BEA WebLogic Portal™

Using WSRP with WebLogic Portal

Version 8.1 Service Pack 3
June, 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

Introduction to WSRP

Establishing WSRP Security

Best Practices for Implementing WSRP

Applying a Look-and-Feel to a Remote Portlet

Monitoring and Logging Remote Portlet Performance

Working with Producers

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 SP3 includes an implementation of WSRP that allows the framework to use WSRP portlets.

This section includes information on the following subjects:

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
- Struts
- Java portlets (JSR168)

You can also use JSP portlets provided you start by creating a Java Page Flow (JPF) and pointing the Begin action to a JSP.

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 SP3 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 SP3 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

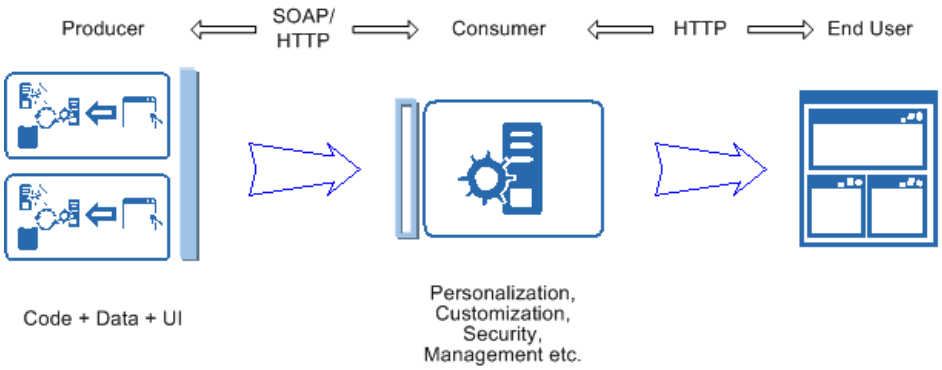
In addition to those listed above, WSRP provides these additional benefits to developers:

- 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 pageflows and Struts applications. 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
- Light-weight, simpler to manage

You can create a simple Producer from a complex Producer so that pageflows and Struts applications available as "portlets" to remote portals. This procedure is described in Creating a Simple Producer from a Complex Producer.

Complex Producers

A complex Producer requires registration, does support URL rewriting in the Consumer, and does support a management interface. By default, all portlets created with WebLogic Workshop 8.1 SP3 are complex Producers.

With complex Producers:

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

Consumers

Consumers 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.

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 1-1:

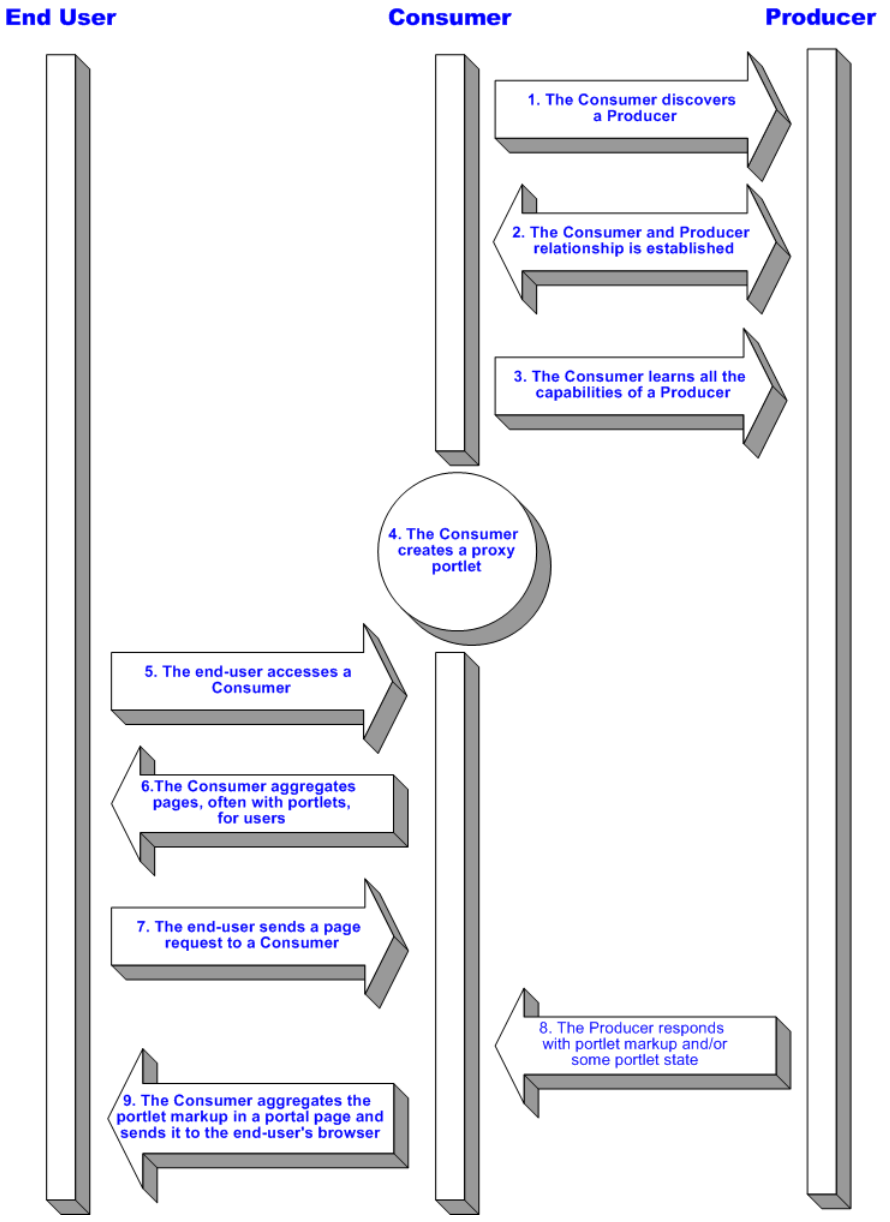
Table 1-1 Features in BEA's WSRP Implementation

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 Web Service Description Language identifier (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	All web projects created with WebLogic Workshop are, by default, Producers. You don't need to do any special coding or add additional content to a project to make it available as a Producer.
Registration	Consumers might be required to register with a Producers. 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, JSR 168, and Struts applications to expose their functionality in remote portlets. They can portletize the application and configure any related properties. Since all projects created with this version WebLogic Workshop 8.1 SP3 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 SP3.
- Customers using applications built on a WebLogic Server 8.1SP3-only installation need to follow the instructions for converting a non-portal web application into a Producer, as described in Creating a Producer from a Non-Portal Web Application.

New applications, after WSRP installation, are automatically configured.

Consumer side The experience would be similar to the current Administration Portal today since remote portlets look like local portlets.

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.

Working with 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,

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.

Note on Localization of Remote Portlets

In service pack 3 of WebLogic Portal 8.1, you cannot localize remote portlets. This feature will be available in a subsequent release of the product.

CHAPTER 2

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
- Obtaining a Signed Certificate
- Configuring the Producer Keystore

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:

- Secure WSRP Messages
- Manage User Identity
- Secure the /producer Path

Secure WSRP 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 2-1.

Listing 2-1 <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.

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).

Obtaining a 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 describes how to use the Java keytool utility to generate a self-signed certificate and then obtain a signed certificate from a CA. It contains information on the following subjects:

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.

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`

Obtaining the Consumer Certificate

To obtain a signed certificate, use this procedure.

Note: Before you can actually create a keystore and generate a certificate, ensure the following has been completed:

- A domain has been created
- A portal web application has been created
- A project has been created

1. At the command line, create the keystore by entering the `-genkey` command; for example:

```
keytool -genkey -keypass password1 -file filename.pem -keystore
C:\working\myWsrpKeystore.jks -storepass password2 -alias myAlias
```

   Where:

   – *password1* is the password used to protect the private key of the generate key pair.

   – /*working*/ is the directory into which you copied the renamed `.jks` file.

   – *`myWsrpKeystore`* is the new `.jks` file.

   – *password2* is the password used to protect the integrity of the keystore.

   – *`myAlias`* is a name you specified as an alias, which is used to access an entity in the keystore.

   Note: The options listed above are just a sample of the options you can use when generating a keystore. 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.

2. At this point, you've generated a self-signed certificate. Because a 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 by doing the following:

   a. Go to a command prompt and enter the `-certreq` command, specifying the appropriate options; for example:

      `-certreq -keystore myWsrpKeystore.jks`

      where *`myWsrpKeystore`* is the renamed `.jks` file.

      The system responds:

      `Enter keystore password:`

   b. Type the password you assigned to the `-storepass` option.

      The system responds:

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

c. Type the password you assigned to the `-keypass` option.

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-----
```

This creates a CSR and puts the request in the file named *myAlias*`.pem` (where *myAlias* is the alias you specified when you created the keystore).

d. Submit the `.pem` file 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.

3. Import the signed certificate by using the `-import` command, specifying the appropriate options; for example:

```
keytool -import
```

With the certificate returned, you will need to store it in the keystore entry identified by *myAlias*. This will replace the self-signed certificate you created with the -genkey command.

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

4. Import the signed certificate, go to a command prompt and enter the `-import` command,

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

a. In WebLogic Workshop, open the application to which the certificate applies.

b. Start WebLogic Server by selecting Tools>WebLogic Server>Start WebLogic Server.

c. Launch the Administration Portal by selecting Portal>Portal Administration

The Administration Portal login page appears

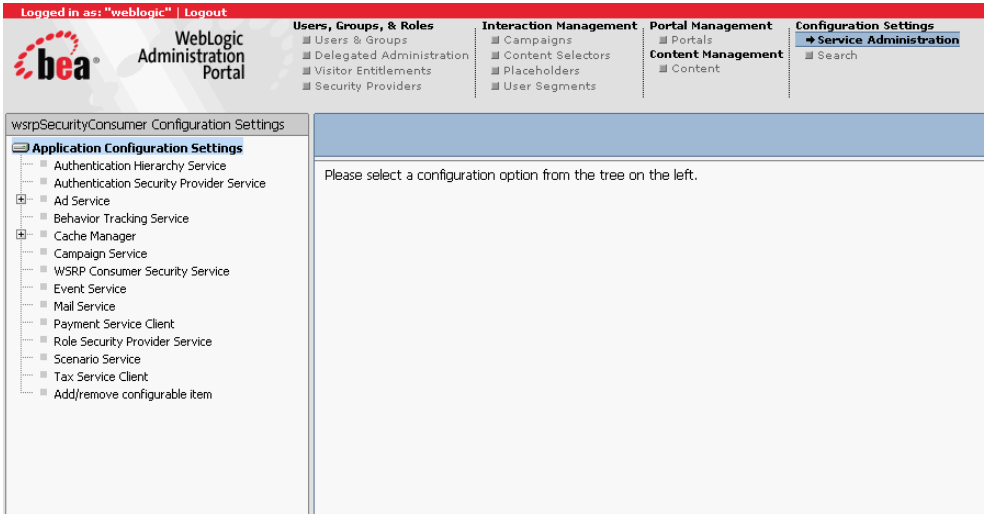
d. Login to the Administration Portal.

The Administration Portal appears.

e. Under Configure Settings, click Service Administration.

The Configuration Setting page appears (Figure 2-1).

Figure 2-1 Configuration Settings Page



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

The Configuration Settings for: dialog box appears in the right pane (Figure 2-2)

Figure 2-2 Configuration Settings for: WSRP Consumer Security Service Dialog Box

g. Update the necessary fields on this dialog box with information from the keystore; for example, Consumer name, keystore name, and passwords.

h. Click Update.

i. Restart the server.

Configuring the Producer Keystore

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.

To configure a keystore on the keystore side, you need to do the following:

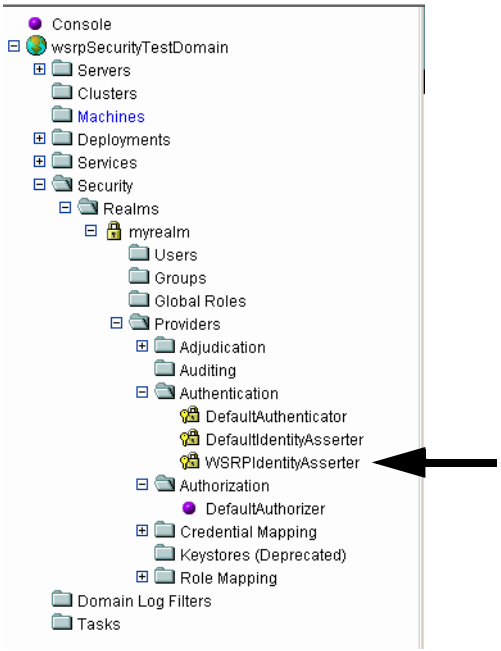
- Update the WSRP Identity Asserter (on the WebLogic Server console).
- Set Up the Producer Keystore by importing the signed certificate that was returned to the Consumer.

Update the WSRP Identity Asserter

To update the WSRP identity asserter, use this procedure:

1. Launch WebLogic Server and open the WebLogic Server console.
2. In the left pane, drill down to the WSRP Identity Asserter node (Security>Realms>myRealm>Producers>Authentication>WSRPIndentityAsserter), as shown in Figure 2-3.

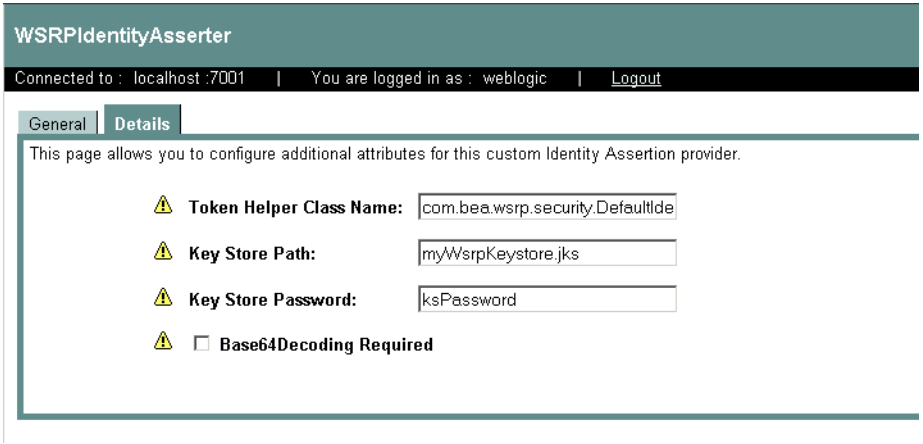
Figure 2-3 WSRP Identity Asserter Drill-down



The WSRP Identity Asserter appears in the right pane.

3. Select the Detail tab to display WSRP identity detail information, as shown in Figure 2-4.

Figure 2-4 WSRP Identity Asserter Detail



4. Update the following fields with the appropriate information:
   - Keystore Path: The name of the `.jks` file of the Consumer to be trusted.
   - Keystore Password: The password created for the `.jks` file of the Consumer to be trusted.
5. Click Apply.

Note: If any of the yellow icons next to the field labels are blinking, you will need to reboot the server.

Set Up the Producer Keystore

To set up the Producer keystore, use the `-import` command on the Producer to import the signed certificate that was returned to the Consumer; for example:

```
keytool -import -file cert_file
```

where *`cert_file`* is the name of the `.pem` file that contains the signed certificate.

CHAPTER 3

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.

- URLs
  - When creating URLs in a portlet, do not create 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

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 3-1.

Listing 3-1 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 proxy portlets.
  - Enable forked rendering for proxy portlets.
  - Set connection timeout by adding `<connection-timeout>120000</connection-timeout>` to `WEB-INF/wsrp-producer-registry.xml`, as shown in Listing 3-2.

Listing 3-2 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: *120000*—in milliseconds—is a suggested timeout period. Your needs might require a longer or shorted timeout.

- Disable logging
  - Undeploy MessageMonitor servlet from `WEB-INF/web.xml`.

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` and 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 4

Applying a Look-and-Feel to a Remote Portlet

The *look and feel* of a remote portlet is what determines such aspects as the portlet's physical boundaries, color, font type and size, images. The look-and-feel of a remote portlet is not linked to a Producer. Therefore, you have the option of modifying the portlet to best suit your needs; for example, to match the appearance of the portal in which the portlet resides.

This section briefly describes the basics of remote portlet look-and-feel and directs you to more detailed information in the WebLogic Workshop online help system. This section contains information on the following subjects:

- The Portlet Look-and-Feel Components
- The Look-and-Feel File
- Skins and Skeletons
- Themes
- Where to Find Look-and-Feel Components
- Look-and-Feel Reference

The Portlet Look-and-Feel Components

Portlet look-and-feel architecture primarily involves the following resources:

- The look-and-feel file
- Skins and Skeletons, which contain:

  - `.css` Files
  - Image files
  - JavaScripts
  - JSPs
- Themes, which are subsets of skins and skeletons

In addition to these portlet-level components, at the portal level, additional components include layouts, navigation menus, and shells. All of these components are described in greater detail in Look & Feel Architecture in the WebLogic Workshop online help system, at:

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

This section will be concerned only with the look-and-feel file, skins, skeletons, and themes.

The Look-and-Feel File

A portal look-and-feel stems from a single `XML` file (that has a `.laf` extension). This file determines the name of the look-and-feel and points to the skeleton and skin that the look-and-feel will use. This simple, extensible framework makes it easy for portal administrators and end users to completely change the appearance of a portal by selecting a different look-and-feel.

Skins and Skeletons

The look and feel of a portlet is determined by that portlet's skins and skeletons, which contain the information needed to render the desired appearance.

- Skins determine the "look" portion of look-and-feel. They are comprised of the cascading style sheet (`.css`) files, images (`.gif` and `.jpg` files), and JavaScripts.
- Skeletons represent the "feel" of a portlet. They are comprised of JSP files that determine the physical boundaries of portal components such as header, footer, book, page, portlet, and portlet title bar.

Skins and skeletons used by a portlet are determined by the `.laf` selected for a portal desktop.

The .css File (Skins)

`.css` files are part of a portlet's skin. These files contain formatting instructions for such visual portlet components as background colors, font size, style, and type, border style, and so on. WebLogic Workshop comes with a number of `.css` files already available. You can use these as

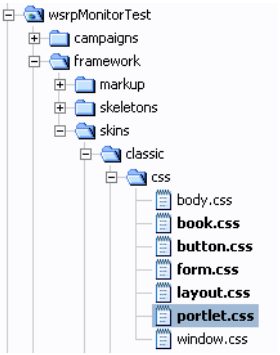
is or modify them as you deem necessary. You can also or create your own `.css` files. If you create your own `.css` file, you will need to store it in the appropriate location, which is in the `/css` directory under the `framework/` directory for a specific project; for example:

`wsrpMonitortest/framework/skins/mySkin/css`

Where *`mySkin`* is the individual skin name.

To make a `.css` available to a skin, simply include the `.css` file in the `/css` directory under that skin's folder; for example, if you want to make the file `portlet.css` available to the skin "classic", you would ensure that it is stored in the directory `skins/classic/css`, as shown in Figure 4-1.

Figure 4-1 .css File in the File Tree



Once you've added the file to the /css directory, you need to register it in the `skins.properties` file, which you can find in root directory of the specific skin; for example, *`myProject`*`/framework/skins/classic/skin.properties` (where *`myProject`* represents the project name).

Image Files (Skins)

Image files contain the graphic elements that will appear on your portlet. These images are for the portlet interface and include buttons, arrows, logos, and so on. These are not images used in the portlet content. Image files are usually `.gif` or `.jpg`/`.jpeg` format as the compressed size of these files makes rendering them a relatively painless process. To make an image file available to a skin, simply include the image file in the `/images` directory under that skin's folder; for example, if you want to make the file `arrow.gif` available to the skin "classic", you would ensure that it is stored in the directory `skins/classic/images`.

JavaScripts (Skins)

The JavaScript (`.js`) files regulate such portlet activity as mouse rollovers, button actions, and menu behavior. You can use the JavaScripts included with WebLogic Workshop or you can create your own. If you create your own, you will need to store it in the appropriate directory (see Where to Find Look-and-Feel Components) and register it in the `skins.properties` file, which you can find in root directory of the specific skin; for example, *`myProject`*`/framework/skins/classic/skin.properties` (where *`myProject`* represents the project name).

JavaServer Pages (Skeletons)

In portlets, JSPs determine the portlet's physical boundaries and those of their title bar. At the portal level, when a portal desktop is rendered, the skeleton JSPs for each portal component (in conjunction with any related classes) perform their logic and insert the resulting HTML into the correct hierarchical locations of the HTML file.

You can use the JSPs included with WebLogic Workshop or you can create your own. If you create your own, you will need to store it in the appropriate directory (see Where to Find Look-and-Feel Components) and register it in the `skeleton.properties` file, which you can find in root directory of the specific skin; for example, *`myProject`*`/framework/skeletons/classic/skeleton.properties` (where *`myProject`* represents the project name).

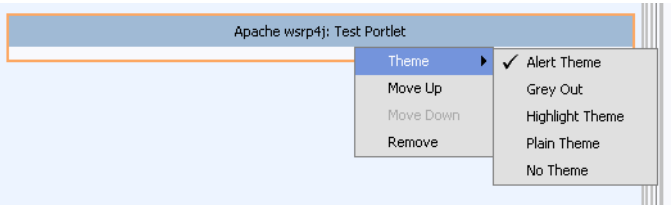
Skin and Skeleton Reference

For more information on creating and using skins and skeletons, please refer to Look-and-Feel Reference. This section contains a complete reference to the documents in the WebLogic Workshop online help system that will guide you through the process of creating and implementing skins and skeletons.

Themes

Skins and skeletons can be applied to portlets in subsets called themes. Themes let you give your portlets (as well as books and pages) a different look-and-feel than the rest of the portal desktop. Themes are identified by a `.theme` file. This XML file identifies the theme and populates a drop-down menu from which portlet developers can select a theme to use (Figure 4-2).

Figure 4-2 Theme Menu in WebLogic Workshop



At its most granular, a theme is a collection of files that contain look-and-feel instructions for a portal component, such as a portlet. You must store these files in a subdirectory, which resides under the skin to which you want to make the theme available. You must give the subdirectory the same `name` XML attribute as the `.theme` file that will be used to identify the theme to a portlet user.

When you select a theme for a portlet, the portal framework looks for theme subdirectories under the main skin and skeleton directories used by the selected look-and-feel. If it finds the theme subdirectory specified, it will render the portlet based upon the instructions within that subdirectory. If it cannot find the specific subdirectory, values for the parent skin and skeleton will be used.

Themes Reference

For more information on creating and using themes, please refer to Look-and-Feel Reference. This section contains a complete reference to the documents in the WebLogic Workshop online help system that will guide you through the process of creating and implementing themes.

Where to Find Look-and-Feel Components

The files used to set the look-and-feel of a remote portlet reside under the project directory under which that portlet resides; for example:

`<Application_Home>/`*`myProject/`*

Where *`myProject/`* is the specific project directory

Table 4-1 lists the actual folder names.

Table 4-1 Look-and-Feel Component File Locations

Component	Located in
Look-and-feel file	`framework/markup/lookandfeel`

Table 4-1 Look-and-Feel Component File Locations

Skins	`framework/skins`
Skeletons	`framework/skeletons`
Themes	`framework/themes` (XML definition file)
`.css` Files	`framework/skins/`*`mySkins`*`/css` Where *`mySkin`* is the specific skin that will use the `.css` file.
Image files	`framework/skins/`*`mySkins`*`/images` Where *`mySkin`* is the specific skin that will use the image.
JavaScripts	`framework/skins/`*`mySkins`*`/js` Where *`mySkin`* is the specific skin that will use the JavaScript.

Look-and-Feel Reference

All aspects of portal and portlet look-and-feel are documented in the WebLogic Workshop online help system. Use the following links to locate this documentation.

General Information on Look-and-Feel

- Creating Look & Feels

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

- Look & Feel Architecture

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

- The Portal User Interface Framework

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportals/fwPortal1.html

- How Look & Feel Determines Rendering

http://edocs.bea.com/workshop/docs81/doc/en/portal/buildportals/fwPortal2.html

Cascading Style Sheets (.css Files)

- Style Sheet Class Reference

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

Skins

- Creating Skins and Skin Themes

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

Skeletons

- Skeleton Reference

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

- Creating Skeletons and Skeleton Themes

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

Themes

- 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

CHAPTER 5

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 5-1.

Listing 5-1 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.

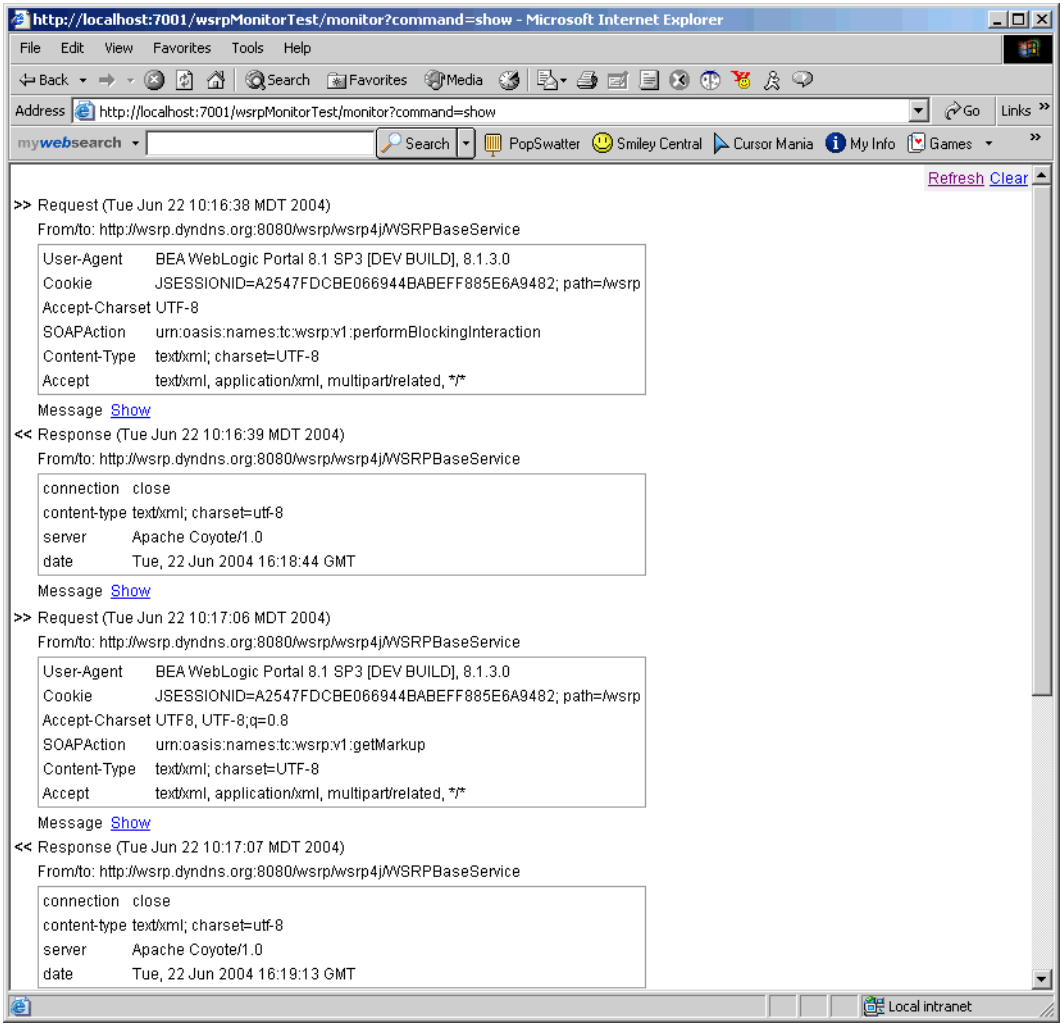
   – *<webProject_name>* is the web project you want to monitor.

   For example:

   `localhost:7001/wsrpMonitorTest/monitor`

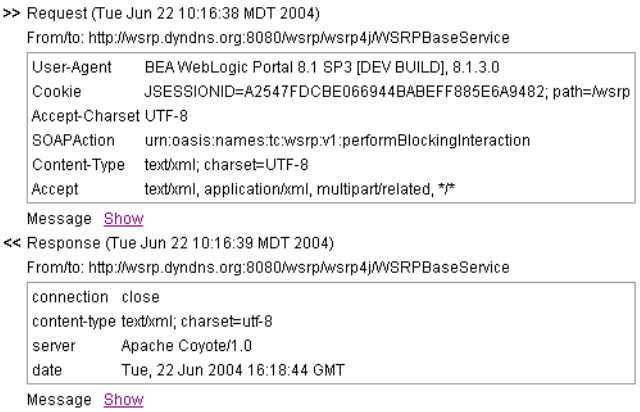
   The Monitor appears in the browser, as shown in Figure 5-1.

Figure 5-1 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 5-2).

Figure 5-2 Monitor Message



– By clicking Show, you can display the content of the request or the response (Figure 5-3).

Figure 5-3 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 Subscribing to Messages at:

`http://e-docs.bea.com/wls/docs81/logging/listening.html#1176551`

Working with Producers

This section describes how you can modify existing applications to create either complex or simple Producers. It includes information in the following subjects:

- Creating a Producer from a Non-Portal Web Application
- Creating a Simple Producer from a Complex Producer

Creating a Producer from a Non-Portal Web Application

You can turn a non-portal web application into a Producer. This is helpful when you want to make Java Page Flow or Struts portlets available as remote portlets. It is also helpful for converting applications created outside of WebLogic Workshop into Producers. You can make this conversion in either a portal application or in a non-portal application.

This section includes information on the following subjects:

- Making the Conversion in a Portal Enterprise Application
- Making the Conversion in a Non-Portal Enterprise Application

Making the Conversion in a Portal Enterprise Application

This section describes how to create a Producer from a non-portal web application.

Making the Conversion

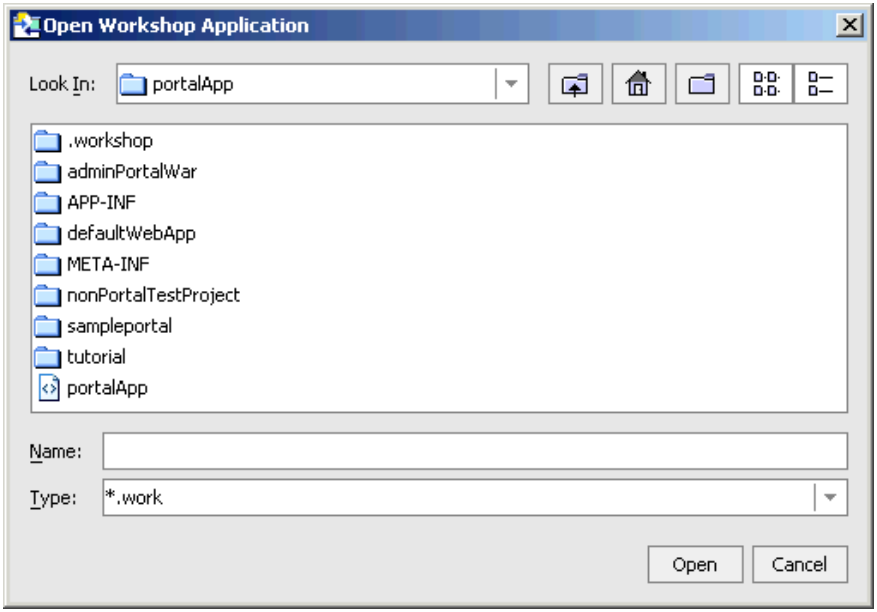
To convert a non-portal web application in a portal enterprise application into a Producer, use this procedure:

Note: WebLogic Workshop must be running.

1. If the enterprise application is currently not open in WebLogic Workshop, open the enterprise application by selecting File>Open>Application (if it is already open, proceed to step 3).

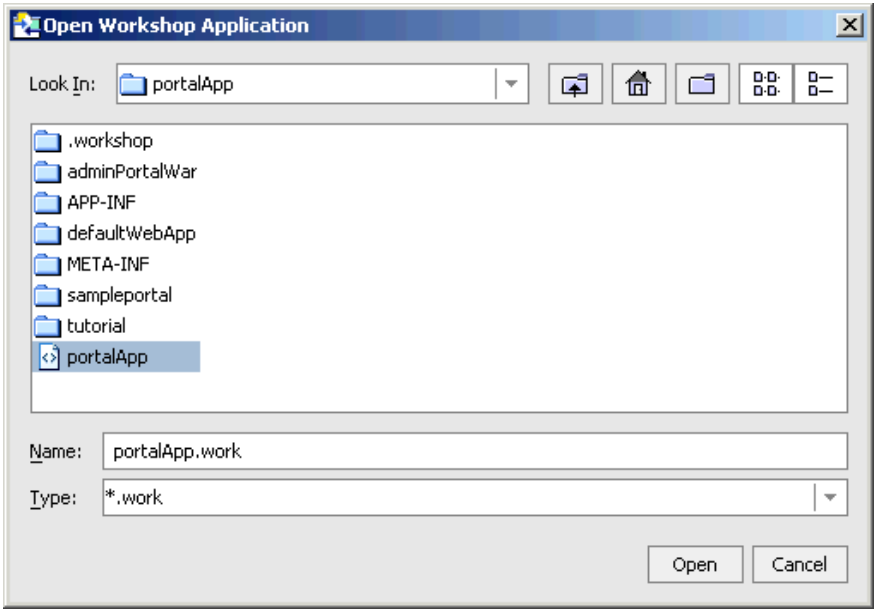
   The Open Workshop Application window appears.

Figure 6-1 Open Workshop Application Window



2. Navigate to and select the appropriate enterprise application (a `*.work` file) and click Open.

Figure 6-2 *.work File Selected



The application appears in the Workshop application tree, as shown in Figure 6-3. Note the non-portal web application listed under the enterprise application.

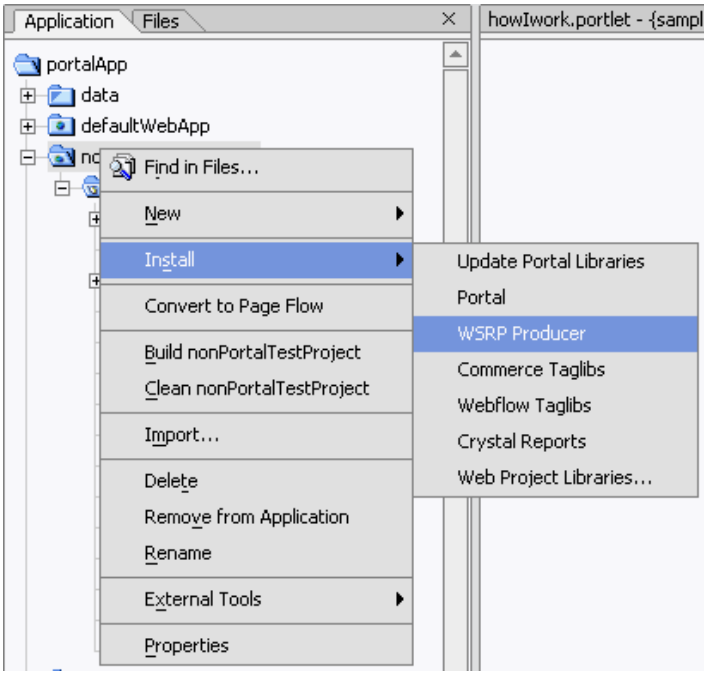
Figure 6-3 Workshop Application Tree Showing Selected Application

3. Copy the following files from the `WEB-INF/lib` folder under the portal web application to the `WEB-INF/lib` folder under the non-portal web application:

- wsrp-struts-adapter.jar
- wsrp-producer-portal-adapter.jar
- netuix_servlet.jar

4. Right-click the web project you want to convert to a Producer and, from the submenu, select Install > WSRP Producer.

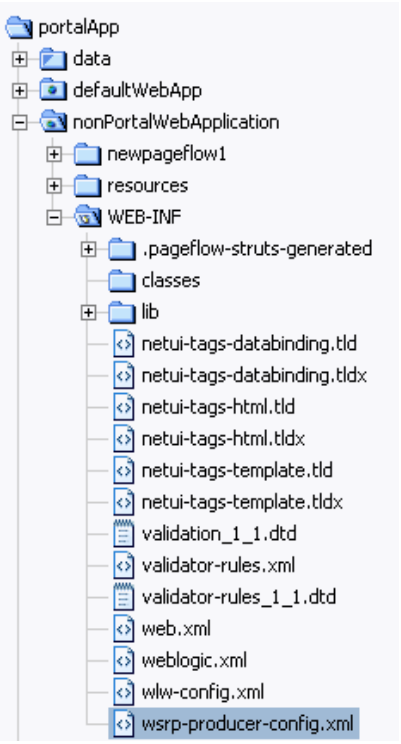
Figure 6-4 Install>WSRP Producer Menus



The project will be installed as a Producer.

5. Verify that the installation was successful by locating the file `wsrp-producer-config.xml` in the `WEB-INF` folder. If it is present, the project was successfully installed as a Producer.

Figure 6-5 Application Tree with wsrp-producer-config.xml Added



Testing the Configuration

To test the configuration, you need to make the Struts or Pageflow application available as a portlet and then view it in a proxy portlet in another web application. To do so, use the following procedure:

1. Enable a Struts or Pageflow portlet by doing one of the following:
   - Copy a Struts or Pageflow `.portlet` file from `<WebLogic813_Home>/portal/portalApp/sampleportal` into the non-portal web application.
   - Create a `.portlet` file inside a portal web application and copy the code sample in Listing 6-1 into that file:

Listing 6-1 Code Sample for Creating a .portlet File

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root
   xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0"
   xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/portal/
      support/1.0.0 portal-support-1_0_0.xsd">
   <netuix:portlet
      definitionLabel="portlet_81b"
                        title="BEA: Non Portal">
      <netuix:titlebar>
      </netuix:titlebar>
      <netuix:content>
         <netuix:pageflowContent contentUri="/Controller.jpf"/>
      </netuix:content>
   </netuix:portlet>
</portal:root>
```

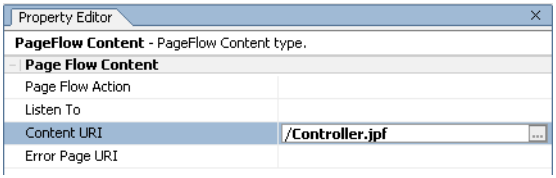
2. If necessary, in the `.portlet` file, update the URI to point to the Struts or Pageflow you want to enable; for example, refer to Listing 6-2

Listing 6-2 Redirecting the URI

```
<netuix:portlet|
   definitionLabel="portlet_81b"
                    title="BEA: Non Portal">
      <netuix:titlebar>
      </netuix:titlebar>
      <netuix:content>
         <netuix:pageflowContent contentUri="/Controller.jpf"/>
      </netuix:content>
</netuix:portlet>
```

You can also update this URI in the WebLogic Workshop property editor, as shown in Figure 6-6

Figure 6-6 Updating the URI by Using the Property Editor



Note: For instructions on steps 3 and 4, please refer to Building a Remote Portlet in the BEA WebLogic Portal IDE online help system.

3. In a different web application, create a proxy portlet in a WebLogic Portal portal and set the WSDL to reference the Struts or Pageflow portlet you just created; for example:

   `http://localhost:7001/`*`webAppName`*`/producer?wsdl`

   Where *`webAppName`* is the web application in which you've stored the `.portlet` file you created in step 1, above.

4. Add the proxy portlet to a portal and view the portal in a browser.

   The Struts or Pageflow portlet you created will appear in the browser.

Making the Conversion in a Non-Portal Enterprise Application

You can also convert a non-portal web application into a Producer from a non-portal enterprise application. This process require adding files that are part of the WebLogic Portal, but you do not need to use WebLogic Workshop.

To convert a non-portal web application into a Producer in a non-portal application, do the following:

1. If your domain is not a portal or a framework domain, you will need to update the domain by doing the following (otherwise, proceed to step 2):

   a. Using a text editor such as Notepad, open the `setEnv.cmd` or `setDomainEnv.cmd` (`setEnv.sh`/`setDomainEnv.sh` for Linux) file and append `%WL_HOME/portal/lib/wsrp/wsrp-common.jar` to the system classpath (`set CLASSPATH=`); for example (Windows):

```
set 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;
```

   or (Linux):

```
CLASSPATH=$WEBLOGIC_CLASSPATH%;%POINTBASE_CLASSPATH%;
   %JAVA_HOME%\jre\lib\rt.jar;%WL_HOME%\server\lib\webservices.jar
   ${WEBLOGIC_CLASSPATH}${CLASSPATHSEP}${POINTBASE_CLASSPATH}
   ${CLASSPATHSEP}${JAVA_HOME}\jre\lib\rt.jar${CLASSPATHSEP}${WL_HOME}
   /server/lib/webservices.jar"export CLASSPATH;$WL_HOME/portal
   /lib/wsrp/wsrp-common.jar;
```

b. Optionally, if you want to enable SSO, deploy the WSRP IdentityAsserter and keystore.

2. In the Producer project's `web.xml` file, do the following:

a. Add the filter `com.bea.wsrp.producer.adapter.pageflow.WsrpPageflowFilter` (Listing 6-3) and map it to the URL pattern `/producer/*` (Listing 6-4).

Listing 6-3 Adding com.bea.wsrp.producer.adapter.pageflow.WsrpPageflowFilter

```
<filter>
   <filter-name>WsrpPageflowFilter</filter-name>
   <filter-class>com.bea.wsrp.producer.adapter.pageflow.WsrpPageflowFilter
      </filter-class>
</filter>
```

Listing 6-4 Mapping com.bea.wsrp.producer.adapter.pageflow.WsrpPageflowFilter

```
<filter-mapping>
   <filter-name>WsrpPageflowFilter</filter-name>
   <url-pattern>/producer/*</url-pattern>
</filter-mapping>
```

b. Add the servlet `com.bea.wsrp.producer.WsrpServer` (Listing 6-5) and map it to the URL pattern `/producer/*` (Listing 6-6) Make sure to set the `<load-on-startup>` parameter in `<servlet>` (Listing 6-5) to ensure that `com.bea.wsrp.producer.WsrpServer` is loaded when the server starts up.

Listing 6-5 Adding com.bea.wsrp.producer.WsrpServer

```
<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>
```

Listing 6-6 Mapping com.bea.wsrp.producer.WsrpServer

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

c. Optionally, if you want to monitor SOAP logs, add the servlet `com.bea.wsrp.logging.MessageMonitor` and map it to `/monitor`.

3. Add the following files to the project's `WEB-INF/lib` folder:
   - `$WL_DIR/portal/lib/wsrp/wsrp-producer.jar`
   - `$WL_DIR/portal/lib/wsrp/adapter/wsrp-jpf-adapter.jar`
   - `$WL_DIR/portal/lib/wsrp/adapter/wsrp-struts-adapter.jar`
4. Add `wsrp-producer-config.xml` to the Producer's `WEB-INF` directory.

Listing 6-7 Contents of 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></description>
```

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

   <supported-locales>
      <locale>en</locale>
      <locale>en-US</locale>
   </supported-locales>

</wsrp-producer-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.

Testing the Configuration

To test the configuration, follow the same steps outlined above in Testing the Configuration. To test this configuration, you will need to create the consumer remote portlet (step 2, below) on a separate machine; otherwise the expense of running two domains on a single machine will severely impact performance.

Note: WebLogic Server must be running.

1. In the non-portal web application created under the non-portal enterprise application, create a Struts or pageflow portlet, updating the URI, if necessary.
2. In a portal web application *on a separate machine*, create a remote portlet and point the WSDL to the non-portal web application under the non-portal enterprise application; for example:

   `http://`*`separate.machine`*`/`*`nonPortalWedApplication`*`/producer?wsdl`

   (Where *`separate.machine`* is the name or IP address of the machine hosting the non-portal web application and *`nonPortalWedApplication`* is the non-portal web application.)
3. Add the remote portlet to a portal and open the portal. If you correctly configured the non-portal web application under the non-portal enterprise application, the Struts or Pageflow portlet you created in step 1 should surface in the portal.

Creating a Simple Producer from a Complex Producer

Unlike a complex Producer, a simple Producer is a non-portal web application that contains Pageflows and Struts applications. It does not depend upon any portal features (for example, customization). It doesn't allow registration, doesn't support URL rewriting in the Consumer, and does not support portlet customization.

A simple Producer is often advantageous because it easier to manage and you don't need to have the complete portal installed to run it. Simple Producers are commonly used in smaller, departmental settings, where more the more advanced features of complex Producers are not necessary.

By default, all portal projects created with WebLogic Workshop 8.1 SP3 are complex Producers. However, you can create a simple Producer from a complex Producer by following the procedures below.

To create a simple Producer from a complex one, use this procedure.

1. From the complex portlet's `WEB-INF/` directory, open the `wsrp-producer-config.xml file` and locate the `<service-config>` element. It should look like the example in Listing 6-8:

Listing 6-8 <service-config> Element of wsrp-producer-config.xml for a Complex Producer

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

2. Change the `<registration required=>` and `<portlet-management required=>` attributes from `"true"` to `"false"`, It should look like the example in Listing 6-9:

Listing 6-9 Updated <service-config> Element for a Simple Producer

```
<service-config>
   <registration required="false" secure="false"/>
```

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

Since the markup `<markup secure=>` and service-description `<service-description secure=>` interfaces are mandatory, you don't need to specify any attributes.

3. Redeploy the project.

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