BEA WebLogic Portal™

Using WSRP with WebLogic Portal

Version 8.1 Service Pack 5
October, 2005

Copyright

Restricted Rights Legend

Trademarks and Service Marks

Contents

About This Document

How this Document is Organized xiii

Product Documentation on the dev2dev Web Site xiv

Contact Us xiv

Documentation Conventions xv

Introduction to WSRP

The WSRP Standard 1-2

WSRP Portlet Type Support 1-2

Why Use WSRP? 1-2

WSRP Decouples the Deployment and Delivery of Applications 1-3

WSRP Delivers both Data and its Presentation Logic 1-3

BEA's Implementation of WSRP Requires Little or No Programming 1-3

Other Benefits of WSRP 1-4

Producers and Consumers 1-4

Producers 1-4

Simple Producers 1-5

Complex Producers 1-5

Complex/Simple Producer Features Matrix 1-5

Consumers 1-6

WSRP and WebLogic Portal 1-6

How WSRP Works 1-7

WSRP-compliant Portlet Lifecycle 1-9
Development Time 1-9
Deployment Time 1-9
Note on Localization of Remote Portlets. 1-10

Working with Remote Portlets

Building a Simple Remote Portlet 2-1
Modifying, Customizing, and Disabling a Remote Portlet 2-2
Setting Preferences on a Remote Portlet 2-3
Applying a Theme to a Remote Portlet 2-3
Other Look-and-Feel Topics. 2-4
Using Backing Files with Remote Portlets 2-4
Setting a Timeout Value on Remote Portlets 2-4
Setting the Default Timeout for Remote Portlets 2-5
Setting the Timeout for Individual Remote Portlets. 2-6

Establishing Interportlet Communications with Remote Portlets

The WebLogic Portal IPC Model 3-1
Event Handlers 3-2
Events 3-2
Event Actions 3-3
How IPC is Implemented. 3-3
Implementing IPC with WSRP: Example 3-5
Step 1: Set Up Your Environment 3-5
Create the Domain. 3-6
Create the Portal Application 3-6
Create the Web Applications (Web Projects) 3-7

Summary 3-8
Step 2: Create the Producer Portlets 3-8
Create the JSP Files and Portlets 3-9
Create the Backing File 3-13
Attach the Backing File 3-16
Add the Event Handler to bPortlet. 3-17
Test the Application 3-19
Summary 3-21
Step 3: Create the Consumer Portlets 3-21
Set Up the Exercise 3-21
Create the JSP Portlet 3-22
Create the Remote Portlet 3-23
Summary 3-26
Step 4: Test the Application 3-26
Build the Portal 3-26
Test the Portal 3-27
Special Considerations for Remote Portlets. 3-28
Understanding Backing Files 3-28
What are Backing Files? 3-28
Which Controls Support Backing Files? 3-29
How Backing Files are Executed 3-29
Thread Safety with Backing Files 3-30

Working with Producers

Using WSRP in a Basic WebLogic Server Domain 4-1
Getting Started 4-2
Configuring the WSRP Producer 4-3
Modify the CLASSPATH for the WebLogic Server Domain 4-3

Modify the Struts Application 4-5
Testing the Producer 4-9
Consuming the Producer Portlet 4-10
Using WSRP in a WebLogic Express Server Domain 4-11
Enabling Portlets on the Producer 4-11

Best Practices for Implementing WSRP

Portlet Programming Guidelines 5-1
Performance Tuning Recommendations 5-3
Avoid Moving Producers 5-4
Upgrading Simple Producers from Service Pack 3 5-5
Other Guidelines 5-5

Implementing Custom Data Transfer

Custom Data Transfer Interfaces 6-1
Implementing the Interfaces 6-2
Implementing Interfaces in a Complex Producer: Example 6-2
Step 1: Set Up the Environment 6-2
Step 2: Create the Producer JSP and Portlet 6-3
Step 3: Federate zipTest.portlet to the Consumer 6-8
Step 4: Create and Attach a Backing File to the Consumer 6-14
Step 5: Test the Application 6-18
Using this Example in a Simple Producer 6-19
Deploying Your Own Interface Implementations 6-28
Implementation Rules 6-28

Local Proxy Support

Why Use Local Proxy Mode? 7-1
Deployment Configuration 7-2

When to Use and Not Use 7-3

Monitoring and Logging Remote Portlet Performance

Monitoring Producer/Consumer Message Logs 8-1
Creating Custom Logs 8-5

Establishing WSRP Security

Access Control 9-1
Security Recommendations 9-2
Setting Security Constraints on Resources 9-2
Creating a Resource Connection Filter 9-3
Secure WSRP Messages 9-4
Manage User Identity 9-4
What is Single Sign-on? 9-4
How Single Sign-on Works with WSRP 9-5
The Signed Certificate 9-5
The Java keytool Utility 9-5
Secure the /producer Path 9-7
Establishing Single Sign-on with Remote Portlets: Example 9-7
Step 1. Set Up the Environment 9-7
Step 2. Create the Login Portlet and Establish SSO with a Remote Portlet 9-8
Create the Log-in Page Flow Portlet 9-8
Create a Log-in Portal 9-14
Create a Portlet on the Producer 9-16
Federate the Producer Portlet to the Consumer 9-20
Test the Log-in Portlet 9-23
Summary 9-25
Step 3: Break the Log-in Portal 9-25

Rename the .jks File 9-25
Retest the Portal 9-26
Step 4: Obtain and Implement a Signed Certificate 9-27
Before You Begin 9-27
Generate a New Keystore 9-27
Create the Certificate Signing Request and Import the Signed Certificate 9-28
Import the Signed Certificate 9-29
Step 5: Update the Consumer mBean 9-30
Step 6: Update the WSRP Identity Asserter 9-33
Step 7: Test the New Keystore 9-35
Securing the WebLogic Administrator's Logon Information 9-36
Encrypting Passwords 9-38
Note on Changing Passwords 9-39

About This Document

Using WSRP with WebLogic Portal explains how to implement Web Services for Remote Portlets (WSRP) with BEA WebLogic Portal 8.1 with service pack 4. It covers the following topics:

How this Document is Organized

This document is comprised of these chapters:

- Chapter 1, "Introduction to WSRP," provides a general overview to WSRP and its place within WebLogic Portal. You should read this chapter before proceeding with the rest of this guide.
- Chapter 2, "Working with Remote Portlets," provides basic instructions for working with WSRP in a WebLogic Portal environment and links to more detailed information.
- Chapter 3, "Establishing Interportlet Communications with Remote Portlets," describes how to establish communication between a federated, or remote, portlet and a local portlet on the same portal. It includes a detailed tutorial that will take you through the process of establishing interportlet communications between federated and local portlets.
- Chapter 4, "Working with Producers," describes how to apply some of the standard portlet functionality to a WSRP-compliant Producer.
- Chapter 5, "Best Practices for Implementing WSRP," provides some proven practices you should follow when implementing WSRP with WebLogic Portal.

- Chapter 6, "Implementing Custom Data Transfer," shows you how a Consumer application can provide data to a Producer application.
- Chapter 7, "Local Proxy Support," describes how to implement local proxy support, which allows co-located Producer and Consumer web applications to short-circuit network I/O and "SOAP over HTTP" overhead.
- Chapter 8, "Monitoring and Logging Remote Portlet Performance," shows you how to monitor activity between Producers and Consumers by using the message monitor servlet and create custom logs to display specific information about WSRP sessions.
- Chapter 9, "Establishing WSRP Security," shows you how to set up a single sign-on between a Consumer and a group of Producers.
- Appendix A, "WSRP Error Messages," lists some of the more common error message you might encounter when trying to implement WSRP with WebLogic Portal.

Product Documentation on the dev2dev Web Site

BEA product documentation, along with other information about BEA software, is available from the BEA dev2dev Web site:

`http://dev2dev.bea.com`

To view the documentation for a particular product, select that product from the list on the dev2dev page; the home page for the specified product is displayed. From the menu on the left side of the screen, select Documentation for the appropriate release. The home page for the complete documentation set for the product and release you have selected is displayed.

Contact Us

Your feedback on the BEA BEA WebLogic Portal 8.1 documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA WebLogic Portal 8.1 documentation.

In your e-mail message, please indicate that you are using the documentation for BEA BEA WebLogic Portal 8.1 ProductVersion.

If you have any questions about this version of BEA BEA WebLogic Portal 8.1, or if you have problems installing and running BEA BEA WebLogic Portal 8.1, contact BEA Customer Support at **http://support.bea.com**. You can also contact Customer Support by using the contact

information provided on the quick reference sheet titled "BEA Customer Support," which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.
`monospace text`	Indicates *user input*, as shown in the following examples: • Filenames: `config.xml` • Pathnames: *`BEAHOME`*`/config/examples` • Commands: `java -Dbea.home=BEA_HOME` • Code: `public TextMsg createTextMsg(`
	Indicates *computer output*, such as error messages, as shown in the following example: `Exception occurred during event dispatching:java.lang.ArrayIndexOutOfBoundsException: No such child: 0`
`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`

Convention	Item
monospace italic text	Identifies variables in code. *Example*: `String expr`
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[ ]	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `java utils.MulticastTest -n name [-p portnumber]`
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. *Example*: `java weblogic.deploy [list|deploy|update]`
. . .	Indicates one of the following in a command line: • That an argument can be repeated several times in a command line • That the statement omits additional optional arguments • That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name] [-f "file1.cpp file2.cpp file3.cpp . . ."`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER 1

Introduction to WSRP

Web Services for Remote Portlets (WSRP) is a web services standard that allows you to "plug-n-play" visual, user-facing web services with portals or other intermediary web applications. It allows you to create a repository of services that users can reference to surface applications in their portlets or to consume applications from WSRP-compliant Producers, even those far removed from your enterprise.

BEA WebLogic Portal 8.1 SP4 includes an implementation of WSRP that allows the framework to use WSRP portlets.

This section includes information on the following subjects:

- The WSRP Standard
- Why Use WSRP?
- Producers and Consumers
- WSRP and WebLogic Portal
- How WSRP Works
- WSRP-compliant Portlet Lifecycle
- Note on Localization of Remote Portlets

The WSRP Standard

BEA's implementation of WSRP is based upon the WSRP 1.0 standard created by OASIS. BEA Systems has been an active member of the OASIS technical group for WSRP 1.0 and continues to work as part of this standard effort for future enhancements to the specification.

You can read the current version on the WSRP standard at:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp

WSRP Portlet Type Support

You can create WSRP-enabled portlets for these portlet types:

- Pageflow
- JSP
- Struts
- Java portlets (JSR168; supported only for complex producers)

WSRP also supports backing files on complex producers

This version of WebLogic Portal supports only homogeneous portlets, therefore, the portlet modes must be compatible with the portlet; for example:

- Pageflows in all modes.
- Struts in all modes.
- Java in all modes.

Why Use WSRP?

WSRP is an attractive option for web development for three main reasons:

- It decouples the deployment and delivery of applications
- It delivers both data and that data's presentation logic.
- Its implementation requires little or no programming.

WSRP Decouples the Deployment and Delivery of Applications

You can surface new applications on your portal, independent of release schedule and where and when the code is physically deployed.

For example, perhaps you have a portal on machine X and another on machine Y. To get a portlet from machine X to machine Y, currently your only method of doing so is to copy the portlet's code, JSPs, and so on, from machine X to the destination machine (Y). By using WSRP, you can access and display that portlet on machine Y simply by referencing it through the Producer's Web Service Description Language identifier (WSDL).

WSRP Delivers both Data and its Presentation Logic

As a "user-facing" web service, WSRP portlets provide both application and presentation logic. This is different from standard web services, or data-oriented web services, which contain business logic but lack presentation logic and thus require that every client implement that logic on its own.

While the data-oriented approach works well in many implementations, it is not well suited for dynamically integrating business applications. For example, to integrate an order status web service into a commerce portal, you would need to write code to display the results of the status services into the portal. Using WSRP, with the presentation logic included in the web service, you can achieve the aggregation of applications and services dynamically. You no longer need to develop the presentation logic in order to do the integration; you can simply request the order status service to show up as a portlet inside the commerce portal at a predetermined location.

BEA's Implementation of WSRP Requires Little or No Programming

You don't have to do a lot of programming to make a portlet remote. In a non-WSRP compliant implementation, integrating remote content and application logic into an end-user presentation usually requires a significant custom programming effort. Typically, vendors of aggregating applications, such as a portal, write special adapters for applications and content providers to accommodate the variety of different interfaces and protocols those providers use.

WebLogic Workshop 8.1 SP4 provides tools that allow you to pick from a rich choice of compliant remote content and application providers, and integrate them with just a few mouse clicks—without writing a line of code. Additionally, applications created with WebLogic Workshop 8.1 SP4 are, by default, WSRP-compliant, which means they can be leveraged into other user's portlets with little or no additional programming required on your part.

Other Benefits of WSRP

In addition to those listed above, WSRP provides these additional benefits to developers:

- Interoperability
- Portability
- Options for deployment
- Support by large players in the industry

Producers and Consumers

WSRP introduces the concepts of Producers and Consumers. By using WSRP, you can aggregate application functionality by integrating WSRP-compliant Producers into WebLogic Portal as a Consumer. Your end users thus will be able to interface with Consumers to view the integrated applications.

Figure 1-1 Web Services Between Producer and Consumer

Producers

Producers host portlets and provide such services as self-description, mark up, registration, and portlet management. Producers can optionally manage the registration of Consumers and require them to pre-register prior to interacting with portlets. A registration establishes a relationship between Consumers and Producers.

Producers are further classified into either simple or complex Producers.

Simple Producers

A simple Producer is a non-portal web application that contains portlets . It does not depend upon any portal features (for example, customization), nor does it require registration, support URL rewriting in the Consumer, or support a management interface.

With simple Producers:

- You can WSRP-enable non-Portal projects, such as WebLogic Server projects.
- You can offer portlets without actually installing WebLogic Portal
- Portlets cannot use Portal APIs/features

"Using WSRP in a Basic WebLogic Server Domain" on page 4-1 describes how to configure a (non-portal) WebLogic Server environment as a WSRP producer so that you can expose portlets based on Struts or Java Page Flows. The exposed portlets can then be consumed as remote portlets running in a regular WebLogic Portal Domain.

Complex Producers

A complex Producer requires registration, does support URL rewriting in the Consumer, and does support a management interface. By default, all portal web projects created with WebLogic Workshop 8.1 SP4 are created in complex Producers.

With complex Producers:

- All Portal Projects are WSRP capable
- Portlets can use Portal APIs/features
- You can offer and consume portlets

Complex/Simple Producer Features Matrix

Table 1-1 lists the features available under complex Producers and simple Producers.

Table 1-1 Complex/Simple Producer Features Matrix

Feature	Complex Producer	Simple Producer
JSR168	X	
Page Flow	X	X

Table 1-1 Complex/Simple Producer Features Matrix

Feature	Complex Producer	Simple Producer
Registration	Required	Not Required
Support for URL Rewriting	X	
Support for Management Interface	X	
Support for JSPs	X	
Support for Backing Files	X (producer-side, only)	

Consumers

Consumers are web applications that aggregate information from Producers and surface it in other portals. Consumers route requests from users to the appropriate Producer, which, in turn processes the request and sends results back to the Consumer. The Consumer aggregates the results coming from various Producers and send the final result back to the user. Consumers provide separation of the traffic flowing between them and the Producers. They also ensure that all interactions are kept private to that specific user during the sessions.

Note: Consumers are web applications. A given consumer can have multiple desktops associated with it. Furthermore, there can be multiple consumers in an enterprise application.

In WebLogic Portal Administration Console, you will notice that producer registrations are scoped to individual consumer web applications. Because there can be multiple consumer web applications in an enterprise application, it is possible that a given producer will need to be registered multiple times within an enterprise application (that is, registered for each consumer web application in which it is used.)

WSRP and WebLogic Portal

In addition to complying with the OASIS WSRP standard, BEA's WSRP implementation adds some additional features to provide you with greater control over remote portlet usage. These features are described in Table 1-2

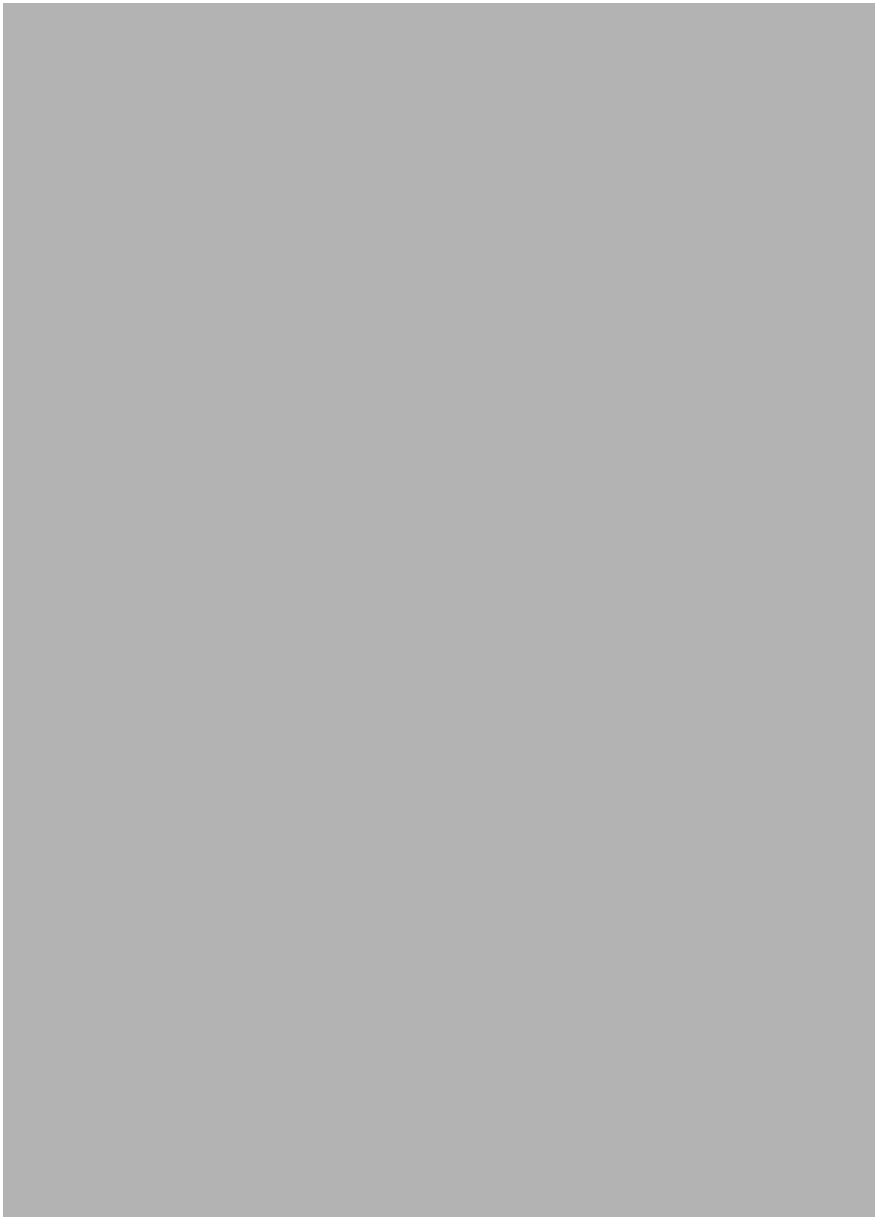
Table 1-2 Features in BEA's WSRP Implementation

Feature	Description
Portlet Wizard	Remote portlets can be easily implemented by using the Portlet Wizard that comes with WebLogic Workshop. You can create and install robust remote portlets with a few simple mouse-clicks and a Producer's WSDL.
Administration Portal	The WebLogic Portal Administration Portal allows you to easily deploy and manage Producer and Consumer portlets for your enterprise from a central location.
Producer-by-Default	Portal web applications created with WebLogic Workshop 8.1 SP4 and later are, by default, created in Producers. You don't need to do any special coding or add additional content on these projects to make them available as a Producer. **Note:** For non-portal web applications, you need install a WSRP Producer in WebLogic Workshop to make that application a Producer.
Registration	Consumers might be required to register with a Producer. Registration allows Producers to identify each Consumer with a unique, Consumer-provided handle. This helps identify what portlets are available to that Consumer.
Service Description	The service description shows what a Producer has to offer. It lets a Consumer discover a Producer and it lists the capabilities and properties that are available from the Producer. As a portlet repository, the service description also lists the portlets available from that Producer.
Markup and User Interaction	Request time operations to initiate or terminate a session. It gets markup for a portlet, which is returned in the body of the message. It submits user interaction request for a portlet.
Portlet Management	The Producer may allow cloning, customization, and deleting of portlets. Customization features allow portal administrators to manage portlet preferences for remote portlets.

How WSRP Works

Figure 1-2 illustrates the WSRP process, including handoffs from end user to Consumer to Producer and back.

Figure 1-2 WSRP Process Flow



WSRP-compliant Portlet Lifecycle

The portlet lifecycle for a WSRP-compliant portlet includes both development time and deployment time capabilities.

Development Time

Producer side (complex Producer) Developers will be able to leverage Java Page Flows, JSP, JSR 168, and Struts applications to expose their functionality in remote portlets. They can portletize the application and configure any related properties. Since all portal projects created with this version of WebLogic Workshop 8.1 are, by default, Producers, developers don't need to be aware of WSRP.

Consumer side Developers declare the Producers that are available to be used in the application. By using the Portlet Wizard in WebLogic Workshop, they can create a remote portlet based on the service description file from the Producer. They will need to:

1. Select "Remote" from a list of portlet types.
2. Configure a few options.
3. Create a new portlet.
4. Drag and drop the WSRP-based portlet to the portal.

See Building a Remote Portlet in the WebLogic Workshop online help system for a detailed description of this process.

Deployment Time

Producer side For applications not built with this version of WebLogic Portal, some changes are required:

- Customers using applications built with previous versions of WebLogic Portal 8.1 need to upgrade to WebLogic 8.1 SP4.
- Customers using applications built on an installation of WebLogic Server 8.1 SP4 need to follow the instructions for converting a non-portal web application into a Producer, as described in Using WSRP in a Basic WebLogic Server Domain.

New applications, after WSRP installation, are automatically configured.

Consumer side Deployment is similar to the current Administration Portal deployment, since remote portlets look like local portlets.

Note on Localization of Remote Portlets

In service pack 4 of WebLogic Portal 8.1, you cannot localize remote portlets. This feature will be available in a subsequent release of the product.

CHAPTER 2

Working with Remote Portlets

Within the context of a portal, WSRP-compliant portlets are designed to behave identically to local portlets. Nearly anything you can do or control on a local portlet, you can do or control on a remote portlet. Despite this similarity, WSRP does present some deviations from other WebLogic Portal implementations. This section describes how to apply some of the standard portlet functionality to a WSRP-compliant remote portlet. In some cases, it will simply provide links to the appropriate documentation elsewhere in the WebLogic Portal documentation set, including WebLogic Workshop's online help system.

This section includes information on the following subjects:

- Building a Simple Remote Portlet
- Modifying, Customizing, and Disabling a Remote Portlet
- Applying a Theme to a Remote Portlet
- Using Backing Files with Remote Portlets
- Setting a Timeout Value on Remote Portlets

The portlets discussed here are built on Consumers unless otherwise noted. For procedures specific to Producers, please refer to Working with Producers.

Building a Simple Remote Portlet

To see how easily you can build a remote Consumer portlet, please see Building a Remote Portlet in the WebLogic Workshop online help system, at:

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportlets/conWsrpConsumer.html

This exercise will show you how to use the Portlet Wizard to create a remote portlet and populate it with an application from a Producer. It will also show you how to add the portlet to a portal and view the portal and the new remote portlet in a browser.

Modifying, Customizing, and Disabling a Remote Portlet

In addition to Building a Remote Portlet, described above, the WebLogic Workshop online help system contains other vital information for developing and using WSRP-compliant portlets. These topics include:

Modifying a Remote Portlet, at:

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportlets/portletModProxy.html

This topic describes how to add states and modes to a remote portlet. Note that with remote portlets:

- The Float mode is not passed from the producer to the consumer proxy portlet.
- The Delete state is not passed from the producer to the consumer proxy portlet.
- Modes and States that are passed from the producer to the consumer cannot be overwritten. This includes minimize state, maximize state, edit mode, and help mode.

Customizing a Remote Portlet, at:

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportlets/portletConsumerCust.html

This topic tells you where to find complete information about and procedures for changing the appearance of a remote portlet.

Disabling A Producer, at:

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportlets/portletDisableProd.html

This topic describes how to modify the Producer configuration file so that Producer application cannot be consumed by a remote portlet.

Setting Preferences on a Remote Portlet

The producer stores and manages portlet preferences for remote portlets. When you view or modify the preferences in a remote portlet (on a consumer), the consumer must fetch the preferences from the producer, and modifications must be sent back to the producer where they are stored.

Note: Portlet preferences are included in the WebLogic Portal implementation of WSRP producers. Other WSRP producer implementations may not support portlet preferences.

You can view and modify the portlet preferences for a remote portlet using the WebLogic Administration Portal. The Administration Portal uses the Portlet Management interface of WSRP to retrieve preferences from the producer and modify them.

Note: It is not possible to create or modify portlet preferences in a remote portlet using WebLogic Workshop.

You cannot add a portlet preference to remote portlets consumed from a simple producer or producers that have portal management disabled in the `wsrp-producer-config.xml file`.

For more information on using portlet preferences with WSRP, please refer to Portlet Preferences at:

`http://dev2dev.bea.com/products/wlportal81/articles/portlet_preferences.jsp`

Applying a Theme to a Remote Portlet

A *theme* determines the appearance of a portlet on the portal desktop. The theme of a remote portlet is not linked to a Producer, giving you the option of modifying the portlet's appearance to best suit your needs; for example, to match the appearance of the portal in which the portlet resides.

Themes are a component of a portal or desktop's *look-and-feel* architecture and a subset of skins and skeletons. These concepts are documented in other documents in the edocs system and in the WebLogic Workshop online help system. Use the following links to locate this documentation:

- Creating Look & Feels

`http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportals/ifLF.html`

- Look & Feel Architecture
  - Portal User Interface Framework Guide, at

    `http://edocs.bea.com/wlp/docs81/lookandfeel/index.html`

- The Portal User Interface Framework
  - Portal User Interface Framework Guide, at

    http://edocs.bea.com/wlp/docs81/lookandfeel/index.html
- How Look & Feel Determines Rendering
  - Portal User Interface Framework Guide, at

    http://edocs.bea.com/wlp/docs81/lookandfeel/index.html

Other Look-and-Feel Topics

- Creating Skins and Skin Themes

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportals/ifLFSkinsThemes.html

- Creating Skeletons and Skeleton Themes

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportals/ifLFSkeletons.html

Using Backing Files with Remote Portlets

This version of WebLogic Portal's implementation of WSRP supports backing files. These files allow you to programatically add functionality to a portlet by implementing (or extending) a Java class, which enables preprocessing (for example, authentication) prior to rendering the portal controls. Backing files are attached to portlets by using the Property Editor in WebLogic Workshop.

For more information on backing files and how they are used with remote portlets, please refer to Understanding Backing Files at

http://edocs.bea.com/wlp/docs81/wsrp/ipc.html#1000978

Setting a Timeout Value on Remote Portlets

Occasionally, a producer is slow to respond to a request from a remote portlet. In this case, the portal application in which the reomote portlet is located remains unresponsive until the remote portlet's response is received.

To avoid these kinds of delays caused by remote portlets without affecting your portal application (and the other portlets it contains) you can set a timeout value. You can set a default timeout limit

for all remote portlets and a timeout limit for an individual remote portlet. The timeout set on an individual portlet takes precedance over the default.

Note: The remote portlet connection timeout only works when a consumer is continually connected to a producer. The timeout is effective only for cases where the producer is slow to respond to a consumer, not for cases where the producer is physically unavailable (the connection is broken), or where a new connection is made. In these cases, the operating system's TCP timeout takes effect.

Setting the Default Timeout for Remote Portlets

Set the default connection timeout for all remote portlets in a web application by setting the `<connection-timeout>` element in `WEB-INF/wsrp-producer-registry.xml`, as shown in Listing 2-1.

Listing 2-1 Setting the Connection Timeout

```
<?xml version="1.0" encoding="UTF-8"?>
<wsrp-producer-registry
xmlns="http://www.bea.com/servers/weblogic/wsrp-producer-registry/8.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/weblogic/wsrp-producer-regi
stry/8.0
wsrp-producer
-registry.xsd">
     <!-- Upload limit (in bytes) -->
     <upload-read-limit>1048576</upload-read-limit>
     <!-- Timeout (in milli seconds) -->
     <connection-timeout-secs>120000</connection-timeout-secs>
```

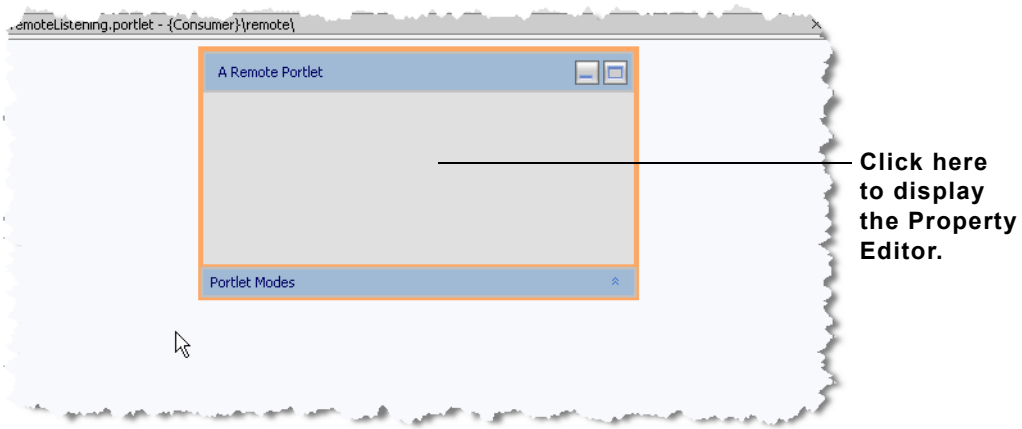
Note: The value *120000*—in milliseconds (not seconds, as the name of the descriptor suggests)—is a suggested timeout period. Your needs might require a longer or shorted timeout.

Setting the Timeout for Individual Remote Portlets

You can set the timeout value for individual remote portlets by setting the Connection Timeout property in the Property Editor in WebLogic Workshop, as described in the following procedure.

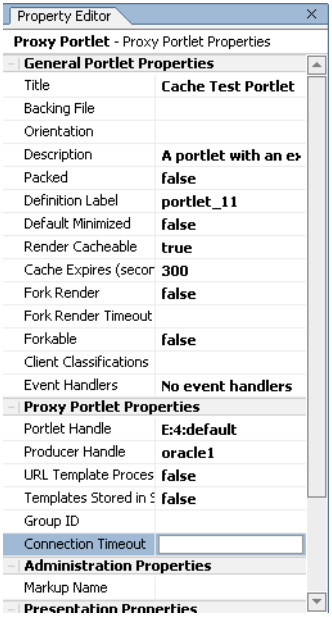
1. Before you begin, ensure that WebLogic Workshop is running and that the remote portlet you want to modify is displayed in the editor window, as shown in Figure 2-1.
2. Display the Proxy Portlet Properties editor by clicking in the center of the remote portlet in the editor window. Figure 2-1 shows you where to click to display the Proxy Portlet Properties editor.

Figure 2-1 Remote Portlet Displayed in the Editor



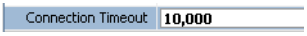
3. In the Proxy Portlet Properties editor, locate and highlight the **Connection Timeout** text field, as shown in Figure 2-2.

Figure 2-2 Proxy Portlet Property Editor—Connection Timeout Selected



4. In the **Connection Timeout** text field, enter a timeout value, in milliseconds, as shown in Figure 2-3.

Figure 2-3 Timeout Value



CHAPTER 3

Establishing Interportlet Communications with Remote Portlets

WebLogic Portal 8.1 with service pack 4 supports interportlet communication (IPC) between Producers and Consumers. In other words, local portlets on a Consumer can react to events on a federated portlet from a Producer and vice versa; for example, minimizing a local portlet would change the contents of a remote portlet that was listening for a minimize event.

This document describes the IPC model used by WebLogic Portal and shows how to apply it to WSRP-compliant portlets. It includes information on the following subjects:

- The WebLogic Portal IPC Model
- How IPC is Implemented
- Implementing IPC with WSRP: Example
- Special Considerations for Remote Portlets
- Understanding Backing Files

Note: For a list of supported portlets, please refer to WSRP Portlet Type Support in Introduction to WSRP.

The WebLogic Portal IPC Model

Earlier versions of WebLogic Portal allowed you to establish interportlet communications by using such techniques are adding `listenTo()` methods or backing files on page flow portlets. WebLogic Portal 8.1 with service pack 4 introduces a new IPC model based upon event handlers, Java objects that listen for predefined events on other portlets in the portal and fire actions when that event occurs.

Event Handlers

Event handlers "listen" for events raised on subscribed portlets and fire an action when a specific event is detected. Event handlers can listen and react to the following types of events:

- Portal Event
- Generic Event
- Page Flow Event
- Custom Event
- Struts Event

Events

Event actions depend upon the type of event being raised. Except for portal events, all other events can be identified in the Events field on the Event Handler tool. Events available with the portal event handler are listed in Table 3-1.

Table 3-1 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	Is in the help mode
onEdit	Is in the edit mode
onView	Is in the view mode
onRefresh	Is refreshed

Event Actions

Event handlers fire an action on the host portlet when that handler detects an event from another portlet in the application; for example, when the user minimizes the appropriate portlet a portal event called onMinimize might cause the handler listening for it to fire an action that invokes a backing file attached to it.

Table 3-2 lists the event actions available.

Table 3-2 Event Actions

This action...	Does this...
Change Window Mode	Changes the window's mode from its current mode to a user-specified mode; for example, from help mode to edit mode.
Change Window State	Changes the window's state from its current state to a user-specified state; for example, from maximized to delete state.
Activate Page	Opens a user-specified page.
Fire Generic Event	Fires a user-specified generic event.
Fire Custom Event	Fires a user-defined custom event. This event needs to be included in the portlet file.
Invoke BackingFile Method	Runs the backing file attached to the portlet. Backing files allow you to programatically add functionality to a portlet to enable preprocessing (for example, authentication) prior to rendering the portal controls. For more information, please refer to Special Considerations for Remote Portlets.

How IPC is Implemented

The IPC Tool included in WebLogic Workshop makes implementing event handlers relatively easy. To launch the tool, do the following:

1. Open a portlet in WebLogic Workshop.
2. In the Property Editor for that portlet, click the ellipses button (...) next to Event Handlers (if no event handlers have been added, the Event handler field will show that. If any event handlers have been added, the field will indicate the number added).

   The tool will appear, as shown in Figure 3-1

Figure 3-1 Event Handler Tool

3. Click Add Handler to open the event handler drop-down menu and select a handler.

   The dialog box will expand, opening up additional fields you can use to set up the handler (Figure 3-2).

Figure 3-2 Expanded Event Handler Tool

The entire process of setting up an event handler can all be handled by using this tool. What you need to do is:

1. Select an event handler.
2. Determine the portlet(s) to which that handler will listen.
3. Select an event that the handler will listen for.
4. Select and configure an action to fire when the event occurs.
5. Save the event handler.

Implementing IPC with WSRP: Example

This section describes the process of setting up interportlet communications between a Consumer and a Producer by using the Event handler tool in BEA WebLogic Workshop. This is a simple example in which minimizing the local portlet on the Consumer will change the text string in the portlet federated from the Producer.

Note: Currently when a change is made to a producer's metadata, consumers are not notified of the change. For example, if you add an event handler to a portlet on the producer *after* you create a remote proxy to that portlet, the remote portlet is not made aware of the change: it will not fire the event. The correct procedure is to add the event handler to the portlet on the producer before you create the remote portlet on the consumer. This is a basic limitation of Web Services in general.

This exercise is comprised of four main steps:

- Step 1: Set Up Your Environment
- Step 2: Create the Producer Portlets
- Step 3: Create the Consumer Portlets
- Step 4: Test the Application

Step 1: Set Up Your Environment

In this step, you will create a domain, an enterprise application, and two web applications (web projects) under the enterprise application.

Note: In this exercise, the root directory for your domain should be `BEA_HOME\user_projects\domains`; for the applications and all of their components,

it's `BEA_HOME\user_projects\applications`. These are default directories that are created when you install BEA WebLogic Platform.

Create the Domain

To create a domain, do the following:

1. Launch WebLogic Workshop.
2. Open the Tools menu and select WebLogic Server>Configuration Wizard.

   The Configuration Wizard launches.
3. Follow the prompts using the value specified in Table 3-3. Click Next when you are finished with each dialog box

Table 3-3 Configuration Wizard Values

On...	Select or Enter...
Create or Extend a Configuration	Create a new WebLogic Configuration
Select a Configuration Template	Basic WebLogic Portal Domain
Choose Express or Custom Configuration	Express
Configure Administrative Username and Password	Username: `weblogic` Password: `weblogic` Confirm Password: `weblogic`
Configure Server Start Mode and Java SDK	Sun SDK 1.4.2_05
Create WebLogic Configuration	Configuration Name:ipcWsrpDomain

4. Once the domain is created, click Done.

Create the Portal Application

To create a portal applications, do the following:

1. Open the File menu and select New>Application...

   The New Application dialog box appears.

2. Select Portal Application and do the following:
   - In Name, enter ipcWsrpTest.
   - In Server, click Browse to display the Select WebLogic Server config.xml File dialog box. Navigate to the ipcWsrpDomain directory and select `config.xml`.
3. Click Create.

   When the application is successfully created, it will appear in the application panel, as shown in Figure 3-3.

Figure 3-3 Producer Portal Application Created

Create the Web Applications (Web Projects)

Create Producer and Consumer portal web applications (Projects) under the ipcWsrpTest portal application by doing the following:

1. In the file tree, right-click ipcWsrpTest and select New>Project...

   The New Project dialog box appears.
2. Select Portal and Portal Web Project and in Name, enter producerWeb.
3. Click Create.

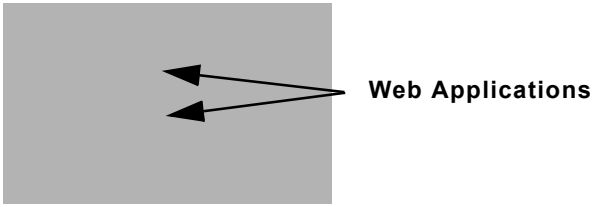
   producerWeb will now appear under the ipcWsrpTest application in the application tree (Figure 3-4).

Figure 3-4 Portal Web Application (Project) Added to the Producer

4. Repeat steps 1 through 3, this time entering consumerWeb in the Name field on the New Project dialog box.

   When the web applications are successfully created, they will appear under ipcWsrpTest in the Application panel, as shown in Figure 3-5.

Figure 3-5 Portal Web Applications (Projects) Added to the Portal Application

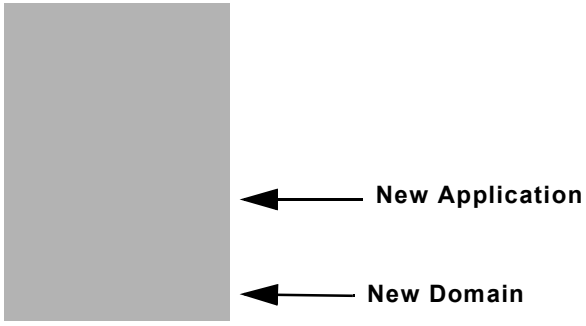


5. Start WebLogic Server by opening the Tools menu and selecting WebLogic Server>Start WebLogic Server.

Summary

With the completion of Step 1: Set Up Your Environment, you have created a domain, a portal applications, and two portal web applications, producerWeb (the Producer) and consumerWeb (the Consumer). You can verify that these components exist by looking in your file system at `BEA_HOME\user_projects\`, as shown in Figure 3-6.

Figure 3-6 File System Showing New Domain and Applications



Step 2: Create the Producer Portlets

In this step, you will create two JSP files on the Producer-side, along with the JSP portlets that will surface these files. You will also create a backing file that will contain the instructions necessary to complete the communication between two portlets (for more information on backing

files, please refer to Understanding Backing Files) and add an event handler to one of the portlets. Once you have created the portlets and attached the backing file, you will test the application in your browser.

Note: Before continuing with this procedure, ensure that WebLogic Workshop is running and the producerWeb Web application node is expanded.

Create the JSP Files and Portlets

To create the JSP files the Producer portlets will surface, do the following:

1. Under the producerWeb node, double-click `index.jsp`.

   The JSP file opens in Design View (Figure 3-7).

Figure 3-7 index.jsp in Design View

2. Click the phrase New Web Application Page to highlight it

   A box will appear around the text and an inner text field will appear in the Property Editor, under General (Figure 3-8).

Figure 3-8 Properties Menu inner Text Field

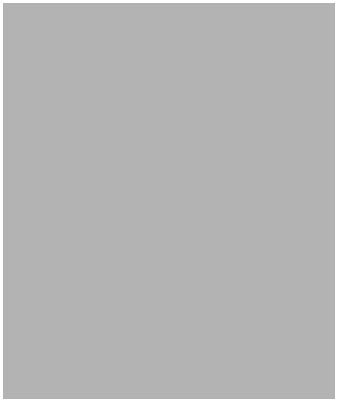
3. In the inner text field, click the ellipses button (...) to open the inner text dialog box and replace New Web Application Page with the phrase Minimize Me!!! Click OK.

   The dialog box closes and Minimize Me!!! appears in the inner text field and in the Design View, as shown in Figure 3-9.

Figure 3-9 New Text Added to the JSP File

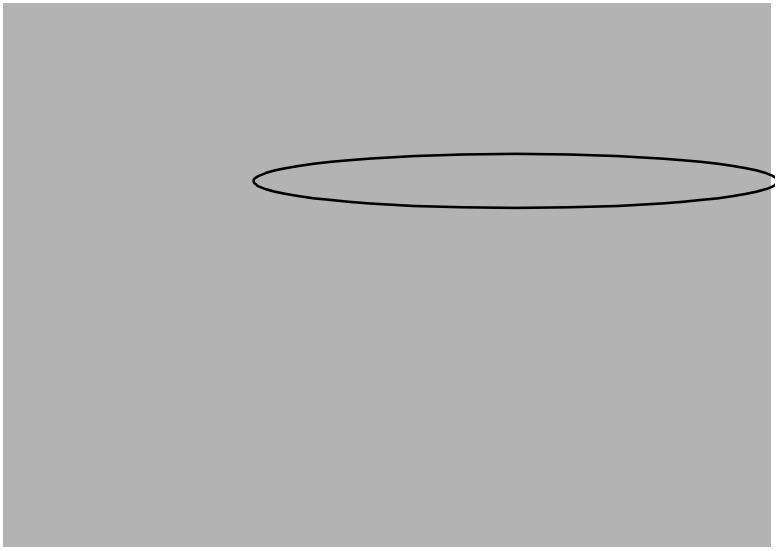
4. Open the File menu and select Save As...
5. Save the file as `aPortlet.jsp`.
6. Right click `aPortlet.jsp` in the Application tree and select Generate Portlet... from the context menu (Figure 3-10).

Figure 3-10 Generating a Portlet from index.jsp



The Portal Details dialog box appears (Figure 3-11). Note that `/aPortlet.jsp` appears in the Content URI field.

Figure 3-11 Portal Details Dialog Box for aPortlet



7. Select Minimizable, Maximizable, and Deletable and click Finish.

   `aPortlet.portlet` will appear under producerWeb in the application tree.

8. When the Portlet Wizard closes, `aPortlet.jsp` should still be open. If it isn't, double-click it in the application tree to reopen it.

9. Open the File menu and select Save as.

10. In the Name field of the Save "aPortlet.jsp" as dialog box, enter `bPortlet.jsp` and click Save.

11. On the JSP display, click Source View.

    The JSP code appears (Figure 3-12).

Figure 3-12 JSP File Source View

12. Copy the code from Listing 3-1 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.

Listing 3-1 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 example in Figure 3-13.

Figure 3-13 Updated JSP Source

13. Save the file either by clicking the save button or by opening the File menu and selecting Save.
14. Repeat steps 6 and 7 to create a JSP portlet for bPortlet.

Create the Backing File

To create the backing file, do the following:

1. In producerWeb, expand the `WEB-INF` node and right-click `src` to open a context menu.
2. Select New>Folder.

   The Create New Folder dialog box appears.
3. In Enter a new folder name, type `backing` and click OK.

   The folder `backing` will appear under `WEB-INF/src`, as shown in Figure 3-14.

Figure 3-14 backing Folder Under WEB-INF/src

4. Right-click backing and select New>Java Class.

   The New File dialog box appears.

5. In Name, enter `Listening.java` and click Create.

   The Source View of the new Java class appears (Figure 3-15).

Figure 3-15 Listening.java Source File

6. Copy the code from Listing 3-2 into `Listening.java`.

Listing 3-2 Backing File Code for listening.java

```
package backing;
```

```
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;
import com.bea.netuix.events.Event;
import com.bea.netuix.events.GenericEvent;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Listening extends AbstractJspBacking

{
    private static boolean minimizeEventHandled = false;

     public void handlePortalEvent(HttpServletRequest request,
        HttpServletResponse response, Event event)

    {
        minimizeEventHandled = true;
    }

    public boolean preRender(HttpServletRequest request, HttpServletResponse
         response)
    {
        if (minimizeEventHandled){

                request.setAttribute("minimizeEvent","minimize event handled");
        }else{
                 request.setAttribute("minimizeEvent",null);
        }

        // reset
        minimizeEventHandled = false;

         return true;
    }
}
```

The source should now look like that shown in Figure 3-16.

Figure 3-16 Listening.java with Updated Backing File Code

7. Save `Listening.java` either by opening the File menu and selecting Save or clicking the save button.

Attach the Backing File

Now you will attach the backing file created in Create the Backing File to bPortlet. Do the following:

1. In the Application tree, double-click `bPortlet.portlet` to open it.
2. In the Property Editor, under Portlet Properties, type `backing.Listening` into the Backing File field, as shown in Figure 3-17 and press Tab.

Figure 3-17 Attaching the Backing File in the Property Editor

3. Save the file.

Add the Event Handler to bPortlet

You now add the event handler to bPortlet. 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, do the following:

Note: `bPortlet.portlet` should be displayed in WebLogic Workshop. If it isn't, locate it under producerWeb in the application panel and double-click it.

1. In the Property Editor, click the ellipses button (...) next to Event Handlers.

   The Event Handler dialog box appears (Figure 3-18).

Figure 3-18 Event Handler Dialog Box

2. Click Add Handler to open the Event Handler drop-down list.
3. Select Handle Portal Event.

   The Event Handler dialog box expands to allow entry of more details (Figure 3-19).

Figure 3-19 Event Handler Dialog Box Expanded

4. Accept the defaults for all fields except Portlet.
5. In Portlet, click the ellipses button (...).

   The Open dialog box for producerWeb appears.
6. Double-click `aPortlet.portlet`.

   The Open dialog box closes and Portal_1 appears in the Listen to: list and the Portlet field (Figure 3-20). Portal_1 is the definition label of the portlet to which the event handler will listen.

Figure 3-20 Adding portlet_1

7. 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 3-21.

Figure 3-21 Event Drop-down List

8. Click Add Action... to open the action drop-down list and select Invoke BackingFile Method, as shown in Figure 3-22

Figure 3-22 Add Action Drop-down List

Invoke BackingFile Method appears on the Events list as a child to Handle Portal Event, as shown in Figure 3-23.

Figure 3-23 Event List with Action Added

9. Open the Method drop-down control and select `handlePortalEvent` (Figure 3-24).

Figure 3-24 Adding the Backing File Method

10. Click OK.

The event handler is added. Note that the Event Handler field in the Property Editor now reads “1 Event Handler.”

Test the Application

To test the application, do the following:

Note: Before you begin, ensure that all files are saved.

1. Create a portal called `ipcLocal.portal` by doing the following:
   a. Right-click producerWeb and select New>Portal.
   b. In the New File dialog box's File Name field, enter ipcLocal.
   c. Click Create.

      When the portal is successfully created, its layout will appear in WebLogic Workshop.
2. Drag both aPortlet and bPortlet from the Data Palette onto the portal layout, as shown in Figure 3-25.

Figure 3-25 Portal Layout with Portlets Added

3. Save the portal either by clicking the save button or opening the file menu and selecting Save.
4. Open the Portal Menu and select Open Current Portal...

   The portal will render in your browser (Figure 3-26).

Figure 3-26 ipcLocal Portal in Browser

5. Minimize aPortlet.

   Note the content change in bPortlet.

Figure 3-27 ipcLocal Portal with aPortlet Minimized

Summary

In this step, you added to the portal application components created in Step 1: Set Up Your Environment two JSP portlets built on the Producer. One portlet, aPortlet, was fairly simple, while the second portlet, bPortlet, surfaced a more complex JSP file, leveraged a backing file, and contained a portal event handler. When you tested the Producer application, you observed how the two portlets communicated when an event occurred on aPortlet. This is called local interportlet communications.

Step 3: Create the Consumer Portlets

In this step, you will create two portlets on the Consumer, one a JSP portlet and the other a remote portlet. As in Step 2: Create the Producer Portlets, the JSP portlet will surface the `aPortlet.jsp` file while the remote portlet will consume bPortlet you created in Create the JSP Files and Portlets.

Set Up the Exercise

Before you continue with this exercise, do the following:

- Ensure that WebLogic Workshop is open and WebLogic Server is running.
- Go to your file system and copy `aPortlet.jsp` from:

  `BEA_HOME \user_projects\applications\ipcWsrpTest\producerWeb`

  to

  `BEA_HOME\user_projects\applications\ipcWsrpTest\consumerWeb.`

  Note: This step is necessary to streamline this exercise and because WebLogic Workshop does not allow you to copy files between folders. If you don't want to copy

`aPortlet.jsp` from producerWeb to consumer Web, you can create it from scratch, as described in Create the JSP Files and Portlets.

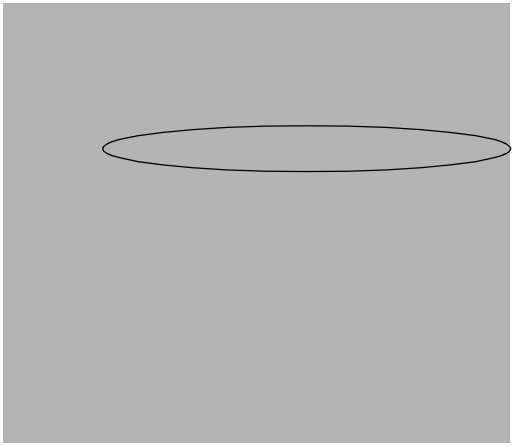
Create the JSP Portlet

To create a the JSP portlet, do the following:

1. Right-click `aPortlet.jsp` and select Generate Portlet...

   The Portlet Details dialog box appears, with `/aPortlet.jsp` already in the Content URI field (Figure 3-28).

Figure 3-28 Portlet Details Dialog Box



2. Fill out the details as described here:
   - Minimizable: select
   - Maximizable: select
   - Deletable: select.
3. Click Finish.

   WebLogic Workshop refreshes and the new portlet layout appears, as shown in Figure 3-29.

Figure 3-29 New Portlet

Create the Remote Portlet

To create the remote portlet, do the following:

1. Right-click consumerWeb and select New>Portlet.

   The New File dialog box appears.

2. In File Name, enter `bPrime.portlet` and click Create.

   The Portlet Wizard is launched, showing the Select Portlet Type dialog box (Figure 3-28).

Figure 3-30 Select Portlet Type Dialog Box

3. Select Remote Portlet and click Next.

   The Find Producer/Select Producer dialog box appears (Figure 3-31).

Figure 3-31 Find Producer/Select Producer Dialog Box

4. Select Find Producer and enter the producer's service description (Web Services Description Language, or *WSDL*) in the service description field. The WSDL communicates interface information between web service Producers and Consumers, allowing you to leverage a service's capabilities without possessing the source code for the service. For this exercise, enter:

   ```
   http://localhost:7001/producerWeb/producer?WSDL
   ```

5. Click Retrieve.

   After a few seconds, the dialog box refreshes, showing the Producer Details.

6. Click Register.

   The Register dialog box appears (Figure 3-32).

Figure 3-32 Register Dialog Box

7. In Producer Handle, enter myProducer and click Register.

   The Find Producer dialog box reappears with myProducer in the Select Producer field.

8. Click Next.

   The Select Portlet From List dialog box appears (Figure 3-33

Figure 3-33 Select Portlet From List Dialog Box

9. Select bPortlet and click Next.

The Proxy Portlet Details dialog box appears.

10. Click Finish.

    The remote portlet will be created and will appear as `bprime.portlet` under consumerWeb in the Application panel.

11. In the Property Editor for bPortlet, set Render Cachable to False (Figure 3-34) and press Tab.

Figure 3-34 Setting Render Cachable Property to False

12. Save the portlet.

Summary

With the completion of the two Consumer portlets, you have now created all of the necessary components to demonstrate interportlet communications between a remote and a local portlet. In the next step, you will add the Consumer portlets to a Consumer portal and raise an event on one portlet that will cause a reaction on the other.

Step 4: Test the Application

In this step, you will test the Consumer application to verify that minimizing aPortlet will change the content of bPrime. You will create a portal and add the two portlets created in Step 3: Create the Consumer Portlets. You will then build the application and view the portal in a browser.

Build the Portal

To create a portal for testing the application, do the following:

Note: Ensure all files are saved.

1. Right-click consumerWeb and select New>Portal.
2. In the New File dialog box's File Name field, enter ipcConsumer.
3. Click Create.

When the portal is successfully created, its layout will appear in WebLogic Workshop.

4. Drag both aPortlet and bPortlet (which identifies bPrime) from the Data Palette onto the portal layout, as shown in Figure 3-35.

Figure 3-35 ipcConsumer Portal Layout

5. Save the portal.

Test the Portal

When the build completes, verify that an event on aPortlet can affect the content of bportlet by doing the following:

1. Open the Portal menu and select Open Current Portal.

A browser opens displaying the ipcConsumer portal (Figure 3-36).

Figure 3-36 ipcConsumer Portal in a Browser

2. In aPortlet, click the minimize button.

   aPortlet minimizes and the contents of bPortlet changes, as shown in Figure 3-37.

Figure 3-37 ipsConsumer Portal in Browser After Minimize Event

Special Considerations for Remote Portlets

If an event has one or more events that must be handled by the Producer (for example., `netuix:invokePageFlowAction`, `netuix:invokeStrutsAction`, `netuix:invokeJavaPortletMethod`, and `netuix:invokeBackingFileMethod`), the Producers adds a `dispatchToRemotePortlet()` method. This element indicates that the Consumer must dispatch the event to the Producer.

Understanding Backing Files

The portal you created in Implementing IPC with WSRP: Example uses backing files to achieve communication between two portlets. Backing files allow you to programatically add functionality to a portlet by implementing (or extending) a Java class, which enables preprocessing (for example, authentication) prior to rendering the portal controls.

What are Backing Files?

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 mimic the controls lifecycle methods (see How Backing Files are Executed) and are invoked at the same time the controls lifecycle methods are invoked.

Which Controls Support Backing Files?

At this time, the following controls support backing files:

- Desktops
- Books
- Pages
- Portlets

How Backing Files are Executed

All backing files are executed before and after the JSP is called. In its lifecycle, each backing file calls these methods:

- `init()`
- `handlePostBackData()`
  - `raiseChangeEvents()`
- `preRender()`
- `dispose()`

On every request, the following occurs:

1. All `init()` methods are called on all backing files on an "in order" basis (that is, in the order they appear in the tree). This method gets called whether or not the control (that is, portal, page, book, or desktop) is on an active page.
2. Next, if the operation is a postback *and* the control (a portlet, page, or book) is on a visible page, all `handlePostbackData()` methods are called. In other words if a portlet is on a page but its parent page is not active, then this method will not get called.
   - If `_nfpb="true"` is set in the request parameter of any `handlePostbackData()` methods called, `raiseChangeEvents()` is called. This method causes events to fire.
3. Next, all `preRender()` methods are called for all controls on an active (visible) page.

4. Next, the JSPs get called and are rendered on the active page by the `<render:beginRender>` JSP tag. Rendering is stopped with the `<render:endRender>` tag.
5. Finally, the `dispose()` method gets called on the backing file.

Thread Safety with Backing Files

A new instance of a backing file is created per request, so you don't have to worry about thread safety issues. New Java VMs are specially tuned for short-lived objects, so this is not the performance issues it once was in the past. Also, `JspContent` controls support a special type of backing file that allows you to specify whether or not the backing file is thread safe. If this value is set to `true`, only one instance of the backing file is created and shared across all requests.

CHAPTER 4

Working with Producers

The main focus of this chapter is to explain how to use WSRP with applications running in a WebLogic Server or WebLogic Express domain. In addition, this chapter explains how to install a producer into a WebLogic Server domain using WebLogic Workshop and how to specify which portlets deployed in a producer application are available to consumers.

This chapter includes the following topics:

- Using WSRP in a Basic WebLogic Server Domain
- Using WSRP in a WebLogic Express Server Domain
- Enabling Portlets on the Producer

Using WSRP in a Basic WebLogic Server Domain

This section explains how to configure a basic WebLogic Server domain as a WSRP producer. The example in this section assumes that you have a functioning Struts module deployed in a WebLogic Server domain.

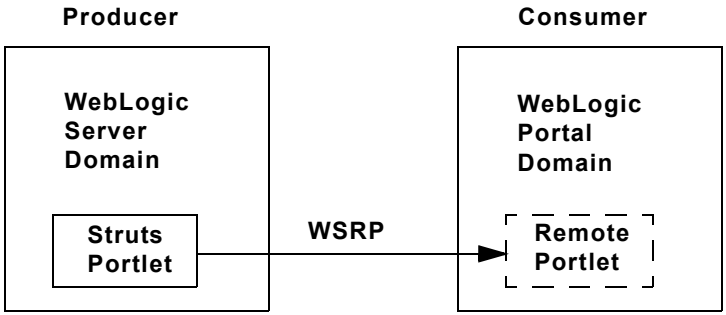
Note: To complete the steps outlined in this section, you must have access to certain JAR files that are part of a WebLogic Portal installation. If you do not have WebLogic Portal installed, we recommend that you obtain a freely available installer from the BEA Systems website and install WebLogic Portal in a separate, temporary area. You can then copy the required JAR files to your producer web application and delete the temporary installation area.

The goal of this procedure is to create a portlet in a producer that can be consumed remotely. To achieve this, you need to modify the configuration of your Struts application, copy appropriate

JAR and TLD files to the application, and create a portlet that references the Struts module in which your application resides.

By following this procedure, you can expose a Struts application as a remote portlet that a WebLogic Portal application can consume, as illustrated in Figure 4-1.

Figure 4-1 WebLogic Portal consumer consumes Struts portlets from a WebLogic Server producer



This section contains the following sections:

- **Getting Started** — Describes the example environment configuration.
- **Configuring the WSRP Producer** — Explains how to configure the WebLogic Server domain as a WSRP producer. This manual procedure explains which library and configuration files are required and how to modify the configuration files to install the producer.
- **Testing the Producer** — Explains how to test the producer configuration.

Getting Started

Before you configure the producer, we assume you have taken the following steps:

- You have created a WebLogic Server Domain. You can create this domain using the BEA WebLogic Configuration Wizard. This domain does not contain any WebLogic Portal components. If you are using WLX, create a WLX Basic Domain.
- You have a working Struts application that is configured as a Struts module.

Note: We recommend that only experienced Struts and WebLogic Portal developers attempt this procedure. It is easy to make simple mistakes that can be difficult to troubleshoot. Whenever possible, we suggest ways to test your progress.

As noted previously, it is crucial that you have a Struts module that you can deploy and run successfully in the WebLogic Server environment before you configure the producer. *Note that the Struts application must be configured as a Struts module.* For detailed information on configuring Struts modules, refer to the Struts documentation. Basically, a module resides in a subdirectory of the web application and has its own `struts-config.xml` file, whose path is specified typically in the `web.xml` file.

> **Tip:** Before trying to use your Struts application in a WSRP environment, try converting the application to a local portlet running in a WebLogic Portal domain. If you can convert the Struts application to a local, standalone portlet in a test environment, then your chances of running it successfully with WSRP are improved. To do this, you need WebLogic Workshop. You can download a free developer's version of WebLogic Workshop from the BEA website. For information on converting a Struts application to a portlet using WebLogic Workshop, see: Integrating Struts Applications into a Portal.

Configuring the WSRP Producer

This section explains how to configure a basic WebLogic Server domain as a WSRP producer. The goal of this procedure is to surface a Struts application as a remote portlet that can be consumed by a WebLogic Portal application.

Modify the CLASSPATH for the WebLogic Server Domain

Note: Before attempting to expose a Struts portlet with WSRP, be sure the underlying application is configured as a module and works properly *before* you attempt to configure the WSRP producer. We also recommend that you try to create a functioning local portlet using WebLogic Workshop. See "Getting Started" on page 4-2 for more information on this recommendation.

1. Modify the `CLASSPATH` for the WebLogic Server domain. Open the following script file:

   On Windows: `DOMAIN_HOME\startWebLogic.cmd`

   On Linux: `DOMAIN_HOME/startWebLogic.sh`

Find the `CLASSPATH` definition, and add the following elements:

Note: Be sure there are no newlines within the `CLASSPATH` elements. Do not delete any of the elements that are already assigned to the `CLASSPATH`.

On Windows:

```
%JAVA_HOME%\jre\lib\rt.jar;
%WL_HOME%\server\lib\webservices.jar;
```

```
%WL_HOME%\portal\lib\wsrp\wsrp-common.jar;
%WL_HOME%\server\lib\xbean.jar;
%WL_HOME%\server\lib\wlxbean.jar;
%WL_HOME%\portal\lib\netuix\system\netuix_schemas.jar;
%WL_HOME%\portal\lib\netuix\system\netuix_system.jar;
%WL_HOME%\server\lib\knex.jar;
%WL_HOME%\javelin\lib\javelin.jar
%WL_HOME%\common\lib\log4j.jar
```

On Linux:

```
$JAVA_HOME/jre/lib/rt.jar;
$WL_HOME/server/lib/webservices.jar;
$WL_HOME/portal/lib/wsrp/wsrp-common.jar;
$WL_HOME/server/lib/xbean.jar;
$WL_HOME/server/lib/wlxbean.jar;
$WL_HOME/portal/lib/netuix/system/netuix_schemas.jar;
$WL_HOME/portal/lib/netuix/system/netuix_system.jar;
$WL_HOME/server/lib/knex.jar;
$WL_HOME/javelin/lib/javelin.jar
$WL_HOME/common/lib/log4j.jar
```

Listing 4-1 shows an example of the completed CLASSPATH for a Windows system.

Listing 4-1 CLASSPATH Example for Windows

```
CLASSPATH=%WEBLOGIC_CLASSPATH%;%POINTBASE_CLASSPATH%;%JAVA_HOME%\jre\lib\
rt.jar;%WL_HOME%\server\lib\webservices.jar;%WL_HOME%\portal\lib\wsrp\
wsrp-common.jar;%WL_HOME%\server\lib\xbean.jar;%WL_HOME%\server\lib\
wlxbean.jar;%WL_HOME%\portal\lib\netuix\system\netuix_schemas.jar;
%WL_HOME%\portal\lib\netuix\system\netuix_system.jar;%WL_HOME%\server\
lib\knex.jar;%WL_HOME%\javelin\lib\javelin.jar;%WL_HOME%\common\lib\
log4j.jar;%CLASSPATH%
```

2. Create the portal-specific directories that you added to the CLASSPATH and copy the appropriate JAR files into them. If you do not have a WebLogic Portal installation, you need to install WebLogic Portal in a temporary area and copy the JAR files from there. See

the note at the beginning of this section for more information on obtaining an installer. For example, the JAR files `wsrp-common.jar`, `netuix_schemas.jar`, `netuix_system.jar` must be on your system in the location referenced by the `CLASSPATH`.

3. **Checkpoint:** Restart your server to ensure that the startup script functions properly. Be sure that your Struts application is deployed and works as well.

Modify the Struts Application

In this section, you will modify the Struts application. This process includes modifying configuration files and adding/replacing files within the application. The configuration files need to be updated to reference BEA-specific classes and tag library files. These components are required for the application to function within a WSRP environment. For instance, the BEA tag libraries ensure that URL rewriting for the portlet is handled properly.

1. It is recommended that you copy the root folder of your Struts application to a new folder. You can then make changes to the new copy and preserve the original.
2. Modify the Struts application's `web.xml` file as follows:

Be sure that the following elements are in your Web application's `web.xml` file. If any of these elements are missing, you must add them:

```
<!-- WSRP Servlet configuration -->

<servlet>
   <servlet-name>com.bea.wsrp.producer.WsrpServer</servlet-name>
   <servlet-class>com.bea.wsrp.producer.WsrpServer</servlet-class>
   <load-on-startup>2</load-on-startup>
</servlet>

<servlet>
   <servlet-name>com.bea.wsrp.logging.MessageMonitor</servlet-name>
   <servlet-class>com.bea.wsrp.logging.MessageMonitor</servlet-class>
   <init-param>
      <param-name>enableSoapMessageLogging</param-name>
      <param-value>true</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>



<!-- WSRP Servlet mapping -->

<servlet-mapping>
   <servlet-name>com.bea.wsrp.producer.WsrpServer</servlet-name>
```

```
   <url-pattern>/producer/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
   <servlet-name>com.bea.wsrp.logging.MessageMonitor</servlet-name>
   <url-pattern>/monitor</url-pattern>
</servlet-mapping>


<!-- Standard Action Servlet Configuration (with debugging) -->

<servlet>

...

   <init-param>
      <param-name>moduleConfigLocators</param-name>
     <param-value>com.bea.struts.adapter.util.ModuleConfigLocator</param-value>
   </init-param>

...

</servlet>


<!-- WSRP Struts Adapter Tag Library Descriptors -->

<taglib>
   <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
   <taglib-location>/WEB-INF/lib/struts-adapter-html.tld</taglib-location>
</taglib>

<taglib>
   <taglib-uri>/WEB-INF/struts-nested.tld</taglib-uri>
   <taglib-location>/WEB-INF/lib/struts-adapter-nested.tld</taglib-location>
</taglib>

<!-- Struts Tag Library Descriptors -->

<taglib>
    <taglib-uri>/tags/struts-bean</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>

<taglib>
    <taglib-uri>/tags/struts-logic</taglib-uri>
    <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
</taglib>

<!-- COMMENTED in favor of WSRP adapter Tag Libraries -->
```

```
<!--
<taglib>
    <taglib-uri>/tags/struts-html</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>

<taglib>
    <taglib-uri>/tags/struts-nested</taglib-uri>
    <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
</taglib>
-->
```

3. Copy the following TLD files to the `WEB-INF\lib` directory of the Struts application:
   - `struts-adapter.jar`
   - `struts-adapter-html.tld`
   - `struts-adapter-nested.tld`

   You can find these files in:

   `%WL_HOME%\portal\lib\netuix\system\ext\web`
4. Copy the following JAR files to the `WEB-INF\lib` directory of the Struts application:
   - `netui-adapter.jar`
   - `netui-util.jar`
   - `netui.pageflow.jar`
   - `netui.scoping.jar`

   You can find these files in:

   `WL_HOME\samples\portal\portalApp\sampleportal\WEB-INF\lib`
5. Copy the following JAR files to the `WEB-INF\lib` directory of the Struts application:
   - `wsrp-producer.jar`
   - `wsrp-struts-adapter.jar`

   You can find these files in:

   `%WL_HOME\portal\lib\wsrp` and
   `%WL_HOME\portal\lib\wsrp\adapters`
6. **Checkpoint:** Start your server and redeploy the Struts application now to test that it still works. If the application does not work properly, you need to recheck the preceding steps.

7. Copy the contents of Listing 4-2 and paste it into a file named `wsrp-producer-config.xml` in the Struts applications's `WEB-INF` directory.

Listing 4-2 wsrp-producer-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<wsrp-producer-config
    xmlns="http://www.bea.com/servers/weblogic/wsrp-producer-config/8.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.bea.com/servers/weblogic/wsrp-producer-config/8
.0 wsrp-producer-config.xsd">
    <description>
       This is a WSRP Producer
    </description>
    <!-- This element describes the capabilities of this producer. Set
    the secure attribute to "true" if you require this producer offer
    any port over SSL. If this webapp is portal-enabled, you may
    enable registration and portlet management services by setting the
    required attribute to "true". -->
    <service-config>
        <registration required="false" secure="false"/>
        <service-description secure="false"/>
       <!-- Set accepts-mime to true to more efficiently process uploaded files
        when the consumer is a WebLogic Portal. -->
        <markup secure="false" rewrite-urls="true"
         transport="string" accepts-mime="false"/>
        <portlet-management required="false" secure="false"/>
    </service-config>
    <supported-locales>
        <locale>en</locale>
        <locale>en-US</locale>
    </supported-locales>
</wsrp-producer-config>
```

Note: Be sure the **`registration required`** and **`portlet-management required`** parameters are set to **`false`**.

8. Add the following line to the `struts-config.xml` file of your Struts module.

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

9. Create a portlet to surface the Struts actions that you want to expose as a remote portlet to consumers. Place the portlet in a `.portlet` file in the directory containing the module you want to surface.

   For example, if you have a `welcome.do` action in a module called `echo` in the `/echo` subdirectory, the portlet file looks like the following:

Listing 4-3 Example Portlet

```
<?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:portlet definitionLabel="portlet_sayHello" title="WSRP Echo
     Example">
        <netuix:titlebar>
            <netuix:minimize/>
            <netuix:maximize/>
        </netuix:titlebar>
        <netuix:content>
            <netuix:strutsContent action="welcome" module="/echo"/>
        </netuix:content>
    </netuix:portlet>
  </portal:root>
```

10. **Checkpoint:** Start the server (if it isn't running), redeploy the Struts application, and make sure it functions properly.

Testing the Producer

This section describes a procedure for testing your producer configuration.

1. Start WebLogic Server with the new `CLASSPATH` setting.

2. Use WebLogic Server Console to deploy (or redeploy) the Struts web application.
3. Open a web browser and test the Struts web module that you want to surface. The module should work normally.
4. Test the WSRP producer by entering its WSDL URL in a browser. For example, if your Struts application is called myStrutsApp, the WSDL URL is:

   `http://host:port/myStrutsApp/producer?wsdl`

   where *host* is the IP address of the server machine, *port* is the server's listening port number, and *myStrutsApp* is the root name of the Struts web application.

   If this test is successful, an XML Webservice WSDL file is returned that looks similar to the file shown in Figure 4-2.

Figure 4-2 XML Webservice WSDL File

```
- <wsdl:definitions targetNamespace="urn:oasis:names:tc:wsrp:v1:wsdl">
    <wsdl:import namespace="urn:oasis:names:tc:wsrp:v1:bind"
    location="http://www.oasis-open.org/committees/wsrp/specifications/version1/wsrp_v1_bindings.wsdl"/>
    <wsdl:import namespace="urn:bea:wsrp:ext:v1:bind" location="wlp_wsrp_v1_bindings.wsdl"/>
  - <wsdl:service name="WSRPService">
    - <wsdl:port name="WSRPBaseService" binding="urn:WSRP_v1_Markup_Binding_SOAP">
        <soap:address location="http://localhost:7001/strutsHello/producer"/>
      </wsdl:port>
    - <wsdl:port name="WSRPServiceDescriptionService"
      binding="urn:WSRP_v1_ServiceDescription_Binding_SOAP">
        <soap:address location="http://localhost:7001/strutsHello/producer"/>
      </wsdl:port>
    - <wsdl:port name="WLP_WSRP_Ext_Service" binding="urn1:WLP_WSRP_v1_Markup_Ext_Binding_SOAP">
        <soap:address location="http://localhost:7001/strutsHello/producer"/>
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Consuming the Producer Portlet

1. On another machine, create a WebLogic Portal Domain. You can use the WebLogic Configuration Wizard to do this. If you cannot use another machine, be sure the server's listen port does not conflict with the port used by the producer server. If necessary, you can obtain a free developer's version of WebLogic Portal by visiting the BEA website.

2. Use WebLogic Workshop to create a Portal Application and associate the application with the new WebLogic Portal Domain. If necessary, you can obtain a free developer's version of WebLogic Workshop by visiting the BEA website.
3. Create a new Portal Web Project to the application. This application is the consumer application.
4. Create a portal in the consumer application.
5. Start the server that hosts the consumer.
6. Create a remote portlet in the Portal Web Project you just created. Point the WSDL to the Struts application on the producer. For example:

   `http://producerHost:producerPort/StrutsApp/producer?WSDL`

   Where *`producerHost:producerPort`* is the IP address and port number of the machine hosting the producer, and *`StrutsApp`* is the name of the context directory for the Struts application that contains the producer portlet(s) that you wish to surface.
7. On the consumer, add the remote portlet to the portal and open the portal. The Struts portlet you created on the producer appears in the portal.

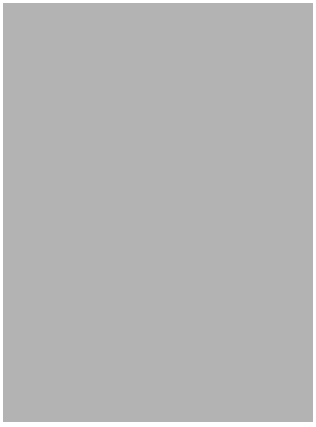
Using WSRP in a WebLogic Express Server Domain

You can configure a WebLogic Express (WLX) server domain as a WSRP producer. To do this, follow the same steps outlined in the previous section, "Using WSRP in a Basic WebLogic Server Domain" on page 4-1. The procedures for configuring a WLX domain and a WebLogic Server domain as a producer are the same.

Enabling Portlets on the Producer

A Producer can have any number of portlets built on it, sometimes into the thousands. By default, all of these portlets are available to consumers as remote portlets. You can, however, specify which portlets are actually available to consumers by setting the offerPortlets property in the Portlet Property Editor.

Figure 4-3 Proxy Portlet Property Editor—Offer as Remote Property Selected



For instructions on how to set this property, please refer to Enabling/Disabling a Portlet for Remote Consumption in the WebLogic Workshop online help system at:

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportlets/portletEnable.html

CHAPTER 5

Best Practices for Implementing WSRP

This section describes programming and tuning practices that you should follow to ensure the best performance of your WSRP-compliant portlets, Producers, and Consumers. It contains the following guidelines:

- Portlet Programming Guidelines
- Performance Tuning Recommendations

Portlet Programming Guidelines

Please follow these guidelines when programming WSRP-compliant portlets:

- Requests and Sessions
  - Don't rely on request attributes across portlets.
  - If your portlets share a session, host them on the same Producer.
- Security
  - To secure messages, implement SSL on any port through which the Producer will be offered.
  - Specify `true` for all secure attributes in the `<service-config>` element of the Producer project's `WEB-INF/wsrp-producer-config.xml` file (for more information, please refer to Establishing WSRP Security).

    Note: If you make any changes to `wsrp-producer-config.xml`, you will need to redeploy or bounce the server before the changes become active.

  - To manage user identity, rely on single-sign-on (SSO), which is set up by default in WebLogic Portal and then have users login to the Consumer portal. WebLogic Portal will manage SSO automatically. Information sharing between the Consumer and the Producer is facilitated by using an external Profile service that can run with the user identity as the key.
  - To secure resources on the producer, see "Setting Security Constraints on Resources" on page 9-2.

- URLs
  - When creating URLs in a portlet, do not use direct links; instead use WebLogic Portal tags and APIs to create URLs.
- Look and Feel
  - Let portlets use standard style attributes and specify those attributes on the Portal skins.
- Backing Files
  - You can use backing files on the Consumer side (remote-portlet) to take some action based on session / request objects or property sets.
- Caching WSRP Portlets
  - Producer - Use <wl:cache> or p13nCache wherever possible.
  - Consumer (remote-portlet) - Use 'RenderCacheable' attribute if you want to cache the remote portlet's rendered HTML. However, this is a session scoped cache and is not configurable.
- Exception Handling
  - Producer; to prevent stack traces from appearing, errors should be handled on the Producer side and a suitable business message provided.
  - Consumer (remote-portlet); in WebLogic Workshop, with your remote portlet open, do the following (in this order):

    Click the Document Structure tab of the remote portlet.

    Click remote Content.

    Click Property Editor.

    Type in the Error URI
- Page Flow

  - If using a Portal Web application as your Producer, all the portal artifacts are available in the web application; however, for any WSRP Producer that is not a portal web application, you cannot use portal features such as property sets. If you need to access portal features in your Producer, use a Portal Web application.

- Development on a WebLogic Server Domain
  - For detailed information on using WSRP with a WebLogic Server domain, see Using WSRP in a Basic WebLogic Server Domain.

Performance Tuning Recommendations

To ensure optimal performance of your Producers and Consumers, we recommend the following performance tuning guidelines:

- On the Producer side:
  - Enable attachment support by adding `<markup transport="attachment"/>` to `WEB-INF/wsrp-producer-config.xml`, as shown in Listing 5-1.

Listing 5-1 Enabling Attachment Support

```
<?xml version="1.0" encoding="UTF-8"?>
wsrp-producer-config
    xmlns="http://www.bea.com/servers/weblogic/wsrp-producer-config/8.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/weblogic/wsrp-producer-conf
ig/8.0
wsrp-producer-cnfig.xsd">

    <service-config>
        <registration required="false" secure="true"/>
        <service-description secure="true"/>
        <markup secure="true" rewrite-urls="true" transport="attachment"/>
        <portlet-management required="false" secure="true"/>
    </service-config>
```

  - Let the Producer create correct URLs by using Consumer-supplied URL templates. This is the default practice.

- On the Consumer side
  - Accept the default behavior to enable caching for remote portlets.
  - Enable forked rendering for remote portlets.
  - Set connection timeout. See "Setting a Timeout Value on Remote Portlets" on page 2-4 for detailed information on setting timeouts.
  - Disable logging
  - Undeploy MessageMonitor servlet from `WEB-INF/web.xml`.

Avoid Moving Producers

When you add producers and create remote portlets, the producer registry (`WEB-INF/wsrp-producer-registry.xml`) and the portal framework database tables contain specific information about the producer, such as its WSDL address and the addresses of ports described in the WSDL. If you propagate or move the producer from one environment to another, this data becomes invalid. In this case, consumers whose proxy portlets reference the producer's portlets will no longer be able to find them.

Note: Currently, WebLogic Portal only supports a shared registration model, where staging and production environments share the same producer registration handle. For more information on shared registration and propagating WSRP producers, see the Production Operation User's Guide.

Tip: You can update the database entries for a producer programmatically. The following class provides methods to get and update producer information:

```
com.bea.wsrp.consumer.management.producer.ProducerManager
```

Upgrading Simple Producers from Service Pack 3

When upgrading a simple producer from WebLogic Portal Service Pack 3 to a later service pack version, WSRP producer web application libraries are not automatically updated. If you upgrade from SP3 to a later service pack version, you must perform the following steps to ensure that your simple producer continues to function properly as a WSRP producer.

Note: A simple producer is a non-portal web application. For more information on simple producers, see "Producers" on page 1-4.

1. Perform the normal procedures for upgrading a WebLogic Portal application from SP3 to a later service pack version.
2. Copy the the following files into the `WEB-INF/lib` directory of your web application:

```
BEAHOME/weblogic81/portal/lib/netuix/web/netui-adapter.jar
BEAHOME/weblogic81/portal/lib/netuix/system/ext/web/struts-adapter.jar
BEAHOME/weblogic81/portal/lib/netuix/system/ext/web/struts-adapter-html.tld
BEAHOME/weblogic81/portal/lib/netuix/system/ext/web/struts-adapter-naming.tld
BEAHOME/weblogic81/portal/lib/netuix/system/ext/web/struts-adapter-nested.tld
BEAHOME/weblogic81/portal/lib/netuix/system/ext/web/struts-adapter-tiles.tld
BEAHOME/weblogic81/portal/lib/wsrp/wsrp-producer.jar
BEAHOME/weblogic81/portal/lib/wsrp/adapters/wsrp-jpf-adapter.jar
BEAHOME/weblogic81/portal/lib/wsrp/adapters/wsrp-struts-adapter.jar
```

3. Redeploy the WSRP producer web application.

Other Guidelines

User sessions on Consumer web applications might be lost if session cookies between Producers and Consumers overlap. To prevent this, open `weblogic.xml`, configure your web applications to include the domain name and web application path for session cookies. Please refer to "session-descriptor" in "weblogic.xml Deployment Descriptor Elements" at:

`http://e-docs.bea.com/wls/docs81/webapp/weblogic_xml.html#1038173`

for details on how to set the domain name and path.

CHAPTER 6

Implementing Custom Data Transfer

Custom data transfer allows portlet developers to exchange arbitrary data between Producers and Consumers; for example,

- You are a portal developer building a portal with a set of location-sensitive portlets deployed on one/more producers. You would like to supply a zip code to each of these portlets at request time so that each portlet can use this zip code to generate location-aware markup.
- You want to send arbitrary data such as theme or style information or user profile data to portlets.

Custom data transfer allows you to easily resolve these situations and many others like them. Both Simple Producers and Complex Producers can take advantage of this feature.

This section describes how to implement custom data transfer. It includes information on these subjects:

- Custom Data Transfer Interfaces
- Implementing the Interfaces

Custom Data Transfer Interfaces

Custom data transfer introduces the following "marker" interfaces:

`com.bea.wsrp.ext.holders.InteractionRequestState`
: Allows the Consumer to send some arbitrary data to the Producer when an interaction (such as a form submission) is taking place.

`com.bea.wsrp.ext.holders.InteractionResponseState`
Allows the Producer to return some arbitrary data to the Consumer after an interaction took place.

`com.bea.wsrp.ext.holders.MarkupRequestState`
Allows the Consumer to send some arbitrary data to the Producer when a portlet is being refreshed.

`com.bea.wsrp.ext.holders.MarkupResponseState`
Allows the Producer to return some arbitrary data to the Producer after portlet is rendered.

Implementing the Interfaces

The following example illustrates custom data transfer using `com.bea.wsrp.ext.holders.SimpleStateHolder`. This class provides a default implementation of the above interfaces. You can use this class to exchange simple name-value pairs of data.

Implementing Interfaces in a Complex Producer: Example

To send data, you need to create an instance of this instance and place it in the request as a request attribute. The Producer/Consumer then transmits that object across the wire and make it available as a request attribute on the other end.

In the following example, we are sending a zip code to a remote portlet.

Step 1: Set Up the Environment

Set up your environment by doing the following:

Note: For instructions on how to create the components required by the following steps, "Step 1: Set Up Your Environment" on page 3-5 in chapter "Establishing Interportlet Communications with Remote Portlets".

1. Either create a domain and a portal application or use an existing domain and portal application. If you are creating a new domain and portal application, name the domain custXferDomain and the application custXfer. Be sure to point the application at custXferDomain.
2. Create two portal web applications under the portal application custXfer and name them Producer and Consumer, respectively.

Step 2: Create the Producer JSP and Portlet

With the environment in place, you will next create a JSP file on the Producer and a portlet to surface that file. This JSP file will get the state from the request. To do so, do the following:

Note: The following procedure assumes that WebLogic Workshop is running. If it isn't, launch it now.

1. Right-click Producer in the application tree to open the context menu and select New>JSP File (Figure 6-1).

Figure 6-1 Creating a New JSP File

The New File dialog box appears (Figure 6-2).

Figure 6-2 New File Dialog Box for New JSP

2. In File name, enter `zipTest.jsp` and click Create.

   The default JSP file appears, in Design View, in the middle pane of WebLogic Workshop (Figure 6-3)

Figure 6-3 Default JSP in Design View

3. Click Source View to display the JSP source (Figure 6-4).

Figure 6-4 Default JSP in Source View

4. Copy the code in Table 6-1 and replace the contents of the JSP source with it.

Listing 6-1 Code to Get State from the Request

```
<%@ page import ="com.bea.wsrp.ext.holders.SimpleStateHolder,
          com.bea.wsrp.ext.holders.MarkupRequestState"%>

<%
    SimpleStateHolder state = (SimpleStateHolder)
    request.getAttribute(MarkupRequestState.KEY);
    String zip = (String) state.getParameter("zipCode");

%>

<%=zip%>
```

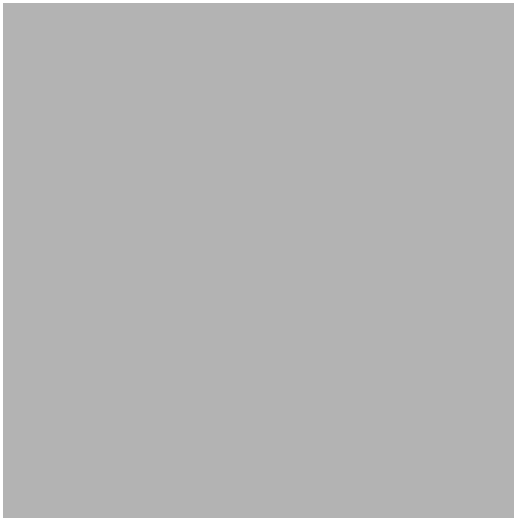
When complete, the Source View should look like that shown in Figure 6-5.

Figure 6-5 New JSP Source for zipTest.jsp

5. Save the file.
6. Right-click `zipTest.jsp` in the application tree and select Generate Portlet...

   The Portlet Details dialog box appears. Note that `zipTest.jsp` already appears in the Content URI field (Figure 6-6).

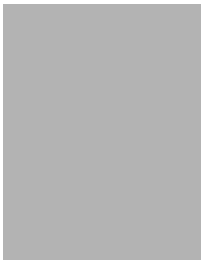
Figure 6-6 Portlet Details with zipTest.jsp Included



7. In State, select Minimizable and Maximizable and click Finish.

   The portlet `zipTest.portlet` will be created and appear in the application tree (Figure 6-7).

Figure 6-7 New JSP Portlet, zipTest.portlet, Added



Step 3: Federate zipTest.portlet to the Consumer

Next, you need to set up a remote portlet on the Consumer to surface in `zipTest.portlet` from the Producer. Use the following steps.

Note: Before you begin, start WebLogic Server by opening the tools file and selecting WebLogic Server>Start WebLogic Server.

1. Right-click Consumer in the application tree and select New>Portlet.

The New File dialog box appears.

2. In File name, enter `zipPrime.portlet` and click Create.

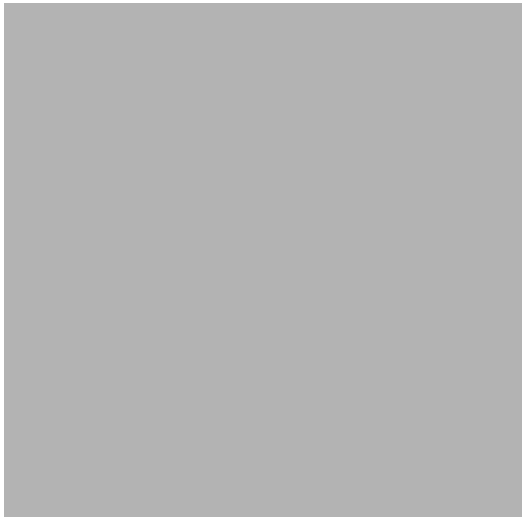
   The Select Portlet Type dialog box appears (Figure 6-8).

Figure 6-8 Select Portlet Type Dialog Box

3. Select Remote Portlet and click Next.

   The Find/Select Producer dialog box appears (Figure 6-9).

Figure 6-9 Find/Select Producer Dialog Box



4. Select Find Producer and, in the field provided, enter:

   `http://localhost:7001/Producer/producer?WSDL`

   and click Retrieve.

   After a few moments, the Find/Select Producer dialog box refreshes, displaying the Producer details, as shown in

Figure 6-10 Producer Retrieved

5. Click Register.

   The Register dialog box appears (Figure 6-11).

Figure 6-11 Register Dialog Box

6. In Producer Handle, enter BEA and leave the other fields blank.
7. Click Register.

The Find/Select Producer dialog box reappears with BEA in the Select Producer field.

8. Click Next.

   The Select Portlet from List dialog box appears (Figure 6-12).

Figure 6-12 Select Portlet from List Dialog Box

9. Select zipTest and click Next.

   The Proxy Portlet Details dialog box appears (Figure 6-13).

Figure 6-13 Proxy Portlet Details Dialog Box

10. Click Finish.

    The Proxy Portlet Details dialog box closes and WebLogic Workshop reappears, showing the design view of the new portlet (Figure 6-14).

Figure 6-14 New Remote Portlet zipPrime.portlet in Design View

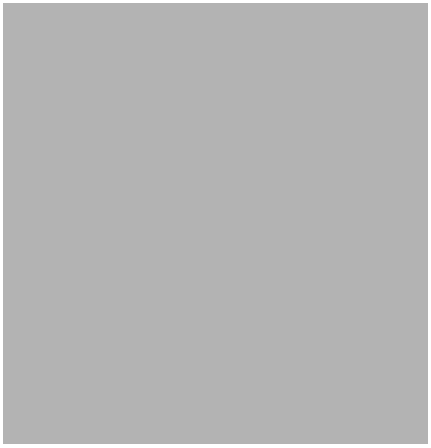
11. If necessary, save the file.

Step 4: Create and Attach a Backing File to the Consumer

In this step, you will create a backing file called `CustomDataBacking.java` on the Consumer side and attach that backing file you created in Step 3: Federate zipTest.portlet to the Consumer. Use the following procedure:

1. Expand the Consumer node to show `WEB-INF/src` and right-click it to open the context menu, and select New>Folder, as shown in Figure 6-15.

Figure 6-15 Creating a New Folder in WEB-INF/src



The Create New Folder dialog box appears (Figure 6-16).

Figure 6-16 Create New Folder Dialog Box

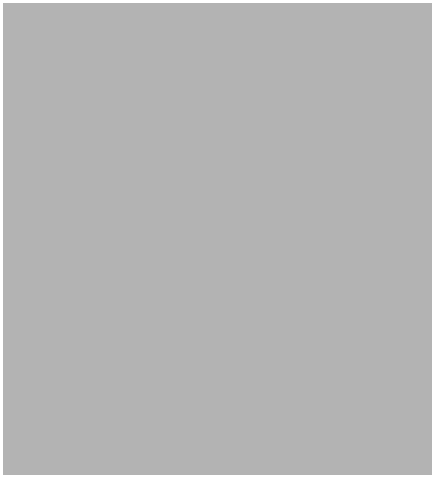
2. In Enter new folder name, type backing and click OK.

   The `backing` folder will appear in the application tree under `WEB-INF/src`, as shown in Figure 6-17.

Figure 6-17 backing Folder Under WEB-INF/src

3. Right-click backing to open the context menu and select New>Java Class (Figure 6-18).

Figure 6-18 Creating a New Java Class



The New File dialog box appears.

4. In File name, enter `CustomDataBacking.java` and click Create.

   A backing file template opens in WebLogic Workshop (Figure 6-19).

Figure 6-19 Backing File Template

5. Copy the code in Table 6-2 into the template and save the file.

Listing 6-2 Adding an Instance of SimpleStateHolder

```
package backing;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking;
import com.bea.wsrp.ext.holders.MarkupRequestState;
import com.bea.wsrp.ext.holders.SimpleStateHolder;

public class CustomDataBacking extends AbstractJspBacking
{

   public boolean preRender(HttpServletRequest request,
      HttpServletResponse response)
   {
      SimpleStateHolder state = new SimpleStateHolder();
      state.addParameter("zipCode", "80501");
      request.setAttribute(MarkupRequestState.KEY, state);
```

```
        return true;
    }
}
```

The template should now look like the example in Figure 6-20.

Figure 6-20 CustomDataBacking.java in WebLogic Workshop

6. Double-click `zipPrime.portlet` to display it in WebLogic Workshop.
7. Add the backing file to `zipPrime.portlet` by typing `backing.CustomDataBacking` in the Backing File field in the Property Editor, as illustrated in Figure 6-21 and press Tab.

Figure 6-21 Adding a Backing File

Step 5: Test the Application

With the application components in place, you can now test the application. If the test is successful, the zip code 80501, provided by the backing file created on the Consumer, will appear in the portlet when it is rendered.

To test the application, do the following:

1. Right-click Consumer in the application tree to open the context menu and select New>Portal.

   The New File dialog box appears.

2. In File name, enter `zipTest.portal` and click Create.

   The portal is created and appears in the WebLogic Workshop design view (Figure 6-22).

Figure 6-22 zipTest.portal in WebLogic Workshop Design View

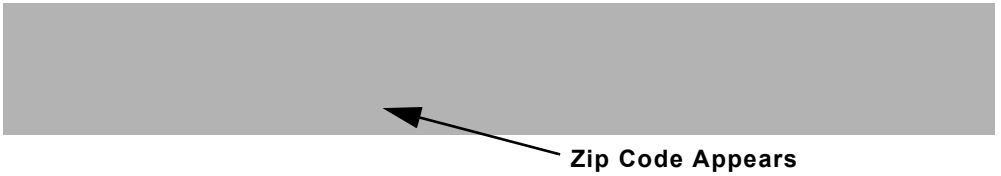
3. Drag zipTest from the data palette into the portal (you can place it in either placeholder; in Figure 6-23, it is in the right-hand placeholder).

Figure 6-23 zipTest.portlet Added to zipTest.portal

4. Save the portal.
5. Open the Portal menu and select Open Current Portal.

   A browser opens, displaying the portal (this might take a few moments). Note the zip code 80501 appearing in the portlet (Figure 6-24).

Figure 6-24 zipTest.portal Successfully Rendered



Using this Example in a Simple Producer

The procedure described in Implementing Interfaces in a Complex Producer: Example applies only to complex producers. If you want to use the preceding example in a simple producer, you

must make a few modifications. To implement the interfaces in a simple Producer, use this procedure:

1. In the application custXfer, create a new project; however, instead of a Portal Web Project, make this project a simple Web Project and call it simpleProducer. See Step 1: Set Up the Environment for instructions.
2. Right-click simpleProducer in the application tree to open the context menu and select Install>WSRP Producer (Figure 6-25.

Figure 6-25 Installing a WSRP Producer

The application tree will update and show `wsrp-producer-config.xml` under the `simpleProducer/WEB-INF` directory.

3. In simpleProducer create a Java Page Flow by doing the following:
   a. Right-click simpleProducer to open the context menu and select New>Page Flow.

      The Page Flow Wizard - Page Flow Name dialog box appears (Figure 6-26).

Figure 6-26 Page Flow Wizard - Page Flow Name Dialog Box

b. In Page Flow Name, enter ZipTestPF. Note that Controller Name becomes `zipTestController.jpf`.

c. Click Next.

The Page Flow Wizard - Select Page Flow Type dialog box appears (Figure 6-27).

Figure 6-27 Page Flow Wizard - Select Page Flow Type Dialog Box

d. Select Basic page flow and click Create.

   WebLogic Workshop reappears, showing the basic page flow in the Flow View (Figure 6-28).

Figure 6-28 Basic Page Flow in Flow View

Note that the page flow now includes the file `index.jsp`.

4. Double-click the `index.jsp` icon.

   `index.jsp` opens in the Design View.

5. Click Source View to display the `index.jsp` source (Figure 6-29).

Figure 6-29 index.jsp Source View

6. Copy the code in Listing 6-3 into `index.jsp`, overwriting its entire default content.

Listing 6-3 Using a Page Flow Portlet for a Simple Producer

```
<%@ page import="com.bea.wlw.netui.pageflow.scoping.ScopedRequest"%>
<%@ page import ="com.bea.wsrp.ext.holders.SimpleStateHolder,
                  com.bea.wsrp.ext.holders.MarkupRequestState"%>
<%
    SimpleStateHolder state = (SimpleStateHolder)
    ((ScopedRequest) request).getOuterRequest().
    getAttribute(MarkupRequestState.KEY);

    String zip = (String) state.getParameter("zipCode");
%>
<%=zip%>
```

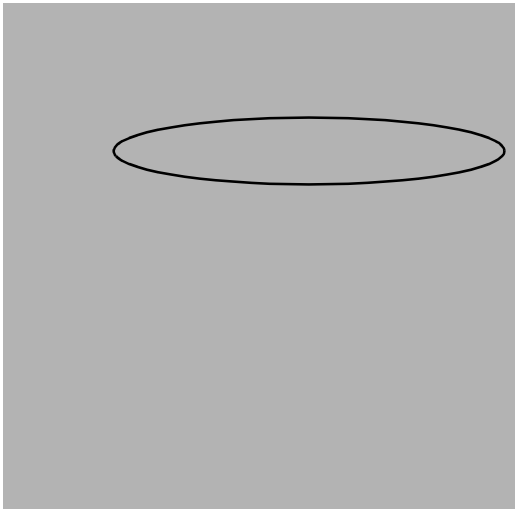
The Source View for `index.jsp` should now look like Figure 6-30.

Figure 6-30 Updated index.jsp in WebLogic Workshop

7. Save the file.
8. Right-click `ZipTestPFController.jpf` in the application tree to open the context menu and select Generate Portlet...

   The Portlet Details dialog box appears (Figure 6-31).

Figure 6-31 Portlet Details for a JPF Portlet

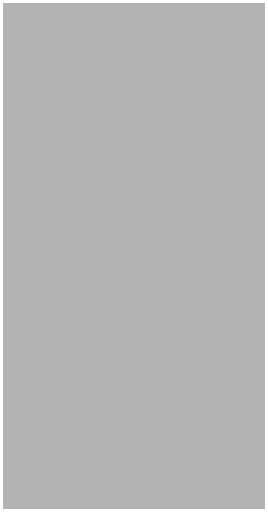


Note that `/zipTestPF/ZipTestPFController.jpf` has already been entered in the Content URI field.

9. Select Minimizable and Maximizable and click Finish.

   The portlet is created; note that `ZipTestPFController.portlet` now appears under `simpleProducerWeb/WEB-INF(`Figure 6-32`)`.

Figure 6-32 ZipTestPFController Added to Application Tree



10. Federate `ZipTestPFController.portlet` to the Consumer portal web application following the steps specified above in Step 3: Federate zipTest.portlet to the Consumer. Make the following changes within the procedure:

    Note: Ensure that WebLogic Server is running.

    – File name: simpleProducer

    – Find Producer: `http://localhost:7001/simpleProducer/producer?WSDL`

      Note: Registration is not required (Requires Registration=false) since you are federating a portlet from a simple producer; therefore, you don't need to open the Registration dialog box as you would for a complex producer.

    – Select Portlet from List: ZipTestPFController

    – Producer Handle: USPS

11. Attach the backing file `CustomDataBacking.java` by typing `backing.CustomDataBacking` in the Backing File field in the Property Editor. Press Tab.

12. Save the portlet file.

13. Test the application by doing the following:

    a. Under the Consumer web application, open `zipTest.portal`.

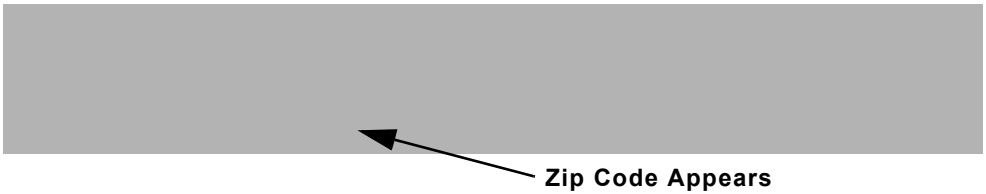
    b. Drag `ZipTestPFController.portlet` from the Data Palette to `zipTest.portal`.

c. Save the portal file.

d. Open the Portal menu and select Open Current Portal.

A browser will open and, after a few moments, the portal will appear, showing the zip code 80501, as shown in (Figure 6-33).

Figure 6-33 zipTest.portal in a Browser



Deploying Your Own Interface Implementations

If you want to deploy your own implementations of these interfaces, consider the following practices:

- The implementation must be serializable.
- The same class version of the implementation must be deployed on both the Producer and Consumer. If the versions are different, the implementations must make sure to have the same `serialVersionUID` for all versions.
- Sending large amounts of data may have performance implications.

Implementation Rules

Whether a Consumer or Producer can send custom data depends on the type of request. These rules apply:

- Consumers can always send `InteractionRequestState`. There are no exceptions.
- Producers can always return `InteractionResponseState`. There are no exceptions.
- Consumers can send `MarkupRequestState` only when there is a need to refresh the portlet. For example, if caching is enabled on the remote portlet, Consumer may not always send a request to the Producer to generate markup.
- Consumers cannot return `MarkupResponseState` if any the following options are enabled:

- Returning markup as an attachment
- Local proxy

In both the cases, the Producer invokes the portlet (typically JSPs) after creating the SOAP response, which is too late to update the SOAP response.

CHAPTER 7

Local Proxy Support

Local proxy support allows co-located Producer and Consumer web applications to short-circuit network I/O and "SOAP over HTTP" overhead. When you enable this feature, the Consumer tries to determine if the Producer is deployed on the same server and, if it discovers that the Producer is so deployed, it uses a local proxy to send requests to the Producer. If the producer is not deployed on the same server, the Consumer uses the default remote proxy. Remote Producers can still be invoked as usual even when the local proxy support is enabled.

This section describes how to implement local proxy support. It includes information on the following subjects:

- Why Use Local Proxy Mode?
- Deployment Configuration
- When to Use and Not Use

Why Use Local Proxy Mode?

Local proxy mode provides a number of advantages over the default remote proxy when you are working with co-located web applications. Among the most significant are that it:

- Avoids local network I/O.
- Avoids serialization and deserialization of SOAP.
- Invokes remote portlets using the same execute thread.
- Writes portlet markup directly to the response without intermediate buffers.

- Enables large file uploads without causing `OutOfMemoryErrors`.

Additionally, by enabling local proxies, customers can take advantage of the decoupling benefits of WSRP without incurring its performance overhead.

Deployment Configuration

To take advantage of local proxy support, do the following:

1. Deploy the Producer and Consumer web applications on the same server. These applications could be in the same enterprise application or across different enterprise applications.
2. Enable local proxy support by setting `<enable-local-proxy>` to `"true"` in `WEB-INF/wsrp-producer-registry.xml` in the Consumer web application, as shown in Listing 7-1:

Listing 7-1 Setting `<enable-local-proxy>` to `"true"`

```
<wsrp-producer-registry
   xmlns="http://www.bea.com/servers/weblogic/wsrp-producer-registry/8.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bea.com/servers/weblogic/wsrp-producer-
      registry/8.0 wsrp-producer-registry.xsd">

   <!-- Upload limit (in bytes) -->
   <upload-read-limit>1048576</upload-read-limit>

   <!-- Timeout (in milli seconds) -->
   <connection-timeout-secs>120000</connection-timeout-secs>

   <!-- Enable local proxy -->
   <enable-local-proxy>true</enable-local-proxy>

...
</wsrp-producer-registry>
```

You can also enable local proxy support by setting a system property `com.bea.wsrp.proxy.LocalProxy.enabled = true`. If this system property is set to true, it will override the `<enable-local-proxy>` setting in `WEB-INF/wsrp-producer-registry.xml`.

Local proxy support is disabled by default in web application templates.

When to Use and Not Use

As powerful a tool as local proxy support is, you should only use it when it will benefit your application. The most common reasons for using local proxy support are:

- When portlets are deployed in self-contained web applications on the same server. The local proxy support provides isolated portlet deployment. In this mode, each portlet web application can be deployed as a WSRP producer. Portlets can therefore be loaded by separate class loaders and have their own servlet context and session. Portlet web applications can de deployed/undeployed without affecting the portal web application.
- When you don't need advanced monitoring software between the Producer and Consumer

On the other hand, you *shouldn't* use local proxy support when interoperating with non-BEA Producers and Consumers.

CHAPTER 8

Monitoring and Logging Remote Portlet Performance

You can monitor activity between Producers and Consumers by using the message monitor servlet installed with WebLogic Workshop. You can also create custom logs to display specific information about WSRP sessions.

This section contains information on these subjects:

- Monitoring Producer/Consumer Message Logs
- Creating Custom Logs

Monitoring Producer/Consumer Message Logs

By default, the message monitor is enabled in the `web.xml` file, as shown in Listing 8-1

Listing 8-1 Enabling Message Monitor in web.xml

```
<!-- WSRP Message Monitor Servlet -->
<servlet>
   <servlet-name>com.bea.wsrp.logging.MessageMonitor</servlet-name>
   <servlet-class>com.bea.wsrp.logging.MessageMonitor</servlet-class>
   <init-param>
      <param-name>enableSoapMessageLogging</param-name>
      <param-value>true</param-value>
   </init-param>
```

```
      <load-on-startup>1</load-on-startup>
</servlet>
```

You easily monitor the messages regarding Producer/Consumer interaction by viewing output of this servlet. To do, use this procedure:

1. Ensure that a WSRP session running (that is, a Consumer portlet is surfacing data from a Producer).
2. Open a new browser.
3. In the new browser's address bar, type the following:

   <host_name>:*<port_number>*/*<webProject_name>*/monitor

   Where:

   – *<host_name>*:*<port_number>* is the host and port you want to monitor.

   – *<webProject_name>* is the web project you want to monitor.

   For example:

   `localhost:7001/wsrpMonitorTest/monitor`

   The Monitor appears in the browser, as shown in Figure 8-1.

Figure 8-1 Monitor Appearing in a Browser

Each time the remote portlet communicates with the Producer, a request-response message header appears on the monitor screen (Figure 8-2).

Figure 8-2 Monitor Message

– By clicking Show, you can display the content of the request or the response (Figure 8-3).

Figure 8-3 Message Content

– Click Hide to close the message content.

Creating Custom Logs

You can create custom logs that display particular information about a WSRP session by using the WebLogic Server `loggers` and `handlers`. These objects allow you to create your own message handlers and subscribe them to the WebLogic Server `Logger` objects; for example, if you want the remote portlet to listen for the messages that the Producer generates, you can create a `handler` and subscribe it to the `Logger` object in the Producer.

`loggers` and `handlers` are WebLogic Server objects. For instructions on using them to create custom logs for your WSPR Consumers and Producers, please refer to the WebLogic Server documentation site.

CHAPTER 9

Establishing WSRP Security

The WSRP standard does not enforce any specific security standard at this time; however, it does recommend that you follow security standards such as WS-Security and SAML when implementing WSRP-compliant portlets. The WSRP standard does emphasize using transport-level security standards, such as SSL/TLS, to address the security issues involved in Consumers invoking Producers on behalf of end-users. These security standards only require that a Producer's WSDL declare ports for an HTTPS service entry point. Consumers can only determine that secure transport is supported by parsing the URL for the service entry point access control.

This section describes some of the security measures we suggest you follow. It contains information on the following subjects:

- Access Control
- Security Recommendations
- Secure WSRP Messages
- Manage User Identity
- Establishing Single Sign-on with Remote Portlets: Example
- Securing the WebLogic Administrator's Logon Information

Access Control

Both Producers and Consumers can control access by using the implemented security measures.

- Consumers can restrict end-users access to portlets and to specific operations on those portlets.
- Producers can implement access control programmatically through the use of facilities such as an authenticated user identity.

Security Recommendations

While the WSRP standard does not specify security requirements, the following recommendations serve as guidelines that will ensure secure implementation of your WSRP-compliant portlets:

- Setting Security Constraints on Resources
- Secure WSRP Messages
- Manage User Identity
- Securing the WebLogic Administrator's Logon Information

Setting Security Constraints on Resources

As a basic rule and best practice, administrators should secure all web applications and their resources running in a production environment. Security is especially important when resource URLs (links to files and images on a producer) are used with WSRP. Per the WSRP 1.0 specification, resource URLs are absolute and become part of the final URL, which is visible in the `wsrp-url` parameter.

When you view portal page's source in a browser, these resource URLs are visible. Because they provide the URL back to the hosting web application on the producer machine, it is important that all resources on the producer are secure. For example, using this information, a user could guess URLs on the producer, such as the WebLogic Server Console, and attempt to access them directly.

Warning: It is the responsibility of the producer's administrator to secure web applications and resources appropriately.

For information on adding security constraints to a WebLogic Server, see "URL (Web) and EJB (Enterprise JavaBean) Resources" and "security-constraint" at:

`http://e-docs.bea.com/wls/docs81/secwlres/types.html#1208206`

`http://e-docs.bea.com/wls/docs81/webapp/web_xml.html#1017885`

Creating a Resource Connection Filter

As discussed in the previous section, the `wsrp-url` parameter exposes resource URLs on the producer machine.

By default, a consumer allows access to URLs hosted on the machines that host WSRP producers that are currently known to the consumer. Such producers are ones added to the consumer with WebLogic Workshop, the Portal Administration Portal, or WebLogic Portal APIs. If you want to override this default behavior, do the following:

1. On the consumer, create a class that implements the interface `com.bea.wsrp.consumer.resource.ResourceConnectionFilter`. This interface includes one method:

   ```
   public boolean allowedURL(String url);
   ```

   Implement this method to return `true` only for URLs that you want to allow access to through the `wsrp-url` parameter.

   Place the implementation class file in `WEB-INF/lib/classes`. For example, `WEB-INF/lib/classes/MyResourceProxyServletImpl.class`.

2. Register the class with the ResourceProxyServlet on the consumer. To do this, add an `<init-param>` element to the ResourceProxyServlet definition in the `WEB-INF/web.xml` file of the consumer web application. You must set the `<param-name>` to `resourceConnectionFilter`, and the `<param-value>` to the fully qualified classname of your ResourceConnectionFilter implementation class.

   Listing 9-1 shows a sample ResourceProxyServlet servlet definition.

Listing 9-1 Configuring ResourceProxyServlet (in WEB-INF/web.xml)

```
<servlet>
   <servlet-name>ResourceProxyServlet</servlet-name>
   <servlet-class>com.bea.wsrp.consumer.resource.ResourceProxyServlet</servlet-class>
   <init-param>
      <param-name>resourceConnectionFilter</param-name>
      <param-value>myClasses.MyResourceProxyServletImpl</param-value>
   </init-param>
</servlet>
```

Secure WSRP Messages

Securing WSRP messages ensures their confidentiality between just the interested parties. When a portlet's messaging is secure, only parties authorized to handle the contents of that portlet's messages can see those messages. To secure WSRP messages:

- Use SSL on any port through which the Producer will be offered.
- Configure the Producer to offer secure portlets by specifying `"true"` for all secure attributes in the `<service-config>` element of the Producer project's `WEB-INF/wsrp-producer-config.xml` file, as shown in Listing 9-2.

Listing 9-2 <service-config> Element Configured for Security

```
<service-config>
   <registration required="true" secure="true"/>
   <service-description secure="true"/>
   <markup secure="true" rewrite-urls="true" transport="string"/>
   <portlet-management required="true" secure="true"/>
</service-config>
```

Note: If you make any changes to `wsrp-producer-config.xml`, you will need to redeploy or bounce the server before the changes become active.

Manage User Identity

To manage user identity:

- Rely on single-sign-on (SSO), which is set up by default in WebLogic Portal.
- Let users login to the Consumer portal. WebLogic Portal will manage SSO automatically.

What is Single Sign-on?

Single sign-on (SSO) is mechanism whereby a single action of user authentication and authorization can permit that user to access all computers and systems to which access permission has been granted, without having to enter multiple passwords. Single sign-on reduces human error, a major component of systems failure and is therefore a feature.

How Single Sign-on Works with WSRP

SSO support is meant to propagate user identity from the consumer to the producer. That is, the consumer provides authentication, and the WSRP stack makes sure that the same user identity is established on each producer with which the user is interacting. The best way to see this in action is to monitor the SOAP message log on the producer or consumer side. After login (on the consumer side), you will see an extra SOAP header in each request.

Each web application has a monitor which logs soap messages. The monitor can be viewed by going to `http://<machine>:<port>/<webapp>/monitor`. For more information on monitoring SOAP messages, please refer to Monitoring Producer/Consumer Message Logs at:

`http://edocs.bea.com/wlp/docs81/wsrp/monlog.html#998933`

The Signed Certificate

Producers authenticate Consumers through the use of client certificates in conjunction with SSL/TLS. Therefore, if you are relying on SSO and allow users to log-in to the Consumer portal, as recommended, the Producer must trust that Consumer. To establish this trust, the Consumer needs a certificate of authentication signed by an approved certificate authority (CA), such as VeriSign, Inc. This section introduces the Java keytool utility that is used to generate a self-signed certificate. Later in this chapter, we use the keytool in a detailed example that includes obtaining and using a signed certificate from a CA.

The Java keytool Utility

When you install WebLogic Platform, part of the installation process installs a Java runtime environment (JRE). Within the JRE you will find a utility call `keytool.exe`. keytool is a key and certificate management utility with which you can administer your own public/private key pairs and associated certificates to use in self-authentication (where the user authenticates himself/herself to other users/services) or data integrity and authentication services by using digital signatures.

keytool Concepts and Terminology

You should be familiar with the following terms when implementing security for WSRP-compliant portlets:

certificate
Also known as a public-key certificate—a digitally signed statement from one entity (the issuer), saying that the public key (and some other information) of another entity (the subject) has some specific value.

Certificate Authority (CA)
An organization, such as VeriSign, Inc. that will accept a CSR and return to the requestor a certificate or certificate chain.

Certificate chain
A set of certificates used to establish trust back to a common certificate authority. The first certificate in the chain contains the public key corresponding to the private key.

Certificate Signing Request (CSR)
A file that is sent to a certificate authority, who will authenticate the certificate requestor (usually offline) and return to the requestor a certificate or certificate chain, used to replace the existing certificate chain (which initially consists of a self-signed certificate) in the keystore.

key pair
The combination of a public key and private key on the same certificate

keystore
A database of private keys and their associated X.509 certificate chains that are used to authenticate the corresponding public keys. A keystore file has the `.jks` extension. The keystore is scoped to the domain, so you will find it in the `<BEA_HOME>/user_projects/domains/`*`specificDomain`* folder (where *`specificDomain`* is the domain to which the application you want to secure points). WebLogic Platform ships with a default keystore called `wsrpKeystore.jks`. *We strongly recommend that you rename this file for each domain in which you plan to run it.* Otherwise, application security will be compromised.

Self-signed certificate
A certificate for which the issuer is the same as the subject (the entity whose public key is being authenticated by the certificate). When `-genkey` generate a new public/private key pair, it wraps the public key into a self-signed certificate.

keytool Reference

keytool was created by Sun Microsystems. For complete information on this utility, please refer to keytool - Key and Certificate Management Tool at:

`http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html`

Secure the /producer Path

By default, the Producer servlet is not protected. In order to restrict access to a Producer, protect the path `<webAppPath>/producer` at the network or firewall level (where *webAppPath* is the URL of the web application).

Establishing Single Sign-on with Remote Portlets: Example

The following example will show you how to establish SSO from a Consumer to a Producer. In this simple example, you will be able to log in to a remote portlet from a local portlet on a Consumer.

In this example, you will do the following:

1. Set up an environment (domain, portal application, and portal web applications) in which to run the example.
2. Create a login portlet and establish SSO with a remote portlet by using the signed certificate supplied with BEA WebLogic Portal 8.1
3. Modify this out-of-the-box keystore file to prevent successful login to the same remote portlet.
4. Use the Java keytool utility to create a new keystore (this utility is described in The Java keytool Utility).
5. Use the keytool utility to request and obtain a signed certificate for that keystore from a certification authority (CA).
6. Add that signed certificate to both the Consumer (you) and the Producer.
7. Reestablish SSO with the remote portlet by using the new signed certificate.

Before you attempt this exercise, please read the preceding section, Manage User Identity.

Step 1. Set Up the Environment

To set up the necessary environment for this example, please refer to Step 1: Set Up Your Environment in Establishing Interportlet Communications with Remote Portlets. Follow the instructions in that section explicitly. If you already completed the environment setup described in Step 1: Set Up Your Environment, you do not need to complete this step.

Note: We recommend that you use the same environment from Establishing Interportlet Communications with Remote Portlets to avoid cluttering your

<BEA_HOME>/user_projects folder. If you want to complete this exercise in a new domain and/or portal application, be sure to substitute the names you select for those components in the following procedures.

Step 2. Create the Login Portlet and Establish SSO with a Remote Portlet

In this step, you will create a log-in portlet on the Consumer using the BEA-supplied log-in controller page flow and a simple JSP portlet on the Producer. You will then federate the JSP portlet to the Consumer. Next, you will attempt to log in to the remote portlet by using the log-in portlet.

Create the Log-in Page Flow Portlet

1. Launch WebLogic Workshop and open the application ipcWsrpTest.
2. Right-click consumerWeb to open the context menu and select New>Page Flow.

   The Page Flow Wizard - Page Flow Name dialog box appears (Figure 9-1).

Figure 9-1 Page Flow Wizard - Page Flow Name Dialog Box

3. In Page Flow Name, enter login.

   Note that Controller Name changes to LoginController.jpf.

4. Click Next

   The Page Flow Wizard Page - Flow Type dialog box appears (Figure 9-2)

Figure 9-2 Page Flow Wizard - Page Flow Type Dialog Box

5. Select Page Flow from a java control and, in the java Control list, select User Login.
6. Click Next.

   The Page Flow Wizard - Select Action dialog box appears (Figure 9-3)

Figure 9-3 Page Flow Wizard - Select Action Dialog Box

7. Click Select All and then click Create.

   The page flow file is created. LoginController.jpf will appear under consumerWeb on the application tree and the page flow schematic will appear in the IDE workspace (Figure 9-4).

Figure 9-4 LoginController.jpf Page Flow in IDE Workspace

8. Click Source View to display the page flow code (Figure 9-5)

Figure 9-5 LoginController.jpf Source

9. In the `public Forward login(LoginForm aForm)` call, locate the string:

```
com.bea.p13n.usermgmt.profile.ProfileWrapper var = myControl.login
  (aForm.username, aForm.password, aForm.request );
```

and replace it with:

```
com.bea.p13n.usermgmt.profile.ProfileWrapper var = myControl.login
   (aForm.username, aForm.password, super.getRequest() );
```

The call should now look like the example in Listing 9-3

Listing 9-3 Updated Forward login(LoginForm aForm) Call

```
 public Forward login(LoginForm aForm)
        throws Exception
    {
        com.bea.p13n.usermgmt.profile.ProfileWrapper var =
```

```
        myControl.login(aForm.username, aForm.password,
            super.getRequest() );
      getRequest().setAttribute("results", var);

        return new Forward("success");
    }
```

10. Save the file.

    `LoginController.jpf` will appear under consumerWeb/login in the application tree (/login is created when you save LoginController.jpf; Figure 9-6).

Figure 9-6 LoginController.jpf Under consumerWeb/login

11. Right-click `LoginController.jpf` in the application tree to open the context menu and select Generate Portlet...

    The Portlet Wizard - Portlet Details dialog box appears (Figure 9-7).

Figure 9-7 Portlet Wizard - Portlet Details Dialog Box for LoginController.jpf

Note that `LoginController.jpf` already appears in the Content URI field.

12. Click Finish.

    The portlet is generated; `LoginController.portlet` appears under `consumerWeb/login` in the application tree.

13. Save all files.

Create a Log-in Portal

In this step, you will create a portal to contain the login portlet you built in Create the Log-in Page Flow Portlet.

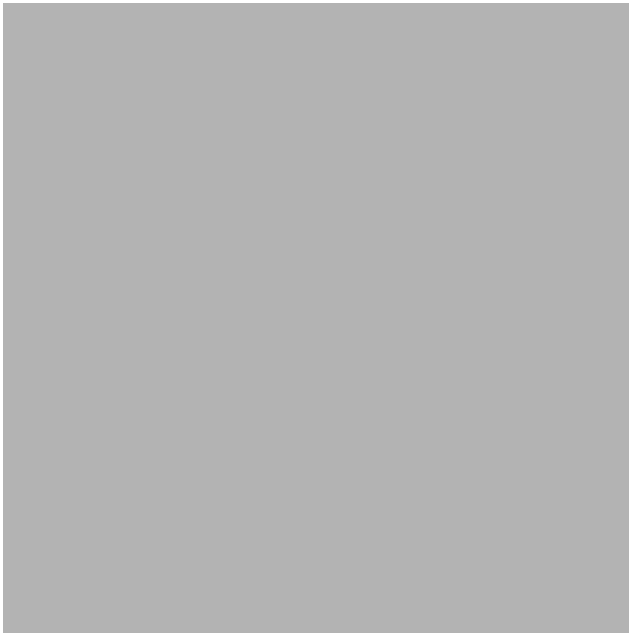
1. Right-click the consumerWeb node of the application tree to open the context menu and select New>Portal.

   The New File dialog box appears.

2. Ensure the Portal is selected in both panes and, in File Name, enter login and then click Create.

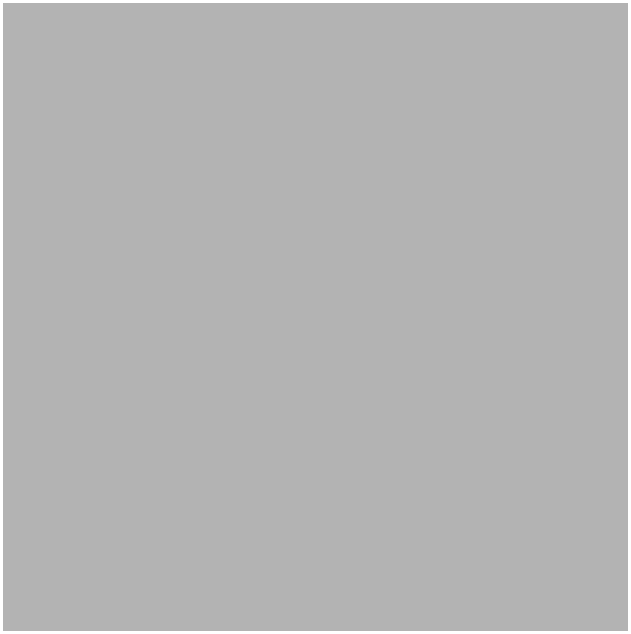
   `login.portal` appears under `consumerWeb/login` in the application tree and the portal layout appears in the WebLogic Workshop workspace (Figure 9-8).

Figure 9-8 login.portal in WebLogic Workshop Workspace



3. Drag `LoginController.portlet` from the Data Palette onto the left-hand column of the portal layout, as shown in Figure 9-9.

Figure 9-9 LoginController.portlet in Portal Layout



4. Save the portal.

Create a Portlet on the Producer

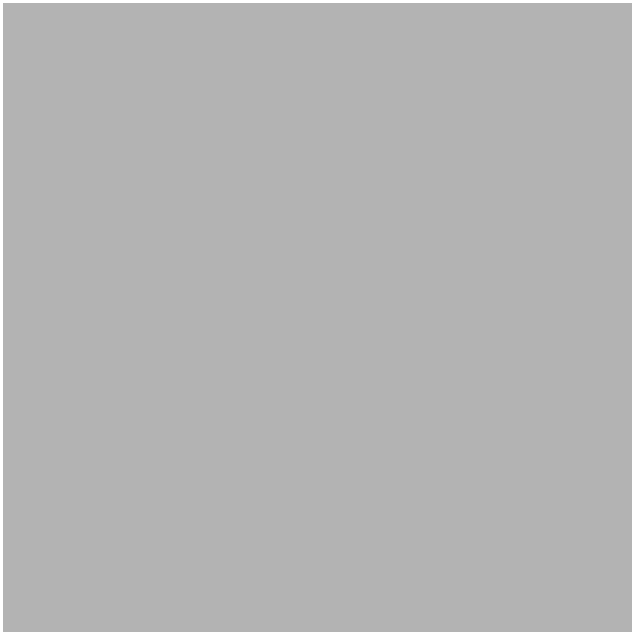
Next, create a JSP file and portlet on the Producer. This is the portlet you will federate to the Consumer portal you created in Create a Log-in Portal and will be the portlet you will attempt to log in to when you test the login portal.

1. To create the portlet on the Producer, do the following:
2. Right-click producerWeb and select New>JSP File

   The New File dialog box appears.
3. Ensure that Web User Interface and JSP File are selected in the left and right-hand panes, respectively; in File name, enter `cPortlet.jsp` and click Create.

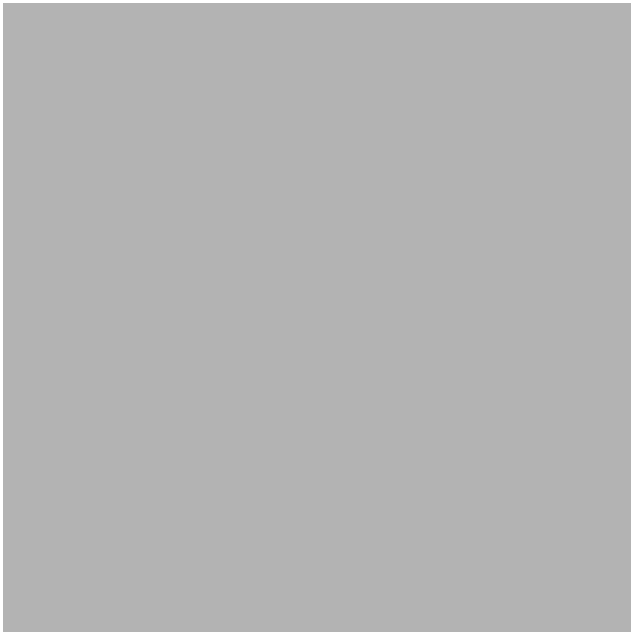
   `cPportlet.jsp` appears in the WebLogic Workshop workspace (Figure 9-10).

Figure 9-10 cPortlet.jsp in Design View



4. Click Source View to display the JSP code for cPortlet.jsp (Figure 9-11).

Figure 9-11 cPortlet in Source View



5. Copy the code from Listing 9-4 and replace the existing `cPortlet.jsp` source code with it.

Listing 9-4 cPortlet.jsp Source

```
<%
   String username=null;
      if(request.getUserPrincipal() !=null ){
      username=request.getUserPrincipal().getName();
   }
%>
Username = <%=username%>
```

The WebLogic Workshop workspace will look like the example in Figure 9-12.

Figure 9-12 New cPortlet.jsp Source

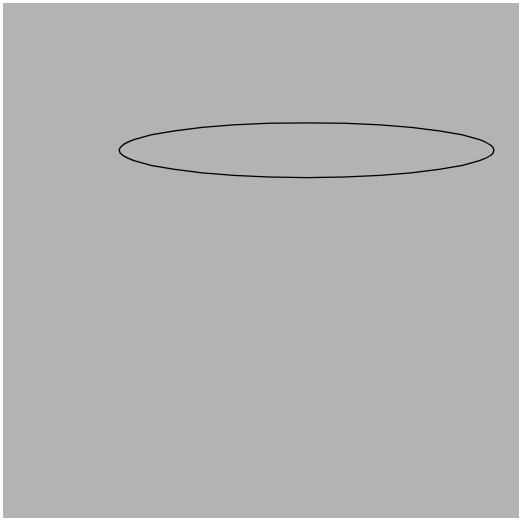
6. Save the file.

   `cPortlet.jsp` appears in the application tree under producerWeb.

7. Right-click `cPortlet.jsp` in the application tree to display the context menu and select Generate Portlet...

   The Portlet Details dialog box for `cPortlet.jsp` appears (Figure 9-13).

Figure 9-13 Portlet Details Dialog Box for cPortlet.jsp



Note that `cPortlet.jsp` appears in the Content URI dialog box.

8. Click Finish.

   `cPortlet.portlet` appears under producerWeb in the application tree.

Federate the Producer Portlet to the Consumer

Next, you will surface the JSP portlet you created in Create a Portlet on the Producer to the Consumer. To do so, use this procedure.

Note: Ensure that WebLogic Workshop is running.

1. If it is not already running, open the Tools menu and select WebLogic Server>Start WebLogic Server.
2. Right-click consumerWeb and select New>Portlet.

   The New File dialog box appears.

3. Ensure that Portal and Portlet are selected; in File name, enter cPrime and click Create.

   The Select Portlet Type dialog box appears.

Figure 9-14 Select Portlet Type Dialog Box

4. Select Remote Portlet and click Next.

   The Find/Select a Producer dialog box appears (Figure 9-15), with Find Producer selected.

Figure 9-15 Find/Select a Producer Dialog Box

5. In the Find Producer field, enter:

   `http://localhost:7001/producerWeb/producer?WSDL`

   and click Retrieve.

   The dialog box refreshes, showing Producer Details.

6. Click Register.

   The Register dialog box appears.

7. In Producer Handle, enter ssoTest and click Register.

   The Find/Select Producer dialog box reappears.

8. Click Next.

   The Select Portlet from List dialog box appears (Figure 9-16).

Figure 9-16 Select Portlet from List Dialog Box

9. From the list, select cPortlet and click Next.

   The Proxy Portlet Details dialog box appears.

10. Click Finish.

    `cPrime.portlet` will appear under consumerWeb on the application tree and the portlet layout will appear in the WebLogic Workshop workspace (Figure 9-17).

Figure 9-17 cPrime Portlet Layout

11. Save the file.

Test the Log-in Portlet

Finally, you need to test the log-in portlet you created in Create the Log-in Page Flow Portlet to ensure that you can log in to a remote portlet. Once that test is successful, you will "break" the application, which will prevent you from logging in to the remote portlet.

Note: Ensure that WebLogic Server is running.

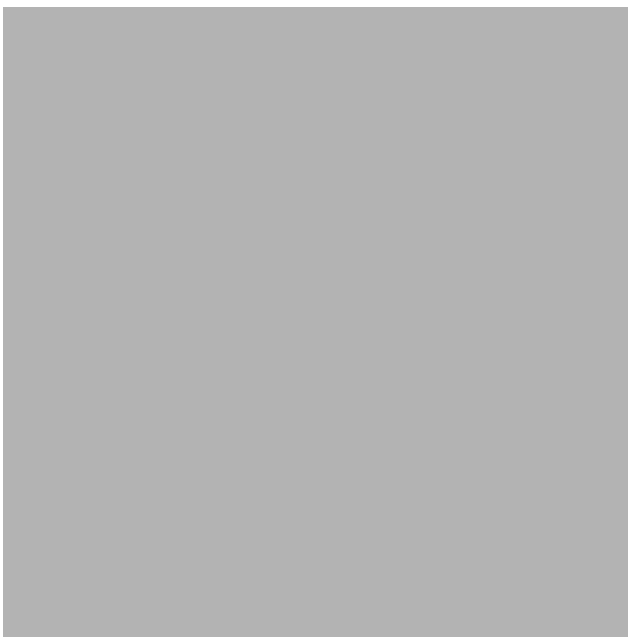
To run this test, do the following:

1. Double-click `login.portal` in `consumerWeb`.

   The portal layout appears in the WebLogic Workshop workspace.

2. Drag cPortlet from the Data Palette into the right-hand placeholder of the portal (Figure 9-18). cPortlet is the name that `cPrime.portlet` is identified by in the Data Palette. If the Data Palette is not visible, select View>Windows>Data Palette.

Figure 9-18 login.portal with cPortlet Added



3. Save the portal.

4. Open the Portal menu and select Open Current Portal.

   After a few moments, the portal will appear in a browser (Figure 9-19).

Figure 9-19 login.portal Rendered in the Browser

Note that cPortlet displays the text Username = null.

5. In LoginController, click Login.

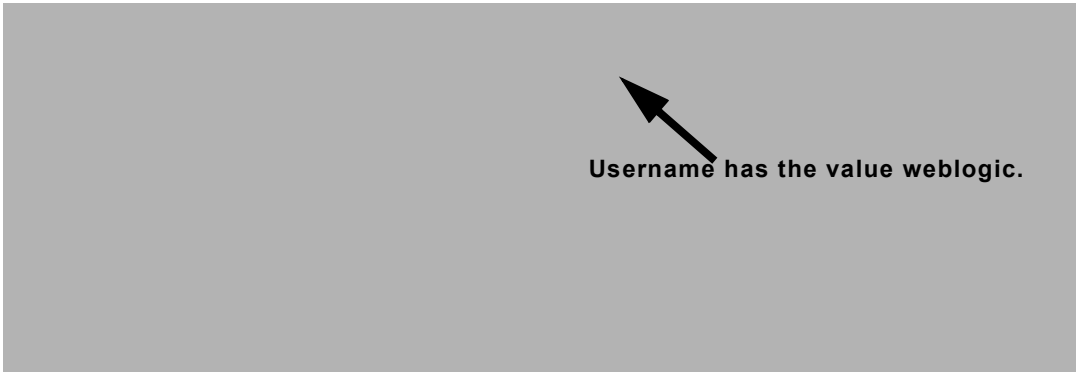
   LoginController refreshes to show log-in fields (Figure 9-20).

Figure 9-20 Log-in Fields on LoginController.portlet

In both Password and Username, enter weblogic and click Submit.

The portal refreshes; cPortlet will now display Username = weblogic.

Figure 9-21 login.portal After Login is Successful



6. Close the portal.
7. Stop WebLogic Server.

Summary

In this step, you created all components required to build a portal that will allow you to log in to a portlet that has been federated to that portal from a producer. You then assembled a log-in portal using out-of-the-box resources, rendered that portal in a browser, and logged in to the remote portlet. In the next step, you will "break" this portal.

Step 3: Break the Log-in Portal

In this step, you will rename the keystore file, which will prevent login to the remote portlet on `login.portal`. Next, you will verify that you cannot log in to the portal by attempting and failing to successfully log in.

Rename the .jks File

To rename the `.jks` file, do the following:

Note: Ensure that WebLogic Workshop is running.

1. Open your file system (for example, Windows Explorer) and navigate to and display the contents of:

   `<BEA_HOME>/user_projects/domains/ipcWsrpTest`

2. In the list of files, locate `wsrpKeystore.jks`.

3. Rename `wsrpkeystore.jks` to `wsrpkeystore_old.jks`.

Retest the Portal

Now, you will launch the log-in portal and attempt to log in to the remote portlet again. To do so, do the following:

1. Return to WebLogic Workshop and restart Weblogic Server.

   Note: The application ipcWsrpTest should be open.

2. Under consumerWeb, double-click login.portal to open that portal in WebLogic Workshop (if `login.portal` is already open when you launched WebLogic Workshop, you can ignore this step).

3. Open the Portal menu and select Open Current Portal...

   The log-in portal appears in a browser.

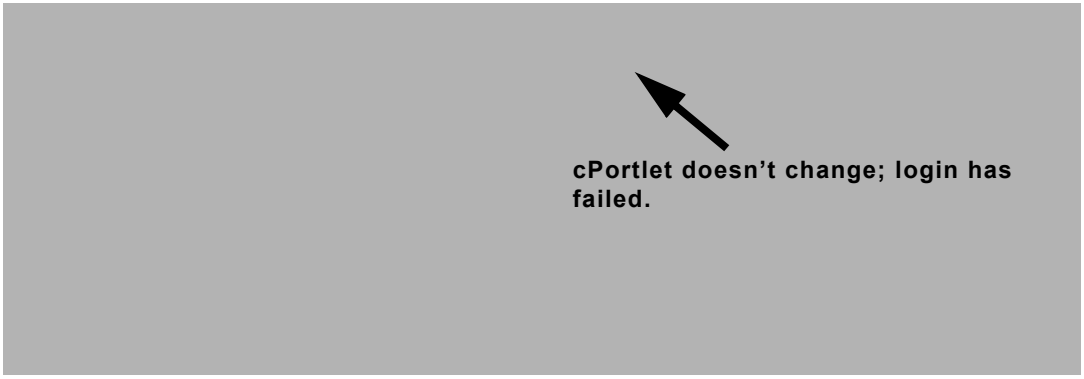
4. Click Login.

   The Password and Username fields appear.

5. For both values, enter weblogic and click Submit.

6. The portal refreshes and cPortlet will read Username = null (that is, it won't change; Figure 9-22). This indicates that the login has failed.

Figure 9-22 login.portal After Failed Login Attempt



7. Close the portal and stop WebLogic Server.

Step 4: Obtain and Implement a Signed Certificate

In this step, you will use the The Java keytool Utility to create a Self-signed certificate, creating a new keystore in the process. You will then generate a Certificate Signing Request (CSR) and send that CSR to a Certificate Authority (CA) who will return to you a signed certificate that you will use to establish log-in to the remote portlet.

Before You Begin

Before attempting this procedure, please read about The Java keytool Utility. You might also want to review the information included in keytool - Key and Certificate Management Tool at:

`http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html`

Generate a New Keystore

To generate a new keystore, use this procedure.

Note: *<BEA_HOME>* is the directory where you installed WebLogic Platform.

1. Open a command prompt on your computer and navigate to the *<BEA_HOME>*`\jdk142_05\bin`

   Note: The keytool utility is scoped to the JVM you are using. If you select another JVM when you configured your implementation of WebLogic Server, for example BEA WebLogic JRockit, you must navigate to and run the following commands from that JVM's directory.

2. At the command line, create the keystore by entering the `-genkey` command as shown here:

   ```
   <BEA_HOME>\jdk142_05\bin>keytool -genkey -keypass testkeypass -keystore
   <BEA_HOME>\user_projects\domains\ipcWsrpDomain\wsrpKeystore.jks
   -storepass teststorepass -alias testalias
   ```

   In this example,:

   - **`testkeypass`** is the password used to protect the private key of the generate key pair.
   - **`<BEA_HOME>\user_projects\domains\ipcWsrpDomain\`** is the domain directory.
   - **`wsrpKeystore`** is the new `.jks` file.
   - **`teststorepass`** is the password used to protect the integrity of the keystore.
   - **`testalias`** is a name you specified as an alias, which is used to access an entity in the keystore.

Note: For a complete list of options, please refer to keytool - Key and Certificate Management Tool at:

`http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html`

The keytool options are *not required*. If you choose *not* to specify them, defaults are used for those that have default values and you will be prompted for any required values.

3. Follow the prompts required to identify yourself.

   If all the information has been entered correctly, the system will respond with a command prompt, such as `<BEA_HOME>\jdk142_05\bin>`. To verify that the keystore was properly generated, go to your file system and in `<BEA_HOME>\user_projects\domains\ipcWsrpDomain\`, locate `wsrpKeystore.jks`

Create the Certificate Signing Request and Import the Signed Certificate

You've now generated a self-signed certificate. Because any certificate is more likely to be trusted by others if it is signed by a Certification Authority (CA), you now need to generate a Certificate Signing Request (CSR) to gain that signature. To create the CSR, do the following:

1. Go to a command prompt and navigate to `<BEA_HOME>\jd142_05\bin` (if you are already at this directory, you can ignore this step).
2. Enter the `keytool -certreq` command, using the options listed here:

   ```
   keytool -certreq -keystore
   <BEA_HOME>\user_projects\domains\ipcWsrpDomain\wsrpKeystore.jks -alias
   testalias
   ```

   ```
   The system responds:
   ```

   ```
   Enter keystore password:
   ```

3. Enter teststorepass.

   ```
   The system responds:
   ```

   ```
   Enter key password for <mykey>:
   ```

4. Enter testkeypass.

   The system will then generate the CSR and respond with a series of characters representing the CSR; for example:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIICYzCCAiACAQAwXjELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAkNPMRAwDgYDVQQHEwdCb3VsZGVy
MRQwEgYDVQQKEwtCRUEgU3lzdGVtczENMAsGA1UECxMERG9jczELMAkGA1UEAxMCRWQwggG3MIIB
LAYHKoZIzjgEATCCAR8CgYEA/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZ
```

```
PY1Y+r/F9bow9subVWzXgTuAHTRv8mZgt2uZUKWkn5/oBHsQIsJPu6nX/rfGG/g7V+fGqKYVDwT7
g/bTxR7DAjVUE1oWkTL2dfOuK2HXKu/yIgMZndFIAccCFQCXYFCPFSMLzLKSuYKi64QL8Fgc9QKB
gQD34aCF1ps93su8q1w2uFe5eZSvu/o66oL5V0wLPQeCZ1FZV4661FlP5nEHEIGAtEkWcSPoTCgW
E7fPCTKMyKbhPBZ6i1R8jSjgo64eK7OmdZFuo38L+iE1YvH7YnoBJDvMpPG+qFGQiaiD3+Fa5Z8G
kotmXoB7VSVkAUw7/s9JKgOBhAACgYBkQ10+BRJVVzMgZTQJiUDYdK+5WOI1EkvXbyZPmvYzAfch
vtR7WKJZMPcbAyq9mtrOXFY7TTEkupXlY4R8c5DdLW0db3YB1eV4gUGQOXn4Y+zE8Z4LxKNhkKLk
yEUQhv0JkyzIReV7sioJahf7AiOwqs2cW1r4dNt4y42duwrdsKAAMAsGByqGSM44BAMFAAMwADAt
AhRARh4iBbioO+Jn3qc/bXOpjr+cqgIVAI78/s8hMqhFkTJxt/qtE3L3F1aP
-----END NEW CERTIFICATE REQUEST-----
```

5. Submit the CSR to a certification authority (CA), such as VeriSign, Inc. They will authenticate you, sign a certificate, and then return it to you. This certificate authenticates your public key.

   Note: Submitting the `.pem` file to a CA is beyond the scope of this document. There are many CAs you can use and your company might already have agreements with a specific CA that defines how to submit the CSR. You can also access the website for any CA to review their submission process.

Import the Signed Certificate

A CA will return two certificates: a consumer certificate and the CA certificate. The CA certificate is used to validate the CA's signature on the consumer certificate. Use the `keytool -import` command to store both the CA-signed certificate and the consumer certificate in the keystore.

Note: Depending upon the CA you use, the CA might return a single file with both certificates imbedded therein. You will have to separate the two using some sort of certificate translation tool, such as the certificate import/export wizards in Internet Explorer. This process is beyond the scope of this document; we recommend you consult your company's security administrator or the CA you used.

Now, you will import both the consumer certificate and the CA certificate into the keystore. Note that when you run the import command, you must use a different alias for each of the certificates, as explained in the following steps.

1. When you obtain the the signed consumer certificate from the CA, save it in `<BEA_HOME>\user_projects\domains\ipcWsrpDomain\` as a `.pem` file called `wsrpKeystore.pem` (if the CA returns the certificates as a `.pem` file, you can skip this step).

2. Use the `keytool -import` command to store the consumer certificate in the keystore entry identified by *`testalias`*. This will replace the self-signed certificate you created with the `-genkey` command. Enter the following command:

```
keytool -import -alias testalias -file
<BEA_HOME>\user_projects\domains\ipcWsrpDomain\wsrpKeystore.pem -keypass
testkeypass -keystore
<BEA_HOME>\user_projects\domains\ipcWsrpDomain\wsrpKeystore.jks -storepass
teststorepass
```

Note: For more information on using the `-import` command, please refer to:

http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html#importCmd

The system responds:

```
Owner: CN=Your Name, OU=WLP Docs, O=WLP, L=Boulder, ST=Colorado, C=US
Issuer: CN=Thawte Test CA Root, OU=TEST TEST TEST, O=CA Name Certification, ST=FO
R TESTING PURPOSES ONLY, C=ZA
Serial number: 121e
Valid from: Mon Dec 13 11:18:17 MST 2004 until: Mon Jan 03 11:18:17 MST 2005
Certificate fingerprints:
         MD5:  87:29:4B:7F:02:7D:2B:90:EF:FA:7D:02:50:82:7D:FF
         SHA1: 25:08:1E:98:7D:76:31:48:B3:6B:4B:5F:81:24:59:1D:41:CA:A2:DB
```

And asks:

```
Trust this certificate? [no]:
```

3. Enter yes.

   The system responds:

   ```
   Certificate was added to keystore
   ```

4. Repeat steps 1 through 3 to import the CA certificate. Save the CA certificate obtained from the CA in a file called `<BEA_HOME>\user_projects\domains\ipcTestDomain\CAcert.pem`, and use the following values for the `.pem` filename (`-file`) and alias (`-alias`) when you run the `keytool -import` command.

   ```
   -file <BEA_HOME>\user_projects\domains\ipcTestDomain\CAcert.pem
   ```

   ```
   -alias CAtestalias
   ```

Step 5: Update the Consumer mBean

Update the Consumer mBean with the new certificate information by doing the following

1. In WebLogic Workshop, open the application to which the certificate applies.
2. Start WebLogic Server by selecting Tools>WebLogic Server>Start WebLogic Server.
3. Launch the Administration Portal by selecting Portal>Portal Administration

The Administration Portal login page appears

4. Login to the Administration Portal using weblogic for both the Username and the Password.

   The Administration Portal appears.

5. Under Configure Settings, click Service Administration.

   The Configuration Setting page appears (Figure 9-23).

Figure 9-23 Configuration Settings Page

6. In the left pane, click WSRP Consumer Security Service.

   The Configuration Settings for: dialog box appears in the right pane (Figure 9-24)

Figure 9-24 Configuration Settings for: WSRP Consumer Security Service Dialog Box

7. Update the necessary fields on this dialog box with information from the keystore. At the minimum, you must update the fields listed in Table 9-1.

Table 9-1 Updating WSRP Consumer Security Service Configuration Information

In...	Enter...
Consumer Name	testalias
Keystore Password	teststorepass
Certificate Alias	testalias
Certificate Private Key Password	testkeypass

8. Click Update.
9. Restart the server.

Step 6: Update the WSRP Identity Asserter

For a Producer to trust a Consumer, it needs to recognize the Consumer's signed certificate. To ensure this, you need to provide the Producer with the Consumer's public key, which the Producer will add to its keystore by updating the WSRP identity asserter.

To update the WSRP identity asserter, use this procedure:

1. Launch an instance of WebLogic Server and open the WebLogic Server console by entering in the address field of a browser the URL:

   ```
   http://localhost:7001/console
   ```

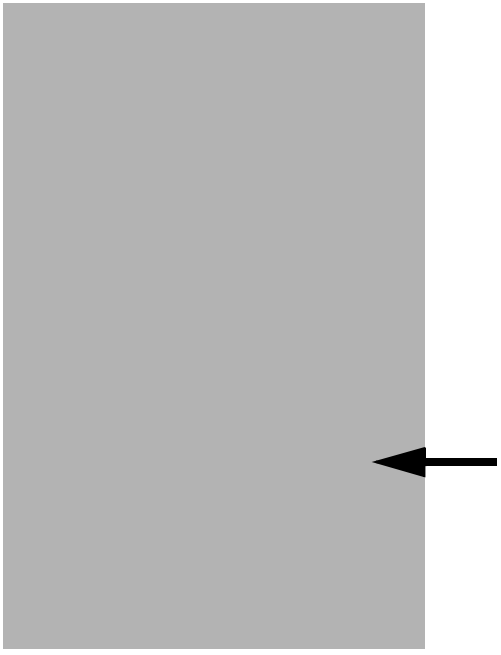
   The WebLogic Server Administration Console login page appears.

2. Log in to the administration console with the user name *weblogic* and the password *weblogic*.

   WebLogic Server Administration Console appears.

3. In the left pane, drill down to the WSRP Identity Asserter node (Security>Realms>myRealm>Providers>Authentication>WSRPIndentityAsserter), as shown in Figure 9-25.

Figure 9-25 WSRP Identity Asserter Drill-down



The WSRP Identity Asserter appears in the right pane.

4. Select the Detail tab to display WSRP identity detail information, as shown in Figure 9-26.

Figure 9-26 WSRP Identity Asserter Detail

5. Update the information in the WSRP Identity Asserter as described in Table 9-2.

Table 9-2 Updating the WSRP Identity Asserter

In...	Enter...
Key Store Path	wsrpKeystore.jks
Key Store Password	teststorepass
Confirm Key Store Password	teststorepass

6. Click Apply.

Note: If any of the yellow icons next to the field labels are blinking, this means that the server must be restarted for the changes to take effect.

Step 7: Test the New Keystore

To test the new keystore, you simply need to try to log in to the remote portlet, as you did in Test the Log-in Portlet. Do the following:

1. Launch WebLogic Workshop, if it isn't already running.

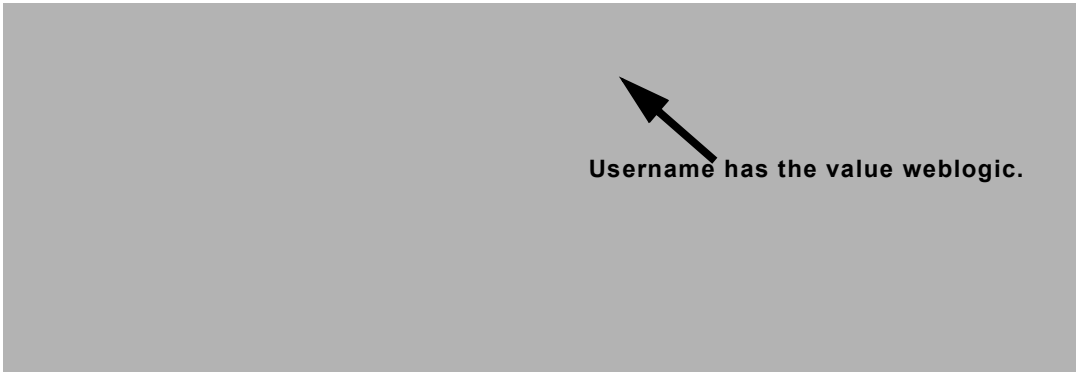
2. Stop (if necessary) and restart the server. Select Tools>WebLogic Server>Stop WebLogic Server (if the server is running). Select Tools>WebLogic Server>Start WebLogic Server to start the server.
3. Once the server starts, go to the consumerWeb node and double-click `login.portal` (if `login.portal` is already open, you can ignore this step).
4. Open the Portal menu and select Open Current Portal.

   `login.portal` renders in the browser.
5. In the LoginController portlet, click Login.

   The portlet refreshes to show the login form.
6. In both Username and Password, enter weblogic and click submit.

   The portal refreshes; cPortlet will now display Username = weblogic.

Figure 9-27 login.portal After Login is Successful



Securing the WebLogic Administrator's Logon Information

A portal domain's portal application's `META-INF` directory contains the `application-config.xml` file. In newly created portal domains where the portal application is deployed, this file will contain unencrypted passwords. These passwords actually remain clear text until the portal administration tool's Service Administration page is used to edit and save attributes contained in this config file, or the application using the password accesses it. (And if the portal application is deployed as an ear file, this file cannot be edited and saved.)

Typically, in development, the administrator will use some sort of generic logon information (logon ID and password), such as *weblogic/weblogic*. For example, Listing 9-5 shows the WSRP element from `application-config.xml` on initial deployment:

Listing 9-5 Development Phase Clear Text Passwords in `application-config.xml`

```
<ConsumerSecurity AdminPassword="weblogic" AdminUserName="weblogic"
   CertAlias="wsrpConsumer" CertPrivateKeyPassword="wsrppassword"
   ConsumerName="wsrpConsumer"
   IdentityAssertionProviderClass="com.bea.wsrp.security.
      DefaultIdentityAssertionProvider"
   Keystore="wsrpKeystore.jks" KeystorePassword="password"
   Name="ConsumerSecurity"/>
```

After the Service Administration tool is used to edit attributes, the file is saved and automatically passwords are encrypted, as shown in Listing 9-6:

Listing 9-6 .Encrypted Passwords in `application-config.xml`

```
<ConsumerSecurity AdminPassword="{3DES}3QrrUeIwN/DxlDI++1ixPw=="
   AdminUserName="weblogicc" CertAlias="wsrpConsumer"
   CertPrivateKeyPassword="{3DES}g7h+VOSAsO9pSlvYSSB2iw=="
   ConsumerName="wsrpConsumer"
   IdentityAssertionProviderClass="com.bea.wsrp.security.
      DefaultIdentityAssertionProvider"
   Keystore="wsrpKeystore.jks" KeystorePassword=
      "{3DES}1OLYVirMWOo+3sEU80cMqw=="

   Name="ConsumerSecurity" />
```

To ensure the security of passwords throughout the applications lifecycle, you need to use the `EncryptDomainString` command-line utility to generate an encrypted password and then place that encrypted password into the `application-config.xml` file while it is still in the development environment. Then you can build the EAR file for the application and deploy it as necessary.

Encrypting Passwords

To encrypt passwords, use this procedure:

1. Open a command box (DOS shell) and navigate to *domain*/portal/ (where *domain* is the domain directory for the application) and run setDomainEnv.cmd.
2. At the prompt, enter

   ```
   java com.bea.p13n.util.EncryptDomainString -targetDomainDir d
   -inputString s
   ```

   where:

   – *d* is the domain directory to which the portal application is being deployed; for example,

   – *s* is the input password to encrypt

   For example:

   ```
   java com.bea.p13n.util.EncryptDomainString -targetDomainDir
   \bea\weblogic81b\samples\domain\portal -inputString weblogic
   ```

   In this example, the input string weblogic represents administrator's password (adminpassword=weblogic; see Listing 9-7). The command line utility prints a domain specific encrypted string.
3. Open WebLogic Workshop and the specific portal application.
4. Select File>Open>File (Figure 9-28).

Figure 9-28 Opening a File in WebLogic Workshop

An Open dialog box appears.

5. Navigate to the application's META-INF folder and open application-config.xml.

Listing 9-7 Clear text Passwords in application-config.xml

```
<ConsumerSecurity AdminPassword="weblogic" AdminUserName="weblogic"
   CertAlias="wsrpConsumer" CertPrivateKeyPassword="wsrppassword"
   ConsumerName="wsrpConsumer"
   IdentityAssertionProviderClass="com.bea.wsrp.security.
      DefaultIdentityAssertionProvider"
   Keystore="wsrpKeystore.jks" KeystorePassword="password"
   Name="ConsumerSecurity"/>
```

6. Replace the clear text passwords with those generated by the `EncryptDomainString` utility, as shown in Figure 9-8. You will need to run `EncryptDomainString` for each password in the `<ConsumerSecurity>` (Listing 9-7) element; for example:
   - `AdminPassword="weblogic"`
   - `CertPrivateKeyPassword="wsrppassword"`
   - `KeystorePassword="password"`

Listing 9-8 Encrypted Passwords Generated by EncryptDomainString Utility

```
<ConsumerSecurity AdminPassword="{3DES}3QrrUeIwN/DxlDI++1ixPw=="
   AdminUserName="weblogicc" CertAlias="wsrpConsumer"
   CertPrivateKeyPassword="{3DES}g7h+VOSAsO9pSlvYSSB2iw=="
   ConsumerName="wsrpConsumer"
   IdentityAssertionProviderClass="com.bea.wsrp.security.
      DefaultIdentityAssertionProvider"
   Keystore="wsrpKeystore.jks" KeystorePassword=
      "{3DES}1OLYVirMWOo+3sEU80cMqw=="

   Name="ConsumerSecurity" />
```

Now you can build the EAR file and deploy the application.

Note on Changing Passwords

If you need to change an administrator's password for any reason, simply changing the password will result in having to rebuild and redeploy the EAR. This is time-consuming and counterproductive. Instead, you can work around this problem by doing the following:

1. Create a special user in the target system for a WSRP administrator. See Create a New User for more information on creating a user.
2. Make that user a member of the administrator group. See Add a User to a Group for more information on adding a member to a group.
3. Insert the new logon information (username and password) into the application's `application-config.xml` file as described in Encrypting Passwords.

WSRP Error Messages

You might encounter one of the error conditions described in Table A-1 while attempting to create and use WSRP-compliant portals.

Table A-1 WSRP Error Messages

Message	Description
`Error: Unable to get the Service Description for the provided WSDL URL`	Producer is not available:
`Fault: {urn:oasis:names:tc:wsrp:v1:types} InvalidRegistration`	Producer is not registered and registration is required Missing registrationHandle?.
`Fault: {urn:oasis:names:tc:wsrp:v1:types} InvalidHandle The given portletHandle [portlet_1] is invalid or none of the supported portlet containers can handle this portlet`	The remote portlet has been changed or it has been deleted