separate keywords; the *or* search connector is assumed when the search is performed. For example, *car, truck* retrieves car or truck. This field allows you to quickly retrieve relevant Discussion Forums when you perform a property search.
   - **Description** – Enter text describing the category. If your System Administrator and Portal Administrator enabled rich text editing for this portlet, you can click **Rich text** to use the formatting toolbar to format your text, or click **Plain text** to enter text without formatting. If rich text editing is disabled for this portlet, the **Rich text** and **Plain text** links are not available.

     Tip: For instructions on enabling rich text editing in a portlet, see Section 16.10, "Setting Up the Rich Text Editor."

   See Figure 16–12.
4. Click **Add**. The category appears in the portlet.
5. Create a Discussion Forum by selecting the category and clicking **Add Forum**.
6. Enter the following information for the new Discussion Forum:
   - **Name** – Enter text describing the purpose of the Discussion Forum.
   - **Keywords** – Enter searchable words that are contained in this discussion. Use a comma or a space to separate keywords; the *or* search connector is assumed when the search is performed. For example, *car, truck* retrieves car or truck. This field allows you to quickly retrieve the Discussion Forum when you perform a property search.
   - **Author Masked** – Select this check box to not display the author.

- **Description** – Enter text describing the Discussion Forum topic. If your System Administrator and Portal Administrator enabled rich text editing for this portlet, you can click **Rich text** to use the formatting toolbar to format your text, or click **Plain text** to enter text without formatting. If rich text editing is disabled for this portlet, the **Rich text** and **Plain text** links are not available.

  Tip: For instructions on enabling rich text editing in a portlet, see Section 16.10, "Setting Up the Rich Text Editor."

7. Click **Add**. The new Discussion Forum appears in the portlet, as shown in Figure 16–13.

Figure 16–12 New Category

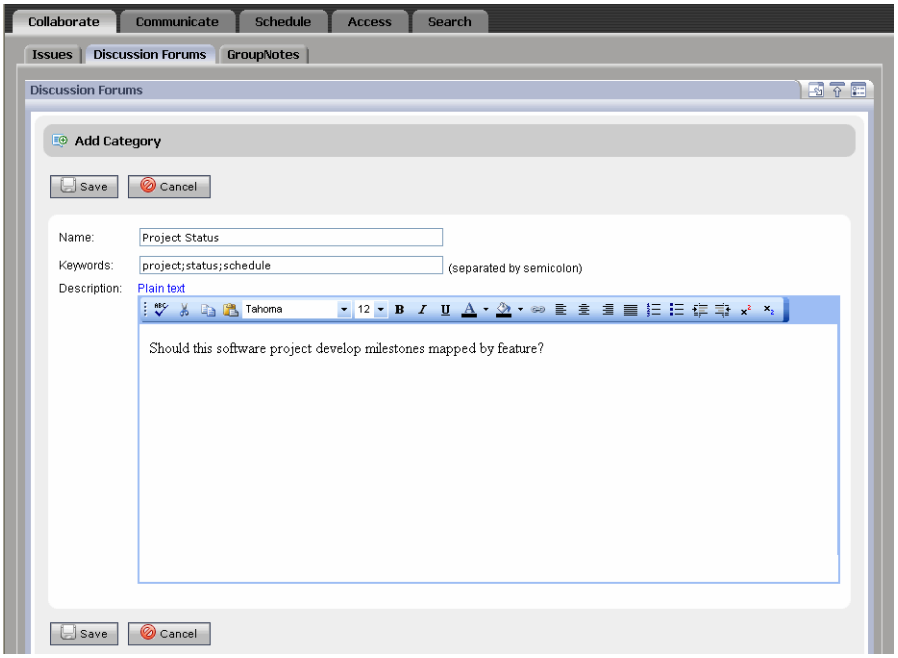
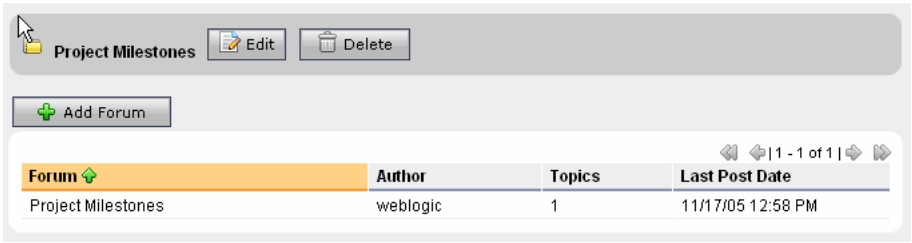


Figure 16–13 New Discussion Forum



16.9.2 Adding a Discussion Topic

After you create a Discussion Forum, Community members can add topics to it.

Perform the following steps to add a topic to a Discussion Forum:

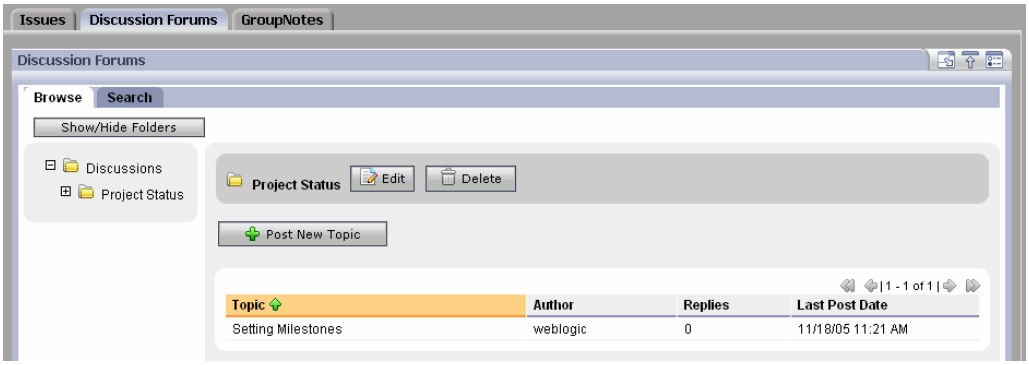
1. Select the **Collaborate** tab and the **Discussion Forums** tab.
2. Select the category and click the name of the Discussion Forum in the Forum column.

3. Click **Post New Topic**.
4. Enter the following information:
   - **Name** – Enter text describing this topic. The topic name must be unique, and the following characters are not permitted: \ and /.
   - **Keywords** – Enter searchable words that are contained in this topic. Use a comma or a space to separate keywords; the *or* search connector is assumed when the search is performed. For example, *car, truck* retrieves car or truck. This field allows you to quickly retrieve topics when you perform a property search.
   - **Author Masked** – Select this check box to not display the author.
   - **Description** – Enter text for your topic. If your System Administrator and Portal Administrator enabled rich text editing for this portlet, you can click **Rich text** to use the formatting toolbar to format your text, or click **Plain text** to enter text without formatting. If rich text editing is disabled for this portlet, the **Rich text** and **Plain text** links are not available.

     Tip: For instructions on enabling rich text editing in a portlet, see Section 16.10, "Setting Up the Rich Text Editor."

   - **Attachments** – Add a file by clicking **Attachments** and clicking **Browse**. Locate your file and click **Open**. Add the file by clicking **Attach**.
5. Click **Save**. The Discussion Topic appears in the Discussion Forums portlet. See Figure 16–14.

Figure 16–14 The New Discussion Topic Appears in the Discussion Forums Portlet



16.9.3 Replying to a Discussion Topic

After you create topics, Community members can post replies to topics.

Perform the following steps to add a topic to a Discussion Forum:

1. Select the **Collaborate** tab and the **Discussion Forums** tab.
2. In the **Browse** tab, select the category and the Discussion Forum.
3. Click the name of the Discussion Forum in the Forum column.
4. Click the topic's name in the Topic column and click **Reply**.
5. Enter your reply information and click **Save**.

16.9.4 Managing Discussion Forums

Owners, creators, and contributors can also perform the following tasks in the Discussion Forums portlet:

- **Edit a Discussion Forum** – Select the Discussion Forum's category, click the Forum's name in the Category column, and click **Edit**.
- **Edit a Category** – Click the category's name in the Category column and then click **Edit**.
- **Delete a Discussion Forum** – Click the Discussion Forum's name and click **Delete**. Deleting a Discussion Forum removes it and all its topics from the database.

In addition to adding a topic to a Discussion Forum, Community members can also perform the following tasks:

- **Edit Your Topic –** Select the category and select the Discussion Forum. Click the topic's name in the Topic column and click **Edit**.
- **Control What You See** – Select the Discussion Forum and the topic, and click **Discussion Overview** to view the topics and replies in the Discussion Forum. Select **Small**, **Medium**, or **Large** for the amount of space available to display topics and replies. You can also select the **Hide Previews** check box.
- **View More Details** – Click **Discussion Details** to view the author of the topic, date and time the topic was posted, and the body of the topic.

16.10 Setting Up the Rich Text Editor

Some Collaboration portlets have a rich text editor to apply formatting to documents or other items. Portal Administrators can allow users to access a portlet's formatting toolbar and a bottom toolbar, which lets them view the item's HTML text or preview the item.

For example, enabling rich text editing in the Tasks portlet allows users to use the formatting toolbar to apply bold, Italic, indentations, colored fonts, and so on. Enabling the bottom toolbar in the portlet lets users see and use the **HTML**, **Edit**, and **Preview** buttons.

You can enable or disable the rich text editor settings for the following portlets:

- Calendar
- Tasks
- AddressBook (in the Contacts portlet)
- Mail
- Discussion Forums

16.10.1 Enabling Rich Text Editing

The System Administrator must edit the `web.xml` file to enable the rich text editor and the bottom toolbar, and then the Portal Administrator sets the portlet preferences.

Perform the following two steps to enable the rich text editor:

1. The System Administrator uses a text editor to edit the `web.xml` file in your `/WEB-INF` directory and add the required context parameters and values, as shown in Example 16–1. After you edit the file, restart the WebLogic server. Use the following guidelines to determine how to set the values in the `web.xml` file:

- If the rich text editor (`RichTextEditorEnabled`) value is set to `false` in the `web.xml` file, rich text editing is not enabled and the bottom toolbar is not enabled and does not appear in the portlet. You cannot override the settings in the Administration Console, as described in Step 2.
- If the bottom toolbar (`RichTextEditorHTMLToolBarEnabled`) value is set to `false` in the `web.xml` file, and the rich text editor value is set to `true`, rich text editing is enabled, but not the bottom toolbar. The bottom toolbar does not appear in the portlet. You cannot override the bottom toolbar settings in the Administration Console.
- If the bottom toolbar value is set to `true` but the rich text editor is set to `false`, the bottom toolbar is not enabled and rich text editing is not enabled. You cannot override the settings in the Administration Console.

2. After the context parameters are added to the `web.xml` file, the Portal Administrator uses the Administration Portal to modify portlet preference settings for each portlet. See Section 16.10.1.1, "Modifying Portlet Preferences for Rich Text Editing."

Example 16–1 Add Context Parameters and Values to the End of the web.xml File

```
<context-param>
     <param-name>com.bea.apps.groupspace.RichTextEditorEnabled</param-name>
    <param-value>true</param-value>
</context-param>

<context-param>
    <param-name>com.bea.apps.groupspace.RichTextEditorHTMLToolBarEnabled
         </param-name>
    <param-value>true</param-value>
</context-param>

<context-param>
    <param-name>collaboration.RichTextEditorEnabled</param-name>
    <param-value>true</param-value>
</context-param>
```

16.10.1.1 Modifying Portlet Preferences for Rich Text Editing

You can use portlet preferences to control the rich text editor and the bottom toolbar for individual portlets.

> **Caution:** If the `web.xml` file's rich text editor values are set to `false`, any edits you make to portlet preferences in the Administration Console are ignored. If the rich text editor value is `false`, but the bottom toolbar value is set to `true`, the toolbar does not appear in the portlet.

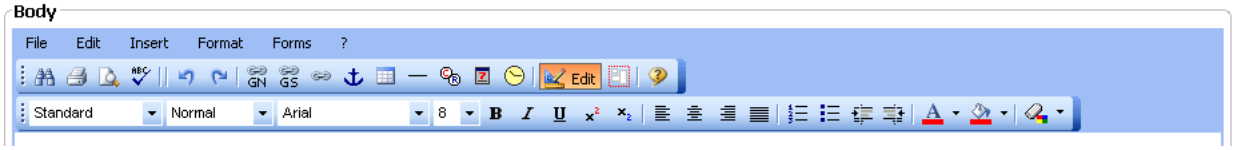
Perform the following steps in the Administration Console to change portlet preferences for rich text editing and the bottom toolbar:

1. In the Administration Console, choose **Portal** > **Portal Management**.
2. In the Portal Resources tree, expand the **Library** folder and the **Portlets** folder.
3. Click **Next** or **Prev** to locate the portlet you wish to modify and select it so that it is highlighted.

4. After you select the portlet, click **Portlet Preferences**.
5. Locate the preference you want to change.
6. Verify that the preference's setting is `true`, to allow the portlet to display the rich text editor. If this preference is set to any value other than `true`, the portlet displays a simple text area for entering information. After you locate the portal preference you want to change, click **Edit**.

   Select the **Is Modifiable** check box for the `RichTextEditorEnabled` preference to allow users to modify this preference. Do not select the **Is Multi-valued** check box. Click **Save**. See Figure 16–15
7. You can also change the preference that controls the display of the bottom toolbar. For example, if you change the `RichTextEditorHTMLToolBarEnbled` preference to `true`, the portlet displays the bottom toolbar when the rich text editor is available. The toolbar controls the **Edit**, **HTML,** and the **Preview** buttons. Clicking **HTML** allows a user to enter HTML directly into the text area. See Figure 16–16.
8. After making your changes to the portlet preferences, click **Save** to see the change appear in your portlet in your Community desktop. If you made preference changes at the library level, click **Propagate to Instances** to force proliferation of this preference to every instance of this portlet. WebLogic Portal overwrites all desktop instance's preferences with the library preferences. For more information on propagating to instances, see the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal*.

Figure 16–15 The Rich Text Editor in a Portlet



Tip: By default, rich text editing is not enabled for any of the portlets.

Figure 16–16 The Bottom Toolbar in a Portlet that Allows Rich Text Editing

Part III

Staging

Oracle recommends that you deploy your portal, including portlets, to a staging environment, where it can be assembled and tested before going live. In the staging environment, you use the WebLogic Portal Administration Console to assemble and configure desktops. You also test your portal in a staging environment before propagating it to a live production system.

For a detailed description of the architecture phase of the portal life cycle, see the *Oracle Fusion Middleware Overview for Oracle WebLogic Portal*.

Part III contains these chapters:

17

Assembling Portlets into Desktops

You perform the tasks described in this chapter to prepare the individual portlets that are part of your portal application for public consumption. After you add portlets to desktops, you can configure and test the application as a whole, and then deploy it to the production environment when it is ready for public access.

Before you perform the tasks described in this chapter, use the *Oracle Fusion Middleware Portal Development Guide for Oracle WebLogic Portal* to create the framework into which you will add the portlets— this includes the portal and its menus, layouts, the Look & Feel components for the overall portal, and the framework of the actual desktop. Also, you must have already created the set of portlets in the portlet *library*, from which you will choose the portlets to add to the desktop.

The primary tools used in this chapter are the WebLogic Portal Administration Console, the WebLogic Portal Propagation Utility (to move database and LDAP data between staging, development, and production), WebLogic Server application deployment tools, and any external content or security providers that you are using.

This chapter contains the following sections:

17.1 Portlet Library

The WebLogic Portal Administration Console organizes portal resources in a tree that consists of Library resources and desktop resources. Understanding the relationship between Library and desktop resources helps you to understand the effects and consequences of propagation.

The following text describes the relationships between the following instances of portal assets:

- **Primary instance** – Created in Oracle Enterprise Pack for Eclipse and stored in a `.portal` or `.portlet` file.
- **Library instance** – Created or updated in the Administration Portal, and displayed in the Portal Resources tree under the Library node.
- **Desktop instance** – Created or updated in the Administration Portal, and displayed in the Portal Resources tree under the Portals node.
- **Visitor instance** – Created or updated in the Visitor Tools.

For more details on portlets in libraries and in desktops, refer to the *Oracle Fusion Middleware Production Operations Guide for Oracle WebLogic Portal*.

17.2 Managing Portlets Using the Administration Console

This section contains instructions for performing portlet-related tasks using the WebLogic Portal Administration Console.

This section contains the following topics:

17.2.1 Copying a Portlet in the Library

You can use this feature of the WebLogic Portal Administration Console to duplicate an existing portlet and use it as a template for a "new" portlet.

Perform these steps:

1. Expand the Library node in the Portal Resources tree and navigate to the portlet that you want to copy.
2. Click **Copy Portlet**. The Copy Portlet dialog displays.
3. Enter a title and description for the copied portlet.
4. Click **OK**. The portlet is added at the bottom of the portlet list.

You can now customize the copied portlet by modifying its properties and preferences.

17.2.2 Modifying Library Portlet Properties

Portlet properties include all of the features and elements that make up the portlet. As a portal administrator, you can modify some of these properties from the Details tab. You can also edit the title, description, and locale information from the Title & Description tab, as described below.

To modify the properties of a portlet that resides in the library, perform these steps:

1. Expand the Library node in the Portal Resources tree and navigate to the portlet that you want to modify.
2. From the Details tab, select the type of property that you want to change. Use the table below for guidance.

Table 17–1 Modifying Library Portlet Properties

Property to Change	Procedure
Change title and description of the portlet in the current locale	1. Click **Title & Description**. 2. Click the locale (for example, **en**) in the **Locale** cell; the Add a Localized Title & Description dialog displays. 3. Enter a new Title and/or Description. 4. Click **Update**.
Add a localized title for the portlet	1. Click **Title & Description**. 2. Click **Add Localized Title**; the Add a Localized Title & Description dialog appears. 3. Enter a Language and Country identifier, Variant if applicable, Title, and a Description for the localized title. 4. Click **Create**.
Portlet Preferences	Refer to Section 17.2.8, "Creating a Portlet Preference" and Section 17.2.9, "Editing a Portlet Preference."
Portlet Theme	1. Click **Appearance**; the Edit Appearance dialog displays. 2. From the drop-down menu, select a Theme. 3. Click **Update**.
Render caching and timeout	1. Click **Advanced Properties**. 2. In the Render Caching Enabled drop-down menu, select True or False. 3. If you selected True, enter a cache expiration value in the Cache Expiration field. 4. Click **Update**.

17.2.3 Modifying Desktop Portlet Properties

Portlet properties include all of the features and elements that make up the portlet. As a portal administrator, you can modify some of these properties from the Details tab. You can also edit the title, description, and locale information from the Title & Description tab, as described below.

To modify the properties of a portlet that resides on a desktop, perform these steps:

1. Expand the Portals node in the Portal Resources tree and navigate to the portlet that you want to modify.
2. From the Details tab, select the type of property that you want to change. Use the table below as a guide.

Table 17–2 Modifying Desktop Portlet Properties

Property to Change	Procedure
Title and Description	You must edit these values within the Library resource tree. Expand the Library node, select the portlet that you want to edit, and follow the instructions in Section 17.2.2, "Modifying Library Portlet Properties."
Portlet Preferences	Refer to Section 17.2.8, "Creating a Portlet Preference" and Section 17.2.9, "Editing a Portlet Preference."
Portlet Theme	1. Click **Appearance**; the Edit Appearance dialog displays. 2. From the drop-down menu, select a Theme. 3. Click **Update**.

17.2.4 Deleting a Portlet

You can delete portlets from the Administration Console only if they were created there; for example, if you used the Copy Portlet feature to duplicate the portlet. Portlets created in Oracle Enterprise Pack for Eclipse cannot be deleted using the Administration Console.

Perform these steps:

1. Expand the Library node in the Portal Resources tree and navigate to the portlet that you want to delete.
2. Click **Delete Portlet**.

17.2.5 Managing Portlets on Pages

The contents of a page include portlets and books. You can view the portlets that are already on your page, and add and remove portlets to construct your page.

17.2.5.1 Adding Portlets to a Page

Library: To add a content to a page, perform these steps:

1. In the Portal Resource tree, expand the Library node and navigate to a page. The Details tab displays.
2. Click **Page Contents**. The Edit Contents tab displays.
3. Click **Add Contents**. The Add Books and Portlets to Placeholder dialog displays.
4. Display the pages that you want to choose from, using the Search area if needed.
5. Choose the portlets that you want to add by selecting the desired check boxes, and click **Add**.
6. When finished, click **Save**.

Desktop: To add a portlets to a page, perform these steps:

1. In the Portal Resource tree, expand the Portals node and navigate to a page. The Details tab displays.
2. Click **Page Contents**. The Edit Contents tab displays.
3. Click **Add Contents**; search for existing portlets if needed, then select the portlets that you want, and click **Add**. When finished, click **Save**.

17.2.5.2 Positioning Elements on a Page

The page layout is the grid structure of a page that holds placeholders for portlets and books on the page. You can select a layout for your portlets/books, and drag and drop them between the placeholders to customize the layout of each page.

Perform these steps:

1. In the Portal Resource tree, expand either the Library node or the Portals node as applicable, and select a page. The Details tab displays.
2. Click **Page Contents**. The Edit Contents tab displays.
3. If you want to change to a different layout, select a layout in the Layout drop-down menu.
4. Select the method that you want to use to position the elements on the page by selecting an option in the Position Elements area. The default is Drag & Drop.

5. Move portlets or books between placeholder columns.
6. If you want to prevent users from moving or deleting elements from a placeholder, select the Lock Placeholder check box.
7. When finished, click **Save Changes**.

17.2.6 Overview of Portlet Categories

Portlet categories provide for the classification of portlets, which is useful when organizing a large collection of portlets into meaningful groupings. The portlet categories are similar to other hierarchical structures in that parent "folders" can contain child folders and/or portlets. You must first create a portlet category, and then you can manage portlets by adding them to a category or moving them between categories.

17.2.6.1 Creating a Portlet Category

To create a portlet category:

1. In the Portal Resources tree, expand the Library folder and select **Portlet Categories**. The Browse Category tab displays.
2. Click **Create New Category**.
3. Type a title and description for the new category in the pop-up window.
4. Click **Create**.

17.2.6.2 Modifying Portlet Category Properties

Portlet category properties include all of the features and elements that make up the category. As a portal administrator, you can modify some of these properties from the Summary tab. You can also edit the title, description, and locale information from the Titles & Descriptions tab, as described below.

Perform these steps:

1. In the Portal Resources tree, expand the Library node and navigate to a portlet category.
2. From the Summary tab, select the type of property that you want to change. Use the table below as a guide.

Table 17–3 Modifying Portlet Category Properties

Property to Change	Procedure
Change title and description of the category in the current locale	1. Click **Title & Description**. 2. Click the locale (for example, **en**) in the Locale cell; the Add a Localized Title & Description dialog displays. 3. Enter a new Title and/or Description. 4. Click **Update**.

Table 17–3 (Cont.) Modifying Portlet Category Properties

Property to Change	Procedure
Add a localized title for the category	1. Click **Title & Description**. 2. Click **Add Localized Title**; the Add a Localized Title & Description dialog appears. 3. Enter a Language and Country identifier, Variant if applicable, Title, and a Description for the localized title. 4. Click **Create**.
Portlets in Category	Refer to Section 17.2.6.3, "Adding Portlets to a Portlet Category."
Categories in Category	1. Click **Categories In Category**; the Browse Category tab displays. 2. Click **Create New Category**; the Create New Category dialog displays. 3. Enter a Title and Description for the new category. 4. Click **Create**. The category is created and added to the currently selected category.

17.2.6.3 Adding Portlets to a Portlet Category

To add portlets into a category:

1. Expand the Library node in the Portal Resources tree and navigate to a portlet category. The Summary tab displays.
2. Click **Portlets In Category**.
3. Click **Add Portlets**.
4. In the Available Portlets area, select the portlets that you want to add, and click **Add** to include them in the Selected Portlets area.
5. Click **Save**.

17.2.7 Overview of Portlet Preferences

A portlet preference is a property in a portlet that can be customized by either an administrator or a user. Your portlet might already have preferences, but if you have the appropriate Delegated Administration rights you can create additional portlet preferences.

17.2.8 Creating a Portlet Preference

To create a portlet preference, perform these steps:

1. Expand the Portals node or the Library node in the Portal Resources tree, as appropriate, and navigate to the portlet for which you want to create a preference. The Details tab displays.
2. Click **Add Portlet Preference**.
3. Fill in the information in the fields. Use the table below as a guide.

Table 17–4 Creating a Portlet Preference

For this field:	Enter this information:
Name	The name you want to give this preference.
Description	A description of this preference.

Table 17–4 (Cont.) Creating a Portlet Preference

For this field:	Enter this information:
Value(s)	A value for a preference.
Is Modifiable? (checkbox)	Select this check box if you want to allow end users to modify this preference.
Is Multi-Valued? (checkbox)	Select this check box if you want to enter multiple values for the preference. If you select this box, an additional data entry field displays for you to enter additional values. Click Add Another Value after entering each value, until you are finished.

4. Click **Save**.

   For library instances of portlets, when you add a preference it automatically proliferates to library page instances and desktop page instances if the instances have not been decoupled.

5. If you want to force proliferation of this preference to every instance of this portlet, click **Propagate to Instances**; WebLogic Portal overwrites all desktop instance's preferences with the library preferences are. When complete, a message appears at the top of the Administration Console.

Here are some tips related to portlet preferences that you might find useful:

- When desktop instances of a portlet have no preferences, they automatically inherit the preferences from the library instance of the portlet.
- When desktop instances of a portlet have their own preferences set, they will not automatically inherit preferences from the library instance.
- If a desktop instance of a portlet has its own preferences set and these preferences are removed, it will automatically inherit all preferences from the library instance.
- If a desktop instance of a portlet has inherited preferences from the library instance and the desktop instance of this preference has been modified, it will no longer automatically inherit new preferences from the library or updates made to the library portlet's instance of this preference.
- If a desktop instance of a portlet has inherited the preferences from the library instance and no desktop instance specific preferences have been set, and the inherited preferences have not been modified in the desktop instance, the desktop instance will inherit all updates to the library preferences.

17.2.9 Editing a Portlet Preference

If you have the appropriate Delegated Administration rights, you can edit a portlet's preferences to change the way a portlet behaves.

To edit a portlet preference:

1. Expand the Portals node or the Library node in the Portal Resources tree, as appropriate, and navigate to the portlet for which you want to edit a preference. The Details tab displays.
2. Click **Portlet Preferences**.
3. Select the portlet preference by clicking its name in the Name column.
4. Edit the information in the fields. Use the table below as a guide.

Table 17–5 Editing a Portlet Preference

For this field:	Enter this information:
Name	The name you want to give this preference.
Description	A description of this preference.
Value(s)	A value for a preference.
Is Modifiable? (checkbox)	Select this check box if you want to allow end users to modify this preference.
Is Multi-Valued? (checkbox)	Select this check box if you want to enter multiple values for the preference. If you select this box, an additional data entry field displays for you to enter additional values. Click Add Another Value after entering each value, until you are finished.

5. Click **Save**.

   For library instances of portlets, when you edit a preference it automatically proliferates to library page instances and desktop page instances if the instances have not been decoupled.

6. If you want to force proliferation of this change to every instance of this portlet, click **Propagate to Instances**. When complete, a message appears at the top of the Administration Console.

17.2.10 Overview of Delegated Administration

In your organization, you typically want individuals to have different access privileges to various administration tasks and resources. For example, a system administrator might have access to every feature in the WebLogic Portal Administration Console. The system administrator might then create a portal administrator role that can manage instances of portal resources in specific desktop views of your portal, and a library administrator role that can manage your portal resource library. Other delegated administration roles only have access to resources if that access has been explicitly granted.

For more information about using delegated administration as a part of your security strategy, see the *Oracle Fusion Middleware Security Guide for Oracle WebLogic Portal*.

17.2.11 Overview of Visitor Entitlements

Visitor entitlements allow you to define who can access the resources in a portal application and what they can do with those resources. This access is based on the role assigned to a portal visitor, allowing for flexible management of the resources.

For more information about using visitor entitlements as a part of your security strategy, see the *Oracle Fusion Middleware Security Guide for Oracle WebLogic Portal*.

18

Deploying Portlets

This chapter discusses deploying portlets.

18.1 Deploying Portlets

Generally speaking, a WebLogic Portal application consists of an EAR file, an LDAP repository, and a database. The EAR file contains application code, such as JSPs and Java classes, and portal framework files that define portals, portlets, and datasync data. The embedded LDAP contains security-related data, such as entitlements, roles, users, and groups. The database contains representations of portal framework and datasync elements used by the portal runtime in streaming mode.

Portlet data can fall into the following two categories:

- **Portal Framework Data** – Refers to desktops, books, pages, and other portal framework elements that are created with the WebLogic Portal Administration Console.
- **EAR Data** – Refers to the final product of Oracle Enterprise Pack for Eclipse development—a J2EE EAR file. The EAR must be deployed to a destination server using the deployment feature of the WebLogic Server Administration Console.

When you deploy or redeploy a portal application EAR file to a server in production mode, .portlet files are automatically loaded into the database.

The primary tools you use to perform portlet deployment are the WebLogic Portal propagation tools and the deployment feature of the WebLogic Server Administration Console. For detailed instructions on deploying a portal and its portlets, refer to the *Oracle Fusion Middleware Production Operations Guide for Oracle WebLogic Portal*.

Part IV

Production

A production portal is live and available to end users. A portal in production can be modified by administrators using the WebLogic Portal Administration Console and by users using Visitor Tools. For instance, an administrator might add additional portlets to a portal or reorganize the contents of a portal.

For a detailed description of the architecture phase of the portal life cycle, see the *Oracle Fusion Middleware Overview for Oracle WebLogic Portal*.

Part IV contains the following chapter:

19

Managing Portlets in Production

During the life cycle of a WebLogic Portal application it moves back and forth between development, staging, and production environments. This chapter contains information about managing portlets that are on a production system.

This chapter contains the following sections:

19.1 Pushing Changes from the Library into Production

Proliferation is the process by which changes made to the Library instance of a portal asset are pushed into user-customized instances of that asset. For example, if a portal administrator deletes a portlet from a desktop, that change must be reflected into user-customized instances of that desktop.

The WebLogic Portal Administration Console includes a configuration setting for Proliferation under **Configuration Settings** > **Service Administration** > **Portal Resources**. The proliferation settings include **synch**, **asynch**, and **off**.

For more information on proliferation, refer to the *Oracle Fusion Middleware Production Operations Guide for Oracle WebLogic Portal*.

19.2 Transferring Changes from Production Back to Development

WebLogic Portal utilities such as the propagation tools and the Export/Import Utility allow you to reliably move and merge changes between environments. The Export/Import Utility allows a full round-trip development life cycle, where you can easily move portals from a production environment back to your Oracle Enterprise Pack for Eclipse development environment.

For instructions on using the propagation tools and Export/Import Utility, refer to the *Oracle Fusion Middleware Production Operations Guide for Oracle WebLogic Portal*.

A

Oracle Enterprise Pack for Eclipse Portlet Database Data

This appendix describes how portlet data is managed by databases, and contains the following sections:

A.1 Database Structure for Portlet Data

When a portlet's data is loaded into the database, the portlet XML is parsed and a number of tables are populated with information about the portlet, including `PF_PORTLET_DEFINITION`, `PF_MARKUP_DEFINITION`, `PF_PORTLET_INSTANCE`, `PF_PORTLET_PREFERENCE`, `L10N_RESOURCE`, and `L10N_INTERSECTION`.

`PF_PORTLET_DEFINITION` is the master record for the portlet and contains columns for properties that are defined for the portlet, such as the definition label, the forkable setting, edit URI, help URI, and so on. The definition label and web application name are the unique identifying records for the portlet. Portlet definitions refer to the rest of the actual XML for the portlet that is stored in `PF_MARKUP_DEF`.

In the Development phase, you use Oracle Enterprise Pack for Eclipse to create portlets and place them onto a portal. In the Staging phase, you use the Administration Console to add portlets to portal desktops. Each time you add a portlet to a desktop, you create an *instance* of that portlet. Portlet instances allow for multiple variations of the same portlet definition.

The following four types of portlet instances are recorded in the database for storing portlet properties:

- **Primary** – Properties defined in development and stored in the `.portlet` file.
- **Library** – Properties defined in the Portal Library, which may be changed using the WebLogic Administration Portal.
- **Admin** – A customized instance of the portlet in a desktop. This allows you to customize a portlet in a particular way for a desktop without affecting other instances of the portlet in other desktops.
- **User** – User-customized instances of the portlet defined in the Visitor Tools.

PF_PORTET_INSTANCE contains properties for the portlet for attributes such as DEFAULT_MINIMIZED, TITLE_BAR_ORIENTATION, and PORTLET_LABEL.

If a portlet has portlet preferences defined, those are stored in the PF_PORTLET_PREFERENCE table.

Finally, portlet titles can be internationalized. Those names are stored in the L10N_RESOURCE table which is linked using L10N_INTERSECTION to PF_PORTLET_DEFINITION.

A.1.1 Removing Portlets from Production

If a portlet is removed from a newly deployed portal application and it has already been defined in the production database, it is marked as IS_PORTLET_FILE_DELETED in the PF_PORTLET_DEFINITION table. It displays as grayed out in the WebLogic Administration Portal, and user requests for the portlet, if it is still contained in a desktop instance, return a message indicating that the portlet is unavailable.

A.2 Portlet Resources in the Database

During the development phase, the `.portlet` files for portal web projects are stored as XML in the portal web application. As a developer creates new `.portlet` files, a file polling thread monitors changes and loads the development database with the `.portlet` information. When a portlet's data is loaded into the database, the portlet XML is parsed and a number of tables are populated with information about the portlet. Changes that you make using the WebLogic Portal Administration Console are directly reflected in the database.

This section contains the following sections:

- Section A.2.1, "Types of Database Tables"
- Section A.2.2, "Management of Portlet Data"
- Section A.2.3, "How the Database Shows Removed Portlets"

A.2.1 Types of Database Tables

Separate database tables store information about portlet resources, including the following:

- **Definitions** – Portlet definition properties including creation date, content URI, whether the portlet is forkable or cacheable, whether it has a backing file, and so on.
- **Instances** (including a subset of tables for WSRP) – Instance properties indicate whether the portlet is minimized by default, title bar orientation (top, left, right, bottom), the parent portlet instance if applicable, and so on.WSRP portlet properties include proxy portlet instance values.
- **Categories** – Portlet categories provide for the classification of portlets, which is useful when organizing a large collection of portlets into meaningful groupings. The database stores values for the category ID and creation/modification dates.
- **Category definitions** – The database stores values for the category ID and creation/modification dates, parent category, and so on.
- **Preferences** – Preference properties, such as whether or not the preference can be multi-valued or whether it is modifiable, are stored in this table.
- **Preference values** – The database stores the actual value of portlet preferences.
- **User properties** – The database table maintains values of portlet user properties for WSRP user profile propagation.

Tip: The tool you use to manipulate these resources varies according to the resource, and the phase of development you are in; for example, you can change portlet preferences using either Oracle Enterprise Pack for Eclipse or the WebLogic Portal Administration Console, but you must use the Administration Console to create portlet categories.

A.2.2 Management of Portlet Data

When a portlet is loaded into the database, the portlet XML is parsed and a number of tables are populated with information about the portlet, including `PF_PORTLET_DEFINITION`, `PF_MARKUP_DEFINITION`, `PF_PORTLET_INSTANCE`, `PF_PORTLET_PREFERENCE`, `L10N_RESOURCE`, and `L10N_INTERSECTION`.

`PF_PORTLET_DEFINITION` is the master record for the portlet and contains rows for properties that are defined for the portlet, such as the definition label, the forkable setting, edit URI, help URI, and so on. The definition label and web application name are the unique identifying records for the portlet. Portlet definitions refer to the rest of the actual XML for the portlet that is stored in `PF_MARKUP_DEF`.

`PF_MARKUP_DEF` contains stored tokenized XML for the `.portlet` file. This means that the `.portlet` XML is parsed into the database and properties are replaced with tokens. For example, the following code fragment shows a tokenized portlet:

```
<netuix:portlet $(definitionLabel) $(title) $(renderCacheable) $(cacheExpires)>
```

These tokens are replaced by values from the master definition table in `PF_PORTLET_DEFINITION`, or by a customized instance of the portlet stored in `PF_PORTLET_INSTANCE`.

The following four types of portlet instances are recorded in the database for storing portlet properties:

- **Primary** – Properties defined in development and stored in the `.portlet` file.
- **Library** – Properties defined in the Portal Library, which you can change using the WebLogic Portal Administration Console.
- **Admin** – A customized instance of the portlet in a desktop. This allows you to customize a portlet in a particular way for a desktop without affecting other instances of the portlet in other desktops.
- **User** – User-customized instances of the portlet defined in the Visitor Tools.

PF_PORTET_INSTANCE contains properties for the portlet for attributes such as DEFAULT_MINIMIZED, TITLE_BAR_ORIENTATION, and PORTLET_LABEL.

If a portlet has portlet preferences defined, those are stored in the PF_PORTLET_PREFERENCE table.

Finally, portlet titles can be internationalized. Those names are stored in the L10N_RESOURCE table which is linked using L10N_INTERSECTION and PF_PORTLET_DEFINITION.

A.2.3 How the Database Shows Removed Portlets

If a portlet is removed from a deployed portal project, and it has already been defined in the production database, the portlet is marked as IS_PORTLET_FILE_DELETED in the PF_PORTLET_DEFINITION table. The portlet displays as grayed out in the Administration Console, and user requests for the portlet (if it is still contained in a desktop instance) return a message indicating that the portlet is unavailable.

For detailed information about the content of WebLogic Portal database tables, refer to the *Oracle Fusion Middleware Database Administration Guide for Oracle WebLogic Portal*.