BEA WebLogic Portal®

Production Operations Guide

Version 10.0
Document Revised: March 2007

Contents

1. Introduction

Part I. Configuration and Deployment

2. Managing a Team Development Environment

3. Configuring a Portal Cluster

4. Deploying Portal Applications

Part II. Propagation

5. Developing a Propagation Strategy

6. Propagation Topics

7. Using Workshop for WebLogic Propagation Tools

8. Using the Propagation Ant Tasks

9. Propagation Ant Task Reference

10.Propagation Tips and Best Practices

11.Using the Export/Import Utility

12. Using the Datasync Web Application

A. Export/Import Utility Files

Introduction

The life cycle of a BEA WebLogic Portal® application requires careful planning and management. During its lifetime, a typical portal moves back and forth between development, staging, and production environments. The process of configuring and managing these environments, and of moving portals between them, is called production operations.

This chapter includes the following topics:

- What Is Production Operations?

- Overview of Production Operations

- Getting Started

What Is Production Operations?

Production operations encompasses the tools, procedures, methodologies, and best practices that allow you to manage the portal life cycle, including portal development, staging, and production environments. As Figure 1-1 shows, portals are typically developed in a *team development environment* by developers using Workshop for WebLogic. Portal components are then moved to a *staging environment*, where portal administrators use the WebLogic Portal Administration Console to create desktops, add entitlements, set up content repositories, and perform testing. The *production environment* is the live environment, where users access and interact with portal applications. The arrows between environments indicate that you can move portals and portal resources back and forth between each of these environments using propagation features provided with WebLogic Portal.

Figure 1-1 Typical WebLogic Portal Environments
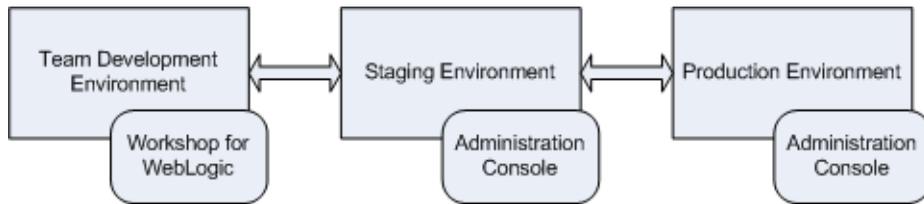
Like considering the architecture of a network or a software system, you should also consider and carefully plan how you will address production operations for your portal system. It is important to consider your particular portal system configuration, how your development team is organized, how you will test and configure portals, how your server is configured, and how you will plan to manage the life cycle of your portal applications. This guide describes the specific methodologies, tools, and best practices to help you achieve the goal of creating solid, manageable environments for portal development, staging, and production.

Overview of Production Operations

This section offers a brief introduction to the major components of production operations:

- Setting Up a Team Development Environment

- Configuring the Portal Cluster

- Building and Deploying the EAR File

- Propagating a Portal Application

- Performing Round-Trip Development

Setting Up a Team Development Environment

Team development of a WebLogic Portal revolves around good source control. Proper use of a source control management system has many benefits, such as close integration between team members, the ability to quickly scale the size of a development team, and protection against data loss.

Chapter 2, "Managing a Team Development Environment" shows you how to configure, store, and manage a common development domain, database data, and portal applications in source control, letting you quickly and consistently develop, build, and update your portal applications.

Configuring the Portal Cluster

By clustering a portal application, you can attain high availability and scalability for that application. Chapter 3, "Configuring a Portal Cluster," discusses how to choose a cluster architecture (single versus multi-cluster) and configure the clustered domain.

Building and Deploying the EAR File

Deployment refers to building an Enterprise archive file (EAR) and deploying it to a destination server. Chapter 4, "Deploying Portal Applications," describes how to prepare a portal application's deployment plans and deploy the EAR file.

Propagating a Portal Application

Propagation refers to the process of moving the database and LDAP contents of one portal domain environment to another. During the typical portal life cycle, portals are moved between the following environments:

- **Development** – In the development phase, developers use Workshop for WebLogic to create portals and portal components, such as portlets.

- **Staging** – In a staging environment, administrators use the Administration Console to build and configure portal desktops, create entitlements, and create content repositories.

- **Production** – A production, or live, environment can be modified by administrators using the Administration Console and customized by users using Visitor Tools.

BEA provides tools to help with portal propagation. These tools not only move database assets and LDAP information, but they also report differences and potential conflicts between the source and the target environments. You can define policies to automatically resolve conflicts, or an administrator can view a list of differences and decide the appropriate actions to take on a case-by-case basis. These tools are described in detail in this guide, and they include:

- The propagation tools, described in Chapter 7, "Using Workshop for WebLogic Propagation Tools" and Chapter 9, "Propagation Ant Task Reference."

- Ant tasks, described in Chapter 8, "Using the Propagation Ant Tasks" and Chapter 9, "Propagation Ant Task Reference."

This guide also helps you through the process of planning a strategy for propagation and provides detailed information on the best practices. See the following chapters for more information:

- Chapter 5, "Developing a Propagation Strategy"

- Chapter 6, "Propagation Topics"

Performing Round-Trip Development

Round-trip development refers to moving portal assets back and forth between a Workshop for WebLogic-based development environment and a staging environment where portal assets are assembled with the WebLogic Portal Administration Console and stored in a database. The Export/Import Utility lets you export portal assets from a database to `.portal`, `.page`, and `.book` files that can be loaded into Workshop for WebLogic. The utility also lets you import `.portal`, `.book`, and `.page` files into a database, as shown in Figure 1-2.

Figure 1-2 The Export/Import Utility Allows Round-Trip Development
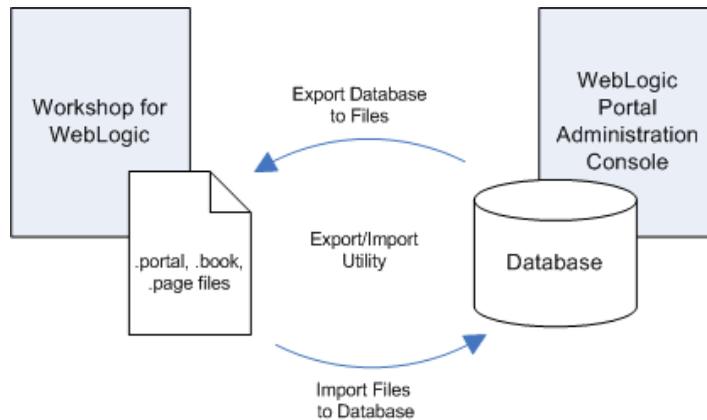
Tip: The Export/Import Utility is also known as the Xip tool (pronounced "zip"). Typically, developers use this utility to move assets back and forth between a development and a staging environment.

In addition, the Export/Import Utility allows you to:

- Merge `.portal` files into a database

- Specify rules to determine how objects are merged

- Specify scoping rules

The Export/Import Utility is described in Chapter 11, "Using the Export/Import Utility."

Getting Started

This section contains the following topics:

- Using this Guide
- Related Guides

Using this Guide

Part I Configuration and Deployment includes topics of interest to managers, developers, and administrators. This part includes information on setting up a team development environment, deploying an EAR file, and configuring a portal cluster.

Part II Propagation includes information on propagating portals between staging and production environments, and round-trip development. In addition, this part includes details on script-based propagation.

Related Guides

For an in-depth discussion of the WebLogic Portal life cycle, see *BEA WebLogic Portal Overview*.

Introduction

Part I   Configuration and Deployment

Part I includes the following chapters:

- Managing a Team Development Environment

  This chapter explains how to configure, store, and manage a common development domain, database data, and portal applications in source control, letting you quickly and consistently develop, build, and update your portal applications.

- Deploying Portal Applications

  This chapter discusses how to prepare your application's Enterprise archive (EAR) file and deploy it.

- Configuring a Portal Cluster

  This chapter describes how to set up a cluster across which your portal application is deployed.

CHAPTER **2**

Managing a Team Development Environment

This chapter discusses how to configure and manage a common environment for WebLogic Portal development. A common development environment allows developers to share the same WebLogic Portal domain, database, and application configuration, and maintain these elements in a source control system. A well-planned team environment allows you to quickly and consistently develop, build, and update your WebLogic Portal applications.

This chapter contains the following sections:

- Introduction

- Choosing a Source Control Vendor

- Creating a Shared WebLogic Portal Domain

- Managing Databases

- Creating and Sharing the Portal Application

- Using J2EE Shared Libraries in a Team Environment

- Sharing Portal Resources: Sample Scenario

- WebLogic Portal Coding Best Practices

- Managing Binary Files in Source Control

- Configuring Facets

- Alternative Domain Sharing Techniques

Introduction

The basic tasks required to configure a team development environment for WebLogic Portal include:

1. Choosing a Source Control Vendor

2. Creating a Shared WebLogic Portal Domain

3. Managing Databases

4. Creating and Sharing the Portal Application

These tasks are described in the following sections.

Choosing a Source Control Vendor

Team development of a WebLogic Portal web site revolves around good source control. Proper use of a source control system has many benefits, such as close integration between team members, the ability to quickly scale the size of a development team, and protection against data loss.

There are a number of source control providers, such as CVS, Subversion (SVN), Perforce, StarTeam, Visual Source Safe (VSS), and PVCS. This chapter assists you with using any of those vendors. However, each vendor has different characteristics when it comes to storing code. An important consideration when choosing your source control management system for team development of portal applications is that it must support an unreserved checkout model for files. This is because there are numerous files in the domain and application that need to be checked into source control management but must be writable by the server. An unreserved checkout model means that multiple users can check out and edit a file simultaneously–individual users do not "lock" the file from other users when they check it out.

Tip: For information on sharing project files using Workshop for WebLogic's Eclipse-based integrated source control features, see the Workshop for WebLogic document "Working with Source Control."

Creating a Shared WebLogic Portal Domain

This section describes the basic steps in creating a shared WebLogic Portal domain, and includes the following topics:

- What is a WebLogic Portal Domain?

- Getting Started

- Creating a WebLogic Portal Domain Template

- Creating the Shared Domain

- Starting WebLogic Server

- Configuring and Tuning the Domain

What is a WebLogic Portal Domain?

A WebLogic Portal domain is the foundation upon which you build portals. The domain includes the configuration files, database, and scripts that define and run your server environment, provides a default security realm and predefined system administrators, and provides server administration tools. The domain also includes files and services for building portals and related functionality.

A basic domain infrastructure consists of one Administration Server and optional Managed Servers and clusters. For a more detailed description of these components, as well as a more complete introduction to domains, see the WebLogic Server document, "Creating WebLogic Domains Using the Configuration Wizard."

In a team development environment, domain-related files are stored in source control, and checked out to individual development machines. Team members share these domain files using source control so that all modifications to existing deployed applications, the addition of new applications, and other settings stored in configuration files and scripts can be shared.

Tip: It is a recommended practice to use domain templates, rather than the domain files themselves, to create and distribute domains to members of a development team. Check these templates into source control. For information on creating domain templates, see "Creating a WebLogic Portal Domain Template" on page 2-4.

Getting Started

Every developer on the team must first install WebLogic Portal on their development machines. It is a recommended practice, but not required, that all developers install WebLogic Portal in a common home directory.

Creating a WebLogic Portal Domain Template

The recommended approach to distributing a commonly configured domain among team members is to create a domain template and check it in to source control.

Tip: If members of your team only require a default WebLogic Portal domain, they can simply create the default WebLogic Portal domain using the WebLogic Configuration Wizard on their individual machines. In this case, no domain files need to be checked into source control. On the other hand, if you need special domain configurations, such as JDBC, JMS, or extra shared libraries, a custom domain template is recommended. For detailed information on using the Configuration Wizard, see the WebLogic Server document "Creating WebLogic Configurations Using the Configuration Wizard."

A domain template is used to create a new domain. A domain template defines the full set of resources within a domain, including infrastructure components, applications, services, security options, and general environment and operating system parameters. You can create a domain template from an existing template or from an existing domain. Given a shared domain template, all developers on a team can effectively create the same domain configuration on their individual development machines.

For detailed information on creating domain templates, see the WebLogic Server document "Creating Configuration Templates Using the WebLogic Configuration Template Builder."

Creating the Shared Domain

After a domain template is created and checked into source control, each developer can check out the template and create their own local domain. Because developers use the same template to create their domains, the contents of their domains are equivalent.

Each developer uses the domain template to create a local domain using either the WebLogic Configuration Wizard or a script. For detailed information on using the Configuration Wizard, see the WebLogic Server document "Creating WebLogic Configurations Using the Configuration Wizard." For an overview of the files that are installed with a domain, see the WebLogic Server document "Domain Configuration Files."

Tip: For detailed information on building a domain programmatically with a script, see the WebLogic Server document, "Creating and Configuring WebLogic Domains Using WLST Offline."

Tip: A common activity in development is the creation of a base set of users that are used to test the system. See also "Updating Users, Groups, Roles, and Entitlements" on page 2-27.

Starting WebLogic Server

Start WebLogic Server using the domain's `DOMAIN_ROOT/bin/startWeblogic` command.

Configuring and Tuning the Domain

With the server running, you can configure the domain. Using the WebLogic Server Administration Console (`http://server:port/console`), you can set up the domain to support the development effort, including the addition of needed data sources. For information on server configuration, see the WebLogic Server document "System Administration for BEA WebLogic Server."

Common tuning activities for a development domain include setting the server logging mode to **Info** from **Warn** (for more verbose console output and outputting JVM messages to the console). In addition, you can limit the maximum size of the log files.

The changes you make are captured in the file `DOMAIN_ROOT/config/config.xml`. You can check this file into source control to share it with members of the development team. Note, however, that the `config.xml` file contains hard-coded paths that each developer might need to modify locally. One technique for sharing this file is to create and check in a string-substitution template and provide a script that each developer can run locally to create their own `config.xml` file.

Managing Databases

WebLogic Portal stores much of its configuration information in the database, and there are occasions where development teams need to share access to this configuration. However, WebLogic Portal does not support running multiple instances of a portal server against the same single database or database schema. Although the default database for a WebLogic Portal domain is PointBase, you might require an Enterprise-quality database for development efforts.

For detailed information on database management for WebLogic Portal, including information on size restrictions imposed by the evaluation version of PointBase that is distributed with WebLogic Portal, see the *WebLogic Portal Database Administration Guide.* For more

information on working with the PointBase database, see "Using the PointBase Database" on page 2-7.

This section includes these sections:

- Developing Against an Enterprise-Quality Database

- Using Different Databases in Development and Production

- Knowing When You are Making Changes to the Database

- Using the PointBase Database

- Removing Unneeded Database Components

Developing Against an Enterprise-Quality Database

Rather than share the PointBase database between developers as a binary files, it is common for each developer to work against their own portal database using Oracle, SQL Server, or another Enterprise-quality database.

Note: For Oracle and DB2, a separate database schema for each developer on a development database is recommended. For Sybase and SQL Server, a separate database and database log file for each developer on a development database instance is recommended. For MySQL a separate database for each developer is recommended.

Each development domain is configured through the domain template to use a specific database, listed in multiple XML files in the directory DOMAIN_HOME/config/jdbc. For details, see the *WebLogic Portal Database Administration Guide*.

Using Different Databases in Development and Production

Generally, it is a good practice to use the same Enterprise-quality DBMS in development that you plan to use in staging and production. For example, if you plan to deploy your application on Oracle it is a good practice to develop your application on Oracle as well. This methodology allows greater performance and easier maintenance of a baseline of data (with proper support from a database administrator and scripts).

It is also possible to use one type of database in development and another in staging and production and use the propagation tools to move data between them. In this scenario, developers might use PointBase in development while the staging and production systems use an Enterprise quality database, such as Oracle. Using the propagation tools, you can export the database inventory from the development system and import it into the staging or production system. For

detailed information on using the propagation tools, see Chapter 7, "Using Workshop for WebLogic Propagation Tools."

Knowing When You are Making Changes to the Database

In general, most activities that are accomplished using the WebLogic Portal Administration Console are persisted to the database, with the exception of entitlements, which are persisted to the embedded LDAP. However, there may be times when you want to develop using test users with user properties assigned to them.

These properties are stored in the database. In addition, service administration configurations are persisted in the application's deployment plan, not to the database. Content repository configurations are also not persisted in the database, although actual content stored in the BEA repository is in the database. (If you are using a File System Repository, only the content metadata is stored in the database.)

For detailed information on entitlements, see the *WebLogic Portal Security Guide*.

Using the PointBase Database

A development licence for PointBase is installed with WebLogic Portal. Note that with the development license, the database size is limited to 30 MB. To increase this limit, you need to purchase a full license. Also, note that PointBase stores data in binary files that grow as you use the database. For information on handling binary files in a team environment, see "Managing Binary Files in Source Control" on page 2-26.

To prevent the PointBase database server from starting and stopping automatically, set your domain's POINTBASE_FLAG to false (the default is true). To do this:

On a Windows system, in the file DOMAIN_HOME/bin/setDomainEnv.cmd, enter:

```
set POINTBASE_FLAG=false
```

On a UNIX system, in the file DOMAIN_HOME/bin/setDomainEnv.sh, enter:

```
POINTBASE_FLAG="false"
```

You can also achieve the same result by running the startWebLogic command with the nopointbase option.

Removing Unneeded Database Components

When you create a WebLogic Portal domain using the default domain template, a datasource called samplesDataSource is automatically included in the domain. You can remove this datasource from the domain if you wish. To remove it, use the WebLogic Server Administration Console. From the main page of the Administration Console, select **JDBC > Data Sources**. Click **Lock & Edit**, then select **samplesDataSource** from the table and click **Delete**.

A sample PointBase database is also installed with the default WebLogic Portal domain. This database is typically used for example and testing purposes only. If all of your domain's data sources are pointed to a non-PointBase database, it is recommended that you remove this database before deploying your application to a production server. To remove the database, simply delete the following files from your domain:

- `weblogic_eval.*`

- `pointbase.*`

Creating and Sharing the Portal Application

After configuring the portal domain, you need to create a new portal application that will be shared by all members of the development team.

Tip: To plan for the sharing of portal application code among team members, it is important to understand the role of Shared J2EE Libraries in a WebLogic Portal application. See the WebLogic Server document "Creating Shared J2EE Libraries and Optional Packages" for information on this important topic. Shared Libraries are also discussed in the *WebLogic Portal Development Guide* and "Deploying J2EE Shared Libraries" on page 4-10.

Like domain creation, application creation occurs in several steps. These steps are explained in this section. They include:

1. Create or Locate the Eclipse Workspace Directory

2. Create a Portal EAR Project

3. Create Portal Web Projects

4. Create a Datasync Project (Optional)

5. Check in the Portal Application

6. Check Out the Workshop for WebLogic Application

Create or Locate the Eclipse Workspace Directory

All projects created with Workshop for WebLogic are created inside a Workspace directory. For more information on creating workspaces and projects, see the *WebLogic Portal Development Guide*.

Create a Portal EAR Project

Use Workshop for WebLogic to create one or more Portal EAR Projects. An EAR Project primarily consists of configuration files that reference Web applications and J2EE Shared Libraries.

When you create an EAR project, you select the Project Facets you want to include in the project, including a set of WebLogic Portal facets, as shown in Figure 2-1.

Figure 2-1 WebLogic Portal Facets
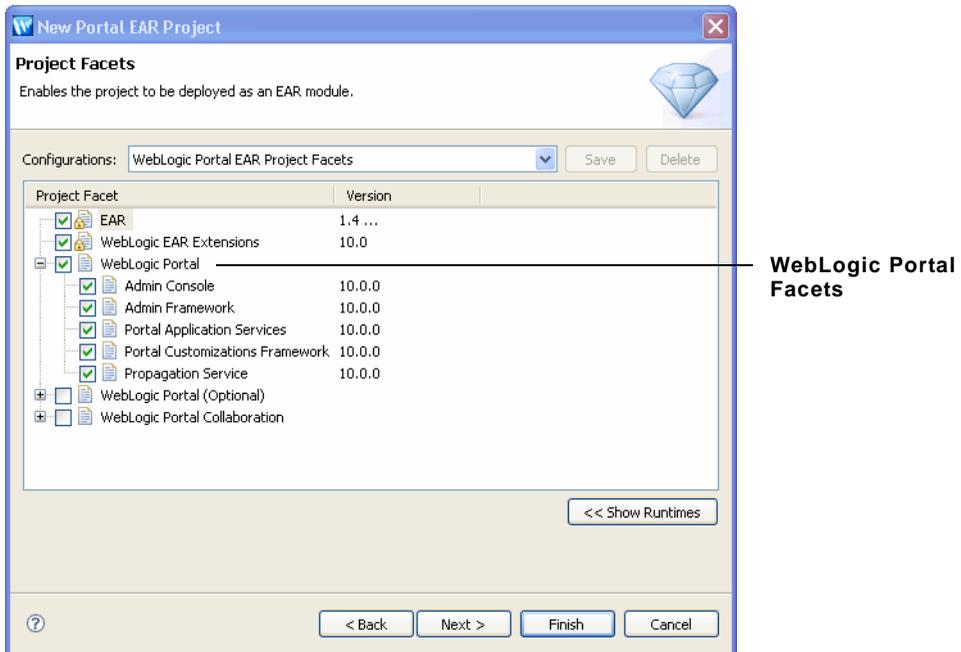
A facet is a convenient way to group a set of J2EE Shared Libraries and IDE functionality that are required for a specific feature. For example, the **Admin Console** facet groups the J2EE Shared Libraries that are required to deploy and run the WebLogic Portal Administration Console. If you do not want to include a facet, you can deselect it. Note that some facets depend on other facets. If you try to remove a facet that has dependencies, the dialog box alerts you and prohibits you from making that particular change.

Tip: It is recommended that, for development purposes, you select all of the WebLogic Portal facets when you create a Portal EAR Project. You can selectively remove some facets before you move your portal to a production environment. For example, you can remove the WebLogic Portal Administration Console before you deploy to your production environment. For information on adding and removing facets, see the *WebLogic Portal Development Guide*. See also "Using J2EE Shared Libraries in a Team Environment" on page 2-14.

For more detailed information on creating Portal EAR Projects, see the *WebLogic Portal Development Guide*.

Create Portal Web Projects

Use Workshop for WebLogic to create Portal Web Projects. A Portal Web Project includes your portal's source code and configuration files that reference J2EE Shared Libraries used by the project.

When you create a Web project, you select the Project Facets you want to include in the project, including the WebLogic Portal facets, as shown in Figure 2-2.

Figure 2-2 WebLogic Portal Facets
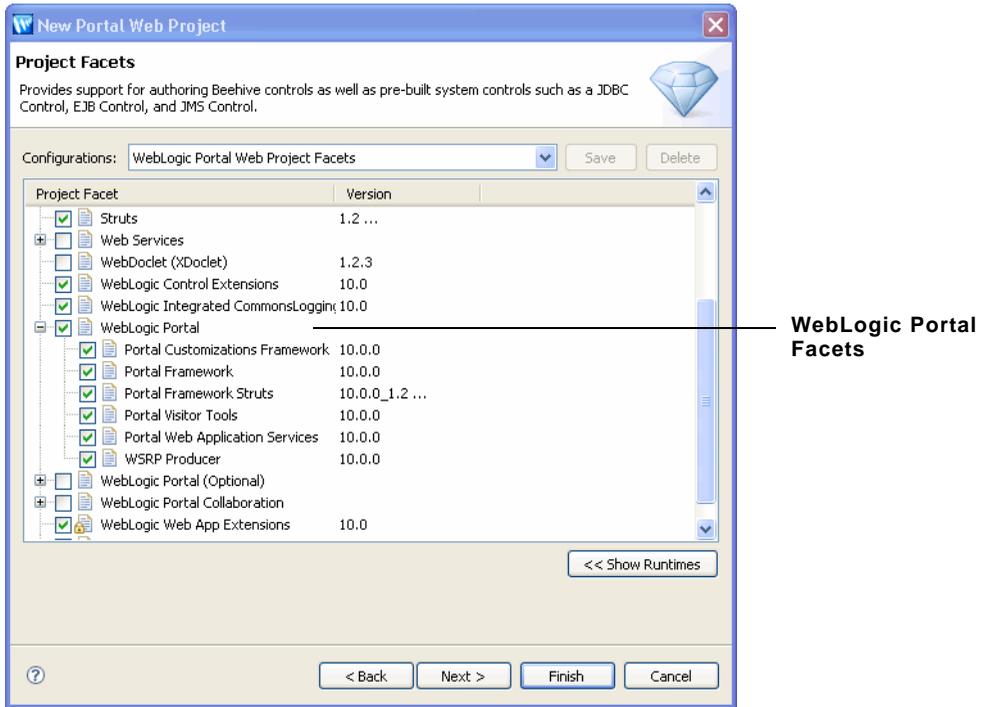
Tip: It is recommended that, for development purposes, you select all of the WebLogic Portal facets when you create a Portal Web Project. You can selectively remove some facet components (features) when you move your portal to a production environment. For more information, see "Using J2EE Shared Libraries in a Team Environment" on page 2-14.

Any application code you write is stored in files within the web project. In a typical application, your code is placed in the `WebContent` directory of the web project. Any domain and application specific code supplied by BEA is stored in J2EE Shared Libraries and referenced by your application.

For more detailed information on creating Portal Web Projects, see the *WebLogic Portal Development Guide*.

Create a Datasync Project (Optional)

A datasync project is an optional project that stores general purpose portal services data that are used in the development of personalized applications and portals. These portal services include user profiles, session properties, campaigns and others.

Tip: You can share a single datasync project among several EAR projects if you wish.

Datasync data must be associated with an EAR to be deployed. If a datasync project is not associated with an EAR, when the EAR is deployed, the datasync data will not be deployed.

You can add a new datasync project to an EAR in two ways:

- When the datasync project is created, you can associate it with an EAR using the Datasync Project Wizard.

- You can associate a datasync project with an EAR by right-clicking the datasync project in WorkSpace Studio and selecting **Properties**. In the **Properties for data** dialog, select Datasync > EAR Projects. In the Ear Projects panel, select the EAR file to add the datasync project to.

For more detailed information on creating Datasync Projects and the Datasync Project Wizard, see the *WebLogic Portal Development Guide*.

Check in the Portal Application

The code you write is physically separated from the domain and application code BEA provides in J2EE Shared Libraries. You only need to store the application code written by you and members of your team in a source control system. Be default, your application code is placed in the `WebContent` directory of your web application. Any Java source code is placed in the `src` directory. Eclipse also stores project settings and other configurations in the project. These files include:

- `.project` files – Contains the Eclipse project information.

- `.classpath` – Contains the Java classpath settings.

- `.settings` – Contains a list of files for the project's facets, J2EE settings, Datasync settings, and other project-specific information.

- `.datasync-project.propeties` – Used in Datasync projects.

As new components are created, many of these new components need to be checked into source control. Developers need to be aware of which files that are created need to be shared in source control.

Exclude from source control any Java output directories that are specific to the web project. Typically, these directories are named `build` or `bin`. Also, avoid checking in any files that contain hard coded paths.

Tip: For more information on files to exclude from source control and on creating a portable workspace ZIP file, see the Workshop for WebLogic document "Working with Source Control."

Three files that are commonly updated during development and need to be checked into source control include the following. These files are located in the enterprise application's `META-INF` directory.

- `application.xml` – Lists the web applications associated with the enterprise application.

- `weblogic-application.xml` – Lists the J2EE Shared Libraries used by the enterprise application.

- `content-config.xml` – Specifies the default content repository used by the application.

Tip: If you want to modify a file that is inside a J2EE Shared Library, you can copy it to your filesystem and modify it. From that point on, the file-based copy takes precedence over the version stored in the J2EE Shared Library. In a development environment, be careful to check any copied J2EE Shared Library files into source control, so that they can be shared by other developers. Not all files can be copied from a shared library. For example, `.class` and `.jar` files cannot be copied.

Check Out the Workshop for WebLogic Application

The fundamental idea when working with source control management and a Workshop for WebLogic application is that developers must be able to check out the application, initiate a build, and start the server without error.

When the EAR project is deployed to the domain by Workshop for WebLogic, it is registered in the domain's `DOMAIN_ROOT/config/config.xml`. This deployment happens automatically when the server is started and the application is built. At this point, the application is added to `config.xml` in a new XML block.

Listing 2-1 shows the block added to `config.xml` for an enterprise application named `myPortalEAR`.

Listing 2-1   Application Added to config.xml File

```
<app-deployment>
    <name>myPortalEAR</name>
    <target>AdminServer</target>
    <module-type>ear</module-type>
    <source-path>D:\users\projects\applications\myWorkspace\.metadata\.plugins\
        org.eclipse.core.resources\.projects\myPortalEAR\beadep
    </source-path>
    <security-dd-model>DDOnly</security-dd-model>
</app-deployment>
```

Because Workshop for WebLogic updates the `config.xml` for the domain automatically, it is not necessary to check a `config.xml` that contains the application name XML block back into source control. Instead, a developer checks out the application, performs a build, and starts the server against a domain without this application reference. The developer's application is then published to the server. Workshop for WebLogic automatically updates it if necessary to add or remove components.

After checking out an application from source control, each developer needs to import it into Workshop for WebLogic. To do this, select **File > Import**. In the Import dialog, select **Existing Projects into Workspace**. Then, follow the online help instructions to locate and import the project.

Tip: Refer to the Eclipse documentation for additional information on sharing projects in Eclipse.

Using J2EE Shared Libraries in a Team Environment

This section includes these topics:

- Overview
- Shared Library Rules of Precedence

● Deployment Descriptors and Shared Libraries

Overview

A J2EE Shared Library is a reusable portion of a J2EE application or web application. At the enterprise application level, a J2EE Shared Library is an EAR file that can include Java classes, EJB deployments, and web applications. At the web application level, a J2EE Shared Library is a WAR file that can include servlets, JSPs, and tag libraries. The difference between a standard EAR or WAR file and a J2EE Shared Library is that shared libraries can be included in an application by *reference*, and multiple applications can reference a single J2EE Shared Library.

One of the most useful aspects of shared libraries for WebLogic Portal development teams is that the code developed by your team and the code that is distributed by BEA remain physically separated. When a WebLogic Portal upgrade or patch is distributed as shared libraries, all you need to do is add the new libraries to the installation directory. The referencing applications, and the code written by your developers, automatically pick up the new modules when the application is redeployed.

Tip: For additional detailed information on J2EE Shared Libraries, see the WebLogic Server document "Creating Shared J2EE Libraries and Optional Packages" and the *WebLogic Portal Development Guide*.

Shared Library Rules of Precedence

Your WebLogic Portal application can reference multiple shared libraries. In turn, libraries can reference other libraries, and so on. Because the J2EE Shared Library code and your own application code is assembled at runtime, rules must exist to resolve potential conflicts. These rules are:

● Any file that is located in your application takes precedence over a file that is in a J2EE Shared Library.

● Conflicts arising between referenced libraries are resolved based on the order in which the libraries are specified in the META-INF/weblogic-application.xml file (for enterprise applications) or the WEB-INF/weblogic.xml file (for web applications).

These precedence rules have an important implication for the development team. For example, the team can choose to copy specific files, such as CSS files, from a J2EE Shared Library. If developers change those files in their development areas, those changes take precedence over the original shared library versions.

Tip: Where possible, copy resources for shared libraries to a new name in your application. This practice can avoid confusion about which version is being used: the local copy or the shared library version. Sometimes this is not possible, and the precedence rules then apply.

Deployment Descriptors and Shared Libraries

In addition to code and other modules and resources, shared libraries include deployment descriptors. The deployment descriptors describe the contents of the library. The following example illustrates the basic structure of an enterprise application that references a J2EE Shared Library.

For this example, assume there is an enterprise application called `myApp.ear` that has the structure shown in Listing 2-2. The application includes two modules, `myEjb.jar` and `myWebApp.war`, plus some additional Java code in `myClasses.jar`.

Listing 2-2 Example Application myApp.ear

```
META-INF/application.xml

    <module>
      <ejb> myEjb.jar </ejb>
    </module>
    <module>
      <web>
        <web-uri>myWebApp.war</web-uri>
        <context-root>myWebApp</context-root>
      </web>
    </module>

META-INF/weblogic-application.xml

    <library-ref>
      <library-name>AppLibOne</library-name>
    </library-ref>

APP-INF/lib/myClasses.jar
myEjb.jar
myWebApp.war
```

Note the element shown in bold in the `META-INF/weblogic-application.xml` descriptor. The `<library-ref>` element specifies a reference to a J2EE Shared Library called `AppLibOne`.

Listing 2-3 shows the actual J2EE Shared Library, `AppLibOne.ear`, referenced by the enterprise application. As you can see, this library includes a `META-INF/MANIFEST.MF` file, shown in bold type, which includes the library name and version information. After a shared library is deployed, it is through this manifest that the server is able to identify it and to assemble it into the deployed application.

Tip: See "Shared Library Manifest File Contents" on page 2-18 for detailed information on the manifest file. For more information on deploying shared libraries, see "Deploying J2EE Shared Libraries" on page 4-10.

Listing 2-3 Shared Library AppLibOne.ear

META-INF/MANIFEST.MF

```
Extension-Name: AppLibOne
Specification-Version: 1.0
Implementation-Version: 1.0
```

META-INF/application.xml

```
<module>
    <ejb> libEjb.jar </ejb>
</module>
```

APP-INF/lib/code.jar

libEjb.jar

Listing 2-4 shows the effective configuration of the final deployed application, after the deployment descriptors in the libraries have all been read and interpreted. The server deploys two EJB JAR files. The file `myEjb.jar` comes from the `myApp.ear` archive, and the other `libEjb.jar` comes from the J2EE Shared Library. Furthermore, the application's classpath is also ordered so that the classes in `myApp`'s `myClasses.jar` override any classes from the `code.jar` file in the `AppLibOne` J2EE Shared Library.

Listing 2-4   Final Deployed Application

```
META-INF/application.xml
```

```
<module>
  <ejb> myEjb.jar </ejb>
</module>
<module>
  <web>
    <web-uri>myWebApp.war</web-uri>
    <context-root>myWebApp</context-root>
  </web>
</module>
<module>
  <ejb> libEjb.jar </ejb>
</module>
```

```
APP-INF/lib/myClasses.jar
```

```
APP-INF/lib/code.jar
```

```
myEjb.jar
```

```
myWebApp.war
```

```
libEjb.jar
```

Shared Library Manifest File Contents

You can enable a J2EE module, such a WAR or EAR, to be a J2EE Shared Library by creating or modifying the archive's `META-INF/MANIFEST.MF` file. This section explains the variables that you can put in the manifest file of a J2EE Shared Library.

A sample `MANIFEST.MF` file is shown in the following listing:

```
META-INF/MANIFEST.MF

    Extension-Name: SomePortlets
    Specification-Version: 1.0
    Implementation-Version: 1.0
```

Table 2-1describes the contents of the file:

Table 2-1 MANIFEST.MF Variables

`Extension-Name`	An optional string value that identifies the name of the shared J2EE library. It is a best practice to always specify a name; otherwise, a name is automatically generated based on the deployment name of the library.
`Specification-Version`	An optional String value that defines the specification version of the shared J2EE library.
`Implementation-Version`	An optional String value that defines the code implementation version of the shared J2EE library. You can provide an Implementation-Version only if you have also defined a Specification-Version.

Tip: For a more detailed description of creating a `MANIFEST.MF` file, see the WebLogic Server document "Creating Shared J2EE Libraries".

In addition to the `MANIFEST.MF` variables shown in Table 2-1, you can also use the following variables to explicitly include or exclude particular files from the **Copy to Project** function. This function lets you copy certain files from a shared library directly to your project workspace. To explicitly include or exclude files from **Copy to Project**, use these variables:

Table 2-2 MANIFEST.MF Include and Exclude Variables

`BEA-EXTRACT-EXCLUDE`	All files except those matching the specified regular expression will be extracted.
`BEA-EXTRACT-INCLUDE`	Only the files matching the specified regular expression will be extracted.

If you specify neither include nor exclude, then all files in the library that can be copied will be copied. If both include and exclude are provided, then first all the files that match the include's regular expression will be found and the excludes will be removed from the list of included files.

The format of the `BEA-EXTRACT-INCLUDE/EXCLUDE` is a regular expression (from javax.util.regex.Pattern) specifies what to exclude or include. For example if the library's directory structure looks like the one shown in Listing 2-5:

Listing 2-5   Sample Library Structure

```
/WEB-INF/web.xml
/WEB-INF/classes/log4j.properties
/includes/test.jsp
/includes/bob.jsp
/css/some.css
```

Then the manifest entry:

```
BEA-EXTRACT-INCLUDE: (/WEB-INF/[_0-9a-zA-Z ]*)|(css)
```

extracts all files that match `/WEB-INF/` and any number of letters

that have a "_" or 0-9 or a-z or A-Z, *or* any file that contains `/css/`. In this case, the following files will be extracted during a **Copy to Project** operation:

```
/WEB-INF/web.xml
/WEB-INF/classes/log4j.properties
/css/some.css
```

The regular expression is not matched against the full path of the file so in the example above, the `/css/` matches all files that are in a path that include "css" somewhere; therefore, these example directories/files would also match the regular expression:

```
/adirectory1/css/bob.joe
/main.css
```

If you want to match against the full path of the file, use the `^` and `$` regular expression characters. For example: `^/hiddenFolder/.*$`

Listing 2-6 illustrates a manifest that includes both variables:

Listing 2-6   Manifest File Using Both Include and Exclude Directives

```
Extension-Name: p13n-app-lib
Specification-Version: 10.0.0
Implementation-Version: 10.0.0
BEA-EXTRACT-INCLUDE: (/META-INF/p13n-config.xml)|(css)
BEA-EXTRACT-EXCLUDE: xml
```

Sharing Portal Resources: Sample Scenario

Shared libraries provide a convenient mechanism for development teams to share the portal resources that they develop with other teams.

This section includes these topics:

- Introduction

- Packaging Resources to Share

- Receiving and Incorporating Shared Resources

Introduction

For example, suppose a team environment consists of one or more portlet development teams and a team that is responsible for assembling and maintaining the overall portal. Figure 2-3 illustrates the example scenario. The portal development team delivers a Shared Library (WAR) file to a portlet development team. This library file contains the portal and its associated resources, such as the portal look & feel. The portlet team receives this file and imports it as a Shared Library into its project space, making it available to the team members and providing a portal in which to test their portlets.

After the portlet team builds a set of portlets, they deliver a Shared Library (WAR) file back to the portal team. The portal team receives the WAR, imports it as a module, and adds the portlets to the portal. For detailed information on creating J2EE Shared Libraries, see the WebLogic Server document "Creating J2EE Libraries and Optional Packages."

Figure 2-3 Sharing Portal Resources in a Team Environment
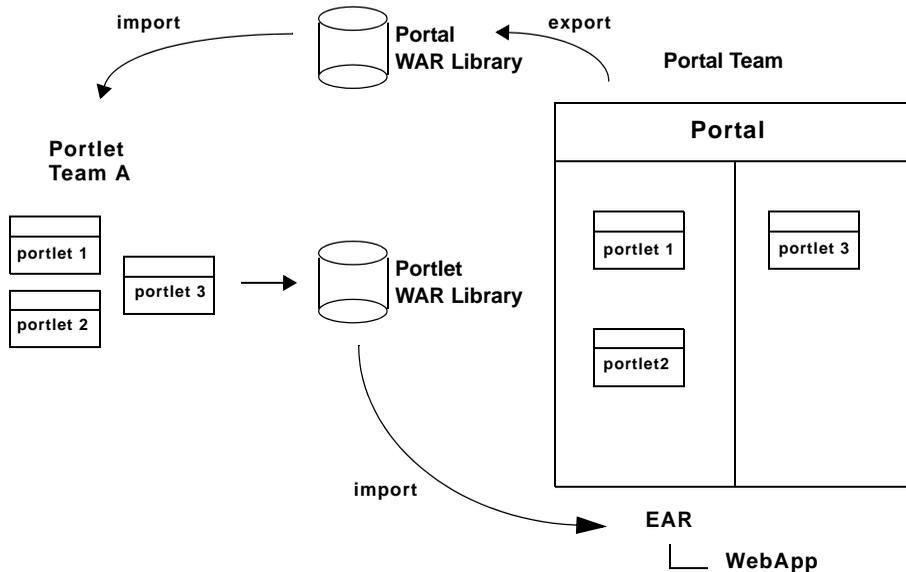
Packaging Resources to Share

The basic steps involved in sharing resources as shared libraries includes the following:

1. The portlet team develops and tests portlets using a test portal environment.

2. The portlet team adds appropriate stanzas to the Shared Library WAR file's
   `MANIFEST/MANIFEST.MF` file to enable the WAR as a J2EE Shared Library. See "Using J2EE
   Shared Libraries in a Team Environment" on page 2-14 for information on enabling a WAR
   as a J2EE Shared Library. For example:

```
META-INF/MANIFEST.MF

   Extension-Name: SomePortlets
   Specification-Version: 1.0
   Implementation-Version: 1.0
```

3. The portlet team uses Workshop for WebLogic to export a project as a WAR file (select **File
   > Export**).

4. The portlet team removes from the Shared Library WAR file any files and code that are not
   strictly required by the portlets. For instance, any test files used by the development team can
   be excluded.

Tip: Choose a naming convention for your portlet development that includes consistent subdirectory names. A consistent naming convention simplifies the process of packaging and delivering your source files.

5. The portlet team sends the Shared Library WAR file to the portal team.

6. The portal team imports the library, as explained in the next section.

Receiving and Incorporating Shared Resources

Upon receiving the Shared Library WAR file, the portal team references the Shared Library WAR file (and, optionally, its version number) in the appropriate configuration files. References to the Shared Library WAR file go in the portal web application's `weblogic.xml` and the domain's `config.xml` files.

The `weblogic.xml` file contains the library name (and, optionally, version number) and the domain's `config.xml` file contains the actual path of the Shared Library WAR file. See "Using J2EE Shared Libraries in a Team Environment" on page 2-14 for information on referencing a J2EE Shared Library.

Tip: Before beginning their development phase, the portlet team might receive from the portal team a set of look & feel files, libraries, and menus to use with their portlets. If these resources are delivered to the portlet team in a J2EE Shared Library file, the portlet team can simply receive and install it by following the same procedures described in this section. In this case, the portlet team may have fewer items to remove from its Shared Library WAR file.

Importing the Shared Library into Workshop for WebLogic

You need to add the library to the Java Build Path of your project, as follows:

1. Right-click the project in the Package Explorer and select **Build Path > Add Libraries**.

2. In the Add Library dialog, select **WebLogic J2EE Library** and click **Next**.

3. In the WebLogic J2EE Library dialog, browse to the library file that you want to add and select it. If you want to specify a version, enter the appropriate version information in the dialog, and click **Finish**.

Importing the Shared Library into a Deployed Application

To incorporate the portlets into a deployed portal, the portal team uses the WebLogic Server Console to deploy the WAR file as a J2EE Shared Library.

WebLogic Portal Coding Best Practices

This section provides guidance for managing portal application source code in a team development environment.

This section includes the following sections:

- Sharing Java Projects

- Supporting Cross-Platform Development

- Editing Definition Labels for Portal Components

- Testing a Cluster Configuration

Sharing Java Projects

Tip: For information on sharing project files using Workshop for WebLogic's Eclipse-based integrated source control features, see the Workshop for WebLogic document "Working with Source Control."

If you have a number of general-purpose Java libraries that will be used by your portals, it is recommended that they be stored in a Java project inside the portal Enterprise archive or packaged as J2EE Shared Libraries. This enables portability of your Java libraries across multiple instances of the server and is a convenient mechanism for packaging libraries for reuse and sharing.

The best practice is to place a JAR file containing your Java libraries in the `APP-INF/lib` directory of your enterprise application, or configure the JAR file as a J2EE Shared Library. For detailed information on creating J2EE Shared Libraries, see the WebLogic Server document "Creating J2EE Libraries and Optional Packages."

Supporting Cross-Platform Development

When coding to develop and deploy in a cross-platform environment, observe the following best practices:

- Do not use spaces in filenames.

- Keep pathnames short.

- When possible, do not hard code pathnames.

- Use forward slashes (/) in path strings when possible.

- Be aware of the difference between case-sensitive operating systems (UNIX) and other operating systems, such as Windows. For example, you could create a file called `myPortletContent.jsp` and specify the file `MyPortletContent.jsp` as the Content URI on windows without problems. However, when this same application is deployed on UNIX, an error that the file `MyPortletContent.jsp` cannot be found is generated.

Editing Definition Labels for Portal Components

A unique identifier called a definition label is generated automatically for each book, page, and portlet that you add to a portal in Workshop for WebLogic. You can view the definition label for a component in Workshop for WebLogic in the Properties view.

With multiple developers creating new portal components, it is possible that different components can have the same automatically generated definition label. To avoid duplicate definition labels, manually change the definition label for each new component using your own naming conventions.

For information on modifying definition labels, see the *Portal Development Guide*.

WARNING: Once you have used the propagation tools to propagate changes among your environments, it is very important that you do not change the definition labels for portlets, pages, and books. The propagation tools use definition labels and instance labels to identify differences between source and destination systems; inconsistent results might occur if you change these labels after propagating a portal.

Testing a Cluster Configuration

Any code you write should be tested often in a clustered environment. Also, keep session data to a manageable size and configure your web applications to support session sharing across the

cluster. Be sure that session data is serializable. For clustering information, see Chapter 3, "Configuring a Portal Cluster."

Tip: WebLogic Server provides a session monitor tool that is useful for debugging HTTP session problems. See the WebLogic Server document "Class SessionMonitor" for more information.

Managing Binary Files in Source Control

A number of binary files in the WebLogic Server domain need to be checked into source control management for the domain to function properly. Some of these files, such as database files, can be modified during the course of WebLogic Portal development.

These binary files may change for various reasons: user-initiated reasons, automatic growth of index files, and so on. It is important to understand what these files are, why they change, and when to check them in and out.

This section explains how to determine when you need to update specific binary files in source control management. Some of these files include: LDAP files, security-related files, and database configuration files.

This section includes these topics:

- General Procedure for Working with Binary Files
- Updating Users, Groups, Roles, and Entitlements
- Updating Other Security-Related Files

General Procedure for Working with Binary Files

With all binary files, there is a consistent process to follow when you make changes to them so they can be shared in source control. To reduce the chances of merge conflicts over the project life cycle, it is recommended that changes to binaries be initiated consistently by a single user.

If, for any reason, you need to modify domain binary file(s) in source control, follow this procedure:

1. Stop the server.

2. Perform a clean checkout of the binary files from source control to ensure you are working from a common base.

3. Start the server.

4. Modify the configuration stored in the binary file(s).

5. Stop the server.

6. Check-in any modified binary files to source control management.

7. Test a clean checkout from another machine.

Updating Users, Groups, Roles, and Entitlements

A common activity in development is the creation of a base set of users and groups that are used to test the system. By default, WebLogic Server stores users and groups in the PointBase RDMBS. A relationship exists between LDAP policy data and database data to support user entitlements. An embedded LDAP server is provided with WebLogic Portal. This LDAP server persists its data store to the filesystem in the
`DOMAIN_HOME/servers/AdminServerName/data/ldap` directory.

For information on BEA's LDAP server, see the WebLogic Server document *Managing the Embedded LDAP Server*.

Because the LDAP server contains information, including role and security policies, that needs to be shared by team members, check the files in the LDAP directory into source control, excluding backup and log files (see Table 2-4, "Domain Files to Exclude from Source Control," on page 2-34).

During project development, there may be occasion to modify the existing users, groups, roles, and entitlements. You can configure users, groups, roles, and entitlements with the WebLogic Portal Administration Console. It is important to maintain database and LDAP changes in source control.

For detailed information on entitlements, see the *WebLogic Portal Security Guide*. For detailed information on users and groups, see the *WebLogic Portal User Management Guide*.

Updating Other Security-Related Files

Other important security files located in the domain are the `SerializedSystemIni.dat`, `DefaultAuthenticatorInit.ldift`, `DefaultAuthorizerInit.ldift`, and `DefaultRoleMapperInit.ldift` files. These files are located in the `DOMAIN_HOME/security` directory, where `DOMAIN_HOME` is the root directory for your domain.

These files contain essential security information needed to start the domain. While not typically modified during the course of development, these files must exist for the server to start. The `DOMAIN_HOME/servers/AdminServer/security/boot.properties` file contains encrypted username and password information for starting the domain. That file is not mandatory, but it is typically used in development environments to allow server startup without requiring authentication.

For more information about security, see the *WebLogic Portal Security Guide*.

Configuring Facets

Using Shared J2EE Libraries to share resources among development teams, as well as third-party development teams, will be sufficient for many cases. In some cases, however, you might want to develop a project facet that includes a bundle of features you have developed and that can be installed by developers when they create a new project in Workshop for WebLogic.

Before WebLogic Portal 9.2, you could create project templates and include them in your project from WebLogic Workshop. The equivalent feature in WebLogic Portal 9.2 and later versions is handled using an Eclipse plugin. Workshop for WebLogic uses the concept of facets (from the Eclipse Web Tools Platform (WTP): `http://www.eclipse.org/webtools`).

A facet is a set of functionality you can add to your project. The facets show up in the wizard when you create a new project. When you select a facet, Workshop for WebLogic installs all of the facet's components into your project.

Facet development is the next step for those partners and developers who want to integrate features directly into Workshop for WebLogic.

WebLogic Portal includes a Library Modules for Project Features extension point, which allows you to define a facet to include one or more J2EE Shared Libraries, as well as other resources to go into the project classpath. Developing such a plugin is straightforward, and does not necessarily involve any code. Typically, you just need to write a `plugin.xml` file. Of course, you can also include code for views or editors or other tools in your plugin, but to simply have your facet show up in Workshop for WebLogic, no code is required.

Alternative Domain Sharing Techniques

If the recommended approach to creating and sharing a portal domain among team members, as discussed in "Creating a Shared WebLogic Portal Domain" on page 2-2, is not sufficient, this section presents alternative techniques.

This section includes these topics:

- Determining the BEA Home Directory

- Creating and Sharing the Portal Domain

Determining the BEA Home Directory

The directory where WebLogic Platform software is installed on a given system is called the BEA home directory. Figure 2-4 shows an example of a home directory. Each development machine in your team environment will have its own BEA home directory.

Figure 2-4 The BEA Home Directory
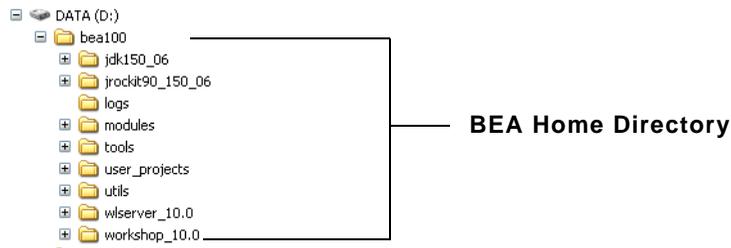
Because Workshop for WebLogic applications and domains each reference the BEA home directory, it is important to carefully consider where to put the BEA home directory. The simplest configuration occurs when every machine in your team environment uses exactly the same BEA home directory (the same drive/directory name). If this is not the case, then you need to choose a strategy for managing the different BEA home directory locations. These strategies are explained in this section.

Importance of the BEA Home Directory

When creating a new portal domain with the domain Configuration Wizard, you choose which BEA home directory you want to reference for that domain. The physical path to this directory is contained in a portal domain's `config/config.xml` file on each development machine, in domain batch scripts such as `startWeblogic.cmd`, and in other domain files (see Table 2-3 for a complete list).

For example, Listing 2-7 shows part of a `config.xml` file. In this example, the `<source-path>` element points to a J2EE Shared Library file in a subdirectory of `D:\myBeaHome`. In this case `D:\myBeaHome` is the BEA home directory.

Listing 2-7   BEA Home Directory Referenced in a config.xml File

```
<library>
  <name>p13n-app-lib#10.0.0@10.0.0</name>
  <target>AdminServer</target>
  <source-path>D:\myBeaHome\wlserver_10.0\common\deployable-libraries\p13n-app-lib.ear
  </source-path>
  <deployment-order>1</deployment-order>
  <security-dd-model>DDOnly</security-dd-model>
</library>
```

If the `config.xml` and other domain files are shared in source control, either all team members must have installed WebLogic Server to the BEA home directory path hard-coded in those files, or another strategy must be used. Strategies for maintaining different BEA home directories on different machines are discussed later in the next section "Managing Multiple BEA Home Directory Locations for Your Team" on page 2-31.

Table 2-3 lists all of the files in a domain that contain hard-coded BEA home directory paths. The files listed in Table 2-3 are relative to the root directory of the domain.

Table 2-3   Domain Files with Hard-Coded Paths

File	Notes
`create_db.*`	`WL_HOME`
`pointbase.ini`	`documentation.home` property
`config/config.xml`	`<source-path>` elements
`bin/setDomainEnv.*`	`WL_HOME`, `JAVA_HOME`, `DOMAIN_HOME`, `LONG_DOMAIN_HOME` variables
`bin/startManagedWebLogic.*`	`trustedCAKeyStore`, `DOMAIN_HOME`
`bin/startPointBaseConsole.*`	`DOMAIN_HOME`
`bin/startWebLogic.*`	`DOMAIN_HOME`
`bin/stopManagedWebLogic.*`	`DOMAIN_HOME`
`bin/stopWebLogic.*`	`DOMAIN_HOME`

Table 2-3 Domain Files with Hard-Coded Paths (Continued)

File	Notes
init-info/domain-info.xml	Output of the Domain Configuration Wizard. This file is needed if you want to use the Configuration Wizard to update the domain.
init-info/startscript.xml	Output of the Domain Configuration Wizard. This file is needed if you want to use the Configuration Wizard to update the domain.

The next section contains strategies to employ when not all team members can use the same BEA home directory.

If all team members *can* use the same BEA home directory, skip to "Creating and Sharing the Portal Domain" on page 2-33.

Managing Multiple BEA Home Directory Locations for Your Team

There are a number of different techniques for sharing a content-equivalent domain with team members with different BEA home directories. These options are described in the following sections.

Option 1: Modifying Configuration Files with String Substitution

You can use a string substitution script to execute search and replace activities on your config.xml and other domain files. For example, you can use the Ant Copy task with a filter to perform the string substitutions. You can make a copy of config.xml (for example, renaming it config-subst.xml), replace hard-coded paths with variables, and check in the copy. Then, each developer can check out the copy with the variables and run the string substitution script.

A string substitution script can be used for more than setting up machines with different BEA home directories: it can provide a way for each developer to work with a separate database instance that shares a common data source configuration.

Option 2: Using a Common Virtual Drive for BEA Home (Windows)

With this option, developers on a team each configure on their machines a common virtual drive letter. This drive is then used as a substitute for the true BEA home directory, which any given developer can place anywhere they wish. The configuration files and start scripts shown in Table 2-3, "Domain Files with Hard-Coded Paths," on page 2-30 can then be edited to use the

virtual drive in place of the BEA home directory, and these files can then be shared among all team members.

For example, if your BEA home directory is currently `D:\BEA-HOME`, the general procedure for creating a substitute drive letter to map to your BEA home directory is as follows:

1. Run the following command from a Windows Command Prompt:

   subst *newDrive*: D:\BEA_HOME

   where `BEA_HOME` is the name of your BEA home directory, for example: `D:\myBEA`. And where *newDrive* is the letter of the drive to which you want to map the BEA home directory, for example `P`.

2. Create a new domain.

3. When you want to use the domain, switch to the new drive and go into the domain directory.

Tip: If other developers install WebLogic Server to a different location, such as `D:\bea`, they can make a similar substitution, such as `subst P: d:\bea`, and share the same `config.xml` and start scripts.

Drawbacks of this option include the following:

- Users must run the `subst` command upon each reboot, though they can type the command in a text file, save the text file with a `.cmd` extension, and put it in their program `/Startup` folder so the command runs automatically at system startup.

- Users must run the created domain and application from the new virtual drive. Running the domain from the "true" install drive and path will result in errors.

- This technique is Windows Operating System specific; however, UNIX developers can follow a similar technique using symbolic links with the `ln` command.

Option 3: Using Relative Paths

If the domain and application directories on each developer's machine are located in a common relative path to the BEA home directory, it is possible to change all file paths in `config.xml` and your start scripts to be relative paths.

Assuming the domain is installed to `D:\myBeaHome\user_projects\mydomain`, where the BEA home directory is `D:\myBeaHome`, the sample `config.xml` entries shown in Listing 2-7 would now look like Listing 2-8, with the changes highlighted in bold type:

Listing 2-8   BEA Home Directory Referenced in a config.xml File

```
<library>
    <name>p13n-app-lib#10.0.0@10.0.0</name>
    <target>AdminServer</target>
<source-path>../../wlserver_10.0/common/deployable-libraries/p13n-app-lib.ear
</source-path>
    <deployment-order>1</deployment-order>
    <security-dd-model>DDOnly</security-dd-model>
  </library>
```

Of course, as with the previous option, all files listed in Table 2-3, "Domain Files with Hard-Coded Paths," on page 2-30 would have to be modified in the same way.

Drawbacks of this option include the following:

● No ability to span multiple drives.

● The domain directory must always be in the exact same relative location to the BEA home directory,

Creating and Sharing the Portal Domain

This section explains an approach to sharing a portal domain among team members. This approach is an alternative to the recommended approach discussed in "Creating a Shared WebLogic Portal Domain" on page 2-2.

Plan a Common Directory for Domains

Create a common domain root directory (%DOMAINNAME) in your source control system for the domain.

Note: Domain creation is discussed in "Creating a Shared WebLogic Portal Domain" on page 2-2. Application creation is discussed in detail in "Creating and Sharing the Portal Application" on page 2-8.

Create the Domain

Create the domain using the WebLogic Configuration Wizard or a script. For detailed information on using the Configuration Wizard, see the WebLogic Server document "Creating WebLogic Configurations Using the Configuration Wizard." For detailed information on building a domain programmatically with a script, see the WebLogic Server document, "Creating

and Configuring WebLogic Domains Using WLST Offline." For an overview of the files that are installed with a domain, see the WebLogic Server document "Domain Configuration Files."

Check the Domain into Source Control

Tip: For information on sharing project files using Workshop for WebLogic's Eclipse-based integrated source control features, see the Workshop for WebLogic document "Working with Source Control."

After you create the domain, but *before you start the server*, check the domain into source control. WebLogic Server creates a number of temporary files and directories in the domain directory at server startup that you are unlikely to want in source control. Table 2-4 lists some files that are created after you start the server, and that you will want to exclude from source control.

Table 2-4 Domain Files to Exclude from Source Control

Path Relative to the Domain Root	File or Files to Exclude from Source Control
`/config/config.xml`	Exclude `config.xml` only if you are using a string substitution script to generate the file.
`/`	`*.log`
`/servers/`*AdminServer*`/logs`	`*`
`/servers/`*AdminServer*`/cache`	`*` (including all subdirectories)
`/servers/`*AdminServer*`/tmp/`	`*` (including all subdirectories)
`/servers/`*AdminServer*`/ldap/` `log/`	`*`

Start the Server

Start WebLogic Server using the domain's `DOMAIN_ROOT/bin/startWeblogic` command. You can find this command in the domain's root directory.

Configure and Tune the Domain

See "Configuring and Tuning the Domain" on page 2-5.

Configuring a Portal Cluster

Before you can deploy a WebLogic Portal application to a clustered environment, you need to use the WebLogic Configuration Wizard to set up and configure a WebLogic Portal domain specifically for a clustered environment. This chapter explains the basic steps to configure a clustered environment for your portal application.

Note: WebLogic Portal requires the use of WebLogic Server multicast clustering. Unicast clustering, as described in the WebLogic Server document *Using Clusters*, is not supported by WLP.

The topics discussed in this chapter include:

- Overview

- Prerequisite Tasks

- Creating Your Clustered Domain

- Configuring the Administration Server

- Setting up JMS Servers

- Creating Managed Server Directories

- Zero-Downtime Architectures

Overview

This chapter describes a set of prerequisite tasks you need to perform before you set up a clustered environment for WebLogic Portal applications. After the prerequisite tasks are complete, this chapter explains how to use the WebLogic Configuration Wizard to set up and configure the cluster, including JMS servers and Managed Server directories.

Prerequisite Tasks

You need to perform several prerequisite tasks before you can configure a clustered production environment for your WebLogic Portal application. Detailed information on most of these prerequisite tasks is beyond the scope of this chapter, and is documented elsewhere (typically in WebLogic Server documentation). Where appropriate, cross-references to source and supplemental documentation are provided.

Note: You can perform these tasks in any order, but they must be addressed before you proceed to configure the cluster environment.

The prerequisite tasks include:

- Set up a Production Database
- Locate JMS Queue and JDBC Data Sources
- Choose a Cluster Architecture
- Determine the Domain Network Layout
- Install WebLogic Portal

Set up a Production Database

To deploy a portal application into production, it is necessary to set up an enterprise-quality database. For detailed information on setting up an enterprise-quality database, see "Managing Databases" on page 2-5. For details on configuring your production database see the *Database Administration Guide*.

Note: An instance of PointBase database is installed when you install WebLogic Server, and is the default database. PointBase is supported only for the design, development, and verification of applications. It is not supported for production server deployment.

Once you have configured your Enterprise database instance, it is possible to install the required database DDL and DML from the command line as described in the *Database Administration*

Guide. You can also create the DDL and DML from the WebLogic Configuration Wizard when configuring your production environment.

Locate JMS Queue and JDBC Data Sources

JMS queues and JDBC names for WebLogic Portal are referenced in the `DOMAIN_ROOT/config/config.xml` file. JMS queues for WebLogic Portal are configured in `DOMAIN_ROOT/config/jms/*.xml` files. JDBC data sources are configured in `DOMAIN_ROOT/config/jdbc/*.xml` files.

Tip: For detailed information on JMS queue configuration, see the WebLogic Server document, "Configuring and Managing WebLogic JMS." For detailed information on JDBC, see the WebLogic Server document, "Configuring and Managing WebLogic JDBC."

Choose a Cluster Architecture

Note: WebLogic Portal requires the use of WebLogic Server multicast clustering. Unicast clustering, as described in the WebLogic Server document *Using Clusters*, is not supported by WLP.

A cluster consists of multiple WebLogic Server instances running simultaneously and working together to provide increased scalability and reliability. A WebLogic Portal application deployed to a cluster appears to clients to be a single application instance.

Note: Multiple managed servers running a WLP application must be deployed to a cluster. Running a WLP application on multiple servers that are not configured as a cluster is not supported.

By clustering a portal application, you can attain high availability and scalability for that application. Use this section to help you choose which cluster configuration you want to use.

This section includes the following topics:

- Single Cluster
- Multi Cluster

Tip: For more detailed information on server clusters, see the WebLogic Server document *Using WebLogic Server Clusters*.

Note: In an early version of WebLogic Portal, it was possible to configure a portal application so that JSPs/Servlets and EJBs were deployed to separate servers. This split configuration is no longer supported.

Single Cluster

When setting up an environment to support a production instance of a portal application, the recommended configuration is to deploy your portal application directly to the cluster. For detailed information on this configuration, refer to the WebLogic Server document, *WebLogic Recommended Basic Architecture*.

Figure 3-1 shows a WebLogic Portal-specific version of the recommended basic architecture.

Figure 3-1 WebLogic Portal Single Cluster Architecture
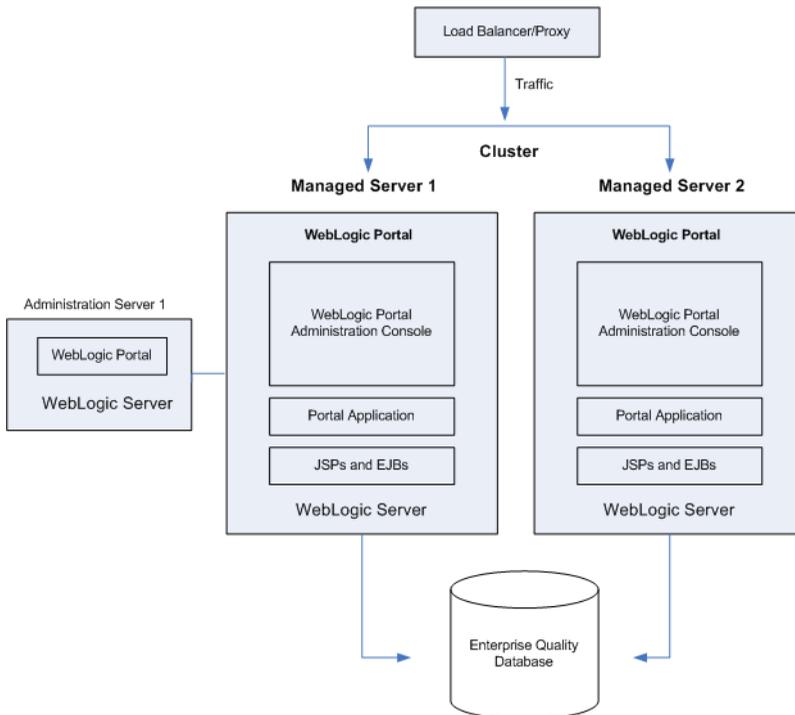
Note: WebLogic Portal does not support a split-configuration architecture where EJBs and JSPs are split onto different servers in a cluster. The basic architecture provides significant performance advantages over a split configuration for WebLogic Portal.

Even if you are running a single server instance in your initial production deployment, this architecture allows you to easily configure new server instances if and when needed.

Multi Cluster

A multi-clustered architecture can be used to support a zero-downtime environment when your portal application needs to be accessible continually. While a portal application can run indefinitely in a single cluster environment, deploying new components to that cluster or server will result in some period of time when the portal is inaccessible. This is due to the fact that while a new EAR application is being deployed to a WebLogic Server, HTTP requests cannot be handled. Redeployment of a portal application also results in the loss of existing sessions.

For more detailed information on the multi-cluster configuration, refer to the WebLogic Server document, "Recommended Multi-Tier Architecture."

A multi-cluster environment involves setting up two clusters, typically a primary cluster and secondary cluster. During normal operations, all traffic is directed to the primary cluster. When some new components (such as portlets) need to be deployed, the secondary cluster is used to handle requests while the primary is updated. The process for managing and updating a multi-clustered environment is more complex than with a single cluster and is addressed in "Zero-Downtime Architectures" on page 3-17. If this environment is of interest, you may want to review that section now.

Figure 3-2 WebLogic Portal Multi-Cluster Architecture
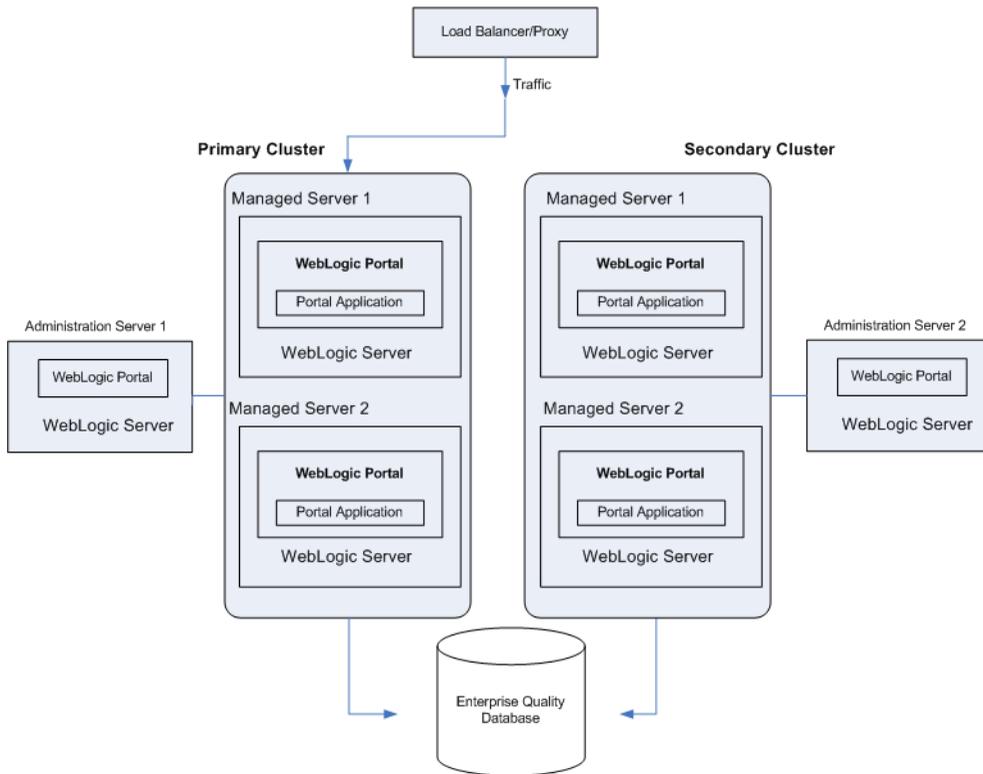
Determine the Domain Network Layout

Before you build your domain with the WebLogic Configuration Wizard, you need to think about the network layout of the domain. Before configuring the domain, consider the total number of Managed Servers in the cluster, including:

- The machines they will run on

- Their listen ports

- Their DNS addresses

In addition, decide if you will use WebLogic Node Manager to start the servers. For information on Node Manager, see *Configuring and Managing WebLogic Server*.

Install WebLogic Portal

WebLogic Portal must be installed on all Managed Server machines and the Administration Server.

Note: The Administration Server must always be running when WebLogic Portal is running in a clustered environment. This ensures that the cluster's policy data stays in sync with the references to that data that are stored by WebLogic Portal in the database.

Creating Your Clustered Domain

This section explains how to set up a clustered environment for a WebLogic Portal domain. This section steps you through the process of setting up a cluster using the WebLogic Configuration Wizard.

Note: To achieve greater scalability, it is common to run a WLP application on multiple managed servers. Multiple managed servers running a WLP application must be deployed to a cluster. Running a WLP application on multiple servers that are not configured as a cluster is not supported.

For more information on clusters, see "Choose a Cluster Architecture" on page 3-3 and the WebLogic Server document "Creating WebLogic Domains Using the Configuration Wizard."

This section includes the following topics:

- What is a Domain?

- Creating the Customized Domain

Tip: You can remove some database files and components from your WebLogic Portal domain after you create it. For details, see "Removing Unneeded Database Components" on page 2-8.

What is a Domain?

To run applications on WebLogic Server you must define and create a domain. To run WebLogic Portal applications, you must create a domain that includes the appropriate WebLogic Portal components.

A domain is the basic administration unit for WebLogic Server. It consists of one or more WebLogic Server instances, and logically related resources and services that are managed,

collectively, as one unit. A basic domain infrastructure consists of one Administration Server and optional Managed Servers and clusters.

Tip: For a more detailed description of these components, as well as a thorough introduction to domains, see the WebLogic Server documents "Creating WebLogic Domains Using the Configuration Wizard and "Understanding WebLogic Server Domains."

The Configuration Wizard guides you through the process of creating or extending a WebLogic Portal domain for your target environment. This process is accomplished using predefined configuration and extension templates containing the main attributes and files required for building or extending a WebLogic Portal domain.

Tip: The Configuration Template Builder guides you through the process of creating custom configuration and extension templates from existing templates or domains. These templates can be used later for creating and updating domains using the Configuration Wizard. For detailed information on domain templates and instructions on using the Configuration Template Builder to create a domain template, see *Creating Configuration Templates Using the WebLogic Configuration Template Builder*. It is a recommended best practice to create a custom domain template for use in a team development environment. For more information, see "Creating a WebLogic Portal Domain Template" on page 2-4.

Creating the Customized Domain

You use the WebLogic Configuration Wizard to create a domain. The procedure for creating a customized domain includes these tasks:

- Initial Configuration

- Customizing the Domain

- Configuring Database and JMS Options

- Completing the Configuration

Note: The following procedure reflects the scope of the WebLogic Configuration Wizard, which handles many complex configuration tasks. **We strongly recommend that you review and refer to the document WebLogic Server document, "Creating WebLogic Domains Using the Configuration Wizard," before and during this**

procedure. This document contains more detailed information than is presented here on each of the wizard options.

Initial Configuration

This section explains how to get started configuring a WebLogic Portal domain using the WebLogic Configuration Wizard.

1. Start the Configuration Wizard. In Windows, choose **Start > Programs > BEA Products > Tools > Configuration Wizard**. You can also start the wizard by executing the file WEBLOGIC_HOME/common/bin/config.cmd (or config.sh). The Welcome dialog appears, as shown in Figure 3-3.

   Tip: You do not need to be running WebLogic Server to start the Configuration Wizard.

Figure 3-3 WebLogic Configuration Wizard Welcome Window

2. In the Welcome dialog, select **Create a new WebLogic domain**, and click **Next**.

3. In the Select Domain Source dialog, select one of the following and click **Next**.

   – If you have not defined a custom domain template, select **Generate a domain configured automatically to support the following BEA products**, and select the **WebLogic Portal** checkbox.

   – If you previously defined a custom domain template, select **Base this domain on an existing template** and use the **Browse** button to select the template file on your system.

4. In the Configure Administration Username and Password dialog, complete the **User name** and **User password** fields and, optionally, a description, such as the name of the administrator. This user is an administrator who can start and stop development mode servers. Click **Next**.

5. In the Configure Server Start Mode and JDK dialog, make the following selections and click **Next**:

   – **Production Mode** – Production mode is the recommended mode for running a production, or live, WebLogic Portal application. When running in production mode WebLogic Server takes advantage of optimizations that enhance performance.

   – **JRockit SDK** – The JRockit SDK is recommended for production environments. This JDK affords better runtime performance and management.

Customizing the Domain

In the next set of steps, the Wizard guides you through the process of customizing the domain by changing default settings.

Tip: For a more detailed description of each configuration setting, refer to the WebLogic Server document "Customizing the Environment."

---

1. In the Customize Environment and Services Settings dialog, select **Yes** and click **Next**, as shown in Figure 3-4.

Figure 3-4 Customize Environment and Services Settings Window
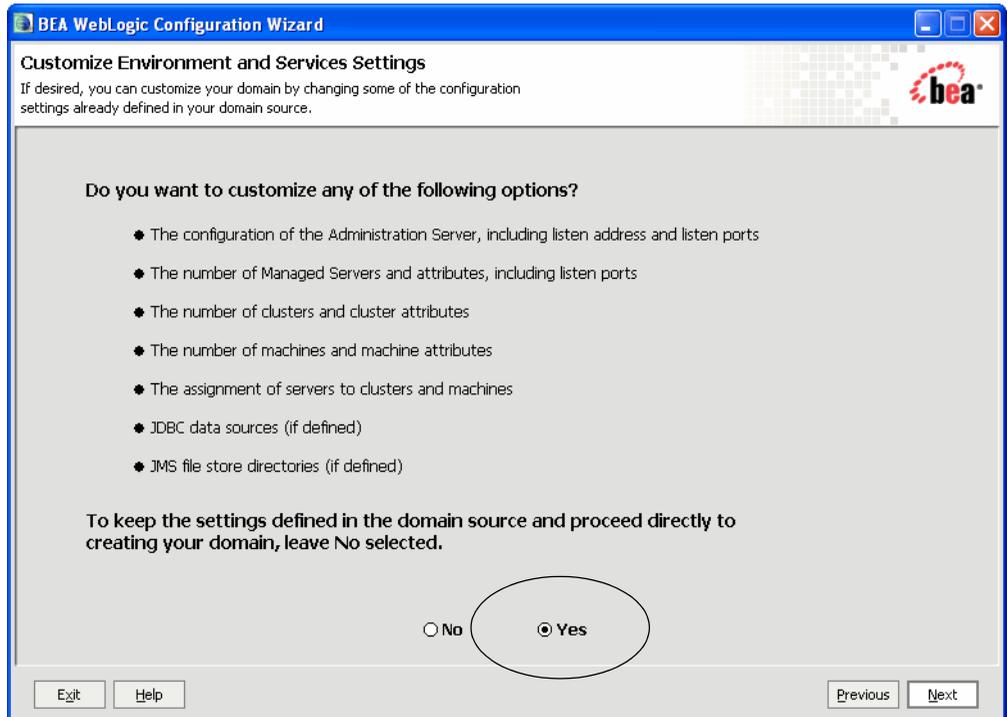
2. Follow the remaining Wizard steps to complete the domain customization. For detailed information, see the WebLogic Server document "Customizing the Environment."

Configuring Database and JMS Options

After you have customized the environment, the Configuration Wizard guides you through the process of configuring database and JMS options.

For detailed information on each of the options in this section, see the WebLogic Server document "Customizing Existing JDBC and JMS Settings." See also "Configuring JDBC Data Sources" in Configuring and Managing WebLogic JDBC."

Note: For nonXA data sources, be sure to select a non-XA driver. For database setup requirements related to using the Asynchronous proliferation setting, refer to the Database Guide.

Completing the Configuration

After the database and JMS options are configured, the Configuration Wizard guides you through the final steps in your domain configuration.

1. The Review WebLogic Domain window allows you to review the detailed configuration settings of your domain before the Configuration Wizard creates it. Figure 3-5 shows a sample window. Click **Next** when you have completed your review.

   Tip: If you need to change anything, click the **Previous** button to return to a previous window.

Figure 3-5 Review WebLogic Domain Window Sample
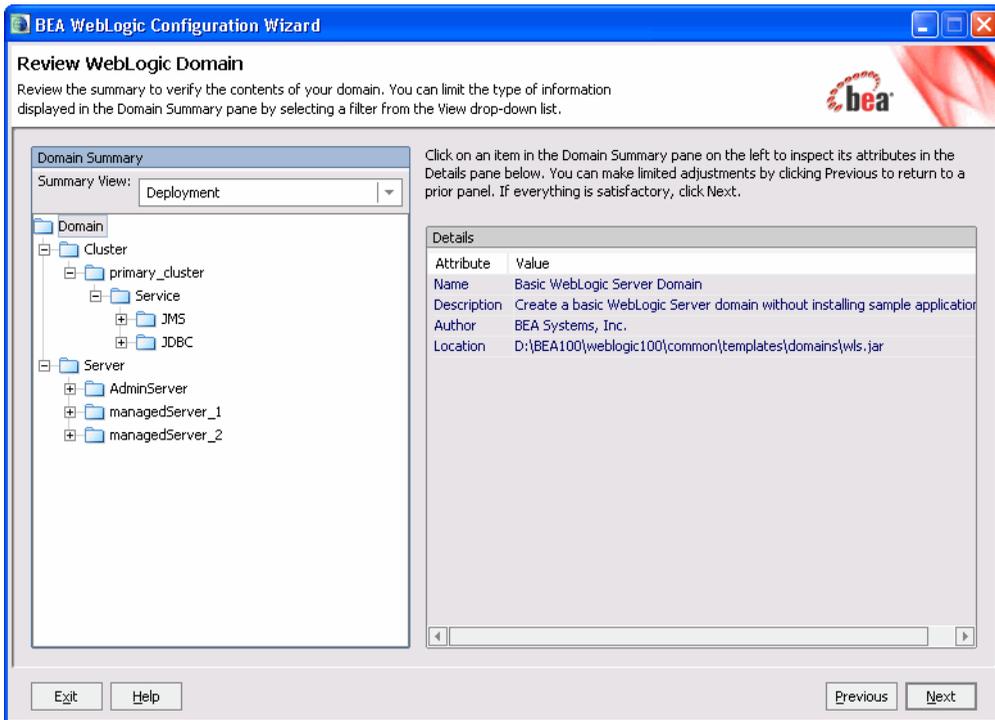
2. The Create WebLogic Domain window is the final window. Enter a name for the domain and a location for it. Click **Create**. A progress window appears indicating the progress of the domain creation.

3. In the progress window, click **Done** after the domain has been created.

Configuring the Administration Server

The Administration Server requires a WebLogic Portal installation; however, *do not deploy your WebLogic Portal applications to the Administration Server*.

Note: The Administration Server must always be running when WebLogic Portal is running in a clustered environment. This ensures that the cluster's policy data stays in sync with the references to that data that are stored by WebLogic Portal in the database.

For detailed information on configuring startup scripts for the Administration Server, see the WebLogic Server document "Managing Server Startup and Shutdown."

Setting up JMS Servers

For detailed information and procedures to configure and manage basic JMS system resources, such as JMS servers and JMS system modules, see the WebLogic Server document "Configuring and Managing WebLogic JMS."

Creating Managed Server Directories

This section on creating Managed Server directories includes the following topics:

- Introduction
- Creating the Managed Server Domains

For more information on Managed Servers, see the WebLogic Server document "Understanding WebLogic Server Domains."

Introduction

Now that you have configured your domain, including defining your Managed Servers, you need to create a server root directory for each Managed Server. There are many options for this, depending on whether or not the Managed Server will reside on the same machine as the Administration Server and whether or not you will use the Node Manager.

- Most of the files in the domain-level directory are not necessary for Managed Servers, so a domain (files directly in the domain directory) is not required on each Managed Server, especially if you are using the Node Manager to start and stop Managed Servers. For example, `config.xml` in a Managed Server domain is not used. Instead, the `config.xml`

file in the Administration Server is used. The only requirement for Managed Servers is to have the `wsrpKeystore.jks` file one directory above the server directory (in the equivalent of a domain-level directory). This file is required if you want to use WSRP with SAML security between WebLogic Portal 8.1x and 9.2 or later domains.

- If the Managed Server will run on a different machine than the Administration Server and you will not use Node Manager, the easiest option is to use the Configuration Wizard to create a full filesystem domain for the Managed Server, as described in the following procedure.

Note: WebLogic Portal must be installed on all Managed Servers.

Creating the Managed Server Domains

This section lists the basic procedure for creating WebLogic Portal domains on the Managed Servers. For more information about Managed Servers, see the WebLogic Server document "Understanding WebLogic Server Domains."

1. Start the Configuration Wizard. In Windows, choose **Start > Programs > BEA Products > Tools > Configuration Wizard**. You can also start the wizard by executing the file `WEBLOGIC_HOME/common/bin/config.cmd` (or `config.sh`). The Welcome dialog appears, as shown in Figure 3-3.

   Tip: You do not need to be running WebLogic Server to start the Configuration Wizard.

**Figure 3-6 WebLogic Configuration Wizard Welcome Window**

2. In the Welcome dialog, select **Create a new WebLogic domain**, and click **Next**.

3. In the Select Domain Source dialog, select one of the following and click **Next**.

    – If you have not defined a custom domain template, select **Generate a domain configured automatically to support the following BEA products**, and select the **WebLogic Portal** checkbox.

    – If you previously defined a custom domain template, select **Base this domain on an existing template** and use the **Browse** button to select the template file on your system.

4. In the Configure Administration Username and Password dialog, complete the **User name** and **User password** fields and, optionally, a description, such as the name of the administrator. This user is an administrator who can start and stop development mode servers. Click **Next**.

5. In the Configure Server Start Mode and JDK dialog, make the following selections and click **Next**:

   – **Production Mode** – Production mode is the recommended mode for running a production, or live, WebLogic Portal application. When running in production mode WebLogic Server takes advantage of optimizations that enhance performance.

   – **JRockit SDK** – The JRockit SDK is recommended for production environments. This JDK affords better runtime performance and management.

   Note: *It is important you choose the same JDK across all instances in the cluster*.

6. In the Customize Environment and Services Settings dialog, select **No** and click **Next**, as shown in Figure 3-4.

Figure 3-7 Customize Environment and Services Settings Window
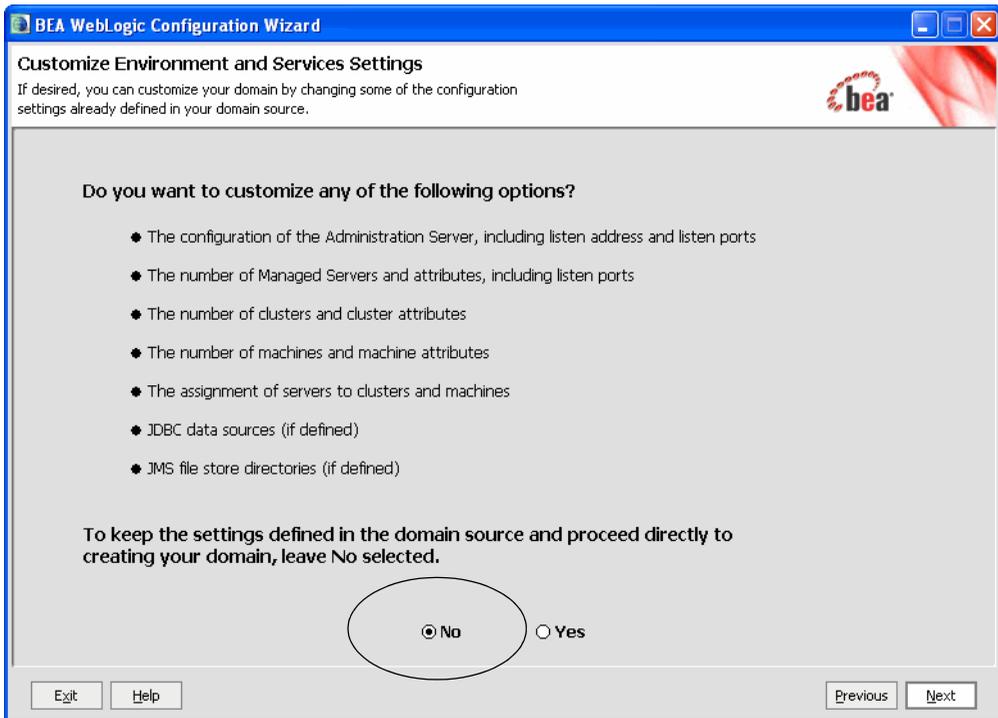
7. In the Create WebLogic Domain dialog, enter a name for the domain and a directory for the domain on the Managed Server, and click **Create**.

8. Follow the remaining Wizard steps as detailed in the WebLogic Server document "Customizing the Environment."

Tip: For detailed information on configuring startup scripts for Managed Servers, see the WebLogic Server document "Managing Server Startup and Shutdown."

Once you have created a domain for a Managed Server, you can reuse the same domain for your other Managed Server on the same machine by specifying different servername parameters to your startManagedWebLogic script, or create new managed domains using the domain Configuration Wizard.

Note: If you decide not to use a full domain for your Managed Servers (that is, not include all files in the domain-level directory), be sure you keep or put a copy of wsrpKeystore.jks in the directory directly above the server directory (in the equivalent of the domain-level directory). This file is required if you want to use WSRP with SAML security between WebLogic Portal 8.1x and 9.2 and later domains.

Zero-Downtime Architectures

This section includes the following sections:

- Overview
- Single Database Instance
- Portal Cache

Tip: WebLogic Portal supports another method of zero-downtime deployment called production redeployment. Production redeployment is only supported for a limited number of use cases. For detailed information on this option, see "Using Production Redeployment with WebLogic Portal" on page 4-19.

Overview

One limitation of redeploying a portal application to a WebLogic Server cluster is that during redeployment, users cannot access the site. For Enterprise environments where it is not possible to schedule down time to update a portal application with new portlets and other components, a multi-cluster configuration lets you keep your portal application up and running during redeployment.

The basis for a multi-clustered environment is the notion that you have a secondary cluster to which user requests are routed while you update the portal application in your primary cluster.

For normal operations, all traffic is sent to the primary cluster, as shown in Figure 3-8. Traffic is not sent to the secondary cluster under normal conditions because the two clusters cannot use the same session cache. If traffic was being sent to both clusters and one cluster failed, a user in the middle of a session on the failed cluster would be routed to the other cluster, and the user's session cache would be lost.

Figure 3-8 During Normal Operations, Traffic Is Sent to the Primary Cluster
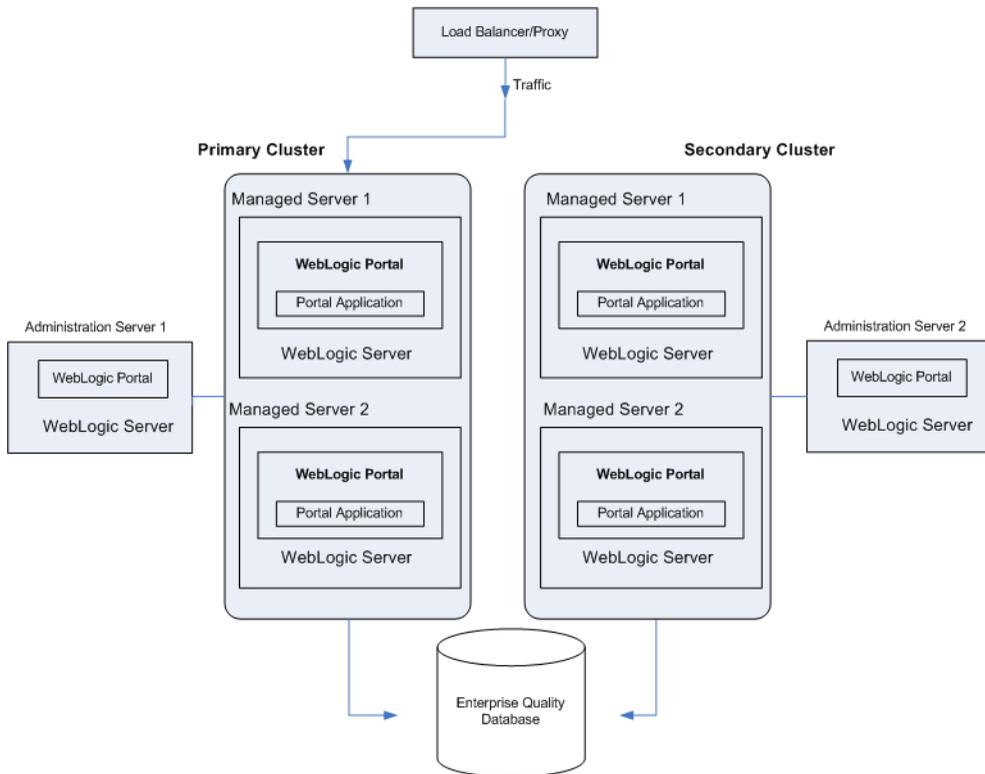
When the primary cluster is being updated, all traffic is routed to the secondary cluster, then the primary cluster is updated with a new Portal EAR, as shown in Figure 3-9. This EAR has a new portlet, which is loaded into the database. Routing requests to the secondary cluster is a gradual process. Existing requests to the primary cluster must first end over a period of time until no more requests exist. At that point, you can update the primary cluster with the new portal application.

Figure 3-9 Traffic Is Routed to the Secondary Cluster; The Primary Cluster Is Updated
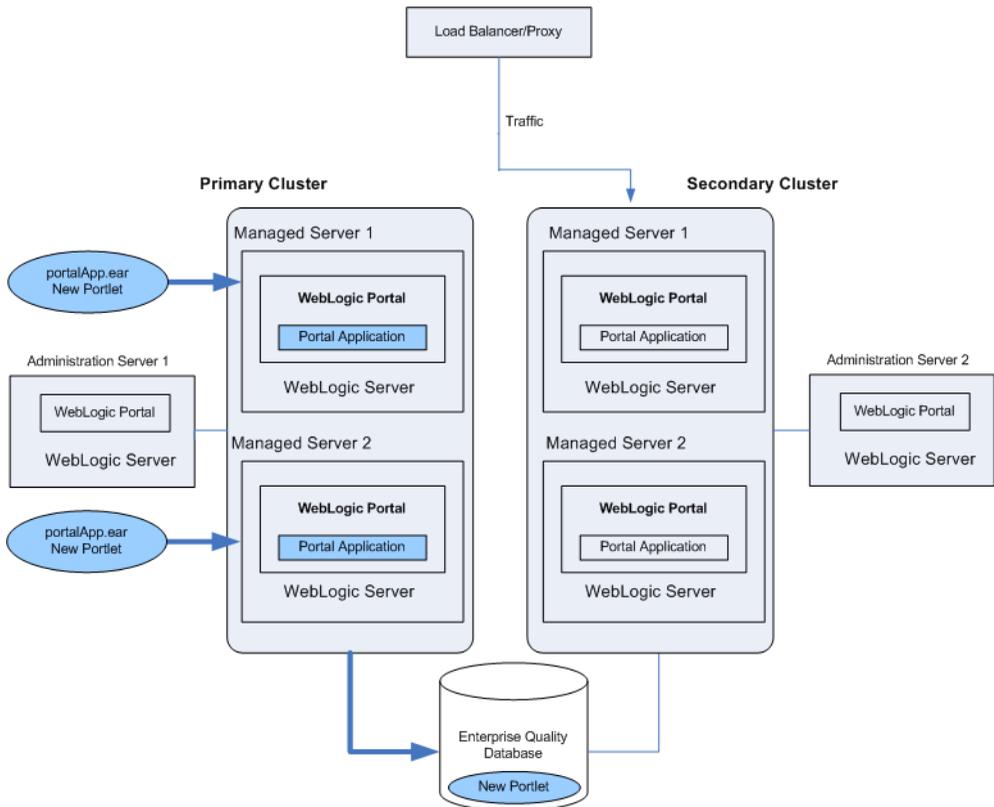
After the primary cluster is updated, all traffic is routed back to the primary cluster, and the secondary cluster is updated with the new EAR, as shown in Figure 3-10. Because the database was updated when the primary cluster was updated, the database is not updated when the secondary cluster is updated.

Figure 3-10 Traffic Is Routed Back to the Primary Cluster; The Secondary Cluster Is Updated
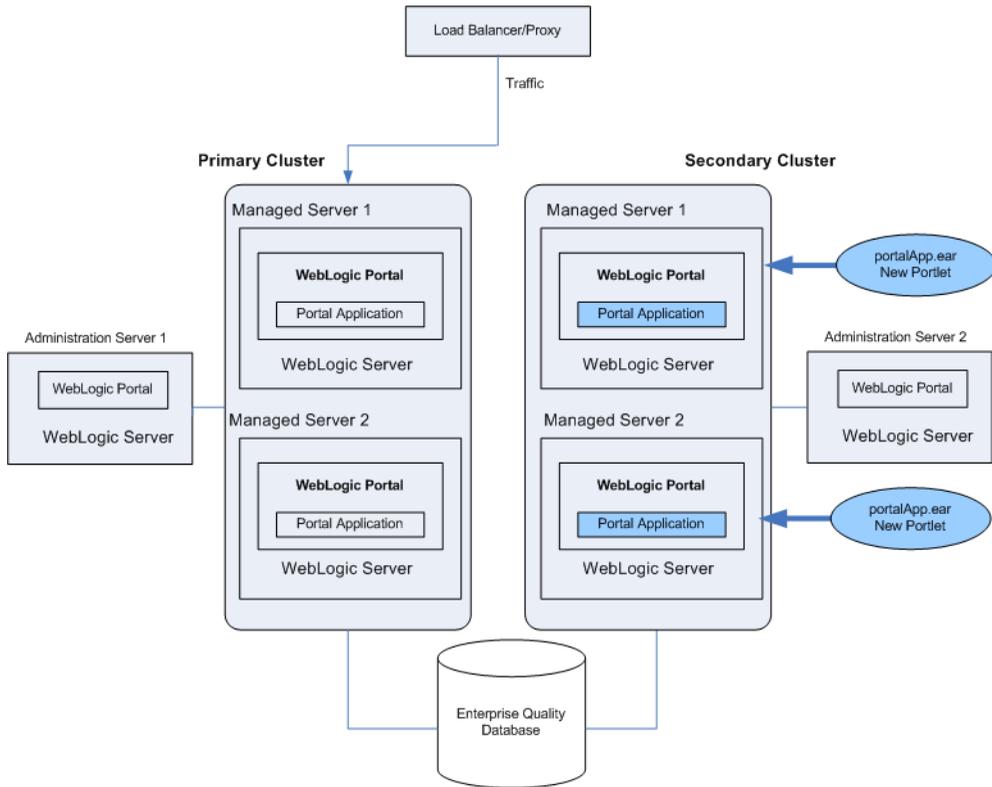
Even though the secondary cluster does not receive traffic under normal conditions, you must still update it with the current portal application. When you next update the portal application, the secondary cluster temporarily receives requests, and the current application must be available.

In summary, to upgrade a multi-clustered portal environment, you switch traffic away from your primary cluster to a secondary one that is pointed at the same portal database instance. You can then update the primary cluster and switch users back from the secondary. This switch can happen instantaneously, so the site experiences no down time. However, in this situation, any existing user sessions will be lost during the switches.

A more advanced scenario is a gradual switchover, where you switch new sessions to the secondary cluster, and after the primary cluster has no existing user sessions you upgrade it. Gradual switchovers can be managed using a variety of specialized hardware and software load balancers. For both scenarios, there are several general concepts that should be understood before

deploying applications, including the portal cache and the impact of using a single database instance.

Single Database Instance

When you configure multiple clusters for your portal application, they will share the same database instance. This database instance stores configuration data for the portal. This can become an issue because when you upgrade the primary cluster it is common to make changes to portal configuration information in the database. These changes are then picked up by the secondary cluster where users are working.

For example, redeploying a portal application with a new portlet to the primary cluster will add that portlet configuration information to the database. This new portlet will in turn be picked up on the secondary cluster. However, the new content (JSP pages or Page Flows) that is referenced by the portlet is not deployed on the secondary cluster.

Portlets are invoked only when they are part of a desktop, so having them available to the secondary cluster has no immediate effect on the portal that users see. However, adding a new portlet to a desktop with the WebLogic Portal Administration Console will immediately affect the desktop that users see on the secondary cluster. In this case, that portlet would show up, but the contents of the portlet will not be found.

To handle this situation, you have several options:

- You can delay adding the portlet to any desktop instances until all users are back on the primary cluster.

- You can entitle the portlet in the library so that it will not be viewable by any users on the secondary cluster. Then add the portlet to the desktop, and once all users have been moved back to the primary cluster, remove or modify that entitlement.

Tip: It is possible to update an existing portlet's content URI to a new location that is not yet deployed. For this reason, exercise caution when updating the content URI of a portlet. The best practice is to update the content URIs as part of a multi-phase update.

When running two portal clusters simultaneously against the same database, you must also consider the portal cache, as described in the next section.

Portal Cache

WebLogic Portal provides facilities for a sophisticated cluster-aware cache. This cache is used by a number of different portal frameworks to cache everything from markup definitions to portlet preferences. Additionally, developers can define their own caches using the portal cache framework.

For detailed information on setting the portal cache, see the *WebLogic Portal Development Guide*.

For any cache entry, the cache can be enabled or disabled, a time to live can be set, the cache maximum size can be set, the entire cache can be flushed, or you can invalidate a specific key.

When a portal framework asset that is cached is updated, it will typically write something to the database and automatically invalidate the cache across all machines in the cluster. This process keeps the cache in sync for users on any Managed Server.

When operating a multi-clustered environment for application redeployment, special care needs to be taken with regard to the cache. The cache invalidation mechanism does not span both clusters, so it is possible to make changes on one cluster that is written to the database but not picked up immediately on the other cluster. Because this situation could lead to system instability, it is recommended that during this user migration window the caches be disabled on both clusters. This is important when you have a gradual switchover between clusters versus a hard switch that drops existing user sessions.

Deploying Portal Applications

The term application deployment refers to the process of making an application or module available for processing client requests in a WebLogic Server domain. This chapter discusses recommended procedures and best practices for deploying WebLogic Portal applications and shared libraries to the server.

This chapter includes the following topics:

- Preparing to Deploy

- Overview of Deployment Descriptors and Config Files

- Using Deployment Plans

- Using Application-Scoped JDBC

- Building a Portal Application

- Deploying the EAR

- Deploying J2EE Shared Libraries

- Creating Content Repositories

- Using Multiple Enterprise Applications in a Single Domain

- Application Tuning Tips

- Deploying JSR-168 Portlets in a WAR File

- Using Production Redeployment with WebLogic Portal

Preparing to Deploy

> **Tip:** Before continuing, we recommend that you review the following WebLogic Server document, "Understanding WebLogic Server Deployment."

Before you deploy a WebLogic Portal application, you need to configure the destination domain. For detailed information on domain configuration, see "Creating Your Clustered Domain" on page 3-7.

Overview of Deployment Descriptors and Config Files

In WebLogic Portal 8.1 and prior versions, an application configuration file named `META-INF/application-config.xml` was used to configure WebLogic Portal components such as cache, campaigns, behavior tracking, content management, and others.

As of WebLogic Portal 9.2, this configuration file has been removed and its contents moved to a collection of new, schema-compliant, descriptors. These new descriptors are each focused on a single service or group of related services. To view the complete set of descriptor files used by an application in WebLogic Workshop, select the Merged Projects view.

These files are processed using the same infrastructure that WebLogic Server uses to process all the J2EE descriptors (like `application.xml` or `web.xml`). These descriptors support merging from J2EE Shared Libraries and also support deployment plans. (For more information on merging, see "Descriptor Merging" on page 4-3.) The WebLogic Portal descriptors are installed with default settings that you can override by copying them into your application and editing them. A significant benefit of this configuration is that the contents of your project's EAR file is focused on your project, and not on the WebLogic Portal product code.

This section includes these topics:

- Descriptor Merging
- Viewing Merged Descriptors
- Portal Web Application Deployment Descriptors
- Enterprise Application Deployment Descriptors
- Configuration Files

Descriptor Merging

When the server processes the deployment descriptors, it merges them. All descriptors from the application and from the shared libraries that the application references are merged. For each separate deployment descriptor, a merged "virtual" descriptor is created, which can be used by the server to deploy the application. The rules that govern descriptor merging are explained in "Shared Library Rules of Precedence" on page 2-15. Remember that the referencing application and its deployment plan always override settings imported from a referenced library.

Viewing Merged Descriptors

If you want to see merge results without deploying an application, you can use the WebLogic Server utility called `weblogic.appmerge`. This utility takes an application that references shared libraries and creates a J2EE application including the merged contents and merged descriptors. This utility is useful for debugging purposes. For details, see the WebLogic Server document, "Using weblogic.appmerge to Merge Libraries."

Workshop for WebLogic provides information about shared libraries (also called library modules). You can use Workshop for WebLogic to copy files from shared libraries to your application. When you do this, you can then modify the copied file. From then on, the local copy takes precedence over the library module copy. Using the Package Explorer view, you can view and browse the contents of shared libraries.

Another utility, called `ddbrowser`, is included with the WebLogic Portal installation. This utility inspects your application and shows you all the "mergeable" descriptors. For each descriptor in the application, you can see the contributions from each library as well as the resulting merged descriptor.

To use `ddbrowser`, set the `WL_HOME` environment variable to point to the WebLogic Home directory of your installation. Then run the following command:

```
java -jar $WL_HOME/common/p13n/lib/ddbrowser.jar.
```

Portal Web Application Deployment Descriptors

A Portal Web Application includes several deployment descriptors. By default, these descriptors are located in the `WebContent/WEB-INF` directory of the web project. The deployment descriptors are listed in Table 4-1.

> **Tip:** For a complete listing and detailed description of deployment descriptors associated with WebLogic Server based applications, see the WebLogic Server document "Overview of WebLogic Server Application Development."

Table 4-1 WebLogic Portal Descriptor Files: Web Application Scoped

Descriptor	Purpose
`web.xml`	The `web.xml` file is a J2EE standard deployment descriptor. Among other settings, it has a set of elements for configuring security for the web application. For more details about web.xml see the WebLogic Server document "web.xml Deployment Descriptor Elements."
`weblogic.xml`	The WebLogic descriptor is a standard WebLogic Server deployment descriptor for web applications that has a number of important descriptor entries. See the WebLogic Server document, "weblogic.xml Deployment Descriptor Elements" for details.
`wlp-template-config.xml`	Content display templates
`wsrp-consumer-handler-config.xml`	SOAP Handlers and interceptors for WSRP Consumer
`wsrp-consumer-security-config.xml`	WSRP Consumer security configuration
`wsrp-user-property-map.xml`	WSRP Producer/Consumer property mappings
`wsrp-producer-portlet-registry-config.xml`	WSRP Producer UDDI registries

Enterprise Application Deployment Descriptors

The J2EE specifications define standard, portable deployment descriptors for J2EE modules and applications. BEA defines additional WebLogic-specific deployment descriptors for deploying a module or application in the WebLogic Server environment. A WebLogic Portal enterprise

application, also called an EAR Project, includes the deployment descriptors listed in Table 4-2. By default, these files are located in the `EarContent/META-INF` directory of the EAR project.

Tip: For a complete listing and detailed description of deployment descriptors associated with WebLogic Server based applications, see the WebLogic Server document "Overview of WebLogic Server Application Development."

Table 4-2 WebLogic Portal Descriptor Files: Enterprise Application Scoped

Descriptor	Purpose
`application.xml`	This J2EE descriptor specifies the web applications that are associated with an EAR project.
`weblogic-application.xml`	This WebLogic descriptor specifies the J2EE Shared Libraries used by the enterprise application. It also specifies web applications, such as the Propagation Servlet, that are deployed to the EAR. See also the WebLogic Server document "Creating Shared J2EE Libraries and Optional Packages."
`p13n-cache-config.xml`	This WebLogic descriptor specifies the cache settings for all caches used by WebLogic Portal. For more information on cache settings, see the *WebLogic Portal Development Guide*.
`p13n-config.xml`	P13N features (events, behavior tracking, etc.)
`p13n-security-config.xml`	Security management, including group hierarchy tree caches and role authorization for management.
`p13n-profile-config.xml`	Unified User Profile adapters (formerly in `p13n-ejb.jar#ejb-jar.xml`)
`netuix-application-config.xml`	Portal Proliferation (sync)
`communities-config.xml`	WebLogic Portal Communities
`content-config.xml`	Content repositories

Table 4-2 WebLogic Portal Descriptor Files: Enterprise Application Scoped

Descriptor	Purpose
`wps-config.xml`	Ads and Campaigns
`wsrp-consumer-portlet-registry-config.xml`	Portlet UDDI registries for WSRP

Configuration Files

Table 4-3 lists Web application configuration files. These files are not descriptor files (they cannot be used in deployment plans and they are not merged).

Table 4-3 WebLogic Portal Configuration Files: Web Application Scoped

Descriptor	Purpose
`wsrp-producer-registry.xml`	This file allows WSRP consumer applications to configure registered producer applications. For example, you can enable local proxy support by setting `<enable-local-proxy>` to true in `WEB-INF/wsrp-producer-registry.xml` in the consumer web application. For more information on WSRP applications, see the *Federated Portals Guide*.
`beehive-netui-config.xml`	This descriptor specifies netui-specific tags and handler classes. For detailed information on the elements of this descriptor, refer to the Apache Beehive Project reference documentation.

Using Deployment Plans

Deployment plans let you easily tune an application's deployment descriptors on a deployment-by-deployment basis without actually modifying the descriptors themselves in the application EAR file. For instance, you might use a deployment plan to adjust descriptors for a

staging environment and another plan to make adjustments for your production environment. Typically, developers set initial baseline descriptor values.

> **Tip:** See also "Application Tuning Tips" on page 4-15.

A deployment descriptor is an XML document used to define the J2EE behavior or WebLogic Server configuration of an application or module at deployment time. A deployment plan is an XML document that resides outside of an application's archive file, and can apply changes to deployment properties stored in the application's existing WebLogic Server deployment descriptors.

For detailed information on deployment plans, see the WebLogic Server document "Configuring Applications for Production Deployment." This document describes deployment plans in detail, discusses the XML schema for deployment plans, and explains how to create, edit, and use them.

Note: The WebLogic Portal Administration Console uses deployment plans to save runtime changes to deployment descriptor values.

Using Application-Scoped JDBC

With application-scoped JDBC, it is possible to use more than one database in a WebLogic Portal domain, allowing each application deployed in the domain to access its own database. WebLogic Server enables this by providing the ability for you to scope JDBC pools to the application level.

> **Tip:** Using application-scoped JDBC pools in a domain with multiple WebLogic Portal applications is a recommended best practice.

When you package your enterprise application, you can include JDBC resources in the application by packaging JDBC modules in the EAR and adding references to the JDBC modules in all applicable descriptor files. When you deploy the application, the JDBC resources are deployed, too. Depending on how you configure the JDBC modules, the JDBC data sources deployed with the application will either be restricted for use only by the containing application (application-scoped modules) or will be available to all applications and clients (globally-scoped modules).

By default, JDBC pools are scoped to the domain level. If you want to scope JDBC pools to the application level, some configuration is required. For detailed information on configuring application-scoped JDBC, see the WebLogic Server document "Configuring JDBC Application Modules for Deployment."

Building a Portal Application

This section explains how to build a portal application and create an EAR file using Workshop for WebLogic or from the command line. This section discusses creating both EAR files and exploded EARs.

Building in Workshop for WebLogic

To deploy a portal application to a production environment, you must first build the application in Workshop for WebLogic to compile necessary classes in the portal application and create a deployable EAR file. The EAR can either be compressed (an EAR file) or uncompressed (an exploded EAR directory).

For detailed information on building an application in Workshop for WebLogic, see the Workshop for WebLogic document "Understanding the Build Process."

To create an exploded EAR directory, first build an EAR file, and then use the Java `jar xf` command to uncompress the file.

Tip: After you create an exploded EAR, you might want to rename the application directory so that it has a `.ear` extension. For example, if the application directory is called `myPortalApp`, rename the directory to `myPortalApp.ear`. Adding the `.ear` makes the configuration common for compressed and exploded deployments, but it is not required.

Building from the Command Line

You can export an Ant build file for your project using Workshop for WebLogic. For more information, see the WebLogic Workshop document "Creating Custom Ant Build Files for an Application."

Deploying the EAR

This section provides instructions for the deployment of your portal application.

Note: Although it was required before WebLogic Portal version 9.2, deploying a WebLogic Portal application to the Administration Server is no longer necessary and is not specifically discussed here.

Tip: WebLogic Server documentation explains the deployment process and multiple deployment scenarios in detail. We recommend you review the WebLogic Server document "Deploying Applications to WebLogic Server" before continuing. For information on automating deployment tasks with WLST see the WebLogic Server document "WebLogic Scripting Tool."

This section includes the following topics:

- Deploying to a Development Environment

- Deploying to a Staging or Production Environment

- Redeploying to a Staging or Production Environment

- Deploying an Exploded EAR

Deploying to a Development Environment

In a typical development environment, developers use Workshop for WebLogic to develop portal components. They typically deploy and run against a local server domain that is configured in Development mode. No Managed Servers are used. The best practice is to use Workshop for WebLogic to deploy (publish) and test your application.

Deploying to a Staging or Production Environment

In a domain with an Administration Server and one Managed Server or multiple Managed Servers in a cluster, it is strongly recommended that you deploy the portal EAR file to the Managed Server or cluster. Do not deploy to the Administration Server; this reserves the Administration Server for administration functions, allows the Administration Server to operate with a smaller heap, and allows specific fire-walling of administration functions on this server.

The best practice is to configure your staging and production servers in Production mode rather than Development mode.

The WebLogic Server feature that allows applications to be deployed directly to a Managed Server is called Managed Server Independent (MSI) mode. See "Understanding Managed Server Independent Mode" in the WebLogic Server documentation for information on MSI and several limitations of MSI. One of the most important limitations is that security provider data cannot be modified if the Administration Server is down. Another limitation is that any WebLogic Portal configuration that must write to a deployment plan for persistence will not work if the Administration Server is down.

Redeploying to a Staging or Production Environment

Redeploying means deploying a new version of an existing/running/deployed application. If you redeploy a portal application to a staging or production environment, follow the same procedure as deploying described in the previous section. However, on redeployment, you must propagate datasync and other WebLogic Portal assets to the target database as a separate operation using the propagation tools. See "General Propagation Scenarios" on page 5-15 for more information on redeploying.

Deploying an Exploded EAR

If you deploy an exploded EAR file to a managed server, extra configuration is required. You must place a new `datasync.properies` file in the end user's application under the `META-INF` directory. In this `datasync.properties` file, add the following property:

```
PERSISTENT_STORE=DATABASE_STORE
```

This change is only required if you deploy an exploded EAR file.

Deploying J2EE Shared Libraries

WebLogic Portal product code is included in your application by referencing several J2EE Shared Libraries. These libraries are part of the WebLogic Portal installation. You can also develop your own shared libraries containing your application code. J2EE Shared Libraries allow you to decouple the development life cycles of various portions of your application. These libraries can be versioned and deployed independently, allowing each library to be developed and distributed independently (rather than as a single large EAR file).

Tip: For detailed information on J2EE Shared Libraries, see the WebLogic Server document "Creating Shared J2EE Libraries and Optional Packages." See also "Using J2EE Shared Libraries in a Team Environment" on page 2-14 and the *WebLogic Portal Development Guide*.

This section discusses several aspects of shared libraries:

- Library Descriptors
- Library Versions

Library Descriptors

Listing 4-1 shows a sample enterprise application, `myApp.ear`. Note that the descriptor `weblogic-application.xml` includes a reference (shown in bold) to a library called `p13n-app-lib`. Note, too, that no other information, other than the name `p13n-app-lib`, is given to describe the library.

Listing 4-1   Example Application myApp.ear

```
META-INF/weblogic-application.xml

    <library-ref>
      <library-name>p13n-app-lib</library-name>
    </library-ref>

APP-INF/lib/myClasses.jar
myEjb.jar
myWebApp.war
```

The enterprise application locates the library because the `<library-ref>` element causes the server at runtime to look for a deployed library with the same name. Since the `<library-ref>` element does not specify any version information, the server looks for the latest, or highest, version number available to it. The name and version number of a shared library is contained in the library itself, in the `META-INF/MANIFEST.MF` file. For example:

```
META-INF/MANIFEST.MF

    Extension-Name: p13n-app-lib
    Specification-Version: 10.0.0
    Implementation-Version: 10.0.0
```

For more information, see "Library Versions" on page 4-12. Note that the same pattern applies to the `WEB-INF/weblogic.xml` file for portal web applications.

To be located by the server, a shared library must be deployed as a library. This deployment differs from the deployment of a runnable application. When a library is deployed *as a library*, the library file is registered with the server, and is therefore available to referencing applications.

To deploy a library, use the WebLogic Server Console, the `weblogic.Deployer` command, the `wldeploy` Ant task, or with a WLST script. For information these methods, see the WebLogic Server documents "The WebLogic Server Administration Console," "weblogic.Deployer Command Line Reference," "wldeploy Ant Task Reference," and "WLST Command and Variable Reference."

For example, you can use the WebLogic Server Console to deploy a shared library in exactly the same way you deploy an application, except that a checkbox is available that distinguishes the shared library from a runnable application.

Listing 4-2 shows an entry in the server domain's `config/config.xml` file for the shared library referenced by the application in Listing 4-1.

Listing 4-2   Configuration File Entry

```
<library>
   <name>p13n-app-lib#10.0.0@10.0.0</name>
   <target>myServer</target>
   <module-type>ear</module-type>
   <source-path>/bea/weblogic100/deployable-libraries/p13n-app-lib.ear
   </source-path>
   <security-dd-model>DDOnly</security-dd-model>
</library>
```

Library Versions

You can deploy several libraries with the same name and different version numbers. The version number that an application uses is specified in the application's configuration file `META-INF/weblogic-application.xml` or `weblogic.xml` for Web applications. If no version number is specified, as in Listing 4-3, the server selects the highest-versioned available deployed library named `p13n-app-lib`.

Listing 4-3   Library Reference With No Version Specified

```
<library-ref>
   <library-name>p13n-app-lib</library-name>
</library-ref>
```

If, on the other hand, you know that version 10.0.0 of `p13n-app-lib` is inadequate, and your application needs to use version 10.0.1 or higher, you can reference the library as shown in Listing 4-4.

Listing 4-4   Library Reference With Version Specified

```
<library-ref>
  <library-name>p13n-app-lib</library-name>
  <specification-version>10.0.1</specification-version>
</library-ref>
```

If you want to use a specific version, and no other version, then you can specify the `<exact-match>` element, as shown in Listing 4-5.

Listing 4-5   Specifying an Exact Match

```
<library-ref>
  <library-name>p13n-app-lib</library-name>
  <specification-version>10.0.0</specification-version>
  <exact-match>true</exact-match>
</library-ref>
```

Creating Content Repositories

If you are using a content repository, you need to create that repository or repositories on the destination server. To do this, use the WebLogic Portal Administration Console as explained in the *Content Management Guide*. This means creating only the root repositories, not the content items and types. To move items and types between environments, use the WebLogic Portal propagation tools. For more information on propagation tools, see Chapter 5, "Developing a Propagation Strategy."

You can also specify the default content repository for an enterprise application in the configuration file `META-INF/content-config.xml`. See "Enterprise Application Deployment Descriptors" on page 4-4 for more information.

Using Multiple Enterprise Applications in a Single Domain

You can create and run multiple enterprise applications in a single-cluster domain. As shown in Figure 4-1, a single domain can host multiple enterprise applications (EARs). Each EAR deployment can host multiple web applications, and any number of desktops can be created based on the web applications. The web applications and desktops associated with one enterprise application (EAR) are not dependent on those in another enterprise application (they are decoupled).

Figure 4-1 Multiple Enterprise Applications in a Single Domain
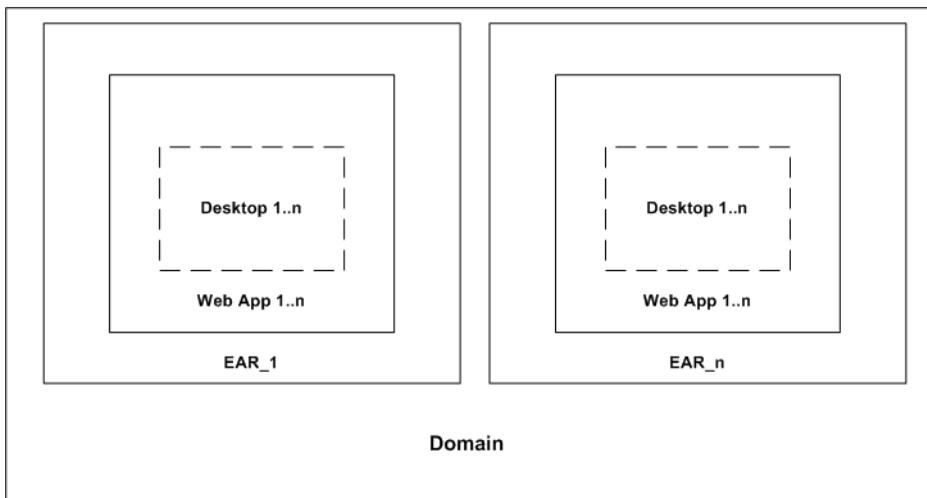
The following restrictions apply to this configuration of multiple enterprise applications in a single domain:

- **Resource names** – Names cannot conflict. For example, for each deployment, web application names must be unique. This applies as well to `context-root` names and their associated `CookieName` names. For each enterprise application, the WebLogic Portal Administration Console shows all of the portal web applications that are contained in that application.

Note: Each enterprise application must be managed through its respective WebLogic Portal Administration Console. Some domain-level resources, such as users and groups, can be viewed and managed across enterprise applications from a single WebLogic Portal Administration Console; however, be aware that data in one application may be cached,

and updates to the same data made from another application's WebLogic Portal Administration Console may not be immediately visible.

● **Content** – If you are deploying multiple enterprise applications within the same domain, and plan to use content management's library services for each application, you must configure each BEA Repository datasource (one per BEA Repository) to be XA-enabled. For more information regarding XA connections, see *Creating WebLogic Configurations Using the Configuration Wizard*.

Note: Content for each enterprise application is managed through its respective WebLogic Portal Administration Console and Virtual Content Repository. Virtual Content Repositories (as well as the WebLogic Portal Administration Console) are unique to each application cannot be shared.

● **Hardware Limitations** – Variables such as heap size, memory, and CPU usage can affect this configuration if the applications are targeted to the same server.

● **Personalization** – The same local property sets cannot be shared between multiple enterprise applications. If common properties must be shared among different enterprise applications, then use Unified User Profile (UUP). Another alternative is to copy and deploy the same property sets to multiple applications. For detailed information on user profiles and property sets and personalization, see the *Interaction Management Guide*.

Application Tuning Tips

The best practice is to use deployment plans to modify descriptor values before deploying an application. See "Using Deployment Plans" on page 4-6. See also the WebLogic Server document "WebLogic Server Performance and Tuning."

Deploying JSR-168 Portlets in a WAR File

WebLogic Portal provides a utility for automatically deploying JSR-168 portlets that are packaged in JSR-168 WAR files. This utility lets you import JSR-168 WAR files containing JSR-168 portlets, and expose the portlets in WSRP producers.

Note: As of WebLogic Portal 10.0, portlets

This section explains how to use the import utility and includes these topics:

● Starting the Import Utility

● Using the Import Utility

- Accessing the Portlets

Starting the Import Utility

To start the utility, do the following:

1. Log into the WebLogic Portal Administration Console. You can do this by entering the following URL in a browser:

   ```
   http://servername:port/earProjectNameAdmin
   ```

   where `servername` is the IP name of Administration Server, `port` is the port number, and `earProject` is the name of the portal Enterprise application that is deployed on the server. For example:

   ```
   http://localhost:7001/myEarProjectAdmin
   ```

2. In a browser, enter the following URL:

   ```
   http://servername:port/earProjectAdmin/jsr168/jsr168import.jsp
   ```

   where `servername` is the IP name of Administration Server, `port` is the port number, and `earProject` is the name of the portal Enterprise application that is deployed on the server. For example:

   ```
   http://localhost:7001/myEarProjectAdmin/jsr168/jsr168import.jsp
   ```

   The utility is shown in Figure 4-1.

Figure 4-1 JSR-168 WAR Import Utility
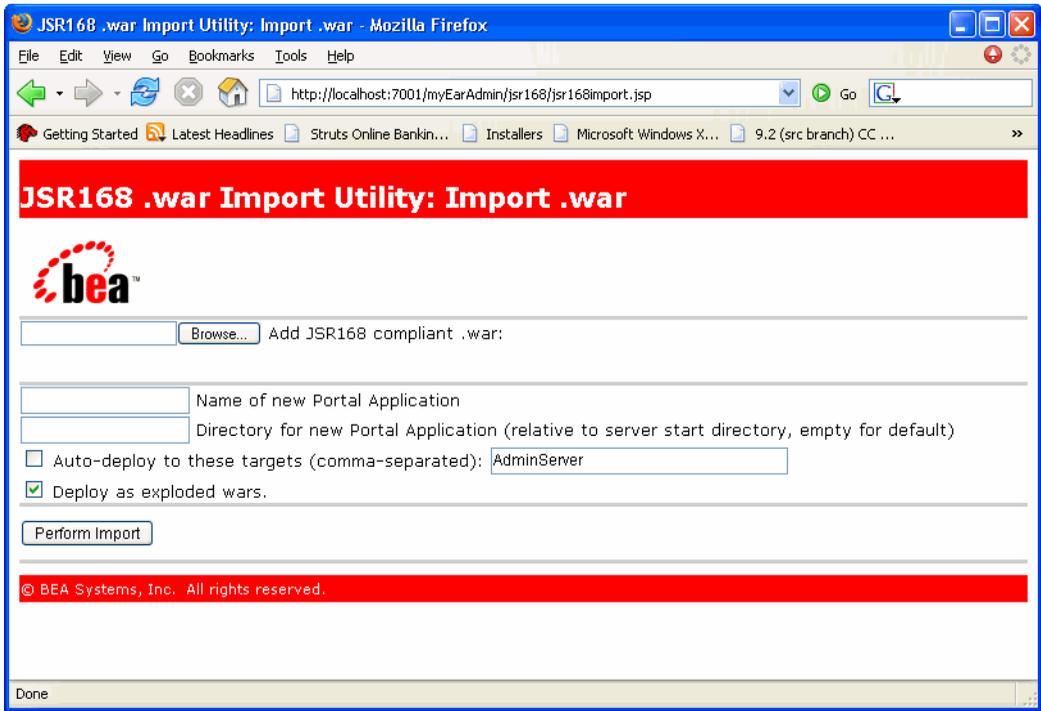
Using the Import Utility

This section explains how to use the import utility to import JSR-168 WAR files into an enterprise application.

1. Use the **Browse** buttons to select a WAR file containing the JSR168 compliant portlets that you want to deploy. The WAR file(s) must be physically located on the same WebLogic Server machine on which the utility is running. You can select multiple WAR files.

    Tip: You can either locate the WAR file using the **Browse** button or by entering the path directly in the **Add JSR-168 compliant .war** text field. If you enter the path in the text field, press the Tab key to accept the path.

2. Enter a name for the new portal application. The utility creates this application (an EAR file). The EAR file is built using the selected templates in the same manner that WebLogic Workshop builds an application. Each JSR-168 WAR file becomes a web application that is packaged in the EAR. Use a unique name for the application. (Do not use a name that matches any existing applications deployed to the server.)

3. Enter a directory pathname in which to place the EAR file. This path is relative to the server start directory. By default, the file is placed in the server start directory (the directory from which the WebLogic Server startup script `startWebLogic.cmd` or `startWebLogic.sh` was run.) For a single server environment, the server start directory is the domain directory. For a managed server, the server start directory is the managed server directory.

4. If you select the **Auto-deploy to these targets** checkbox, the new portal application is automatically deployed to the specified targets. By default, the EAR is deployed to the Administration Server. Typically, this option is only used for simple development or test deployments. If you want to upload the new application to a more complex environment, such as a staging or production environment, it is recommended that you do not select this checkbox. Instead, use the WebLogic Server Console to deploy the EAR.

5. Select **Deploy as exploded wars** to deploy the WAR files exploded inside the EAR file. Only do this if you plan to perform some additional manipulation on individual web application files.

6. Click **Perform Import** to create the new application. If **Auto-deploy to running server** is selected, the newly created EAR file is deployed automatically. If you did not select the checkbox, you need to deploy the EAR using the WebLogic Server Console. You can find the new EAR file in the directory you specified previously in Step 3.

Accessing the Portlets

After the new EAR file is deployed, you can add the portlets contained in the imported WAR file(s) to your application. To do this, you need to add the web application(s) as WSRP producers. After a web application is added as a producer, you can incorporate the application's portlets as you would with any WSRP producer using the WebLogic Portal Administration Console. See the *Federated Portals Guide* for details.

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

Using Production Redeployment with WebLogic Portal

This section discusses using the production redeployment technique to redeploy a WebLogic Portal application to a production environment.

Caution: Production redeployment is known to work for a limited number of use cases. The most reliable and common use case is to add, remove, or update web-tier (file based) resources. For example, typical uses of production redeployment include updating JSP code or updating image files.

If your changes require changes to the portal database or other data sources, production redeployment is not generally recommended. Although other more complex uses of production redeployment are possible, they will depend on a number of other factors that are described in this section.

This section includes these topics:

- What is Production Redeployment?

- Conceptual Overview and Limitations

- Application Redeployment Scenarios

- Production Redeployment Issues and Limitations

- Side Effects of Production Redeployment

What is Production Redeployment?

Production redeployment enables a portal administrator to redeploy a new version of an application in a production environment without stopping the deployed application or otherwise interrupting the application's availability to clients. Production redeployment works by deploying a new version of an updated application alongside an older version of the same application. WebLogic Server automatically manages client connections so that only new client requests are directed to the new version. Clients already connected to the application during the redeployment continue to use the older, retiring version of the application until they complete their work.

Tip: If you plan to use production redeployment with a WebLogic Portal application, we recommend that you also consider using federated portal techniques (WSRP) to maintain and update specific parts of your portal. Such a strategy can augment your redeployment

strategy by reducing the overall scope of a given deployment. For detailed information on federated portals, see the *Federated Portals Guide*.

Production redeployment support in WebLogic Portal is based on support from WebLogic Server. Before continuing, we recommend that you review the following WebLogic Server documentation:

- "Using Production Redeployment to Update Applications"
- "Developing Applications for Production Redeployment"

Conceptual Overview and Limitations

Figure 4-2 depicts a typical production redeployment scenario for a WebLogic Portal application. As the diagram shows, a new portal version, App V2, has been deployed side-by-side with the original App V1. During production redeployment, existing users continue to interact with App V1 while new users interact with App V2. When all user sessions connected to App V1 end or time out, App V1 is undeployed.

The key to understanding how production redeployment works for a portal application is to note that during production redeployment, *both the old and new application share the same database, security repository, and content repository*.

Figure 4-2 Production Redeployment Overview
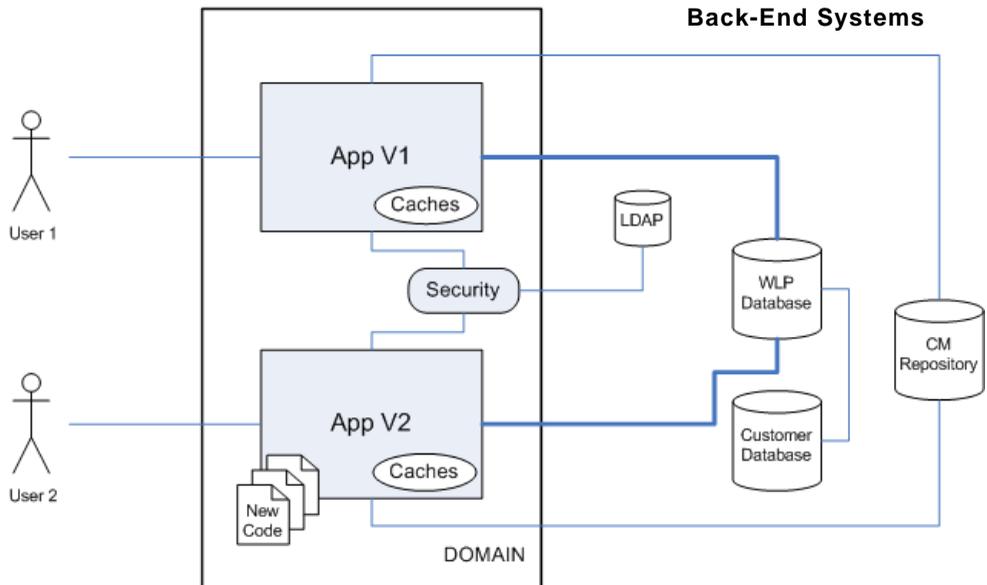
Because both applications share the same back-end systems, side effects can occur during production redeployment. To illustrate this, consider the following scenario. Developers have added a new portlet to the App V2 portal's home page. When the new portal is deployed, that portlet and the updated page definition containing the new portlet are placed in the production database. Now, because both the old and new applications share the same database, App V1 may attempt to display the new portlet. Unfortunately, any supporting file-based code that underlies the portlet will not exist in the App V1 deployment, resulting in an error.

This simple example scenario illustrates that side effects during production redeployment typically occur because the shared database contains application-specific references. Potentially, differences that exist between the underlying deployments of the "side-by-side" applications can result in errors. Keep in mind that all external touch points to your application, such as application resources that are referenced in the database, must be consistent between the two applications. For example, the class name of a portlet backing file is stored in the database along with the portlet's definition. If you rename or remove a backing file, existing users accessing the original application may encounter errors. Similarly, if you add a new page to your portal that contains new portlets, during production redeployment the page and portlet definitions will be added to the database and could inadvertently show up in the original application. In this case users of the original application could experience errors.

> **Tip:** If the only underlying differences between the old and new application are in the collection of files and application code that are not referenced by the database, the production redeployment is likely to be successful.

These scenarios illustrate possible side effects that can occur during production redeployment of a WebLogic Portal application. Later sections discuss additional, possible side effects as well as recommended use cases for production redeployment.

Overview of Basic Steps

> **Caution:** It is recommended that you perform all the required steps for production redeployment in a staging environment first, before redeploying in a live production environment.

Figure 4-3 illustrates the basic steps for production redeployment with a WebLogic Portal installation.

Figure 4-3 Basic Steps to Update a Portal Using Production Redeployment

<u>**Deployment Time Line**</u>

Place Old Application (V1) in Maintenance Mode

Deploy New Application (V2)   **Administration Mode**

Propagate (if necessary)

Verify

V2 Goes Live   **Allow New Client Connections**

V1 Sessions Time Out

V1 Undeployed
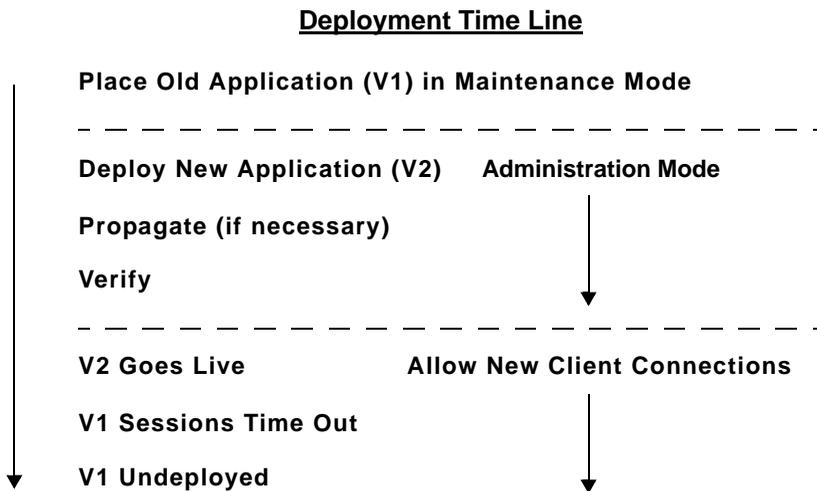
1. Place the old application in maintenance mode. See "OnlineMaintenanceModeTask" on page 9-11 for details.

Caution: If your WebLogic Portal site allows visitor customization or administrator activity, it is strongly recommended that you use Maintenance Mode or another strategy to restrict or eliminate such activity during production redeployment. Maintenance mode prevents administrators from making changes to the portal through the WebLogic Portal Administration Console. You can enable Maintenance Mode in the WebLogic Portal Administration Console by selecting **Configurations > Service Administration**. For more information, refer to the Administration Console online help.

2. Follow the instructions in the WebLogic Server document, "Using Production Redeployment to Update Applications," to deploy the new application "side-by-side" with the old application in the same domain.

Tip: The WebLogic Server document, "Using Production Redeployment to Update Applications," offers a detailed, step-by-step procedure for production redeployment that applies to any application deployed on WebLogic Server, including portal applications.

Caution: It is recommended that you initially deploy the new application version in Administration mode. Administration mode makes the application available only via a configured Administration channel. External clients cannot access an application that has been distributed and deployed in Administration mode. For more information, see the WebLogic Server document, "Starting a Distributed Application in Administration Mode."

3. Use the propagation tools to propagate the updated application from the staging environment to the production environment. Propagation is a required step if any changes to the portal were made using the Administration Console in the staging area.

Tip: For background information on propagation, including use cases, see Chapter 5, "Developing a Propagation Strategy."

4. After propagation, administrators can run the new application and verify that it is functioning properly in the new environment.

5. Finally, you can switch off Administration mode. The application begins to allow new client connections. At that point all new user connections are directed to the new application. After all user sessions end or time out on the old application, the old application is undeployed.

The following sections discuss possible scenarios for production redeployment and the known limitations of each.

Application Redeployment Scenarios

This section describes several scenarios in which application redeployment can be used with WebLogic Portal.

Caution: If you use production redeployment in any case where database changes are made, you cannot roll back or revert the application to its original state. The database contains representations of portal framework and datasync elements used by the portal at runtime. For example, values for features such as user customization and entitlements are stored in the database.

This section discusses the following redeployment scenarios:

- Adding, Removing, or Updating a Non-Portal Asset

- Adding, Removing, or Updating Portal Assets with Database Changes

- Deploying and Propagating

Adding, Removing, or Updating a Non-Portal Asset

Adding, removing, or updating a non-portal asset refers to a adding or removing an asset that does not require any change to the database. In this scenario, production redeployment is a safe and recommended technique for achieving zero-downtime redeployment.

Examples:

- Adding new images to a portal

- Presentation changes, such CSS changes

- New servlets or EJBs that are not referenced by a portlet

Adding, Removing, or Updating Portal Assets with Database Changes

Caution: Anytime database changes or changes to any back-end data source are required, production redeployment is not recommended. If you decide to use production redeployment, use extreme caution. Thoroughly test your redeployment in a staging environment first. For more information, see "Side Effects of Production Redeployment" on page 4-28.

In this scenario, assets that require database changes are added to or removed from the new version of the portal. If you change the structure of a portal, such as by adding a new page to an existing book, upon deployment, the changes are automatically pulled into the database.

In this scenario, WebLogic Portal does not fully support production redeployment. This is because application database rollback is not supported. WebLogic Portal cannot revert the changes in the database once they have been applied.

Note: Application rollback is not supported whenever you use the propagation tools to propagate portal assets. See "Rolling Back an Import Process" on page 6-29 for more information.

Examples:

- Adding, removing, or modifying portlets, books, pages

- Adding, removing, or modifying datasync files

Deploying and Propagating

In this scenario, the newly deployed application requires portal assets to be propagated. Because propagation alters the portal database, application rollback is not supported in this case. For an introduction to propagation, see Chapter 5, "Developing a Propagation Strategy."

Caution: Anytime database changes or changes to any back-end data source are required, production redeployment is not recommended. If you decide to use production redeployment, use extreme caution. Thoroughly test your redeployment in a staging environment first. For more information, see "Side Effects of Production Redeployment" on page 4-28.

Note: Application rollback is not supported whenever you use the propagation tools to propagate portal assets. See "Rolling Back an Import Process" on page 6-29 for more information.

Production Redeployment Issues and Limitations

This section discusses several issues and limitations related to production redeployment of a WebLogic Portal application. Please review this section before attempting to use application redeployment with a WebLogic Portal application.

This section includes these topics:

- Application Rollback

- Memory and CPU Requirements

- Application Rollback

- Propagation

- Database Instances

- Database Changes

- Cache Invalidations

- Content Repositories

- Applications that Use Expression-Based Roles

- External Systems

Application Rollback

Reversing or "rolling back" the production redeployment process switches the state of the active and retiring applications and redirects new client connection requests accordingly. Reverting the production redeployment process might be necessary if you detect a problem with a newly-deployed version of an application, and you want to stop clients from accessing it. Application rollback is described in the WebLogic Server document "Rolling Back the Production Redeployment Process."

Generally, anytime your production redeployment changes WebLogic Portal database entries, application rollback is not supported for WebLogic Portal production redeployment. Again, because the old and new applications share the same database, if the database changes during redeployment, those changes cannot be rolled back, leading to possible side effects for users of the original application.

For more information and examples of scenarios where database changes occur, see "Application Redeployment Scenarios" on page 4-24.

Memory and CPU Requirements

Because the production redeployment feature executes two full version of the WebLogic Portal application at the same time, you need to consider memory and CPU requirements. While it is impossible to specify the exact amount of memory and CPU that will be required for production redeployment, a good rule of thumb is to double the current running load on the server. For example if the application is taking one gigabyte of memory and using 40% of the current CPU then it should be assumed that the second version will also require the same about of memory and CPU. For this reason, you may want to consider scheduling your production redeployments when there is less user activity and load on the WebLogic Portal server instances.

Propagation

The WebLogic Portal propagation tools do not support database rollback. For this reason if a deployment scenario requires propagation to be used, roll back of the application cannot be performed.

See "Rolling Back an Import Process" on page 6-29 for more information.

Database Instances

It is recommended that you use the same data store between versions of a WebLogic Portal application.

Database Changes

WebLogic Portal does not support changes to the database schema when using production redeployment.

Cache Invalidations

The WebLogic Portal cache framework only invalidates cache entries for the current application. Any applications that are running during production redeployment will not see cache invalidations. Because of this, it is possible to see stale cache entries between different version of the application.

Content Repositories

File-based content repositories are not supported for production redeployment.

Applications that Use Expression-Based Roles

Production redeployment is not supported for any application that uses expression-based role policies. For information on defining roles using expressions, see the *WebLogic Portal Security Guide*.

Note: Because GroupSpace applications use expression-based role policies, WebLogic Portal does not support production redeployment of GroupSpace applications.

External Systems

External systems that depend on a WebLogic Portal application can be affected when using production redeployment. Because production redeployment only uses HTTP session

management to manage which version of an application a user is connected to, the following non-HTTP frameworks always reference the newer version of the application:

- **WSRP** – A WSRP consumer maintains a session state with a producer on behalf of a user. Therefore, any external application consuming a WSRP portlet will transition to the new application version after the session times out.

- **EJB** – Do not access EJBs from outside of the WebLogic Portal application. See the WebLogic Portal document "Using Production Redeployment to Update Applications," for more information.

- **JMS** – Any MDB processing messages or messages sent to the WebLogic Portal application will be processed by the new version of application.

- **External Databases** – Any databases or database tables that your portal accesses will be processed by the newer version of the application.

- **Content Management Repositories** – If the content management system interacts with portal assets stored in the database, the interactions must be consistent between the old and new versions of the redeployed application. For example, if the newer application is configured to do full text searches, but the original application is not, full text searches will inadvertently appear in the original application.

- **Other Back-End Resources/Systems** – Any back-end systems must not change schema or API or protocol between application versions.

Side Effects of Production Redeployment

This section lists known side-effects of using production redeployment with a WebLogic Portal application. This section includes these topics:

- Customizations

- Portal Resources

- Entitlements

- WSRP

Customizations

Customization to desktops are not immediately viewable in new or old versions of a portal application. For example, if an administrator alters users' desktops in the new version of an application, the change may not be immediately viewable. Because customizations are cached for

users of the old version of the application, users will not see the new customization until the cache times out.

Portal Resources

When the new version of the application is deployed, WebLogic Portal alters the portal database. For example, when a new portlet is added that does not exist in the old application, the database is updated for the new application. Because of this, users currently using the older version of the application may see the portlet but not have the ability to render it because the file system assets do not exist in the older version.

Entitlements

Entitlements have both LDAP and RDBMS artifacts, meaning they will be shared between versioned applications which use the same database. One side effect of this is that a WebLogic Portal Administration Console administrator in the older application could affect entitlements in the versioned application (and vice versa).

WSRP

A WSRP consumer maintains a session state with a producer on behalf of a user. Therefore, any external application consuming a WSRP portlet will transition to the new application version after the session times out.

System Classpath Changes

Any changes to system classpaths will require the server to be restarted.

Part II   Propagation

Part II includes the following chapters:

- Developing a Propagation Strategy

  This chapter explains what you need to consider before you attempt to move or propagate portal assets. Tools for moving portal assets include the Workshop for WebLogic propagation tools, the Export/Import Utility, and the Datasync web application.

- Propagation Topics

  This chapter provides information that you need to know before using the propagation tools.

- Using Workshop for WebLogic Propagation Tools

  This chapter explains how to use the propagation tools provided by Workshop for WebLogic.

- Using the Propagation Ant Tasks

  This chapter gives an overview of the propagation Ant tasks and discusses related topics such as scoping and policies.

- Propagation Ant Task Reference

  This chapter provides information about each page of the propagation Ant tasks.

- Using the Export/Import Utility

This chapter explains how to use the Export/Import Utility. The chapter includes background information on the utility and its purpose. In addition, detailed examples are provided that illustrate common use cases.

- Using the Datasync Web Application

The Datasync Web Application is deprecated.

Developing a Propagation Strategy

The steps you take to successfully move a portal configuration from one environment to another depend on many variables. It is impossible to prescribe a single procedure that applies to all circumstances. Therefore, before you attempt to move or propagate a portal application, it is important to plan your strategy, pick the appropriate tools, and develop a set of procedures based on recommended best practices. This chapter explains what you need to consider before you attempt to move or propagate portal assets.

The topics included in this chapter are:

- What is Propagation?

- What Tools Does BEA Provide to Assist with Propagation?

- What Kind of Data Can Be Propagated?

- Choosing the Right Propagation Tool

- Propagation Roadmap

- Assessing Your Portal System Configuration

- General Propagation Scenarios

- Production Mode Versus Development Mode

- Propagation and Proliferation

What is Propagation?

Specifically, propagation refers to moving the database contents of a portal application from one server environment to another. To accomplish this, BEA provides these tools:

- A Propagation Session feature in Workshop for WebLogic

- Ant tasks that let you develop propagation scripts

- The Export/Import Utility, for moving portal assets from the database of a staging environment to files that can be imported back into Workshop for WebLogic. This utility also lets you move and merge portal assets from a Workshop for WebLogic environment to a staging environment's database.

Each of these utilities are discussed further in the following section, What Tools Does BEA Provide to Assist with Propagation?.

Figure 5-1 shows the three typical environments between which portal assets are moved.

Figure 5-1 Moving Portals Between Environments
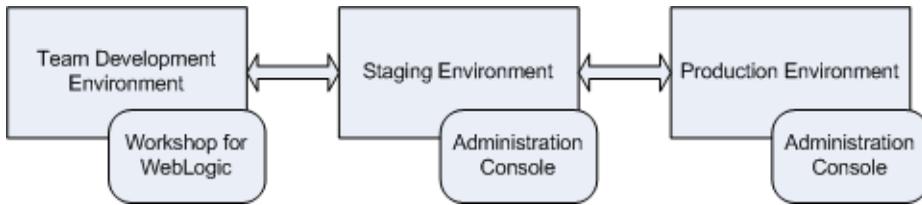
These three environments include:

- **Development** – In the development environment, a team of developers uses Workshop for WebLogic to create portals and portal components, such as portlets.

- **Staging** – In the staging environment, administrators use the Administration Console to build and configure portal desktops, create entitlements, and set up content repositories.

- **Production** – The production environment is the "live" website. In a production environment users access the web application and, optionally, customize it using Visitor Tools. Administrators can also use the WebLogic Portal Administration Console to modify the production environment.

What Tools Does BEA Provide to Assist with Propagation?

As you develop a propagation strategy, it is important that you know what tools are available to help you propagate a portal from one environment to another. This section introduces the primary tools at your disposal and their purposes.

WebLogic Server Administration Console (EAR Deployment)

Use the WebLogic Server Administration Console to deploy an Enterprise Application's EAR file to a target server. EAR deployment is almost always the first step in any propagation. The EAR file must be redeployed any time changes are made using Workshop for WebLogic. For example, if developers add or remove pages from a `.portal` file, define content selectors, create new portlets and related Java resources (such as JSPs), then the EAR file must be built and deployed to propagate those changes to a new environment.

For detailed information on EAR deployment, see Chapter 4, "Deploying Portal Applications."

Workshop for WebLogic Propagation Tools

Workshop for WebLogic provides tools that guide you through the process of propagating the configuration contents, including portal framework, datasync, and security data, of one portal domain environment to another. For example, you can create a Propagation Session to move a portal application from a staging environment to the production environment.

Tip: Before propagating a portal, always deploy the EAR file in the target environment first.

Features of the propagation tools include:

- **Exporting portal assets** – Export the portal assets stored in the portal application's database into an XML file.

- **Differencing** – View the differences between portal assets in the source application and the target application. For instance, you can tell at a glance if a particular page in a portal was added or removed from the target configuration.

- **Importing portal assets** – Portal assets that have been exported from a source configuration can be imported into the target. Before importing, you can view the differences between the assets stored in the two configurations. On import, merge decisions are made based on well-defined policies.

- **Setting Policies** – Policies are user-definable rules that determine which source assets will (or will not) be merged into a target configuration. For instance, a policy may state that if an asset exists on the source, but not on the destination, add it to the destination. Another policy may state that if an asset exists on the destination, but not on the source, delete it from the destination. A Propagation Session allows you to customize the policies that govern these merge operations. For more information on policies, see "Using Policies" on page 6-22.

- **Setting Scope** – Scope defines which portal artifacts will be *included* in a propagation. By default, scope is set to the entire Enterprise application, including the contents of content repositories; however, you can modify the scope. For detailed information, see "Understanding Scope" on page 6-11.

For detailed information on using the WebLogic Workshop propagation tools, see Chapter 7, "Using Workshop for WebLogic Propagation Tools."

Propagation Ant Tasks

Instead of using Workshop for WebLogic to propagate a portal, you can use Ant tasks to create customized propagation scripts. In many cases, task based propagation provides an even greater measure of control over your propagation than is provided by Workshop for WebLogic. In fact, the Ant tasks underlie the Workshop for WebLogic propagation tools. For detailed information on the Ant tasks, see Chapter 8, "Using the Propagation Ant Tasks" and Chapter 9, "Propagation Ant Task Reference."

Manual Propagation Steps

The propagation tools provided by BEA handle most of the work necessary to move a portal web application from one environment to another. However, there are some manual steps that you may need to perform to ensure a successful migration. In some cases, these steps are required by design. For instance, some security data is intentionally not automatically propagated by the tools.

For detailed information on manual propagation steps, see "Make Required Manual Changes" on page 6-5.

Export/Import Utility

The Export/Import Utility allows you to export desktops, books, and pages from a database to a `.portal` file. This `.portal` file can then be opened with Workshop for WebLogic, modified, and then merged back into the database using the Export/Import Utility. This utility allows developers

to move portal assets in a "round trip" between a development environment and a staging environment, or between development environments.

The utility lets you scope its operations to the following levels:

- **Library level** – Assets that exist in the Library of the WebLogic Portal Administration Console

- **Administration level** – Assets that exist in desktops created in the WebLogic Portal Administration Console

- **Visitor level** – Asset instances that have been customized by users through Visitor Tools or user customizations

In addition, the utility lets you specify a set of rules to determine how the objects are merged. You can also specify different scoping rules, from the Enterprise Application scope (at the highest level) down to pages within books. This flexibility helps ensure that the user's and administrator's customizations will not be lost when the assets are merged.

For detailed information on the Export/Import Utility, see Chapter 11, "Using the Export/Import Utility."

Database Vendor Tools (Not Supported)

BEA does not support the use of database vendor tools as a means of propagating any WebLogic Portal assets from one database environment to another.

What Kind of Data Can Be Propagated?

Generally speaking, a WebLogic Portal application consists of an EAR file, an LDAP repository, and a database. The EAR file contains application code, such as JSPs and Java classes, and portal framework files that define portals, portlets, and datasync data. The embedded LDAP contains security-related data, such as entitlements, roles, users, and groups. The database contains representations of portal framework and datasync elements used by the portal runtime.

This section divides portal data into four categories, and lists the types of data that fall into each category. The categories include:

- **Portal Framework Data** – Refers to desktops, books, pages, and other portal framework elements that are created with the WebLogic Portal Administration Console.

- **Datasync Data** – Refers to definition files, such as user profiles and content selectors that WebLogic Portal allows you to create. Within Workshop for WebLogic, these definitions

are placed within a Datasync Project folder. This data is deployed with the EAR file. If the destination server is running in production mode, datasync data is placed in the database.

- **Security Data** – Refers to policy, role, and delegated administration data. This data is generated by administrators using the WebLogic Portal Administration Console.

- **EAR Data** – Refers to the final product of Workshop for WebLogic development—a J2EE EAR file. The EAR must be deployed to a destination server using the deployment feature of the WebLogic Server Administration Console.

- **Content Management Data** – WebLogic Portal's content management system allows you to store content, track its progress, and incorporate content in your portal applications. It provides an easy integration between creating content and delivering that content to your users.

Table 5-1 lists the specific kinds of data that comprise each of these categories.

Table 5-1 General Categories of Data to Propagate

Portal Framework Data	Datasync Data	Security Data	EAR Data	Content Management Data
• Desktops • Portals • Books • Pages • Portlets • Portlet Preferences • Community Templates	• Catalog data • Content selectors • Discounts • Events • Placeholders • Request properties • Segments • Session properties • User profiles • Campaigns	• Policies (visitor entitlement and delegated administration) • Roles (global, visitor entitlement, and delegated administration) The following delegated administration policies: • portal resources (Library and Desktop level) • content management • user group management • content selectors • campaigns • visitor roles • placeholders • segments • security providers Note that content management workflows are not supported.	• `.portal` • `.portlet` • `.shell` • `.theme` • `.menu` • `.layout` • `.laf` • `.jsp` • `.class` • `.book` • `.page`	• items • types • metadata • folders

Note: EAR data consists of files that are generated by developers using Workshop for WebLogic. If you make a change to EAR data in Workshop for WebLogic (for instance, modifying a theme), the only way to move those changes to a staging environment is to redeploy the EAR file. Note, however, that desktops, books, pages that are created in the WebLogic Portal Administration Console usually contain references to EAR data. For instance, an administrator can assign a specific theme to a page using the WebLogic Portal Administration Console, and a reference to the actual theme file is maintained in

the database. When you propagate a desktop to another environment using the propagation tools, those references are propagated, but the actual file the reference points to (for example, a `.theme` file) is not. EAR deployment and the propagation tools are described in the following sections.

The following sections discuss the tools that WebLogic Portal provides for moving portal data between environments.

Choosing the Right Propagation Tool

Table 5-2 shows the appropriate propagation tool to use depending on the type of propagation and the specific kinds of changes to propagate. Items in the Changes to Propagate column are defined in "What Kind of Data Can Be Propagated?" on page 5-5, and the tools listed are summarized in "What Tools Does BEA Provide to Assist with Propagation?" on page 5-3.

For example, the first row of the table indicates that if you are making an initial deployment from a development (Workshop for WebLogic) environment to a staging environment, you simply deploy the EAR file. However, on redeployment, you need to also use the propagation tools to move datasync data to the target (see also "Moving the Datasync Data" on page 5-17).

Table 5-2 Choosing a Propagation Method

Source Environment	Destination Environment	Changes to Propagate	Propagation Method
Development	Staging • Production mode * • Initial deployment	• Portal framework files (`.portal`, `.book`, etc.) • Datasync data • Other EAR data (`.jsp`, `.java`, etc.)	Deploy EAR
Development	Staging • Production mode * • Redeployment	• Portal framework files (`.portal`, `.book`, etc.) • Other EAR data (`.jsp`, `.java`, etc.)	Deploy EAR
Development	Staging • Production mode * • Redeployment	• Datasync data	Propagation tools

Table 5-2 Choosing a Propagation Method

Source Environment	Destination Environment	Changes to Propagate	Propagation Method
Staging • Production mode *	Development	• Portal framework *database data* (desktops, books, and pages)	Export/Import Utility
Staging • Production mode *	Development	• Datasync data	Propagation tools
Staging • Production mode * • WebLogic Portal Administration Console changes	Production • Production mode * • WebLogic Portal Administration Console changes	• Portal Framework data • Datasync data • Security data	Propagation tools **Note:** The same EAR file that was deployed to the staging system must also be deployed at least once to the production system.

* It is a best practice to always run staging and production servers in production mode. See also "Production Mode Versus Development Mode" on page 5-22.

Propagation Roadmap

As you plan your propagation strategy, it is important to develop a roadmap of your system. This section describes the interrelationships between a typical system that includes development, staging, and production environments. The roadmap, Figure 5-2, shows how portals are moved across these environments and the tools that are used to move them.

This section is intended to help you understand not only the connections between the different systems on which portals exist, but the tools and methods used to move or propagate portals between those systems. The numbered parts of the figure are described in detail in the remainder of this section. Please refer to the figure as you read the following sections.

Figure 5-2 Propagation Roadmap
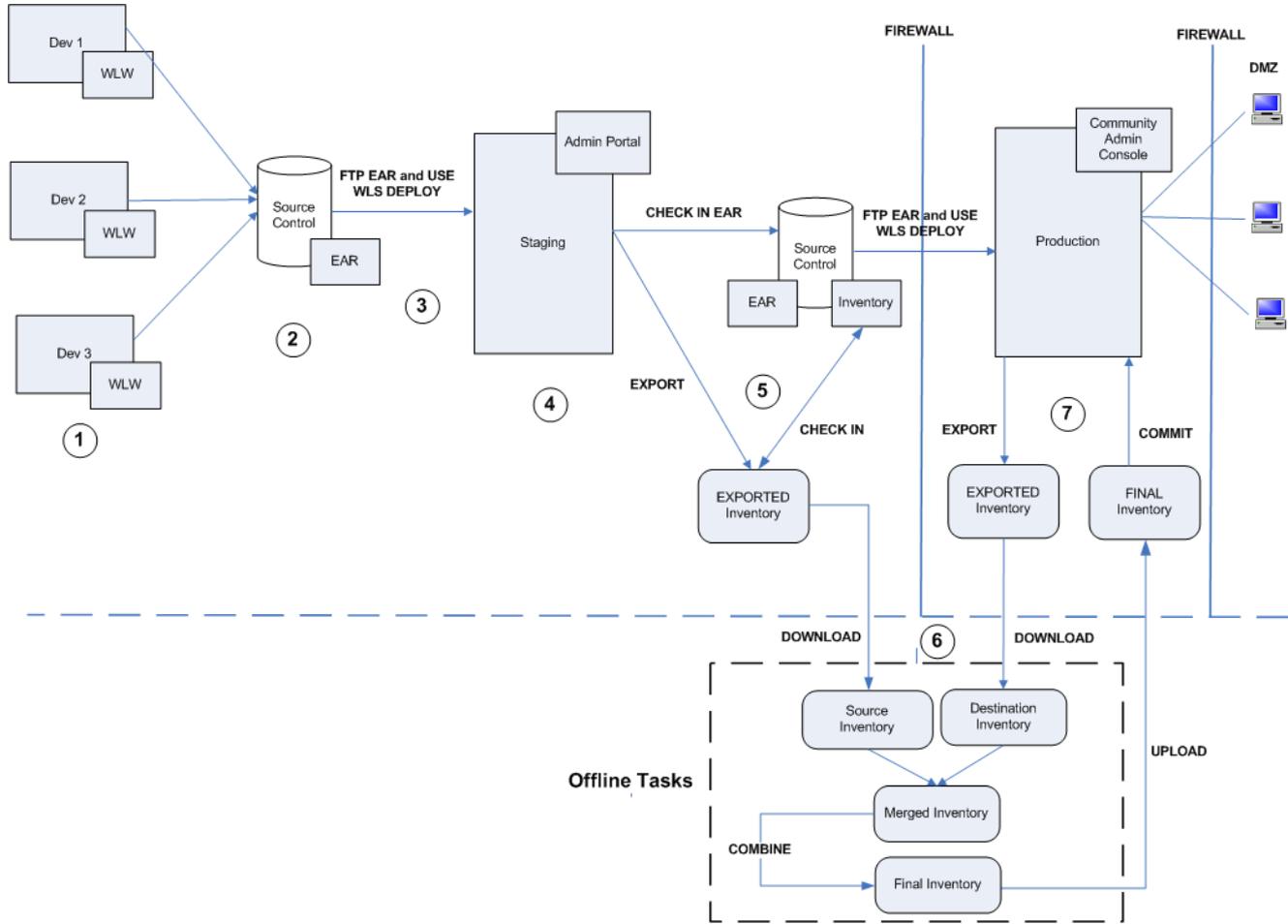
① Development Environments

Portal development is typically spread out among members of a team using Workshop for WebLogic on individual client machines. Refer to Chapter 2, "Managing a Team Development Environment" for detailed information on setting up a multi-developer portal development environment.

It is important to remember that files are the primary, and often only, product of the IDE-based environment. Workshop for WebLogic allows developers to create Java components that are used by portals, such as Java Pageflows, Controls, and JSPs. Workshop for WebLogic also lets developers create portlets, portals, look and feels, layouts, and so on. All of these components are stored in files, such as `.java`, `.jsp`, `.jpf`, `.jcs`, `.portal`, `.page`, `.book`, `.portlet`, `.laf`, and others.

② Source Control

Development typically occurs in conjunction with a source control system. As explained in Chapter 2, "Managing a Team Development Environment," a common domain is established for the team, but the web application itself is checked into source control where individual developers can check it out, create and modify files, and check them back into source control. Once built, the EAR file can also be checked into source control.

③ Moving from Development to Staging

When development is complete, an EAR file can be deployed to the staging environment. The easiest way to do this is to use FTP to move the EAR to the staging server, and then to use the WebLogic Server Deployment feature to deploy the EAR into the J2EE server environment. See Chapter 4, "Deploying Portal Applications" for more details.

Note: It is possible to run the Administration Console from Workshop for WebLogic. Whenever the Administration Console is used, the output is stored in a database, not in a file. Therefore, if you are in development and use the Administration Console to create Desktops, users, groups, entitlements, or other administrative features, that information is stored in a database. Assets in the database are not included in the EAR file, and will not be transferred when you move and deploy the EAR.

Tip: The best practice in portal development is to use the WebLogic Portal Administration Console only in a staging or production environment. If you use the Administration

Console in a development environment, then you must use the propagation tools to move the database assets to the staging environment.

④ Staging Environment

In the staging environment the WebLogic Portal Administration Console is used to assemble the portal components that were created in development into desktops, to create users and groups, assign administrative privileges, configure delegated administration rights, modify books and pages, and so on. It is important to remember that anytime you use the WebLogic Portal Administration Console to modify a portal, all portal assets from that point on are stored in the database: the connection to the original `.portal` and other files created in Workshop for WebLogic is lost.

Tip: It is possible to return portal framework assets stored in the database back to files using the Export/Import Utility. To move portal assets to a production environment (database to database), the best practice is to use the WebLogic Workshop propagation tools or Ant tasks.

For details on the Export/Import Utility, see Chapter 11, "Using the Export/Import Utility." For details on the propagation tools, see Chapter 7, "Using Workshop for WebLogic Propagation Tools."

⑤ Source Control in the Staging Environment

It is a best practice to employ source control in the staging environment. Two components to store in source control are the web application's EAR file and the application's portal-specific assets, or *inventory*. You can extract the inventory from a web application using the Workshop for WebLogic based propagation tools or the propagation Ant tasks to export the inventory into a ZIP file.

For details on the propagation tools, see Chapter 7, "Using Workshop for WebLogic Propagation Tools." For details on the propagation Ant tasks, see Chapter 9, "Propagation Ant Task Reference."

⑥ Perform Offline Tasks

You work with exported inventories offline. That is, you work with them directly on the filesystem, with no connection to the source or destination server. The offline tasks include creating a merged inventory, viewing and tuning the merged inventory, and combining the merged inventory into a final inventory.

Note: At this point, you can view the differences between the source and destination inventories (the *merged view*) and decide whether or not to go ahead with the propagation. Differences can include portal assets that have been added, deleted, or updated. For example, if a page was added to a desktop in staging the propagation tools will report that a page was added if it does not exist in the production environment. Similarly, if a page was deleted from the production environment, the propagation tools will report that it was deleted and give you the option of adding it back or not (if it still exists on the staging server).

Before you can perform offline tasks, you need to download a source and a destination inventory. When you finish the offline tasks, you upload the final inventory to the destination system. You can upload and download inventories using the OnlineUploadTask and OnlineDownloadTask Ant tasks or use the Import and Export features in Workshop for WebLogic.

⑦ Committing the Final Inventory

After you have uploaded the final inventory to the destination server, you must commit it. You can use the OnlineCommitTask Ant task to commit the final inventory, or use the Export feature in Workshop for WebLogic.

Assessing Your Portal System Configuration

It is important for each site that deploys WebLogic Portal to develop a strategy for propagating portal applications from one environment to another. When you plan a propagation strategy, it helps to assess carefully the structure of your site and your methods of portal development. Some questions to ask include:

- **What is the portal life cycle pattern for your site?** Will you be propagating primarily from a development server to a production server, or is there also an intermediate staging/testing environment? For example, in development, developers typically use Workshop for WebLogic to create portal assets. In staging and production, administrators use the WebLogic Portal Administration Console to create and configure portal desktops. In addition, users of the production system can use Visitor Tools to customize portals.

Also, extensive testing typically occurs in the staging environment. For an overview of the environments you can propagate to and from, see the previous section, "Propagation Roadmap" on page 5-9.

- **What *mode* is your production server in?** For more information on modes, see "Production Mode Versus Development Mode" on page 5-22.

- **Are you deploying your application for the first time, or are you redeploying it?** For details, see "General Propagation Scenarios" on page 5-15.

- **If you are redeploying an existing application, what kinds of changes were made to it, and how were they made?** Did developers use Workshop for WebLogic to modify, create, or remove portal resources (such as adding a new book and pages to a portal, and adding portlets to the pages)? Was your application modified in a staging environment by administrators using the Administration Console? For details, see "General Propagation Scenarios" on page 5-15.

- **What is the scope of your propagation?** Do you wish to propagate the entire Enterprise application, a single web application, or a desktop? For details on scope, see "Understanding Scope" on page 6-11.

- **Do you want to push a portal from a staging or production environment back to a** Workshop for WebLogic **environment?** For information on accomplishing this task, see Chapter 11, "Using the Export/Import Utility."

- **Do you want to push a portal from a production environment back to a staging environment?** This can be accomplished with the propagation tools or the propagation Ant tasks. See Chapter 7, "Using Workshop for WebLogic Propagation Tools" and Chapter 8, "Using the Propagation Ant Tasks."

- **Have you defined entitlements for portal assets, such as portlets?** If so, you need to be aware that users and groups are, by design never propagated by the propagation tools, and you may need to manually recreate specific user and group definitions on the production server. For information on how security elements are handled during propagation, see "Security Information and Propagation" on page 6-10.

The next section discusses typical propagation scenarios and the tools and methods that are used in each.

General Propagation Scenarios

After you familiarize yourself with the available propagation tools, the next challenge is to decide when and where to use them. This section presents several general propagation scenarios and offers suggested best practices.

Example Environment

The scenarios outlined in this section assume an environment where there are separate development, staging, and production servers, as shown in Figure 5-3.

Figure 5-3 Example of a Portal Development and Production Environment
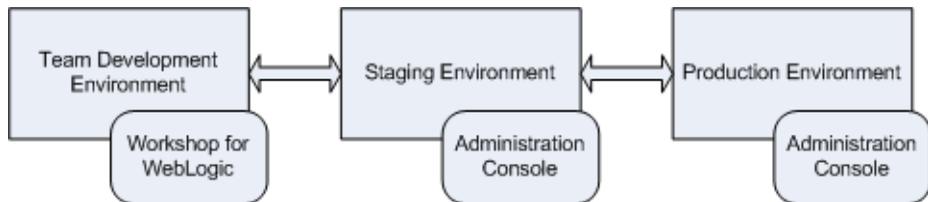
In a *development environment*, developers use Workshop for WebLogic to create portals and portlets. In this environment, all portal-related data is file-based (stored in XML files, such as `.portal` and `.portlet` files). At the completion of development, these file-based assets, including Java and JSP files, configuration files, and datasync files, are assembled and compressed into an EAR file.

A *staging environment* has been established on a server that is separated from the development environment. The staging server will be used to test the application and to further configure it. In the staging environment, administrators use the Administration Console to create and arrange portal desktops, entitle portal resources, create users and groups for testing purposes, and so on.

The *production environment* is the "live" web site. In this environment, users are accessing and using portal applications. Both administrators and users can make changes to the portal in the production environment. Administrators use the Administration Console to effect changes, and users use the Visitor Tools to customize their individual portal views.

Scenario 1: Deploying the EAR file for the first time

If you are deploying an EAR file to a server for the first time, the procedure is relatively easy. Typically, developers have used Workshop for WebLogic to create portals, portlets, and other application features, and they wish to deploy the new application to a staging server.

For detailed information on deploying an EAR file, see Chapter 4, "Deploying Portal Applications."

Note: When you begin using the WebLogic Administration Console, data stored in the EAR is automatically pulled out of the EAR and stored in the staging server's database. At this point, all subsequent modifications to the portal occur in the database only. Changes are not reflected back into the EAR file.

Scenario 2: Redeploying an EAR file

As with the first deployment described previously, the first steps in redeploying are to build the EAR file, move it to the staging server, and use WebLogic Server Console to deploy the EAR on the staging server.

Tip: For detailed information on using WebLogic Server Console to redeploy an application, see "Redeploying to a Staging or Production Environment" on page 4-10.

When you redeploy the application, there are caveats depending on whether the server you are deploying to is in development mode or production mode. Recall that you choose the server's mode when you create the domain. A server in production mode is optimized to run more efficiently than one in development mode. The caveats are explained in the following sections.

If the Target Server is in Development Mode

- Any changes made to the EAR file, including datasync data, are automatically detected and deployed to the target server.

- New portlets are automatically detected and moved into the Library of the Administration Console.

- New books and pages added to a portal are *not* automatically added to the Library. They are added only if you use the Administration Console to create a new Desktop that uses the new books and pages. This includes .book and .page files that you add to your portal project.

If the Target Server is in Production Mode

- *Datasync data is ignored if the EAR is redeployed to a server in production mode.* In this case, you must propagate the datasync data using one of the propagation tools. See the following section, Moving the Datasync Data, for information on moving the datasync data.

- New portlets are automatically detected and moved into the Library of the Administration Console.

- New Books and Pages added to a portal are *not* automatically added to the Library. They are added only if you use the Administration Console to create a new Desktop that uses the new books and pages.

Moving the Datasync Data

The way in which datasync data is handled during deployment depends on whether the server is in development mode or production mode. (See also "Production Mode Versus Development Mode" on page 5-22.)

As Table 5-3 shows, when you deploy an Enterprise application, datasync data is placed in the /data directory (it is deployed to the filesystem) except in the case where you redeploy an EAR file to a server in production mode. In the latter case, use appropriate WebLogic Portal propagation tools to move the datasync assets to the target server's database.

Table 5-3 Where Datasync Data Is Stored When You Deploy an Enterprise Application

Mode of Target Server	Deploy an Exploded EAR	Deploy an EAR File
Development Mode	• Datasync data saved on filesystem (META-INF/data) • Not recommended for cluster deployment	• Not recommended because application files are not editable • Datasync data is read from the filesystem (read-only) • Not recommended for cluster deployment
Production Mode	• Generally, EAR files are preferred for production mode. **On Initial Deployment** • Datasync data is bootstrapped from the filesystem (META-INF/data) to the database. Thereafter, datasync data is always in the database. **On Redeployment** • Datasync data is stored in the database. • You must propagate datasync assets to the target database as a separate operation, as explained in this section.	**On Initial Deployment** • Datasync data is bootstrapped from the filesystem (META-INF/data) to the database. Thereafter, datasync data is always in the database. **On Redeployment** • Datasync data is stored in the database. • You must propagate datasync assets to the target database as a separate operation, as explained in this section.

When you redeploy an EAR file (not an exploded EAR) to a server that is in production mode, you have to propagate the datasync data as a separate operation. To do this, use the propagation tools to propagate the database from the source to the target server. The propagation tools let you view the differences between the two environments and automatically handle merging those differences based on configurable policies.

Scenario 3: Propagating from Staging to Production: Default Scope

When propagating a portal application that is scoped to the highest level (enterprise scope) from a staging to a production environment, it is assumed that the target server is in production mode. In this case, everything that can be propagated is propagated.

Tip: For detailed information on scoping an inventory, see "Scoping an Inventory" on page 8-10.

Having the production server in production mode is the best practice. However, nothing prevents a site from running the "live" production server in development mode. Be aware that if the target server is running in development mode, datasync data is automatically moved when the EAR is redeployed. If the target is running in production mode, EAR-based datasync data is ignored.

The basic steps for propagating from staging to production environments are:

1. Deploy the EAR file.

2. Use the propagation tools to propagate the database assets from the staging server to the production server.

Note: Typically, the two environments will be out of sync. It is possible that changes were made to both the production and staging servers since the previous propagation. See the section "Scenario 5: Propagating from Production to Staging: Both Have Changed" on page 5-20.

Tip: Be aware that users and groups, by design, are not propagated. Therefore, typically, items such as groups (which contain users) must be manually recreated on the production server (Although, if the staging and production servers share the same LDAP directory, and this authentication information is stored in LDAP, it is not necessary to recreate the users and groups.). For more information on LDAP propagation, see "Security Information and Propagation" on page 6-10.

Scenario 4: Propagating from Staging to Production: Desktop Scope

If you do not wish to propagate the entire enterprise application, you can adjust the scope. For example, you can use the propagation tools to propagate only a desktop. In this case, you must be aware that datasync data will not be propagated because it resides at the Enterprise application scope and is not included in the propagation that is scoped to the desktop level. Therefore, you need to propagate the datasync data separately in this circumstance using the propagation tools.

Tip: For detailed information on scoping an inventory, see "Scoping an Inventory" on page 8-10.

The basic steps for propagating an inventory that is scoped to the desktop level are the same as enterprise scope, described in the previous section.

Note: It is recommended as a best practice to initially scope to the Enterprise application level, and then to the desktop level.

Scenario 5: Propagating from Production to Staging: Both Have Changed

It is sometimes necessary to return a production environment configuration to a staging environment. Typically, this allows administrators and developers to add new features and fix known problems. The problem associated with this propagation is that both administrative and user customizations could have been made to the production system. For instance, an administrator may have added a new desktop, or removed a page. A user may have customized her individual view of a portal using Visitor Tools.

To complicate the scenario, additional development and customization may have occurred in the staging environment since the application was last propagated. The problem, then, is to identify the differences between the two environments and decide which changes to keep and which to remove.

The propagation tools report differences between the source and target environments. While the propagation policies allow you to set rules for merging portal assets, it is up to you to review the differences and verify that the changes you intend to make to the target are made.

Note: User customizations are not propagated, but they are preserved on the destination. Changes introduced through the propagation tools have the same impact on user customizations as similar changes made through the WebLogic Portal Administration Console. For more information see "User Customizations and Propagation" on page 6-28.

Tip: If you make a change to the staging environment using the WebLogic Portal Administration Console, the changes you make are saved in the database if the EAR file is compressed. If the EAR is uncompressed (exploded) changes are written to the filesystem. If you redeploy the EAR file at a later time, you must be sure to propagate datasync elements that have been changed. To do this, use the propagation tools. For more information, see "Production Mode Versus Development Mode" on page 5-22.

Scenario 6: Round-Trip Development

The Export/Import Utility allows you to propagate desktops, books, and pages back and forth between development (Workshop for WebLogic) and staging environments. Propagating from development to staging and back to development is called a round trip.

The Export/Import Utility exports desktops, books, and pages from the staging database to `.portal`, `.book`, and `.page` files that can be read into Workshop for WebLogic. The utility also allows you to import `.portal`, `.book`, and `.page` files into a staging database. You can set the scope of imports and exports to the library, desktop, or visitor level.

WARNING: When a new asset, such as a new book or page, is created in the Administration Console, a unique identifier is generated for that asset. **It is a best practice to avoid changing these definition labels once they have been propagated (or moved with the Export/Import Utility) for the first time.** If you change a resource's definition label, the Export/Import Utility views the resource as a new resource rather than an updated resource. As a result, the Export/Import Utility will perform an add operation rather than an update operation.

Figure 5-4 shows the definition label for a page, displayed in the Properties view in Workshop for WebLogic.

Figure 5-4 Definition Label in the Properties View
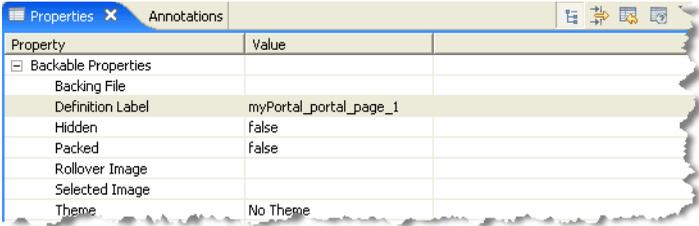
Tip: An important feature of the Export/Import Utility is that it allows you to merge file-based assets from the development environment into a database-based staging environment. In other words, if you export an application to a file-based development environment, and then make changes in the development environment, you can use the Export/Import Utility to merge those changes back into the database of the staging environment.

Production Mode Versus Development Mode

When you configure a domain, you are given a choice between Development mode and
Production mode.

Tip: As a best practice, always run staging (testing) and production (live) environments in
production mode. For developers using Workshop for WebLogic, the best practice is to
run the server in development mode.

Knowing the server's mode is important for understanding how certain portal assets are
propagated. For instance, when you deploy an EAR file to a server that is in production mode,
datasync data is ignored. See "Scenario 2: Redeploying an EAR file" on page 5-16.

Propagation and Proliferation

Proliferation refers to the process by which changes made to the Library instance of a portal asset
are pushed into user-customized instances of that asset. For example, if a portal administrator
deletes a portlet from a desktop, that change must be reflected into user-customized instances of
that desktop. Before you propagate a portal, consider the way in which proliferation is configured
for your portal.

If your desktops include a large number of user customizations, we recommend that you change
the **Portal Resources Proliferation of Updates Configuration** setting to either **Asynchronous**
or **Off**. This change reduces the amount of time required to complete the propagation.

You can do this in the WebLogic Portal Administration Console under **Configuration Settings
> Service Administration > Portal Resources > Portal Resources Proliferation of Updates
Configuration**. The proliferation settings include **Asynchronous**, **Synchronous**, or **Off**.

Propagation Topics

This chapter covers a range of topics related to WebLogic Portal propagation. We recommend that you read this chapter before you attempt to propagate a portal.

Note: WebLogic Portal inventories saved with WebLogic Portal 8.1 or 9.2 cannot be used with WebLogic Portal 10.0 propagation tools.

This chapter includes the following topics:

- Overview

- Before You Begin

- Propagation Reference Table

- Security Information and Propagation

- Understanding Scope

- Using Policies

- Previewing Changes and Tuning a Merged Inventory

- User Customizations and Propagation

- Reviewing Log Files

- Rolling Back an Import Process

- Federated Portal (WSRP) Propagation

- Increasing the Default Upload File Size

- Configuring the Propagation Servlet

Overview

Figure 6-1 shows the basic flow of a propagation session. The basic steps in any propagation session include:

1. First, you export the portal inventory from the source system and the destination system.

2. Merge the inventories. These propagation tools let you view and modify the merged inventory files before producing a final inventory file.

3. Create a final merged inventory. The final inventory represents the combination of the two inventories after all scoping and policy rules have been applied. Scoping is discussed in "Understanding Scope" on page 6-11. Policies are discussed in "Using Policies" on page 6-22.

4. The final inventory file is then uploaded and committed to the destination server.

Figure 6-1 Flow of a Propagation Session
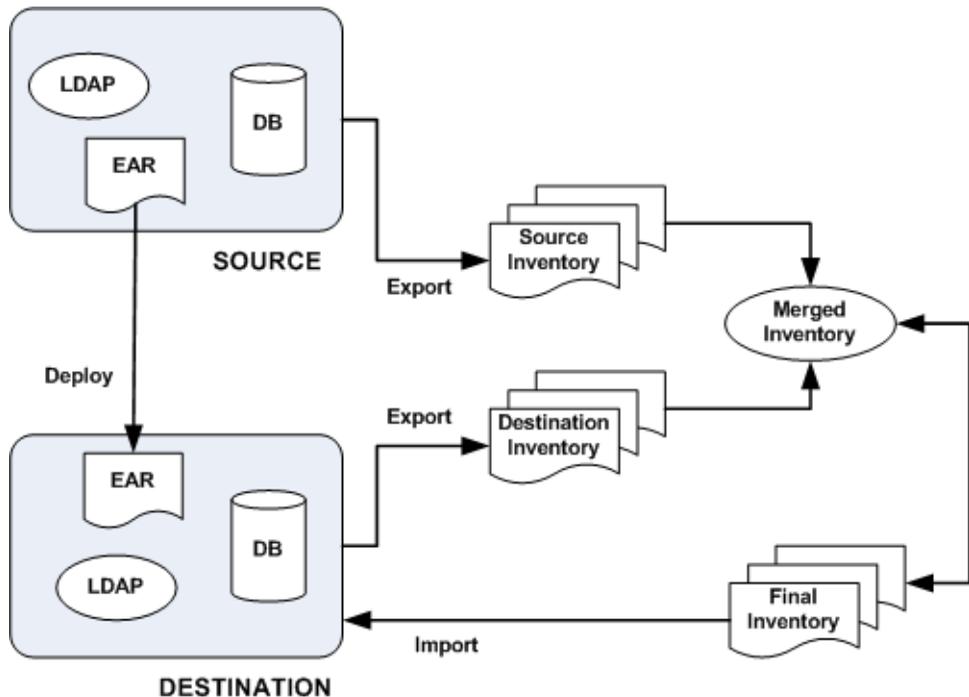
Before You Begin

The following sections describe some preparation tasks to perform before propagating a portal application. For additional information about the propagation planning process, see Chapter 5, "Developing a Propagation Strategy."

Note: It is not recommended to propagate between source and destination systems that are running JVMs with different default locales. If the locales differ between the source and destination system, performance will be slower and the localization node of all Portal Framework assets will always appear to be different in the change manifest.

Start the Administration Server

The Administration Server must be running when you perform a propagation to allow certain LDAP data to be updated. A propagation can cause the following kinds of LDAP data to be

added, deleted, or updated: visitor roles, delegated administration roles, entitlement policies, and delegated administration policies.

Perform a Data Backup

Back up your WebLogic LDAP repository using the steps in the "Avoiding and Recovering from Server Failure" section of the WebLogic Server documentation.

Back up your database using the vendor tools appropriate for your environment. Save a copy of the deployed application that you are propagating; you will be deploying an EAR to the existing application, which will overwrite the current configuration.

Plan to Inactivate the System During the Import Process

Before you import a final inventory to a destination system, take steps to prevent users from changing portal data. When the final inventory is being committed to the destination server, any changes made through the WebLogic Portal Administration Console can lead to unwanted side effects, such as lost data.

The OnlineMaintenanceModeTask Ant task can be used to prevent changes to the destination server. You can also use the WebLogic Portal Administration Console to place a server in maintenance mode. To do this, select the **Configuration Settings > Service Administration > Maintenance Mode**. For detailed information on OnlineMaintenanceModeTask, see "OnlineMaintenanceModeTask" on page 9-11. Workshop for WebLogic does not provide a maintenance mode feature.

Install the Propagation Tools

The Workshop for WebLogic propagation feature is installed automatically when you install WebLogic Portal (unless you explicitly exclude it). The propagation Ant tasks require some additional configuration before you can use them. For information on installing and using the Ant tasks, see Chapter 8, "Using the Propagation Ant Tasks."

Configure Log Files (Optional)

For information on configuring verbose log files in Workshop for WebLogic, see "Enabling Verbose Logging" on page 7-22.

Deploy the J2EE Application (EAR)

If you created new resources for your web application using Workshop for WebLogic, including portlets, portals, books, pages, custom layouts, look and feels, menus, shells, themes, JSPs, or Java classes, you must deploy the J2EE application from the source system to the destination system at some point prior to committing the changes with the propagation tools.

Tip: It is a good practice to deploy the EAR file before you export the destination inventory. By doing this, you can reduce the number of manual changes you need to make after you commit the final inventory to the destination.

When you deploy the J2EE application, changes reflected in the EAR file may or may not be propagated automatically, depending on whether your server is in production mode or development mode. For more information, see "General Propagation Scenarios" on page 5-15 and "Scenario 2: Redeploying an EAR file" on page 5-16. For detailed information on deploying EAR files, see "Deploying Portal Applications" on page 4-1.

If you commit propagation changes without deploying the J2EE application, the inventory listing includes the new resources, but the resources are not displayed in the portal library on the destination system and the following destination database tables are not updated with new data:

- PF_MENU_DEFINITION

- PF_LAYOUT_DEFINITION

- PF_LOOK_AND_FEEL_DEFINITION

- PF_SHELL_DEFINITION

- PF_THEME_DEFINITION

- PF_PORTLET_DEFINITION

For more information on these database tables and how they are used, see the *BEA WebLogic Portal Database Administrator Guide*.

Make Required Manual Changes

While the propagation tools propagate most portal data, there are some exceptions. Any portal assets that are not propagated must be added to the destination server using the WebLogic Portal Administration Console.

What Kinds of Data Require Manual Changes?

In general, the propagation tools do not propagate the following kinds of data:

- Users and groups

  Users and groups are, by design, not propagated.

- New content repositories

  You must create the new content repositories on the destination system before you can propagate content in those repositories.

- Out of scope dependencies

  If you use the scoping mechanism to exclude assets that other assets depend upon, you must manually add the dependencies to the destination server. See "Understanding Scope" on page 6-11 for more information.

Tip: For a comprehensive list of propagated and non-propagated portal resources, see "Propagation Reference Table" on page 6-7.

There might be cases where propagated data depends on other data that is not propagated. For example, delegated administration roles are propagated to the destination but the related users and groups are not, so you must manually add those related users and groups on the destination system.

Where are Manual Changes Reported?

Manual changes are reported to you in the following places:

- The OfflineDiffTask and OfflineExtractTask propagation Ant tasks can optionally write a list of manual changes to a file. For details on these tasks, see Chapter 9, "Propagation Ant Task Reference."

- All of the propagation Ant tasks allow you to optionally write messages to a log file. Any manual changes that are required are listed in the log file.

- A Workshop for WebLogic propagation session reports manual changes in the following ways:

  – The Console view displays manual changes.

  – In the Properties view, the **Manual Steps** field indicates if propagating the asset requires any manual changes.

– In the **Merged Files** tab, the editor highlights assets that must be manually updated as follows: the asset's name is shown in yellow and an Eclipse warning badge is shown on the asset's icon.

Propagation Reference Table

Table 6-1 provides a comprehensive list of the categories and types of data that comprise a portal and whether or not the propagation tools move them. Any type of data that is *not* propagated by the propagation tools is noted in the **Notes** column. Data that is not propagated might require a manual change to be made on the destination server. For more information on manual changes, see "Make Required Manual Changes" on page 6-5.

Table 6-1 Data Propagation Reference Table

Data Category	Data	Notes
Framework	Portals	
	Desktops	
	Books	
	Pages	In addition, Look and Feel references, such as a dependency between a page and its layout, are propagated.
	Portlets	
	Portlet Preferences	
	Portlet Categories	
	Community Templates	
Framework, *continued*	Shells	
	Themes	
	Menus	
	Layouts	
	Look and Feels	

Table 6-1 Data Propagation Reference Table (Continued)

Data Category	Data	Notes
Framework, *continued*	User Customizations	Not propagated. User customizations are preserved on the destination system, but may be modified when imported changes are applied to the destination. For more information, see "User Customizations and Propagation" on page 6-28.
Datasync	Catalog Property Definitions	
	Content Selectors	
	Discount Definitions	
	Event Definitions	
	Placeholders	
	Request Properties	
	Segments	
	Session Properties	
	User Profile Definitions	
	Campaign Definitions	
Runtime Data	Behavior Tracking Events	Not propagated.
	Orders	Not propagated.
	User Profiles	Not propagated.

Table 6-1 Data Propagation Reference Table (Continued)

Data Category	Data	Notes
Security	Global Roles (created using WebLogic Server)	**Note:** Definitions that you create using public entitlement APIs are not propagated.
	Visitor Entitlement Policy Definitions	**Note:** Definitions that you create using your own custom code (APIs) are not propagated.
	Delegated Administration Policy Definitions	**Note:** Only the roles that are required (that a given asset depends upon) are propagated. These delegated administration policies are propagated: • Portal resources (Library and Desktop level) • Users and groups • Content selectors • Campaigns • Visitor roles • Placeholders • Segments • Security providers
	Delegated Administration and Visitor Entitlement Roles	**Note:** Only the roles that are required (that a given asset depends upon) are propagated. **Note:** Roles that you create using your own custom code (APIs) are not propagated. **Note:** Users and groups are not propagated, but user and group identification is preserved; you must manually create users and groups that correspond with propagated roles.
	Users	Not propagated; the hashed passwords cannot be propagated.
	Groups	Not propagated.
WSRP	WSRP Registration Data Remote portlets	For more information on WSRP propagation, see "Federated Portal (WSRP) Propagation" on page 6-30.
Content Management	All Content Management information, items, types, metadata, and folders	Propagation will not move the checkout status (if a user has an item checked out in staging, the user will not have it checked out in production after propagation). **Note:** Workflows are not propagated.

Security Information and Propagation

Security information consists generally of authentication data and authorization data. As Table 6-2 shows, authorization data consists of roles and policies, while authentication consists of users and groups.

Table 6-2 Types of Security Data

Authorization	Roles	• Visitor entitlement • Delegated administration • Global (defined in WebLogic Server) • Delegated role hierarchy	Stored in internal providers or external providers, such as a custom Authorization provider.
	Policies	• Visitor entitlements • Delegated administration	
Authentication		• Users • Groups	Stored in internal providers or external providers, such as an RDBMS user store or an OpenLDAP. Also stored in provider configurations and audit trails.

Users and groups are never propagated by the propagation tools. In addition to user and group information, the propagation tools do not propagate the following:

- Provider configurations (although they are listed when an application is exported)

- Data from external providers

- Audit Trails

Because roles contain user and group information, you need to consider how user and group information is stored for your staging and production system; these two systems may or may not share the same authentication repositories.

If your systems do not share the same authentication repositories for managing users and groups, then after propagating a portal, you must manually edit each role to add the appropriate users and groups on the destination system. If the systems do share the same authentication repositories, then no manual changes to roles need to be made after a propagation.

For more information on manual propagation changes, see "Make Required Manual Changes" on page 6-5.

Note: It is a best practice to reference groups, but not specific users, in roles. This practice makes it easier to maintain roles when users need to be added or removed from the system.

Understanding Scope

You can think of a WebLogic Portal inventory as a large tree structure, with multiple levels of parent and child nodes. Each node in the tree represents the inventory for a part of the portal application. If a node is declared to be in scope, then that node and all of its descendents are in scope. That is, they will be included in the exported inventory file.

This section includes these topics:

- Overview

- Why Use Scoping?

- What are the Risks of Scoping?

- Best Practices for Scoping

- How to Set Scope

- The Effects of Scoping

- Scope and Library Inheritance

Overview

If you do not want to propagate an entire portal application, you can use scoping to limit the propagated assets. The most common use cases for scoping include:

- propagating only a content repository

- propagating part of a content repository, such as the contents of a folder

- propagating one or more specified desktops

The Workshop for WebLogic based propagation tools and the propagation Ant tasks both support scoping. The propagation tools let you manually make scoping decisions by selecting and unselecting nodes in the merged inventory tree. The Ant tasks let you control scoping with a `scope.properties` file that specifies which portal items are considered to be in scope.

> **Tip:** It's important to remember that scope defines which portal artifacts will be *included* in a propagation. By default, scope is set to the entire Enterprise application, including content repository data. When you specify scope either manually or in a properties file, you are specifying what to include. If you specify a single desktop, for instance, only that desktop and its contents (books, pages, and so on) are propagated. Any other desktops in the portal, content, and other items are not propagated. If you limit the scope to a single content folder, then that folder, its contents, and its parent folders are propagated; however, the contents of parent folders are not propagated.

Why Use Scoping?

Performance is the primary use case for scoping. Because a scoped inventory reduces the overall size of an exported WebLogic Portal inventory, propagation tasks typically run faster. If you have a large content repository, you might not want to export it when you propagate a portal. In this case, you can move the content repository, and all of its content, out of scope.

Another reason to scope is security-related. You may want to exclude certain sensitive content from an inventory if you plan to transport that inventory through insecure channels.

What are the Risks of Scoping?

The primary risk associated with scoping is that a scoped inventory file can include assets with dependencies that cannot be resolved. When an inventory is scoped, entire nodes (and descendants of the nodes) are excluded from the inventory. As a result, if included artifacts depend on excluded artifacts, those dependencies cannot be resolved, and the propagated portal may not function properly.

> **Tip:** If you want to exclude specific nodes from an inventory, we recommend that you use policies instead of scoping. Policies are explained in "Using Policies" on page 6-22.

Best Practices for Scoping

The following are best practices if you intend to consider scoping an inventory:

- If you want to selectively remove parts of a portal from an exported inventory, use policies rather than scoping.

- Scope to the highest level possible. This ensures the that the destination environment will closely mirror the source environment. If you scope to a lower level, such as a specific desktop, additional quality assurance testing may be required on the destination.

- If at all possible, avoid using scoping on an inventory file that has already been exported. It is preferable to use scoping only when you are extracting an inventory, such as with the Ant task OnlineDownloadTask.

- Do not scope an exported file that has already been scoped. If you do this, you cannot use the scoping features in the Workshop for WebLogic based propagation tools.

- Use scoping to exclude content repository data from an exported inventory, while including portal data.

- Use scoping to exclude portal data from an exported inventory, but include content repository data.

How to Set Scope

You can set the scope of a propagation in the Workshop for WebLogic based propagation tools and in the Ant tasks.

This section includes these topics:

- Using Workshop for WebLogic to Set Scope
- Setting Scope with Ant Tasks

Using Workshop for WebLogic to Set Scope

The propagation tools provide two methods of scoping. The first is in the New Propagation Session wizard, and the second is in the Merged Inventory view.

Tip: For detailed information on using the propagation tools, see Chapter 7, "Using Workshop for WebLogic Propagation Tools."

Setting Global Scope in the New Propagation Session Wizard

When you create a new Propagation Session, you select a source and a destination inventory to merge. Before the merge, however, you are given the chance to make scoping decisions in the Select Global Scope dialog, shown in Figure 6-2. As shown in the figure, when you select a node to be in scope, the node's ancestors are automatically selected.

Note: It is important to remember that when you select items in the Select Global Scope dialog, *you are selecting items to be in scope*. All other items in the enterprise application will be out of the scope of the propagation. Therefore, in Figure 6-2, only the selected shell definition and its ancestors will be propagated. For example, a content item's parent folder is included, but content types on which the content item depend are not automatically included.

To make selections in the Select Global Scope dialog, you must first select the **Limit the scope to the specified nodes** checkbox.

Figure 6-2 Select Global Scope Dialog
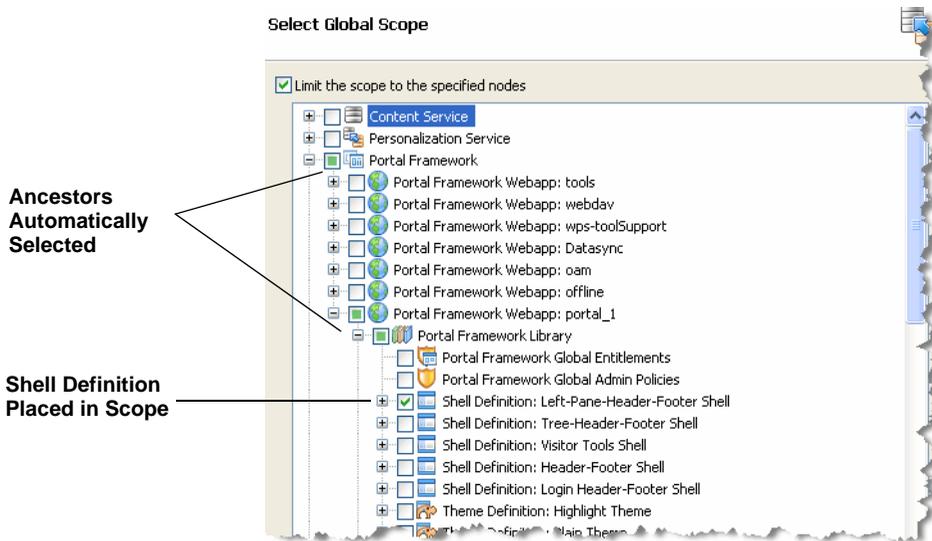
Scoping in the Merged Inventory View

You can refine the scope of a merged inventory in the Merged Inventory View. This view lets you make final changes to the scope of the merged inventory before you generate a final inventory file. Unlike the Select Global Scope dialog, all the nodes in the Merged Inventory View are selected. That is, they are all considered to be in scope by default. In Figure 6-3, the **Desktop_update_1** desktop is deselected. This removes that desktop and its dependents from the scope of the propagation.

Figure 6-3 Setting Scope in the Merged Inventory View
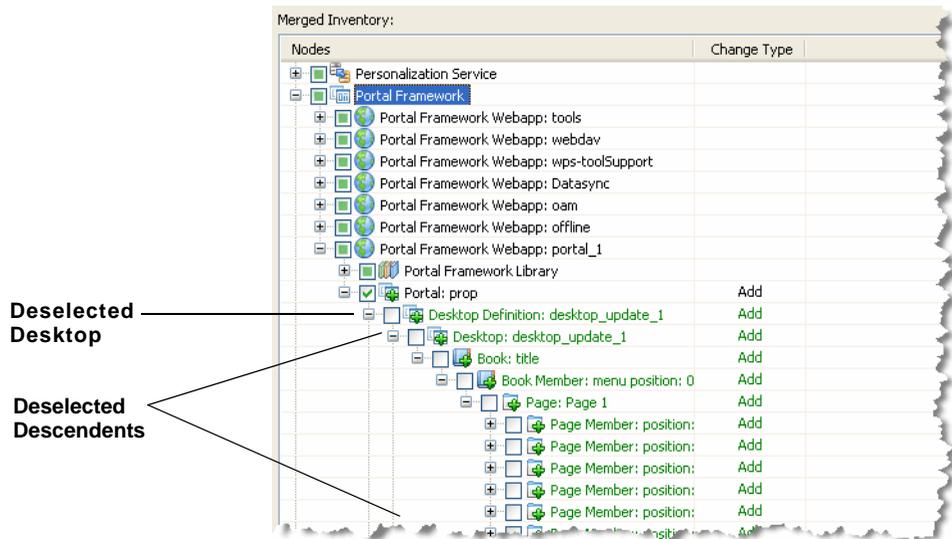
Setting Scope with Ant Tasks

You can use the OnlineDownloadTask to scope an inventory file. This task takes a
`scope.properties` file attribute. You can edit this file to include the items you want to include
in the inventory file. If you do not specify a `scope.properties` file, then the entire enterprise
application is considered to be in scope. For detailed information on the OnlineDownloadTask
and other propagation Ant tasks, see Chapter 9, "Propagation Ant Task Reference."

The Effects of Scoping

WebLogic Portal gives you a great deal of freedom to scope your portal inventories. When you
select an inventory node to be in or out of scope, that choice usually affects other nodes that are
either dependent on or a dependent of the selected node. WebLogic Portal automatically
determines these dependencies. In Workshop for WebLogic, you can see the dependencies
visually: when you select a node, other related nodes are automatically selected.

This section helps you to understand the effects of four kinds of scoping decisions:

- Scoping to the Enterprise Application Level (the default)

- Scoping to the Desktop Level

- Scoping to a Repository

- Scoping to a Content Folder

Note: When you deploy the EAR file to a server in development mode, datasync data and some portal framework data is automatically extracted and placed in the database. If you intended to exclude some or all of the datasync data with scoping, then you will find that it is added automatically by the EAR deployment, effectively negating the scoping intention.

Scoping to the Enterprise Application Level

By default, all propagations are scoped to the Enterprise application level. This means that all of the artifacts that make up the portal that can be propagated are propagated. By default, the entire Enterprise application is *in scope*, as Figure 6-4 illustrates.

Figure 6-4 Scoping at the Enterprise Application Level
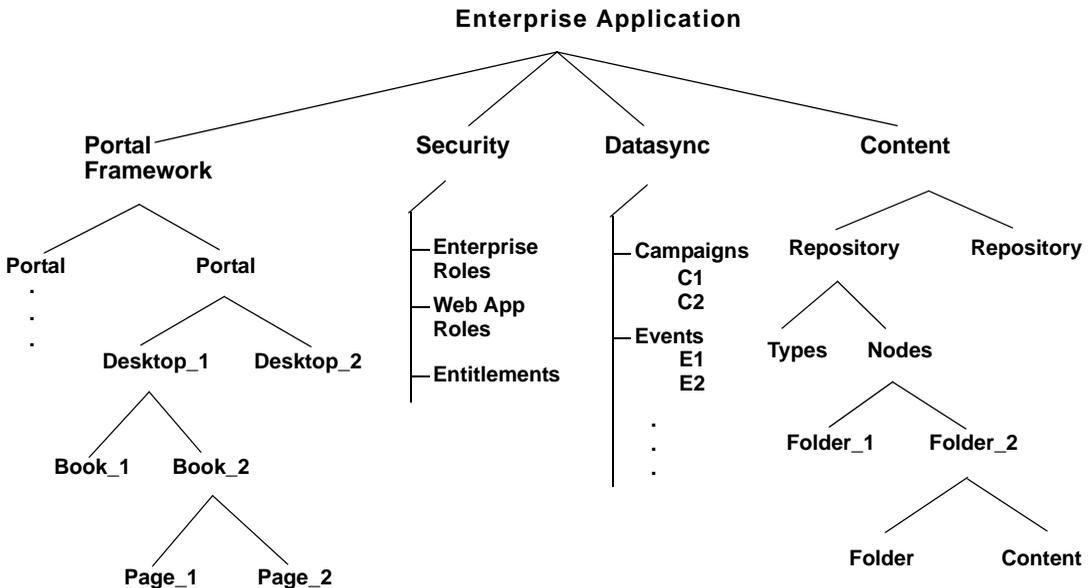
There are no preset scoping levels. Both the propagation tools and Ant tasks let you specify scoping however you want. The Ant tasks let you specify a scope.properties file, in which you specify which artifacts to propagate. (See also "Understanding a Scope Property File" on page 8-13.)

In Workshop for WebLogic, you scope visually by selecting or deselecting nodes in the inventory tree. The selected nodes are considered to be in scope and are propagated. All other nodes are not propagated.

There are four major categories of portal artifacts that you can scope to:

- **Content Service** – Includes content repositories, folders, content, types, and entitlements.

- **Personalization Service** – Includes datasync data, such as campaigns, content selectors, and events.

- **Portal Framework** – Includes portals, desktops, books, pages, and so on.

- **Security** – Includes delegated admin roles and policies, visitor roles, global roles, security providers, and others.

Each of these top-level nodes are represented in the New Propagation Session wizard's Select Global Scope dialog, as shown in Figure 6-5. Using this dialog, you can drill into one of the top-level categories to refine the level of scoping. However, as previously noted, the best practice is to scope only to one of the top-level categories, such as the Content Service.

Figure 6-5 Select Global Scope Dialog
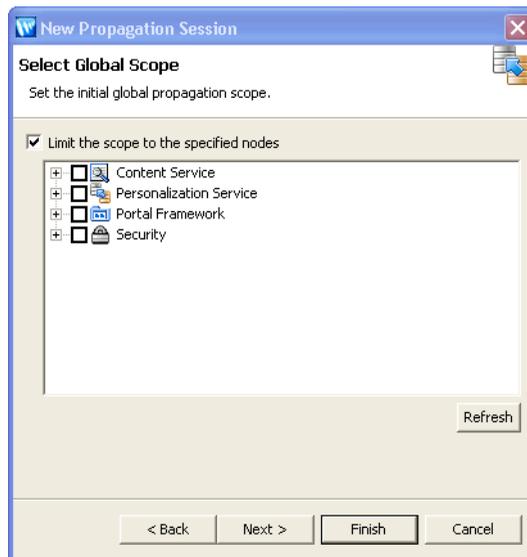
Scoping to the Desktop Level

One of the most common scoping use cases is scoping to the desktop level. This means that you identify one or more specific desktops to be in scope. The rest of the enterprise application is then considered to be out of scope and is not propagated. As Figure 6-6 illustrates, when you scope to Desktop_1, the desktop and its children (books, pages, and so on) are placed in scope. Any other desktops and the rest of the Enterprise application are then out of scope. See "What are the Risks of Scoping?" on page 6-12 for information on the risks of scoping at this level.

Figure 6-6 Scoping to Desktop_1
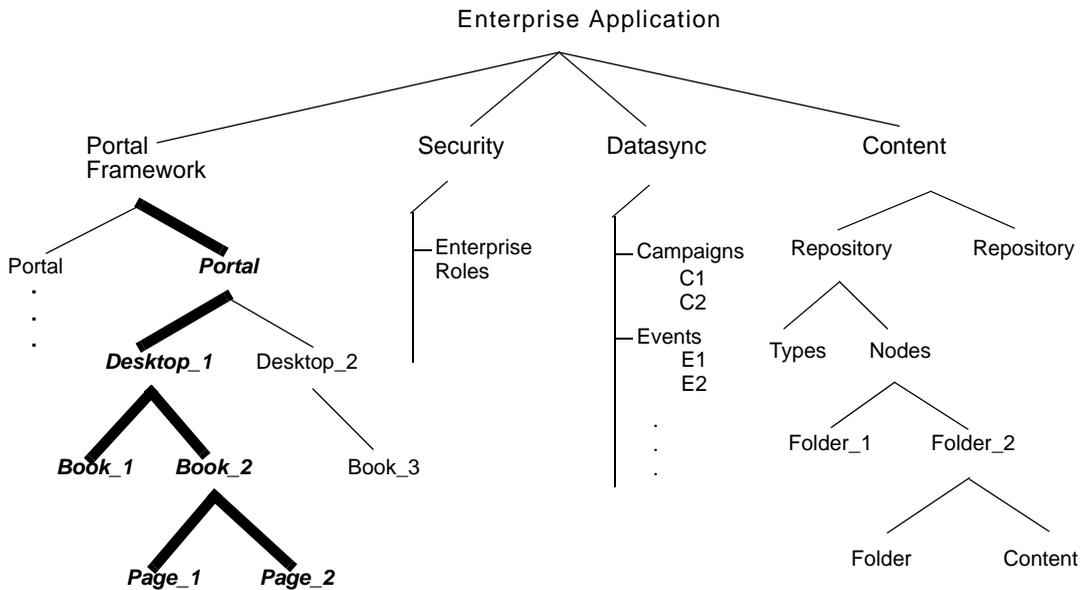
Scoping to a Repository

Another common use case it scoping to a repository. When you scope to a repository, the entire repository is propagated, including all types, folders, and content, as illustrated in Figure 6-7. Any other repositories are not propagated, and no other part of the Enterprise application is propagated.

Figure 6-7 Scoping to a Repository
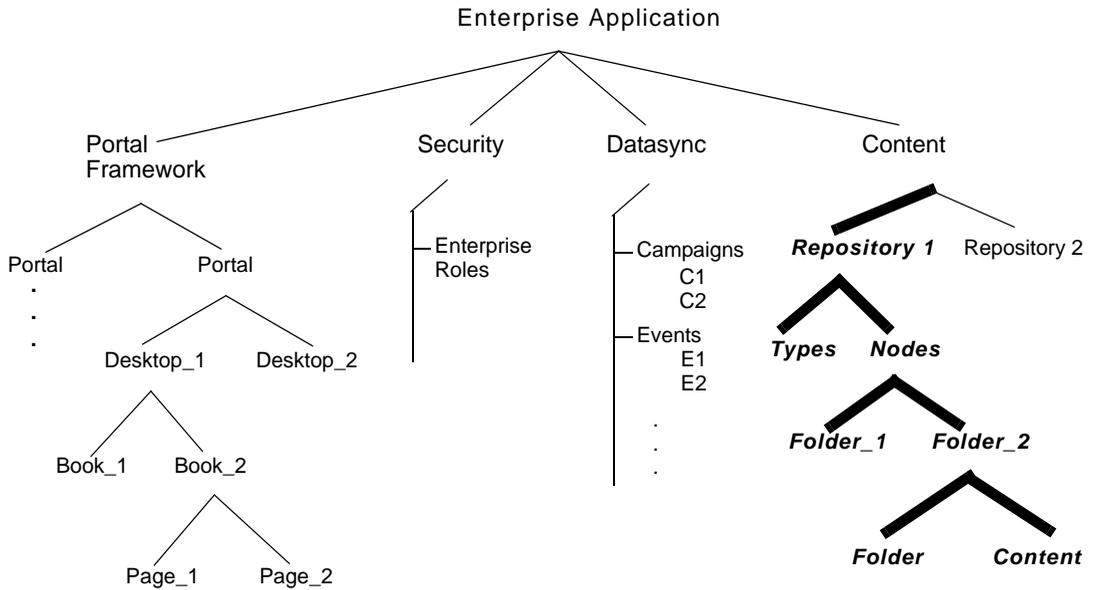
Scoping to a Content Folder

When you scope to a folder in a content repository, that folder, its contents, and all of the folder's parent folders are propagated. Note that the contents of parent folders are not propagated, only the parent folders themselves, as illustrated in Figure 6-8. See "What are the Risks of Scoping?" on page 6-12 for information on the risks of scoping at this level.

Figure 6-8 Scoping to Content Folder_3
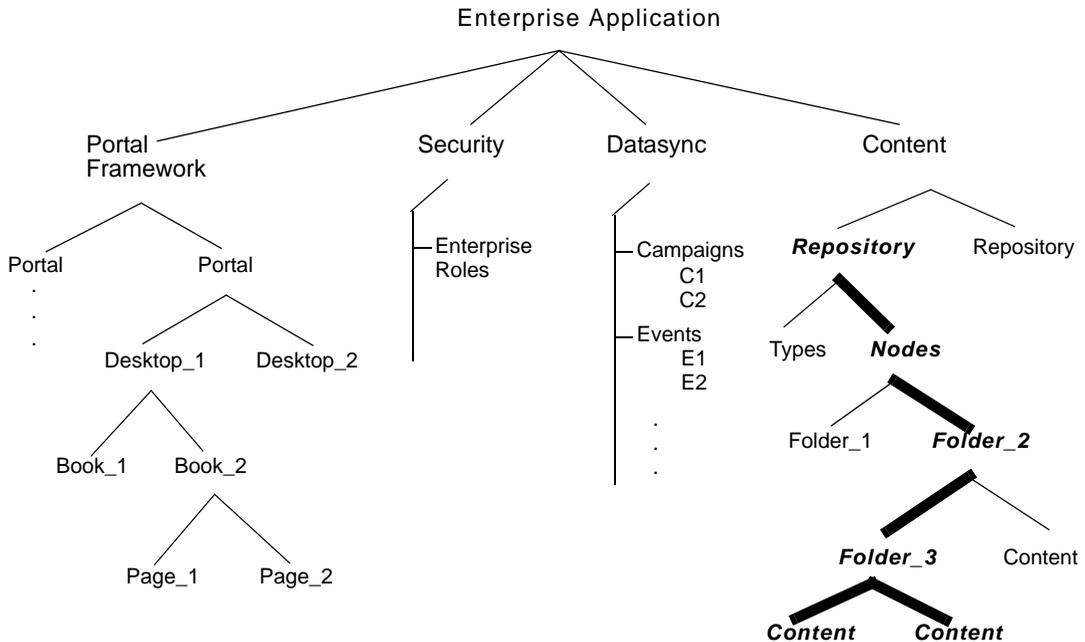
Scope and Library Inheritance

When you propagate portal assets, such as desktops containing pages and books, new pages and books are added to the Portal Library if they do not already exist there.

The WebLogic Portal Administration Console organizes portal resources in a tree that consists of Library resources and desktop resources. Understanding the relationship between Library and desktop resources helps you to understand the effects and consequences of propagation.

Portal Asset Instances and Inheritance

The following text describes the relationships between the following instances of portal assets:

- **Primary instance** – Created in Workshop for WebLogic and stored in a `.portal` or `.portlet` file

- **Library instance** – Created or updated in the WebLogic Portal Administration Console, and displayed in the Portal Resources tree under the Library node

- **Desktop instance** – Created or updated in the WebLogic Portal Administration Console, and displayed in the Portal Resources tree under the Portals node

- **Visitor instance** – Created or updated in the Visitor Tools

Creating a New Desktop and Disassembling to the Library

When you create a new desktop using the WebLogic Portal Administration Console, you can use an existing portal template. Using a template means that you will be taking the portal resources for your desktop directly from a `.portal` file that was created in Workshop for WebLogic. (The `.portal` file is also called the primary instance.) When you create a desktop, the portal assets are removed from the `.portal` file, placed in a database, and surfaced in both the Library and desktop trees of the WebLogic Portal Administration Console. Taking the assets from a new desktop instance and placing them in the Library is called disassembling.

At this point, the assets (books, pages, and so on) in the Library (Library instances) are hierarchically related to their corresponding desktop instances. A change to a Library resource, such as a name change, is automatically inherited by the corresponding desktop instance. On the other hand, a change to the desktop instance is not reflected back up the hierarchy.

Note: Changes made to assets are never "reverse inherited" up the hierarchy. A change to a desktop asset is never inherited by its corresponding Library instance. Likewise, a change to a Visitor instance is never inherited by a desktop or Library instance.

New books and pages that you create in a desktop are not disassembled—they are considered to be private to that desktop.

Decoupling of Property Settings

If an administrator or a visitor (using Visitor Tools) changes the Book Properties of a book or the Page Properties of a page in a desktop, those property settings become decoupled from the settings in the parent book or page in the Library. Page properties include layout and theme, while Book Properties include menus and layout. These properties can be modified in the WebLogic Portal Administration Console. When a portal is propagated, any assets that are decoupled in the source application will remain decoupled in the destination.

For more details on the specific data propagated, see "Propagation Reference Table" on page 6-7.

Using Policies

Policies let you control how source and destination inventories are merged into a final inventory file. This section describes policies and presents examples to help you understand how to apply them.

This section includes the following topics:

- Global Policy Examples

- Local Policy Overrides

- Using Local Policies with Desktops

Introduction

Policies let you specify how to handle the following three merge cases:

- **Additions** – If an asset exists in the source inventory, but not in the destination inventory, add it to the destination.

- **Deletions** – If an asset exists in the destination inventory, but not in the source inventory, delete it from the destination.

- **Updates** – If an asset exists in the source inventory and in the destination inventory, update the destination with the artifact in the source inventory.

Through the propagation tools, you can set two kinds of policies:

- **Global** – Global policies apply to all assets in the source inventory. During a merge, each artifact in the source inventory is evaluated with respect to the destination inventory. The global policies dictate the outcome of each evaluation.

- **Local** – Local policies apply to specific desktop and content management assets and override global policies for those assets. You can elect to apply a local policy to any desktop or content management asset in the source inventory. During a merge, the local policy overrides the global policy for that asset.

Global Policy Examples

Figure 6-9 illustrates a simple, but common, example. In this case, the default global policies define how two inventories are merged. *Portlets 2* and *3* on Desktop A are added to the destination's *Desktop A*. *Portlet 5* from *Desktop B* is also added. However, *Portlet 6* on the destination is deleted, because it did not exist in *Desktop B* on source system.

Tip: By default global policies are set to accept adds, deletes, and updates.

Figure 6-9 Accepting Adds and Deletes
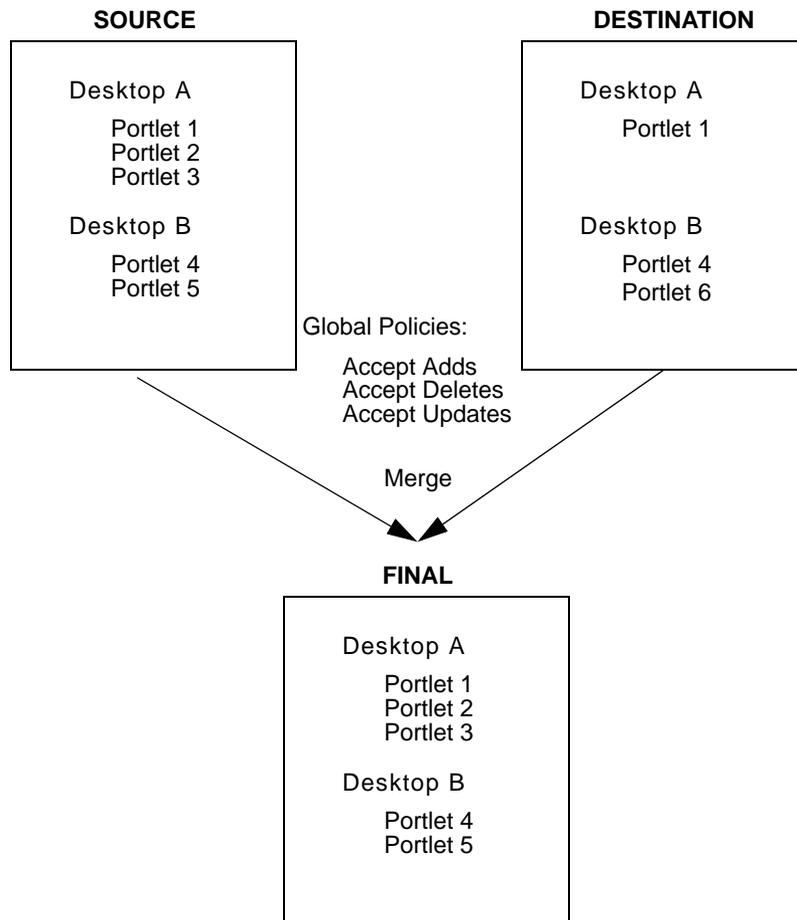
Figure 6-10 shows the same source and destination system; however, in this case the global policy specifies that adds are ignored. In this case, *Portlet 6* is removed from the destination desktop because it did not exist in the source desktop. None of the additional portlets from the source are added, because the global policy specifies that adds are to be ignored.

Figure 6-10 Ignoring Adds and Accepting Deletes
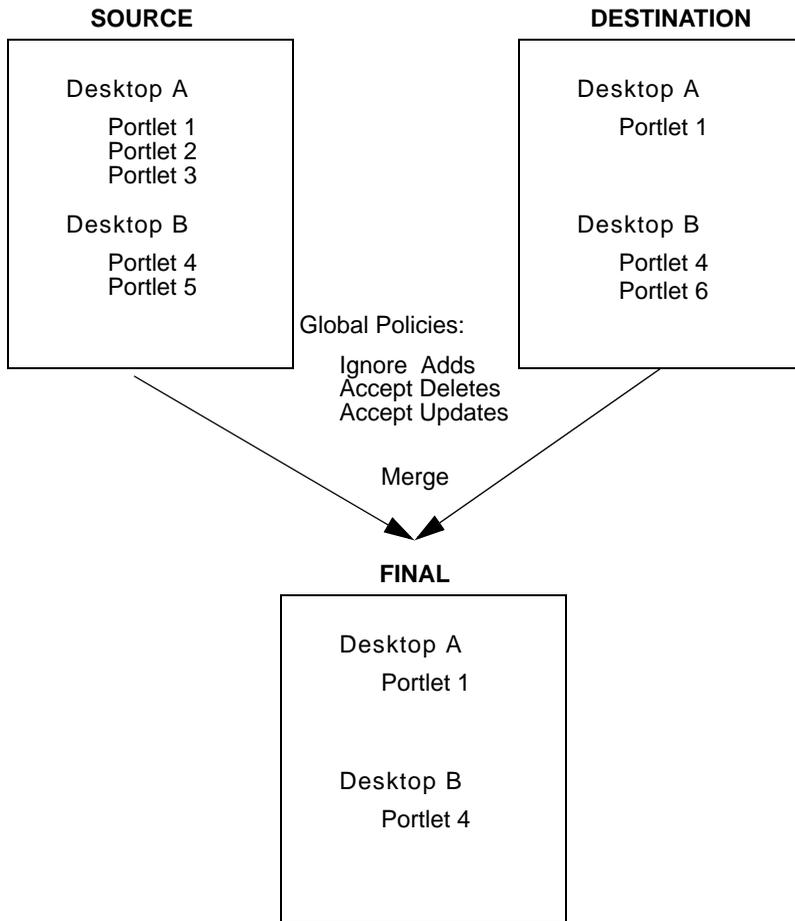
Local Policy Overrides

Figure 6-11 shows how a local policy overrides a global policy. In this example, the global policy is to accept adds; however, a local policy is placed on *Portlet 1* in *Desktop A*. The local policy states that the portlet should be deleted from the destination. In this case, *Portlet 1* on *Desktop A* is not propagated. The local policy overrides the global policy.

Figure 6-11 Local Policy Overrides Global Policy
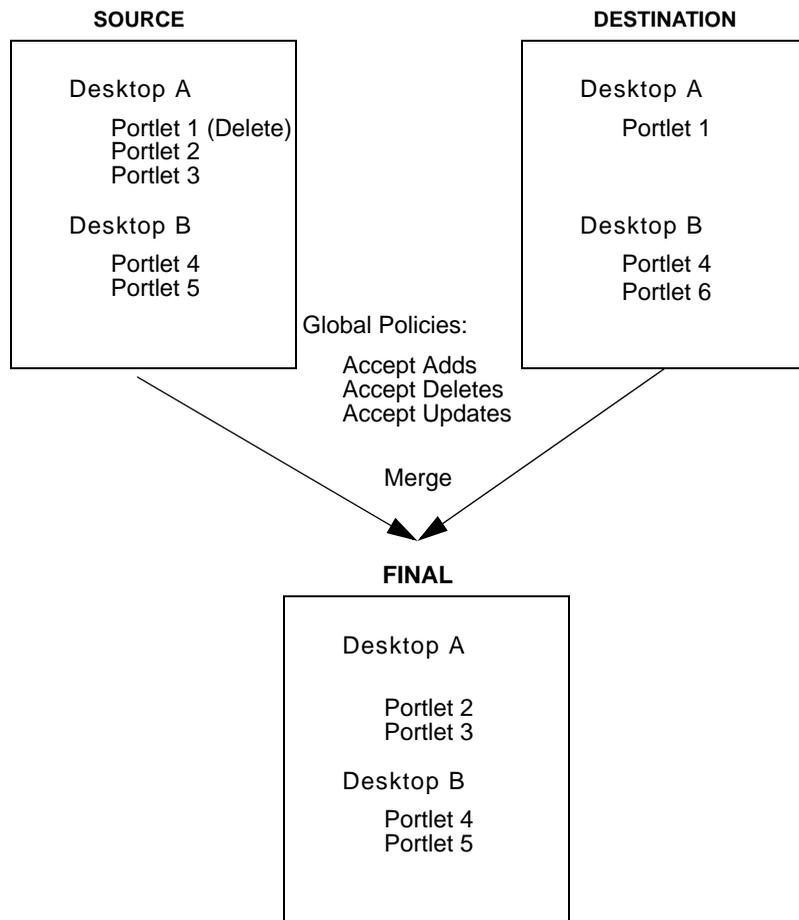
Using Local Policies with Desktops

In WebLogic Portal, multiple desktops can contain instances of the same library object, such as a portlet.

Tip: It is important to understand the difference between a library instance and a desktop instance of an artifact, such as a portlet. For a detailed summary of this difference, see "Scope and Library Inheritance" on page 6-20 and "Export and Import Scope" on page 11-6.

Because global policies can be overridden at the desktop level, it is possible that the policies for instances of the same library object (such as a portlet) might conflict. The propagation software resolves these conflicts as follows:

- **Adds and Deletes** – If after the application of global and local policies, any in-scope desktop requires a specific library object, then the library object will be included in the propagation.

- **Updates** – If after the application of global and local policies any in-scope desktop requires a specific library object, then that library object will be updated during the propagation.

For example, consider this scenario: A user adds *Page 1* to *Book 2* on the source system, but another user adds the same *Page 1* to *Book 3* on the destination. The rules are defined such that additions to the source are added to the destination, and "deletions" on the source are ignored (that is, if an asset exists on the destination and not on the source, it remains on the destination). Because WebLogic Portal requires that a given page cannot exist in more than one book, the propagation tools resolve the conflict by adding *Page 1* to *Book 2*, and removing it in *Book 3* on the destination system.

For another example, consider the following scenario:

- *Desktops A* and *B* both include instances of *Portlet X*.

- The global policy is set to accept adds.

- A local policy to ignore adds is set on *Desktop A*.

- There are no local policies set on *Desktop B*.

In this scenario, illustrated in Figure 6-12, there is a conflict between the local policy for *Desktop A* and the global policy inherited by *Desktop B* with respect to additions. Because *Desktop B* includes *Portlet X*, *Portlet X* is required in the propagation of *Desktop B* (*Desktop B* cannot be

propagated properly without this portlet). Therefore, according to the rules stated previously, the library object for *Portlet X* will be included in the propagation, despite the policy override applied to *Desktop A* indicating that adds should be ignored.

Figure 6-12 Local Policies and Library Instances
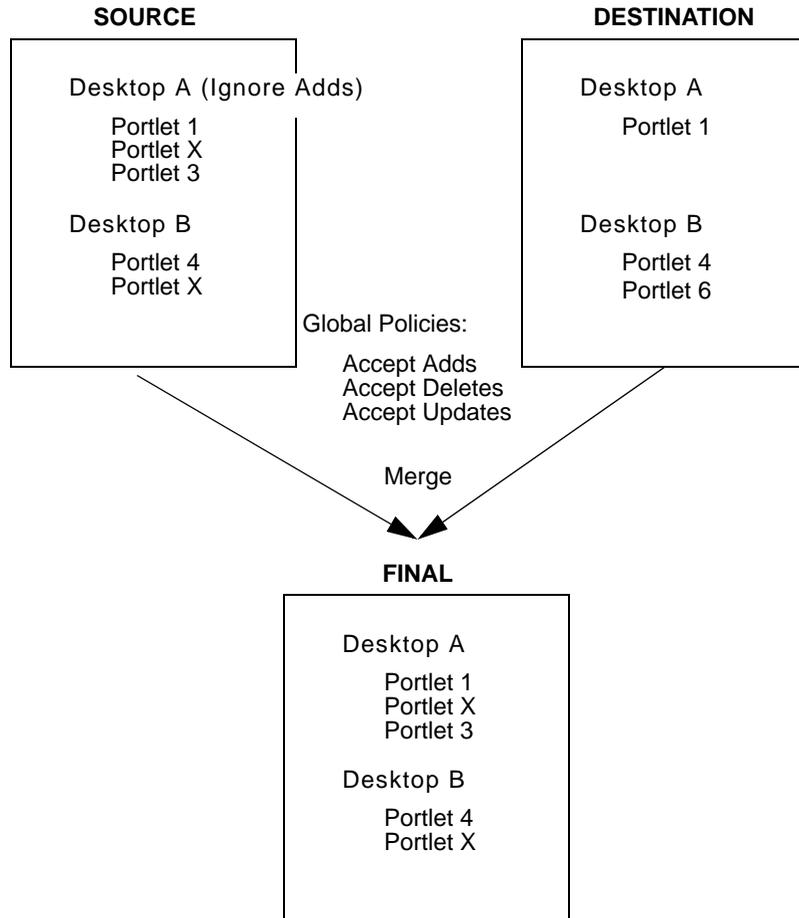
Reporting Changes Based on Policies

When a propagation conflict is resolved by a propagation Ant task, the resolutions are listed in the log file that is created during processing. The Workshop for WebLogic based interface lists any changes that occurred in order to resolve conflicts in the Console view.

For information on log files, see "Reviewing Log Files" on page 6-29.

Previewing Changes and Tuning a Merged Inventory

The basic propagation process starts with a source inventory and a destination inventory. You then merge these two files into a merged inventory file using either Workshop for WebLogic or the propagation Ant tasks.

Before you generate and commit a final inventory file, you have an opportunity to examine the pending changes that will be made to the destination server. You can view these changes in the following ways:

- In a Workshop for WebLogic propagation session, click the **Merged** tab in the editor. This tab displays the merged inventory and highlights assets that represent additions, deletions, and updates. In addition, you can make last-minute changes to the inventory by selecting or deselecting specific nodes in the inventory tree. For more information on "Viewing and Tuning the Merged Inventory" on page 7-14.

- If you are using the Ant tasks, you can examine a change manifest file. This file is an XML file that lists the adds, deletes, and updates that will be applied when the final inventory is generated. The OfflineExtractTask can extract the change manifest file from a merged inventory file. You can edit the manifest file to apply further changes before you produce a final merged inventory file with the OfflineCombineTask. For details on these tasks, see Chapter 9, "Propagation Ant Task Reference."

User Customizations and Propagation

The typical propagation occurs from a staging environment to a production system. In this scenario, users typically do not have access to the staging environment, and no user customization changes that require propagation would exist. For this reason, user customizations are not propagated.

In general, when changes on a staging system are propagated to the production system, the merging of staged changes and user customizations will occur just as it would if the staged changes were made directly on the production system using the Administration Console.

User customizations are preserved on the destination system, but when you propagate to a destination system, those customizations might be removed or modified on the destination server under certain conditions. For instance, an administrator might make changes through the WebLogic Portal Administration Console that potentially conflict with a given user's customizations. For example, an administrator might delete a particular portlet from the portal

that has been added by a user to one of their pages using the Visitor Tools. In this situation, the user's customization is "lost" in the sense that the portlet no longer exists, and as such, will not appear in the user's portal upon the next login.

Reviewing Log Files

Tip: It is a good practice to always review the verbose log after every propagation to ensure that all of your propagation elections were processed and to review the server log to check for propagation errors.

Propagation operations and tasks generate log messages in the following locations:

- The server log.

- The verbose log for the propagation application. For information on enabling, disabling and configuring the verbose log file, see "Configuring the Propagation Servlet" on page 6-38.

- Log files specified in the parameters of individual propagation Ant tasks. See Chapter 9, "Propagation Ant Task Reference" for detailed information on the Ant tasks.

- You can enable verbose logging for the propagation tools in Workshop for WebLogic. For more information, see "Enabling Verbose Logging" on page 7-22.

Rolling Back an Import Process

The propagation tools are not transactional, and therefore do not support roll-back capability. If you must revert to a pre-propagated environment, use the backups that you created before beginning the propagation process. For more information, see "Perform a Data Backup" on page 6-4.

Federated Portal (WSRP) Propagation

This section discusses propagation of federated portals.

Tip: If you are upgrading to WebLogic Portal 10.0, it is recommended that you read the section on upgrading federated portals in the *WebLogic Portal Upgrade Guide* before reading this section. The *Upgrade Guide* explains how to upgrade federated portals from WebLogic Portal 8.1 to 10.0 and from 9.2 to 10.0.

This section includes these topics:

- Introduction

- WSRP Propagation Procedure

- If Only Producer(s) are Upgraded to WebLogic Portal 10.0

- Listing Producer Handles

- Updating Producer Registration Handles

Introduction

WebLogic Portal 10.0 supports features of WSRP 2.0 that permit a more flexible and practical approach to propagation of federated portals. With WebLogic Portal 10.0, the consumer applications in staging and production environments can point to separate producers. The primary advantage of this new capability is that you can create and modify remote (proxy) portlets in a staging environment in isolation from the production environment.

Before version 10.0, if you wanted to propagate WSRP consumer applications, the consumers on the source and destination systems had to point to the same producer. This configuration, which is described in detail in "WSRP Propagation" in the *WebLogic Portal 9.2 Production Operations Guide*, included several limitations. For detailed information on using WSRP with WebLogic Portal, see the *Federated Portals Guide*.

WSRP Propagation Procedure

If both the producer and consumer applications are at WebLogic Portal 10.0 (either new installations or upgrades), then propagation of consumers is fully supported. There is no requirement for the staging and production consumers to point to the same producer. This

recommended configuration is illustrated in Figure 6-13. For detailed information on upgrading federated portals to WebLogic Portal 10.0, see the *WebLogic Portal Upgrade Guide*.

Figure 6-13 Recommended Configuration: Consumers Point to Separate Producers
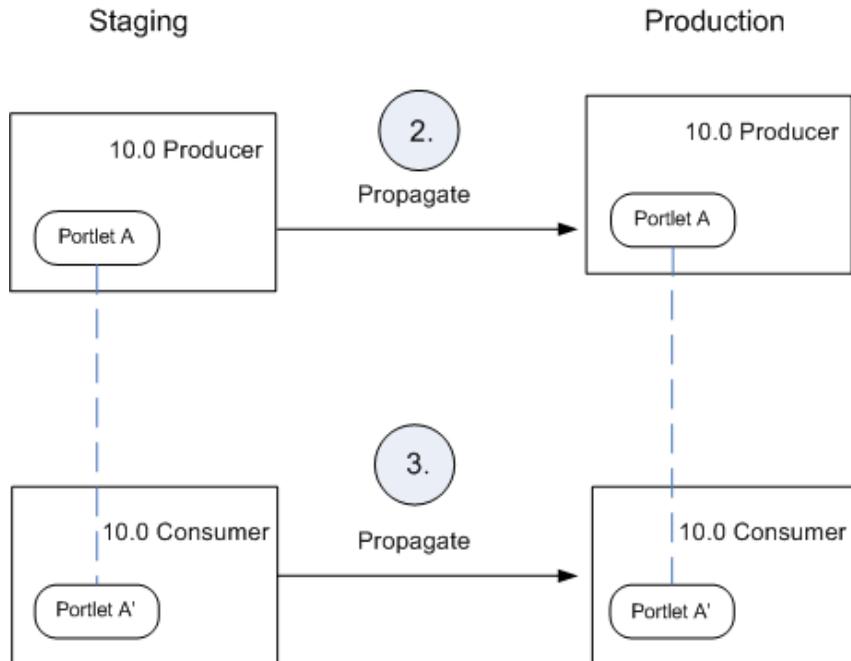
The procedure for propagating a federated portal where both the producer and consumer applications have been upgraded to WebLogic Portal 10.0 is:

Note: The following steps apply whether or not the producer required the consumer to register itself.

1. Before propagating, add the producer to the consumer application on the destination system. For information on adding a producer, refer to the WebLogic Portal Administration Console online help. If the producer requires registration, you must perform the necessary registration steps. *The producer handle you select **must match** the handle that was used to register the producer in the source system.*

   For example, if you added or registered the producer with the consumer in staging with the producer handle `myProducer`, you must also use `myProducer` as the producer handle in the production area.

Note: You only need to perform Step 1 the first time you propagate the portal.

2. Propagate the producer(s). You must propagate to the destination system any new portlets that were deployed on any producers in the staging environment if those portlets are being accessed by consumers.

Note: You also need to perform Step 2 if you made any changes to the portlet definitions on the producers.

3. Propagate the consumer portal application(s) from the source system to the destination system.

When the propagation is completed, the consumer portal in the production environment is functionally equivalent to the consumer portal in the staging environment.

Note: If you do not upgrade all producers and consumers to WebLogic Portal 10.0, then the configuration where consumers point to separate producers (Figure 6-13) is not supported. For detailed information on upgrading federated portals to WebLogic Portal 10.0, see the *WebLogic Portal Upgrade Guide*.

If Only Producer(s) are Upgraded to WebLogic Portal 10.0

The following steps are mandatory if you have upgraded a producer to WebLogic Portal 10.0, but consumers(s) are not upgraded:

1. Obtain the producer registration handles from each consumer application. To do this, run the List Producers JSP utility as described in "Listing Producer Handles" on page 6-33.

2. Update the producer registration handles for each upgraded producer application. To do this, run the Update Registrations JSP utility as described in "Updating Producer Registration Handles" on page 6-34.

3. Propagate the consumer application(s).

If Only Consumer(s) are Upgraded to WebLogic Portal 10.0

If you upgrade only the consumer(s) to 10.0, you are required to use the propagation model described in WSRP Propagation in the *WebLogic Portal 9.2 Production Operations Guide*. In this model, consumer applications in both staging and production environments must point to the same producer.

Listing Producer Handles

Note: You only need to run the utility described in this section on the systems where your consumer applications are deployed.

This section explains how to run the List Producer JSP utility (`listProducers.jsp`) on both the staging and production systems on which your consumer applications are deployed. This utility obtains the registration handles for producers that have been previously added to your consumers.

Note: You must have administration privileges to run the JSPs.

Note: If your consumer application is deployed in a WebLogic Portal 9.2 or 8.1.5 (or later) environment, you must perform steps 1 and 2 below. If your consumer environment was already upgraded to WebLogic Portal 10.0, steps 1 and 2 are not required.

1. This step is only required if your consumer application is deployed in a WebLogic Portal 9.2 or 8.1.5 (or later) environment. Extract the `listProducers.jsp` file from the following J2EE Shared Library. You can use an archive tool, such as WinZip, to extract the `listProducers.jsp` file from the archive:

   `WEBLOGIC_HOME/portal/lib/modules/wlp-propagation-web-lib.war`

2. This step is only required if your consumer application is deployed in a WebLogic Portal 9.2 or 8.1.5 (or later) environment. Copy the `listProducers.jsp` file you extracted into the web applications in which your consumers are deployed. Do this on both the staging and production systems.

3. Run the List Producers JSP utility on the staging or source system. For a 10.0 installation, open a browser and enter the following URL:

   `http://host:port/EarProjectNamePropagation/wsrp/listProducers.jsp`

   Where `EarProjectName` is the name of the enterprise application deployed to the server. For example:

   `http://localhost:7001/myEarPropagation/wsrp/listProducers.jsp`

   For a 9.2 or 8.1.5 (or later) installation, the URL depends on where you placed the JSP. For example:

   `http://host:port/EarProjectNamePropagation/mydir/listProducers.jsp`

4. When prompted, enter the correct user name and password for the server.

5. In the List Producers JSP, select the name of a consumer web application in the source system.

6. Click **Submit Query**. The JSP returns a table that lists the producer handle, WSDL, and registration handle for each producer that was added to the consumer.

7. Print or copy the information in this table. You will need this information to complete the steps in the next section, .

8. Repeat these steps on the production/destination system.

Updating Producer Registration Handles

Note: You only need to run the utility described in this section on the systems where your producer application is deployed.

This section explains how to run the Update Registrations JSP utility (`updateRegistrations.jsp`) on both the staging and production systems. This utility updates the registration handles for each consumer application the currently references a given producer.

Note: If your producer application is deployed in a WebLogic Portal 9.2 or 8.1.5 (or later) environment, you must perform steps 1 and 2 below. If your producer environment was already upgraded to WebLogic Portal 10.0, steps 1 and 2 are not required.

1. This step is only required if your producer application is deployed in a WebLogic 9.2 or 8.1.5 (or later) environment. Extract the `updateRegistrations.jsp` file from the following J2EE Shared Library. You can use an archive tool, such as WinZip, to extract the `updateRegistrations.jsp` file from the archive:

   ```
   WEBLOGIC_HOME/portal/lib/modules/wlp-propagation-web-lib.war
   ```

2. This step is only required if your producer application is deployed in a WebLogic 9.2 or 8.1.5 (or later) environment. Copy the `updateRegistrations.jsp` file you extracted into the web application in which your producer is deployed. Do this on both the staging and production systems.

3. On the destination system, run the Update Registrations JSP. To do this, To do this, open a browser and enter the following URL:

   ```
   http://host:port/EarProjectNamePropagation/wsrp/updateRegistrations.jsp
   ```

   Where `EarProjectName` is the name of the enterprise application deployed to the server. For example:

   ```
   http://localhost:7001/myEarPropagation/wsrp/updateRegistrations.jsp
   ```

4. When prompted, enter the correct user name and password for the server.

5. In the JSP, enter the source and corresponding destination registration handles that you obtained from the consumer applications by running the List Producers utility described previously in "Listing Producer Handles" on page 6-33.

6. Click **Submit**.

7. Repeat this procedure for each consumer involved in the propagation that is registered with the producer.

Note: The Update Registrations JSP copies registrations so that both consumers have access to the remote portlet.

Increasing the Default Upload File Size

The propagation management servlet has a configuration setting to help mitigate "denial of service" attacks. The servlet is configured with a maximum size allowed for uploaded files (files uploaded over HTTP). By default, this is set to 1 MB. If any given file inside the inventory ZIP file is larger than this value, it will be rejected. This section explains how to either work around this limitation or change the propagation servlet's configuration to allow larger files.

- Copying the Inventory to the Server

- Modifying a Deployment Plan

- Modifying the web.xml File

Copying the Inventory to the Server

The simplest workaround to this file size limit is to physically copy your file, through FTP or another means, from the source to the destination server. After you copy the file to the destination, you can use the OnlineUploadTask's `readFromServerFileSystem` attribute to perform the upload. For information on this task, see "OnlineUploadTask" on page 9-15.

Modifying a Deployment Plan

You can override the 1MB default file upload limit using a deployment plan. A deployment plan is an optional XML file that configures an application for deployment to WebLogic Server. A deployment plan works by setting property values that would normally be defined in the WebLogic Server deployment descriptors, or by overriding context parameter values already defined in a WebLogic Server deployment descriptor.

> **Tip:** The creation and use of deployment plans is thoroughly discussed in the WebLogic Server documentation. For more information, see "Configuring Applications for Production Deployment" and "Updating Applications in a Production Environment."

To modify the file size limit using a deployment plan, follow these steps:

1. Create a deployment plan file. There are several ways to create a deployment plan, as discussed in the WebLogic Server documentation. Listing 6-1 shows a sample deployment plan configured to modify the propagation web application.

   > **Tip:** The location of the deployment plan must be specified in your domain's `config.xml` file. A sample stanza is shown in Listing 6-2. Refer to the WebLogic Server documentation for more information.

2. Add `<variable-definition>` and `<variable-assignment>` elements to change the context deployment descriptor parameter `maximum_inventoryfile_upload_size`. These elements are highlighted in bold type in Listing 6-1. In the example, the upload file size limit is changed to 3 MB. Note that the overridden descriptor is associated with the application's `web.xml` file.

Listing 6-1 Sample Deployment Plan

```
<deployment-plan xmlns="http://www.bea.com/ns/weblogic/90"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <application-name>drtApp</application-name>
  <variable-definition>
    <variable>
      <name>CHANGEUPLOAD</name>
      <value>3000000</value>
    </variable>
  </variable-definition>

  <module-override>
    <module-name>propagation.war</module-name>
    <module-type>war</module-type>
    <module-descriptor external="false">
      <root-element>weblogic-web-app</root-element>
      <uri>WEB-INF/weblogic.xml</uri>
    </module-descriptor>
    <module-descriptor external="false">
      <root-element>web-app</root-element>
```

```
     <uri>WEB-INF/web.xml</uri>
     <variable-assignment>
        <name>CHANGEUPLOAD</name>
      <xpath>/web-app/context-param/
      [param-name="maximum_inventoryfile_upload_size"]/param-value</xpath>
     </variable-assignment>

   </module-descriptor>
  </module-override>
</deployment-plan>
```

Listing 6-2   Deployment Plan Location in Sample config.xml

```
<app-deployment>
    <name>drtApp</name>
    <target>portalServer</target>
    <module-type>ear</module-type>
    <source-path>/myProject/myApplication</source-path>
            <deployment-order>101</deployment-order>
            <plan-dir>/myProject/applications/plan</plan-dir>
            <plan-path>plan.xml</plan-path>
    <security-dd-model>Advanced</security-dd-model>
  </app-deployment>
```

3. Redeploy the application, as described in the WebLogic Server document, "Updating Applications in a Production Environment."

Modifying the web.xml File

Note: This method is not recommended.

You can configure the file upload size directly in the propagation application's web.xml file. To modify this file, do the following:

1. Locate the propagation application's WAR file. This file is located in:

WEBLOGIC_HOME/portal/lib/modules/wlp-propagation-web-lib.war

2. Unpack the WAR file and open the web.xml file for editing.

3. Add the stanza to the web.xml file shown in Listing 6-3. In this example, the default file size is changed to 3 MB.

Listing 6-3   Changing the Default Upload Size

```
<context-param>
    <description>Maximum upload size (in bytes), if not specified
        defaults to 1 MB.
    </description>
    <param-name>maximum_inventoryfile_upload_size</param-name>
    <param-value>3000000</param-value>
</context-param>
```

4. Repackage the WAR file.

5. Redeploy the application.

Configuring the Propagation Servlet

A number of configuration parameters that affect the propagation servlet are specified in the propagation web application's `web.xml` file. The preferred method for modifying these parameters is by using a deployment plan.

A deployment plan is an optional XML file that configures an application for deployment to WebLogic Server. A deployment plan works by setting property values that would normally be defined in the WebLogic Server deployment descriptors, or by overriding context parameter values already defined in a WebLogic Server deployment descriptor.

Tip: The creation and use of deployment plans is thoroughly discussed in the WebLogic Server documentation. For more information, see "Configuring Applications for Production Deployment" and "Updating Applications in a Production Environment."

Tip: See also the section, "Increasing the Default Upload File Size" on page 6-35, for information on configuring the propagation servlet using a deployment plan.

The following sections show context parameters for the propagation application's `web.xml` file. To override these parameters, use a deployment plan. The parameters are used to set the following configurations:

● Inventory Export Directory

- Description Text
- Verbose Logging
- Verbose Log File Location

Inventory Export Directory

The example stanza shown in Listing 6-4 is used to specify a temporary directory in which to place the exported inventory ZIP file.

Listing 6-4 Inventory Export Directory

```
context-param>
    <description>Base folder path for runtime data, such as
        inventory exports.
    </description>
    <param-name>inventoryWorkingFolder</param-name>
    <param-value>D:\dev\src\wlp\propagation\test\inventories</param-value>
</context-param>
```

Description Text

The example stanza shown in Listing 6-5 is used to specify a short description to be placed in the `export.properties` file. This file contains summary information about the inventory; including who exported it, when it was exported, how many nodes are in the export, and other information.

Listing 6-5 Description Text

```
<context-param>
    <description>The name of the operating environment.</description>
    <param-name>environment_name</param-name>
    <param-value>Staging</param-value>
  </context-param>
```

Verbose Logging

The example stanza shown in Listing 6-6 is used to enable or disable verbose logging.

Listing 6-6   Verbose Logging

```
<context-param>
    <description>Enable verbose logging for the servlet.</description>
    <param-name>enable_verbose_logging</param-name>
    <param-value>true</param-value>
  </context-param>
```

Verbose Log File Location

The example stanza shown in Listing 6-5 is used to specify the location of the verbose log file. By default, this file is stored in a temporary directory in the domain.

Listing 6-7   Verbose Log File Location

```
<context-param>
    <description>Specify the folder to put verbose logs.</description>
    <param-name>verbose_log_folder</param-name>
    <param-value>D:/dev/src/wlp/propagation/test/inventories</param-value>
  </context-param>
```

Using Workshop for WebLogic Propagation Tools

This chapter explains how to use Workshop for WebLogic propagation tools to propagate a WebLogic Portal. Workshop for WebLogic provides propagation tools that guide you through the process of downloading and uploading portal inventories, merging portal inventories, setting scopes and policies, and committing a final inventory. If you would like to explore a more programmatic, automated approach to propagation, see Chapter 8, "Using the Propagation Ant Tasks."

Note: WebLogic Portal inventories saved with WebLogic Portal 8.1 or 9.2 cannot be used with WebLogic Portal 10.0 propagation tools.

Tip: Before reading this chapter, we recommend that you review Chapter 5, "Developing a Propagation Strategy" and Chapter 6, "Propagation Topics."

This chapter includes these topics:

- Overview
- Overview of the Propagation Perspective
- Downloading an Inventory File
- Creating a Propagation Project
- Viewing and Tuning the Merged Inventory
- Creating a Final Merged Inventory File

- Uploading the Final Inventory to the Server

- Enabling Verbose Logging

Overview

Workshop for WebLogic provides a set of tools that let you perform a complete portal propagation. The tools provided in the Workshop for WebLogic Propagation let you:

- Download a portal inventory from a source or destination system.

- Merge source and destination inventories.

- Set scopes and policies on merged inventories.

- Upload a portal inventory to a destination system.

- Commit a final inventory on the destination system.

Workshop for WebLogic lets you merge inventory files, graphically depicts this merged inventory, and highlights the artifacts that have been added, removed, or updated during the merge. You then have a chance to tune the merged inventory and produce a final inventory file that you can upload to a destination server.

Tip: All of the propagation features that are available in Workshop for WebLogic are also available through the propagation Ant tasks. The Ant tasks offer a programmatic approach to propagation and offer additional features, such as the ability to place the server in maintenance mode, that are not offered in Workshop for WebLogic. See Chapter 8, "Using the Propagation Ant Tasks" for more information.

Propagating a portal using Workshop for WebLogic involves these major steps. Each step is explained in this chapter.

1. Use the import feature to download portal inventory files from the source and destination systems.

2. Use the Propagation Session wizard to create a propagation project, import source and destination inventory files, and create a merged inventory file.

3. Use the Propagation Perspective to view and tune the merged inventory file. This perspective provides a graphical view of your merged inventory, highlighting additions, deletions, and updates, and allowing you to make changes before generating the final merged inventory.

4. Create a final merged inventory file.

5. Upload and commit the final merged inventory file to the destination server.

Overview of the Propagation Perspective

Workshop for WebLogic provides a Propagation perspective which provides views for viewing and editing inventory files. This section introduces the Propagation perspective.

Figure 7-1 shows an example Propagation Perspective. The numbered areas are described briefly below the figure.

Figure 7-1 Example Propagation Perspective
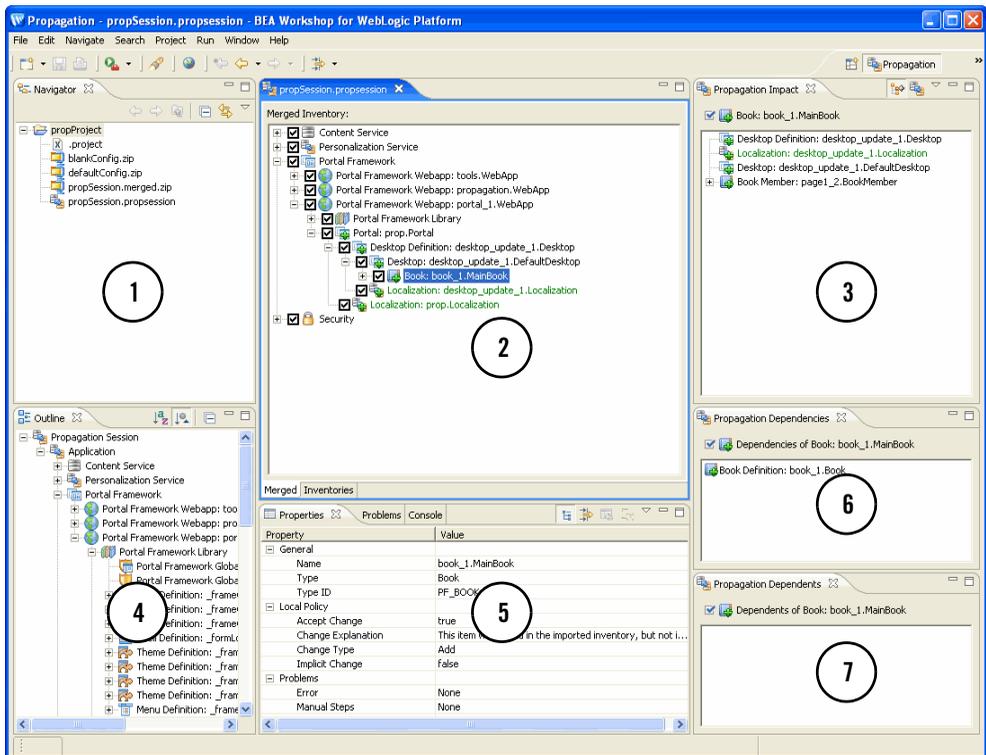
1. Navigator – Shows the contents of the propagation session, including source and destination files, merged inventory files, and properties files.

2. Merged Inventory/Inventories – This view lets you switch between a hierarchical representation of the merged inventory file or the source and destination inventories. To switch between these two views, click the **Merged** or the **Inventories** tab below the view.

3. Propagation Impact – This view shows you the inventory nodes that are affected when you modify a merged inventory file. For instance, if you deselect a node in the merged inventory, to remove it from the propagation scope, the Propagation Impact view highlights nodes that are dependent on the deleted node.

4. Outline – This view lets you view and navigate the portal inventory.

5. Properties – The Properties view displays information about the selected inventory node. This view also lists the policy that applies to the node and any manual changes that will be required if the policy is applied.

6. Propagation Dependencies – This view shows you assets upon which the selected asset depend. Removing any of the dependent assets will make the selected asset invalid.

7. Propagation Dependents – This view shows you the assets that depend on the selected asset. If you remove the selected asset, these dependent assets will become invalid.

Downloading an Inventory File

This section explains how to download a portal inventory file from a server. To propagate a portal, you need to download the source and the destination inventories, combine them, and then upload and commit the final combined inventory.

Note: This operation extracts the portal inventory and attempts to write it to a ZIP file. If the ZIP file created exceeds 4 GB, this operation fails and a message is written to the server log and the verbose log. If this occurs, try scoping your inventory to limit the size of the resulting archive file. See "Understanding Scope" on page 6-11 for more information.

Tip: You can also use the OnlineDownloadTask Ant task to download an inventory file. See Chapter 9, "Propagation Ant Task Reference" for more information.

1. Select **File > Import**.

2. In the Import – Select dialog, select **Other > Propagation Inventory from Server**, as shown in Figure 7-2, and click **Next**.

Figure 7-2 Import – Select Dialog
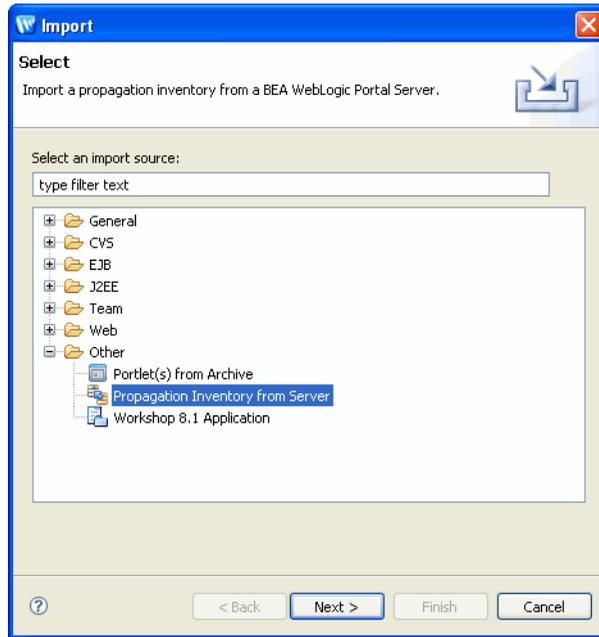
3. In the Import Inventory from Server dialog, complete the Server URL, Username, and Password fields. The server URL is the URL of the propagation servlet that is deployed on the target server. This URL is formed as follows:

```
http://servername:portnumber/earProjectNamePropagation/inventorymanagement
```

Where `earProjectName` is the name of the EAR project that contains the portal application that you are propagating. For example: `myEARProjectPropagation`. For example:

```
http://localhost:7001/myEARProjectPropagation/inventorymanagement
```

> **Tip:** The propagation servlet is deployed, by default, in all WebLogic Portal EAR projects. This servlet enables communication with remote propagation clients, such as the Workshop for WebLogic propagation tools and Propagation Ant Tasks. The servlet allows remote clients to perform online operations, such as downloading, uploading, and committing WebLogic Portal inventories.

A sample Import Inventory from Server dialog is shown in Figure 7-3.

Figure 7-3 Import From Server Parameters
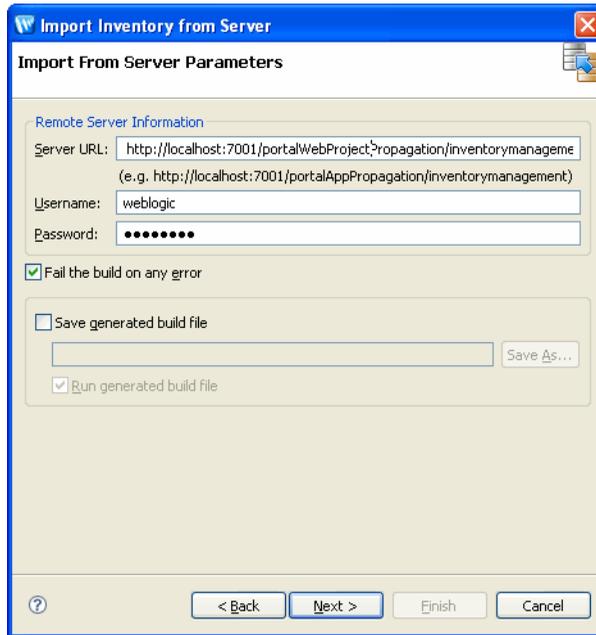
4. Click **Next**. The Choose Inventory File dialog appears.

5. In the Choose Inventory File dialog, select the folder in which to place the inventory file, and enter a name for the file, as shown in Figure 7-4.

Figure 7-4 Choose Inventory File Dialog
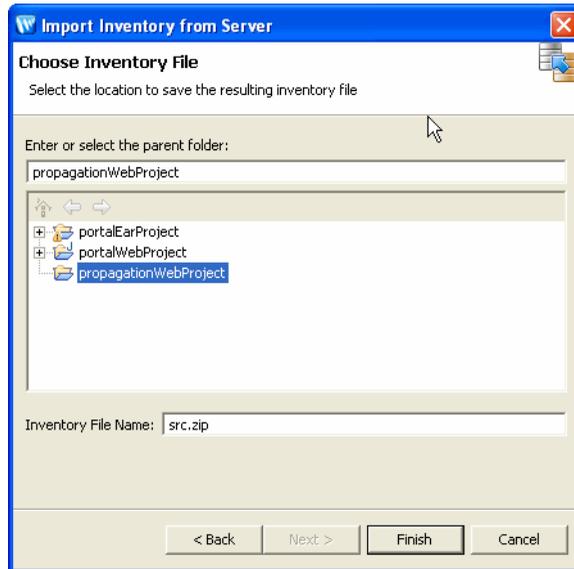
Creating a Propagation Project

This section explains how to use the Propagation Session wizard to set up a propagation project in Workshop for WebLogic. This section discusses the following tasks:

- Create a Simple Project

- Begin a Propagation Session

- Import the Inventory Files

- Create a Merged Inventory File

Create a Simple Project

A propagation project must reside in a project folder. If you do not currently have a project folder in which to put your propagation project, you need to create one first. In this initial task, you create a simple project for this purpose.

To create a simple project, do the following:

1. Start Workshop for WebLogic.

2. Select **File > New > Project**.

3. In the New Project – Select a Wizard dialog, open the **General** folder, select **Project**, and click **Next**.

4. In the New Project dialog, enter a name for the project and click **Finish**. The project appears in the Package Explorer, as shown in Figure 7-5.

Figure 7-5 New Simple Project Folder
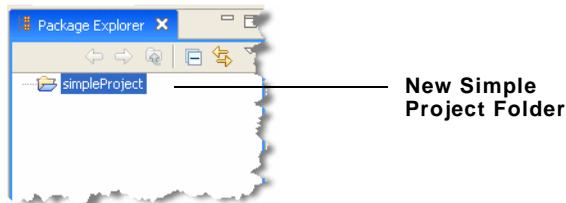
Begin a Propagation Session

A propagation session provides a wizard that guides you through the process of importing inventory files, viewing them, and merging them.

To begin a propagation session, do the following:

1. Select **File > New > Other**.

2. In the New – Select a Wizard dialog, open the **WebLogic Portal** folder, select **Propagation Session**, and click **Next**.

3. In the New Propagation Session dialog, select a parent folder, and enter a name for the session. The parent folder can be any project folder, such as the Simple Project folder you created previously.

4. Click **Next**. The New Propagation Session – Choose source inventory files dialog appears.

Checkpoint: In the next task, you will import a source and a destination inventory file. Workshop for WebLogic then lets you view these files, adjust the scope and policies, and merge the files into a final merged inventory file that you can upload and commit to the destination server.

Import the Inventory Files

Note: WebLogic Portal inventories saved with WebLogic Portal 8.1 or 9.2 cannot be used with WebLogic Portal 10.0 propagation tools.

The propagation tools operate on a pair of WebLogic Portal inventory files: a source and a destination file. In this task, you import a previously downloaded source inventory file into the project.

Tip: You can download a portal inventory using Workshop for WebLogic or the OnlineDownloadTask Ant task. The Workshop for WebLogic method is described in "Downloading an Inventory File" on page 7-4. The Ant task is described in Chapter 8, "Using the Propagation Ant Tasks."

The imported inventory file describes the entire Enterprise application environment, as contained in the database, for that system. The source inventory file is stored in the form of XML files, which are grouped into a single ZIP file.

WARNING: Never edit the individual XML files in the exported inventory ZIP file.

1. In the The New Propagation Session – Choose source inventory files dialog, select the link **Click to import files**, as shown in Figure 7-6. The Import – Select dialog appears.

Figure 7-6 Choose Source Inventory File Dialog
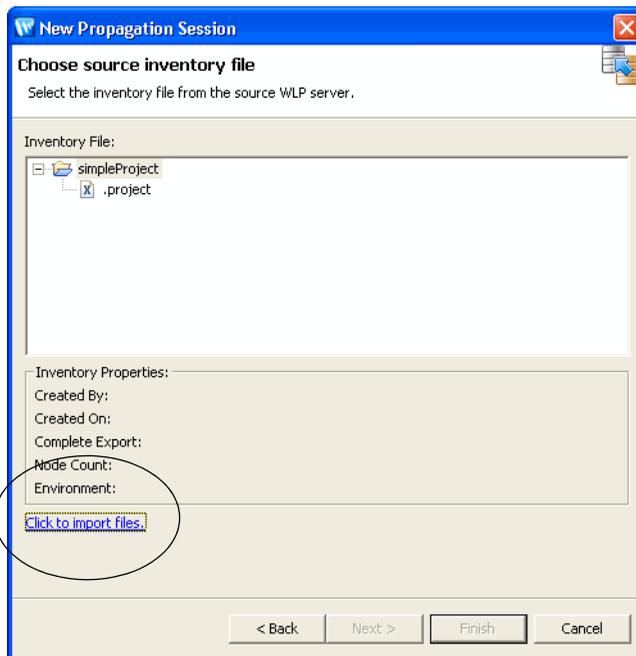
2. In the Import – Select dialog, select **General > File system**, as shown in Figure 7-7, and click **Next**.

> **Tip:** If you select the Import Inventory from Server option, the wizard lets you import the inventory directly from the server. If you choose this path, the basic steps are identical to the steps outlined in "Downloading an Inventory File" on page 7-4.

Figure 7-7 Import – Select Dialog
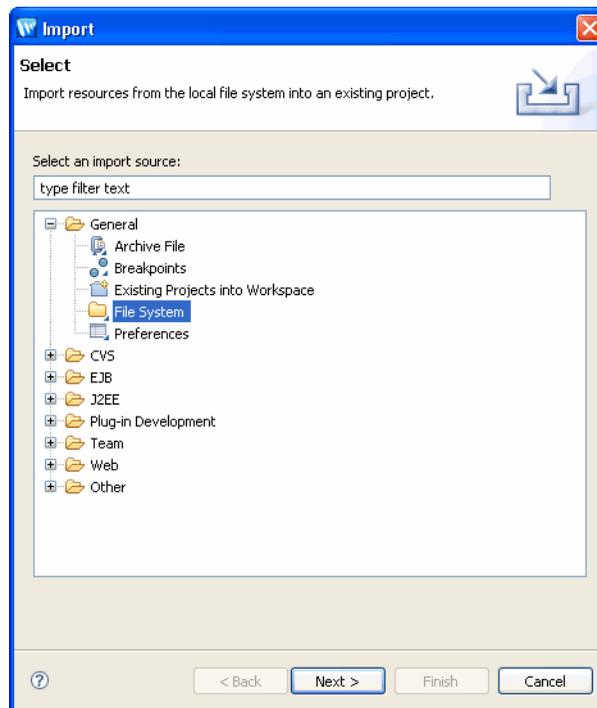
3. In the Import – File System dialog, click **Browse** to locate the folder containing the inventory or inventories you want to import, and select the inventory files from the list box, as shown in Figure 7-8.

Figure 7-8 Import – File System Dialog
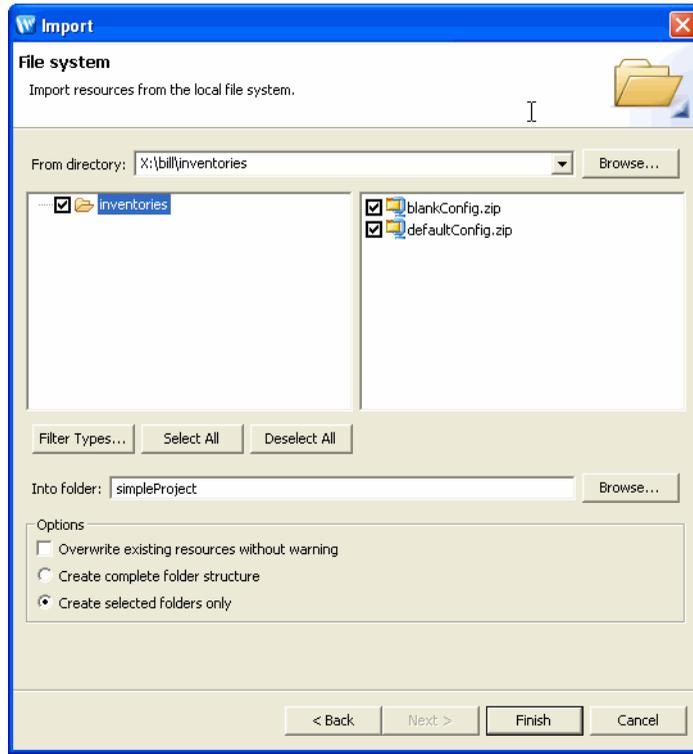
4. In the Into folder field, enter (or select using the **Browse** button) the Propagation Session folder you want to import the inventories into.

5. Click **Finish**. This returns you to the New Propagation Session – Choose Source Inventory File dialog. Do not close the dialog; you will continue using it in the next section.

6. If necessary, repeat this procedure to download the destination inventory file.

Tip: BEA recommends that you store all propagation session files within a source control system.

Checkpoint: You now have imported the source and destination inventory files into the propagation project. Now you will create a merged inventory file from the source and destination files.

Create a Merged Inventory File

In this task, you select the source and destination inventory files and generate a merged inventory file. A merged inventory file contains all of the artifacts resulting in the union of the source and destination files. Later, you can view and refine the merged inventory file before creating a final inventory file.

The files are merged using a set of rules, which include:

- Global policies
- Global scope
- Local policy overrides

For detailed information on these rules, see Chapter 6, "Propagation Topics." For the example in this chapter, default rules are used.

Select the Source Inventory File

1. In the Choose Source Inventory File dialog, select the source inventory file, as shown in Figure 7-9, and click **Next**.

Figure 7-9 Choose Source Inventory Dialog
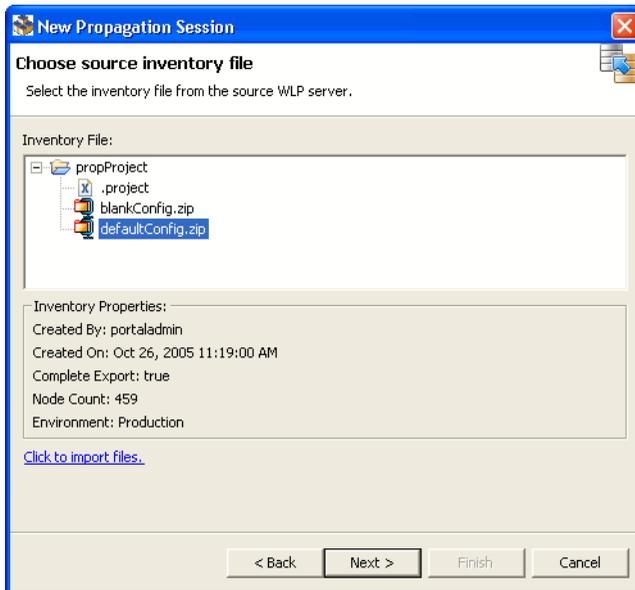
Select the Destination Inventory File

1. In the Choose Source Inventory File dialog, select the destination inventory file, as shown in Figure 7-10.

Figure 7-10 Destination Inventory File Selected
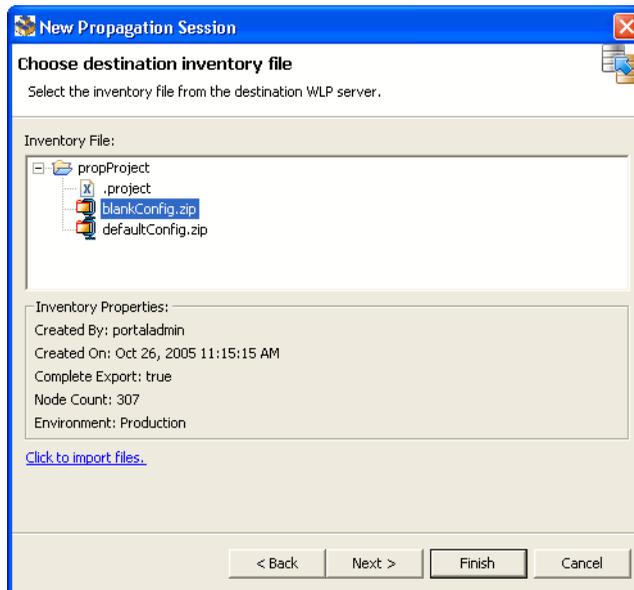
2. Click **Finish**. The Open Associated Perspective dialog appears.

> **Tip:** By clicking **Finish**, you bypass the remaining dialogs of the Propagation Session wizard, accepting all of the default values. If you want to learn more about these options, see Chapter 6, "Propagation Topics."

3. In the Open Associated Perspective dialog, click **Yes**. This action causes the Propagation Perspective to open. This perspective lets you view the merged inventory file and make changes to it if you want.

> **Tip:** You can always open the Propagation Perspective manually by selecting **Window > Open Perspective > Other**, and selecting **Propagation** from the Select Perspective dialog.

The interim merged inventory file is created. By default, this file is named:

```
propSessionName.merged.zip
```

where *propSessionName* is the name of the propagation session file you created previously.

Checkpoint: At this point, you have imported a source and a destination inventory file into your propagation session and generated a merged inventory file, which is the union of the contents of the source and destination files. Next, you will view and tune the merged inventory file.

Viewing and Tuning the Merged Inventory

After you generate the merged inventory file, it appears in the Merged Inventory Tree View, as shown in Figure 7-11.

Note: You must be in the Propagation Perspective to see the merged inventory. You can open this perspective by selecting **Window > Open Perspective > Other**, and selecting **Propagation** from the Select Perspective dialog.

The tree view is a hierarchical description of the content of the merged inventory files.

Figure 7-11 Viewing the Merged Inventory
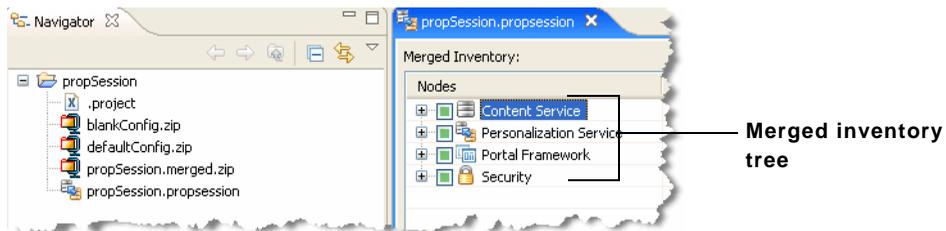
By expanding the merged inventory tree, you can see the detailed contents of the inventory, including the artifacts that have been added, deleted, or updated. Figure 7-12 shows that as you drill down into an inventory view, added artifacts are indicated with a + (Plus) icon and by a notation in the Change Type column.

Tip: You can turn the gridlines on or off in the Merged Inventory view by selecting **Window > Preferences**. In the Preferences dialog, select **WebLogic Portal**, and then select **Propagation Tool**.

Figure 7-12 Expanded Inventory
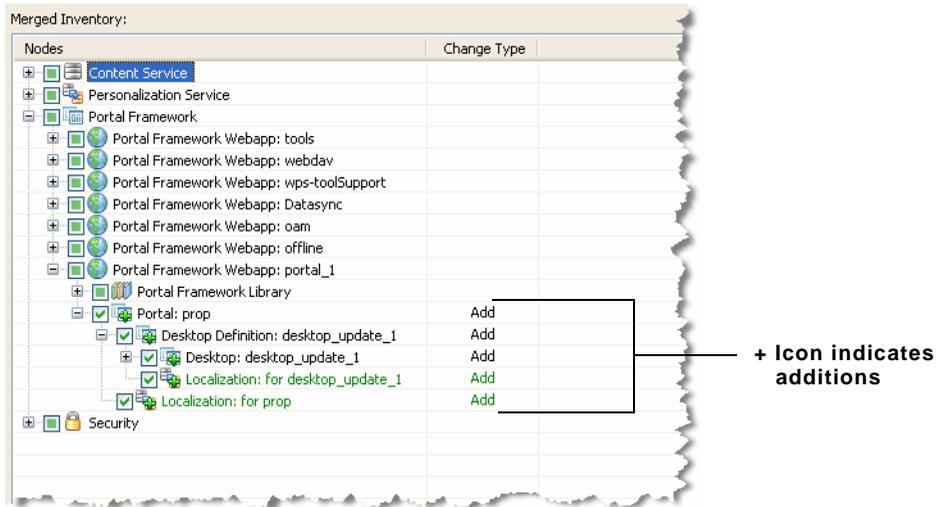
The merged inventory file graphically shows the state of each node with a combination of color and special icons badges. These visual cues include:

Table 7-1 Default Colors and Icon Badges in Merged Inventories

Icon Badge	Default Color	Meaning
+ (Plus sign)	Green	The artifact exists in the source, but not the destination. It will be added to the destination.
– (Minus sign)	Red	The artifact exists in the destination, but not the source. It will be deleted from the destination.
[ ] (Square)	Blue	The artifact exists in the source and destination. It has changed in the source and it will be updated in the destination.
Standard Eclipse error badge	Red	Error. Indicates that the change is impossible to make, possibly because of a conflicting dependency. In addition, a message is displayed in the Problems view.
Standard Eclipse warning badge	Yellow	Warning. Indicates a manual change is required. In addition, a message is displayed in the Problems view.
None	Green	Implicit change.

Tuning the inventory refers to manually selecting or deselecting nodes in the inventory tree. If you deselect a node, the node and any dependent nodes will not be included in the final merged inventory file. Because many artifacts are dependent on other artifacts, deselecting one artifact often results in other artifacts automatically becoming deselected. Because the dependencies between portal artifacts are often complex, any discussion of modifying the merged inventory is beyond the scope of this chapter. For more information on editing the inventory, see Chapter 6, "Propagation Topics."

If a particular node cannot be modified, the checkbox is greyed out (you can't change it). If the unmodifiable node has children, the node's label is shown in grey text. If the unmodifiable node does have children, the text is shown in the normal color.

The default colors listed in Table 7-1 can be changed in the Eclipse IDE. To change the color assigned to the name of an unmodifiable node or any of the colors listed in Table 7-1, do the following:

1. Select **Window > Preferences**.

2. In the Preferences dialog, select **General > Appearance > Colors and Fonts**.

3. In the Colors and Fonts part of the dialog, select **WebLogic Portal > Propagation Tool**.

4. Select and modify the color you which to change. (Note that Inactive Change refers to the color assigned to the text of an unmodifiable node).

Tip: You can right-click a node to access additional functions, such as view filters. For example, you can filter the view to just show additions. Refer to online help in Workshop for WebLogic for more information.

Tip: The Propagation Dependencies and Propagation Dependents views help you asses the impact of adding or deleting portal assets from the merged view. For more information on these views, see "Overview of the Propagation Perspective" on page 7-3.

Checkpoint: After viewing and modifying the merged inventory, you are ready to create a final merged inventory file.

Creating a Final Merged Inventory File

After you are satisfied with the state of the interim merged inventory, you need to generate a final version of the merged file. At this time, any changes you made to the interim file are executed

and a final, merged inventory file is created. The final inventory file is always smaller than the interim file. This is because the interim file has to maintain all of the artifacts resulting in the union of the source and destination files, while the final file contains only the merged contents of the two files.

1. Select **File > Export**. The Export – Select dialog appears.

2. In the Export – Select dialog, select **Other > Propagation Session to Inventory**, as shown in Figure 7-13, and click **Next**.

Figure 7-13 Export – Select Dialog
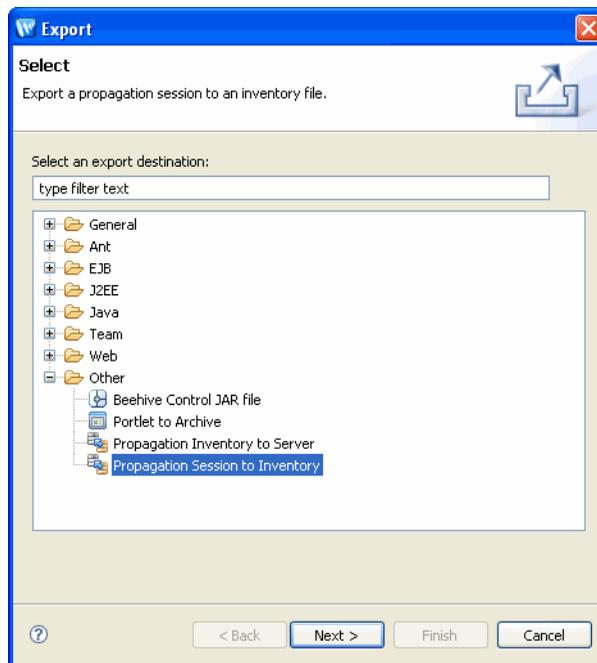
3. In the Choose Propagation Session dialog, select the propagation session file for the propagation session you are currently working on. To do this, you must open the project folder and select the propagation session file, as shown in Figure 7-14. After selecting the propagation file, click **Next**.

Figure 7-14 Choose Propagation Session Dialog
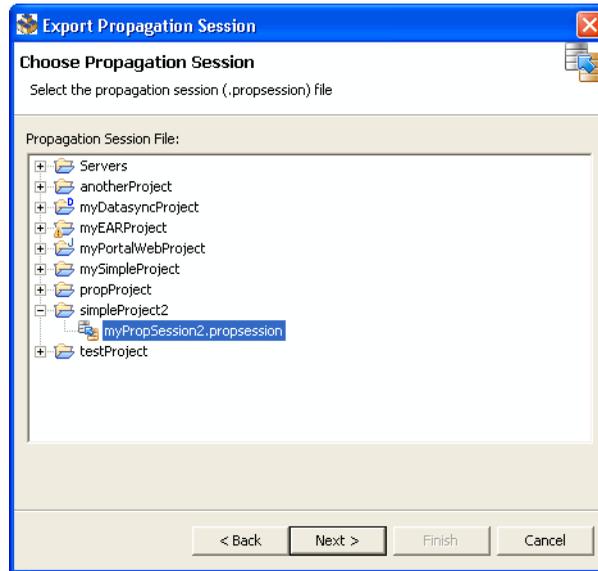
4. In the Choose Inventory File dialog, select the folder in which you want to put the final inventory file, enter a name for the file, and click **Finish**.

    The final inventory file appears in the Package Explorer in the folder you designated, as shown in Figure 7-15.

Figure 7-15 Final Inventory File
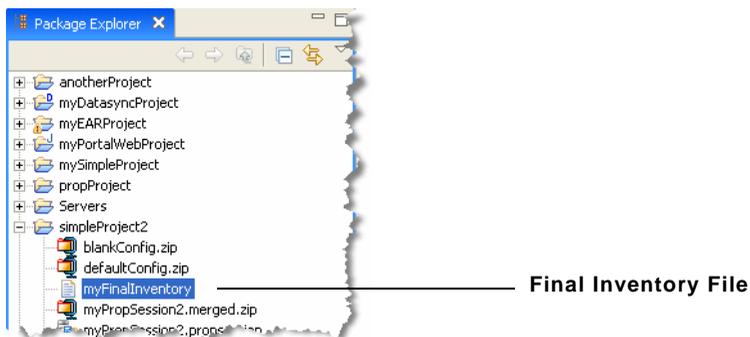
Uploading the Final Inventory to the Server

This section explains how to upload the final inventory to a destination server.

Note: Workshop for WebLogic does not automatically place the server in maintenance mode to prevent users from changing portal data using the WebLogic Portal Administration Console or Visitor Tools on the production system. If a user makes changes on the production system after the inventory listing has been imported and validated, propagation results might be inaccurate and changes could be unsuccessful. If you want to place the destination system in maintenance mode, you can use the OnlineMaintenanceModeTask Ant task, which is described in Chapter 9, "Propagation Ant Task Reference."

WARNING: **It is very important that you make no changes to the production system during the final upload.**

Deploy the EAR File

Before you upload the final inventory, the source application EAR file must be deployed on the destination server. In other words, if you are propagating an application from the staging environment to the production environment, you must deploy the EAR from staging to production before you upload the inventory.

For more information, see "Deploy the J2EE Application (EAR)" on page 6-5.

Propagate the Final Inventory to the Destination Server

When you propagate the inventory to the server, Workshop for WebLogic propagates the database assets in the final inventory file to the production server, according to the scope you assigned and the policies you selected previously.

Tip: In this step, Workshop for WebLogic uploads the file to the server and commits it. The propagation Ant tasks OnlineUploadTask and OnlineCommitTask can be used for the same purpose.

---

1. Select **File > Export**. In the Export – Select dialog, select **Other > Propagation Inventory to Server**, as shown in Figure 7-16, and click **Next**.

Figure 7-16 Select Export Propagation Inventory to Server
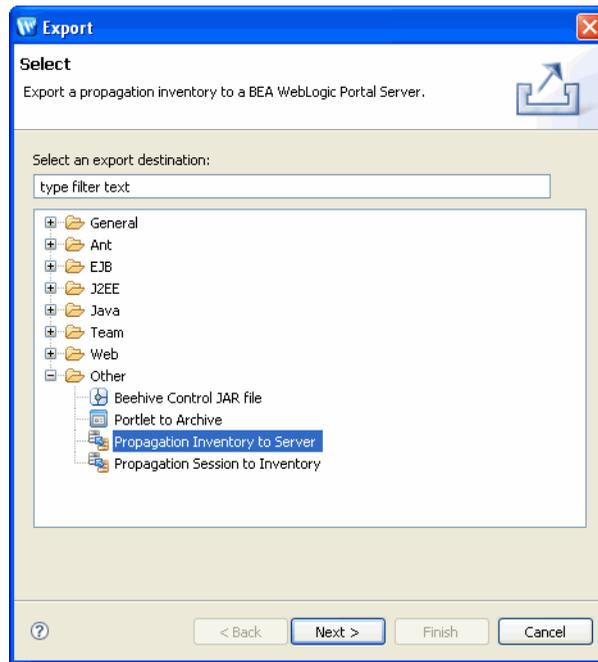
2. In the Choose Propagation Inventory dialog, select the final merged inventory file, as shown in Figure 7-17, and click **Next**.

Figure 7-17 Choose Propagation Inventory Dialog
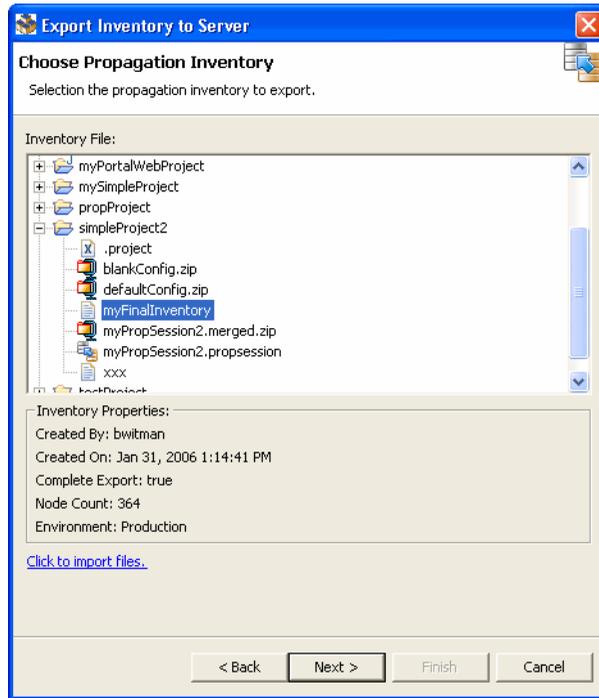
3. In the Export to Server Parameters dialog, enter the URL of the propagation servlet running
   in the target web application, and the required username and password information, as shown
   in Figure 7-18, and click **Finish**.

   The servlet URL is:

```
http://servername:portnumber/earProjectNamePropagation/inventorymanagement
```

   Where `earProjectName` is the name of the EAR project that contains the portal
   application that you are propagating. For example: `myEARProjectPropagation`. For
   example:

```
http://localhost:7001/myEARProjectPropagation/inventorymanagement
```

   Tip: The propagation servlet is deployed, by default, in all WebLogic Portal EAR projects.
   This servlet enables communication with remote propagation clients, such as the
   Workshop for WebLogic propagation tools and Propagation Ant Tasks. The servlet

allows remote clients to perform online operations, such as downloading, uploading, and committing WebLogic Portal inventories.

Figure 7-18 Export To Server Parameters Dialog
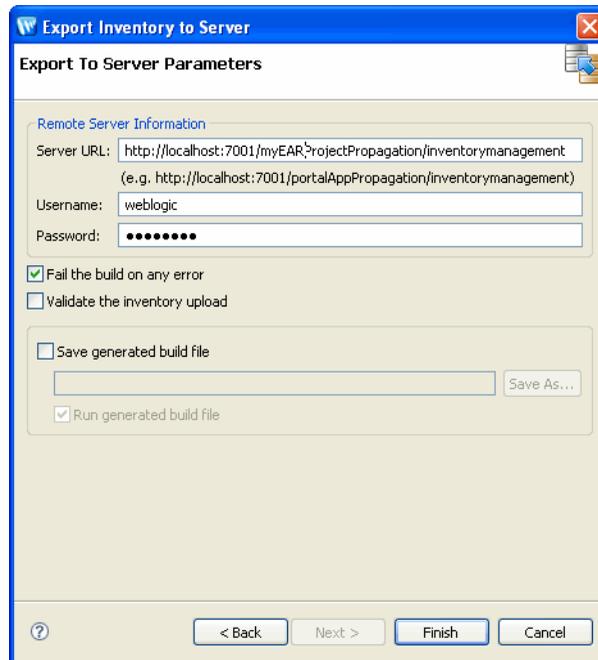
Workshop for WebLogic deploys the merged inventory file to the server.

Enabling Verbose Logging

Verbose logging is useful in the event that a problem occurs with your propagation. You can choose to enable verbose logging in Workshop for WebLogic by following these steps:

1. Exit Workshop for WebLogic.

2. Open the following file in an editor:

   ```
   BEA_HOME\workshop_10.0\workshop4WP\workshop4WP.ini
   ```

3. Add the following property to the `workshop4WP.ini` file:

   ```
   -Dcom.bea.wlp.eclipse.proptool.verbosefolder=D:\propagation\elogs
   ```

where *D:\propagation\elogs* is the directory in which you want the verbose logs to be saved.

4. Restart Workshop for WebLogic.

Note: The verbose logs will be written to the specified directory. You must purge this directory from time to time, to avoid unwanted disk space use.

Using the Propagation Ant Tasks

The propagation Ant tasks provide a full set of tools that you can use to propagate WebLogic Portal assets from one environment to another programmatically.

This chapter introduces the propagation Ant tasks and discusses related topics such as scoping and policies. This chapter includes the following sections:

- Introduction
- Before You Begin
- Installing the Ant Tasks
- Overview of Online Tasks
- Overview of Offline Tasks
- Scoping an Inventory
- Using Policies
- Combining and Committing Inventories

Introduction

The Ant tasks let you perform all of the functions that you can perform with the Workshop for WebLogic propagation tools, plus additional options and functions, such as placing the server into maintenance mode. The Ant tasks:

- Allow you to automate the propagation process.

- Provide a richer set of features than the Workshop for WebLogic propagation tools.

- Let you perform a propagation entirely on the destination system.

Before You Begin

Note: The Administration Server must be running when you perform a propagation to allow certain LDAP data to be updated. A propagation can cause the following kinds of LDAP data to be added, deleted, or updated: visitor roles, delegated administration roles, entitlement policies, and delegated administration policies.

Before you attempt to propagate a portal web application using the Ant tasks described in this chapter, it is important to be familiar with the basic concepts of WebLogic Portal propagation. For detailed information on planning a propagation strategy, see Chapter 5, "Developing a Propagation Strategy." The Ant tasks provide the same basic features as the Workshop for WebLogic propagation tools. The basic concepts and considerations outlined in Chapter 6, "Propagation Topics" apply to both the propagation tools and to the Ant tasks.

Tip: We also recommend that you read Chapter 7, "Using Workshop for WebLogic Propagation Tools" before designing an Ant-based propagation script. This chapter describes how to propagate a portal using the Workshop for WebLogic propagation tools. These tools step you through the propagation process and provides a visual interface for merging, viewing, and tuning inventories. Reviewing and understanding the workflow used by the propagation tools can be useful to you as you plan your Ant-based propagation.

Installing the Ant Tasks

The Ant tasks are divided into two categories: online and offline. The online tasks interact with a WebLogic Portal application that is deployed and running. For example, the OnlineDownloadTask lets you extract a portal inventory from a live server and store it in a file. To accomplish this, the task communicates with a servlet that is deployed with the enterprise application. The online tasks are summarized in "Overview of Online Tasks" on page 8-7.

The offline tasks operate on inventory files that have already been extracted and saved. The offline tasks do not require network connectivity. The offline tasks are summarized in "Overview of Offline Tasks" on page 8-9.

To use either the online or offline Ant tasks, you need to put the JAR files containing the tasks in your CLASSPATH. To use the online tasks, you also need to deploy a library module with the Portal EAR Project on the source and destination systems.

This section includes these installation topics:

- Deploying the Propagation Servlet

- Testing the Ant Installation

- Using the Ant Tasks Outside of a WebLogic Portal Environment

Deploying the Propagation Servlet

A servlet that handles propagation requests is provided with your WebLogic Portal installation. If you intend to use the online Ant tasks, this servlet must be deployed in the Portal EAR Project that contains the portal application(s) you want to propagate.

By default, the Propagation Service facet is included in a Portal EAR Project, as shown in Figure 8-1. This facet includes the propagation servlet. If you created the Portal EAR Project with the Propagation Service facet selected, then the propagation servlet is automatically deployed when you deploy the EAR to the server.

Figure 8-1 Propagation Service Module
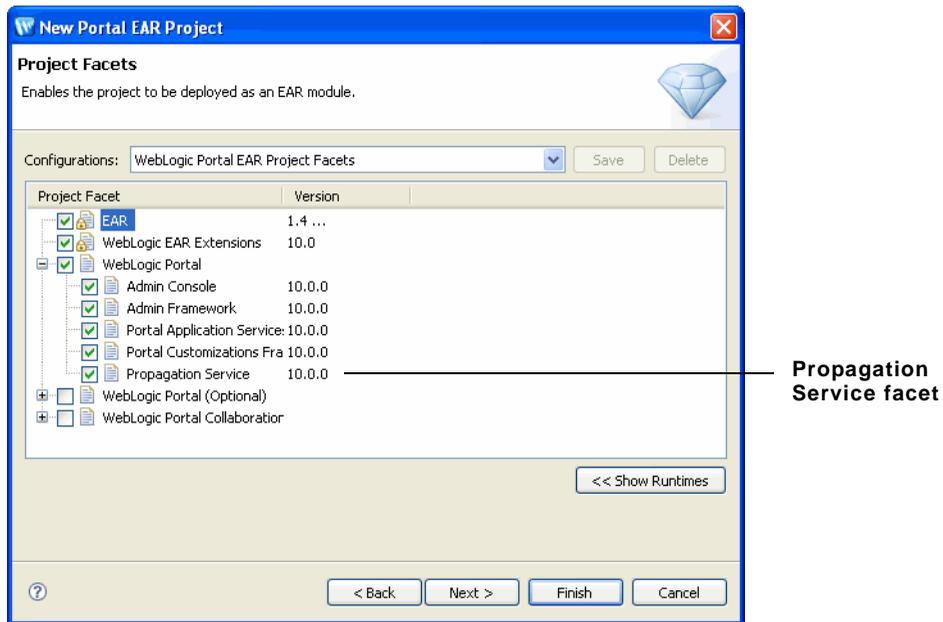
When it is deployed, the name of the servlet is *earProjectName*Propagation.

Where *earProjectName* is the name of the EAR project that contains the portal application that you are propagating. For example: myEARProjectPropagation

---

Tip: You can see that the servlet has been added to your application by looking in the application's META-INF/weblogic-application.xml file. Listing 8-1 shows the stanzas added for the example application, myEarProject.

Listing 8-1   Propagation Servlet Configuration

```
<library-ref>
    <library-name>wlp-propagation-app-lib</library-name>
</library-ref>
<library-context-root-override>
    <context-root>propagation</context-root>
```

```
    <override-value>myEarProjectPropagation</override-value>
</library-context-root-override>
```

You need to know the name of the servlet when you use the online Ant tasks, because these tasks use the servlet to process their requests. For example, the OnlinePingTask takes a **serverURL** parameter that requires the servlet name:

```
servletURL="http://localhost:7001/myEarProjectPropagation/
inventorymanagement"
```

You can check whether the propagation servlet is included in your EAR project by doing the following:

1. Open Workshop for WebLogic.

2. Right-click the EAR project in the Package Explorer, and select **Properties**. The Project Facets dialog appears, as shown in Figure 8-2.

Figure 8-2 Checking the Propagation Service Facet
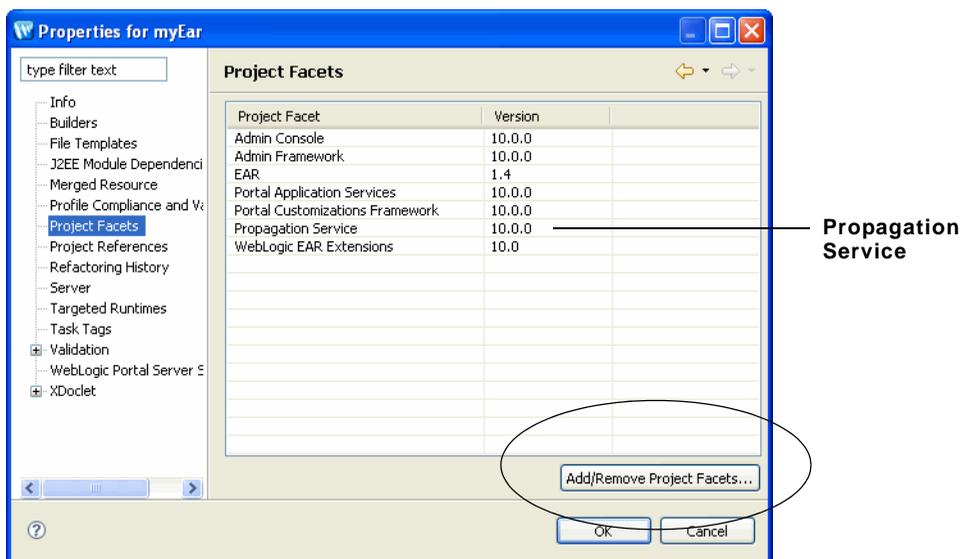
3. In the Project Facets dialog, be sure that the **Propagation Service** facet is included. If it isn't, select **Add/Remove Project Facets** and add this facet.

4. Click **OK** to complete the operation.

Testing the Ant Installation

WebLogic Portal provides a sample Ant build script that you can use to execute each of the propagation Ant tasks. This section explains how to use the sample build script to test that the propagation servlet is installed correctly. The sample build script is:

```
WEBLOGIC_HOME/portal/bin/propagation/propagation_ant.xml
```

1. Start the WebLogic Portal server in which your portal EAR project is deployed.

2. Edit the sample build script appropriately to include correct values for your WebLogic Portal installation directory, a directory in which to write script output, the propagation servlet URL, and other information.

Tip: The script file includes detailed information about the variables you need to provide. Be sure to read the instructions in the script carefully.

---

3. After you have properly configured the build script variables, run the OnlinePingTask. To do this, enter the following command.

```
ant -f propagation_ant.xml pingSrc
```

If the operation succeeds, you will see the output shown in Listing 8-2.

Listing 8-2 Successful Ping Message

```
pingSrc:

     [echo] ****** Start: pingSrc ******

[onlinePing] INFO: Using the logger delegate
(com.bea.p13n.management.inventory.util.OfflineInventoryLoggerDelegate)
specified in the propagation_context.properties file.

[onlinePing] WARNING: This command is using HTTP, which will transmit the
administrator password in plain text. It is recommended to set allowHttp to false.

     [echo] ****** Finish: pingSrc ******
```

Tip: You can copy the `propagation_ant.xml` build script and use it as a basis for creating your own custom propagation build scripts. Note that the sample script includes examples of running each task both with and without a condition property. The condition property, provided by the Ant ConditionTask, allows you to implement simple flow control in the Ant script.

Using the Ant Tasks Outside of a WebLogic Portal Environment

To use the online or offline Ant tasks outside of a WebLogic Portal environment, you need to copy a set of JAR files from a WebLogic Portal installation to your environment. These JAR files must be placed in the `CLASSPATH` for the Ant tasks on the system where you intend to use them.

Required JARS for Online Tasks

```
WEBLOGIC_HOME/platform/lib/p13n/p13n_common.jar
WEBLOGIC_HOME/portal/lib/propagation/propagation.jar
WEBLOGIC_HOME/portal/lib/propagation/propagation_ant.jar
WEBLOGIC_HOME/portal/lib/propagation/content_prop.jar
WEBLOGIC_HOME/portal/lib/propagation/netuix_prop.jar
```

Required JARS for Offline Tasks

```
WEBLOGIC_HOME/platform/lib/p13n/p13n_common.jar
WEBLOGIC_HOME/portal/lib/propagation/propagation.jar
WEBLOGIC_HOME/portal/lib/propagation/propagation_ant.jar
WEBLOGIC_HOME/portal/lib/propagation/content_prop.jar
WEBLOGIC_HOME/portal/lib/propagation/netuix_prop.jar
WEBLOGIC_HOME/server/lib/api.jar
WEBLOGIC_HOME/server/lib/webserviceclient.jar
WEBLOGIC_HOME/platform/lib/wlp/wsrp-common.jar
BEA_HOME/modules/com.bea.core.xml.beaxmlbeans_2.2.0.0.jar
BEA_HOME/modules/com.bea.core.weblogic.stax_1.0.0.0.jar
```

Overview of Online Tasks

The online tasks operate on a WebLogic Portal application that is deployed and running. They communicate with the portal application through a servlet that is deployed with the enterprise

application. For example, online tasks let you create and download inventory files from the server to another machine. For information on the propagation servlet, see "Deploying the Propagation Servlet" on page 8-3.

Note: The online Ant tasks use HTTP or HTTPS to reach WebLogic Server, therefore you must make sure that the intervening firewalls allow that.

This section includes these topics:

- Online Task Summary

- Troubleshooting Online Tasks

Online Task Summary

Table 8-2 summarizes the online propagation Ant tasks.

Table 8-1 Online Tasks

OnlineCheckMutexTask	Verifies that the propagation servlet is not currently in use by another process.
OnlineCommitTask	Performs a final inventory file merge and commits the final merged inventory to the destination server.
OnlineDownloadTask	Downloads the inventory from a currently running WebLogic Portal application to a specified ZIP file.
OnlineMaintenanceModeTask	Prevents administrators from making changes to the portal through the WebLogic Portal Administration Console.
OnlinePingTask	Tests if the propagation management servlet is running on the designated server.
OnlineUploadTask	Uploads an inventory file to a temporary location associated with a running WebLogic Portal application.

Using Online Tasks with HTTPS

To use HTTPS with the online Ant tasks, follow the standard WebLogic SSL instructions found in the WebLogic Server document, "Configuring Identity and Trust."

Note: When formatting the URL to the remote WebLogic server, be sure to specify the actual host name of the system rather than `localhost` if you are on the same system. Also, be sure to use the HTTPS port number for the server. The default is 7002.

Troubleshooting Online Tasks

If an online propagation Ant task fails, typically the failure is caused by one of these reasons:

- The server on which the source or destination portal application is deployed has not been started. Verify that the server or cluster has been started.

- The propagation servlet is not deployed. Follow the instructions in "Deploying the Propagation Servlet" on page 8-3 to deploy the servlet.

- There is a network problem. Use an operating system utility to make sure the server can be reached.

- The propagation servlet is in use by another process. Only one thread can use the propagation servlet at one time. You can use the OnlineCheckMutexTask to check to see if another process is currently using the servlet.

Overview of Offline Tasks

Offline Ant tasks operate exclusively on previously exported inventory files. These tasks operate on a single inventory file or on two inventory files to explore and manipulate the contents of the inventor(ies). The offline tasks provide features similar to the Workshop for WebLogic propagation tools.

These tasks do not require connectivity to a live WebLogic Portal application. For instance, you can use offline tasks to combine and compare source and destination inventory files.

This section includes these topics:

- Offline Task Summary
- Troubleshooting Offline Tasks

Offline Task Summary

Table 8-2 lists the offline propagation Ant tasks.

Table 8-2 Offline Tasks

OfflineCheckManualElections	Tests for the presence of manual elections (changes).
OfflineCombineTask	Combines `src.zip` and `dest.zip` into a new inventory, `combined.zip`.
OfflineDiffTask	Differences `src.zip` and `dest.zip` and writes the results to `diff_cm.xml`.
OfflineElectionAlgebraTask	Allows for algebraic operations on two change manifest files.
OfflineExtractTask	Extracts the working artifacts out of `combined.zip` for viewing.
OfflineInsertTask	Inserts updated working artifacts into `combined.zip`.
OfflineListPoliciesTask	Exports the valid policies from an inventory file.
OfflineListScopesTask	Lists the scopes in `src.zip` that will be used to combine two inventories.
OfflineSearchTask	Finds nodes with a specific string in their name, to help verify the wanted node was exported.
OfflineValidateTask	Makes sure `combined.zip` is a valid ZIP file.

Troubleshooting Offline Tasks

If an offline propagation Ant task fails, typically the failure is caused by one of these reasons:

- The input parameters are not properly specified.

- An unexpected error occurred, such as encountering a full disk.

Scoping an Inventory

Scoping refers to limiting the number of artifacts in an exported WebLogic Portal inventory, and, therefore, the number of artifacts that must be added, deleted, or updated during propagation. In general, scoping reduces both the duration and complexity of propagation operations.

Tip: Before continuing, we recommend you review the detailed discussion of scoping, in "Understanding Scope" on page 6-11.

This section includes these topics:

- Scoping with Ant Tasks
- Sample Scoping Workflow

Scoping with Ant Tasks

Several of the Ant tasks either create or use a property file, by default called `scope.properites`, that specifies scoping rules.

For example, you can edit the `scope.properties` file to adjust the propagation scope and then use the file as a parameter to the OnlineDownloadTask. This file declares how the task is to treat each node in the inventory as either in scope or out of scope. The task looks at the scoping rules and applies them to the exported inventory. As a result, the scoped inventory file is usually a subset of the full inventory. For more information on the `scope.properties` file, see "Understanding a Scope Property File" on page 8-13.

Sample Scoping Workflow

This section explains the basic workflow for scoping an inventory using the Ant tasks. The objective of this workflow is to reduce the size and complexity of an inventory file through scoping. The result of this workflow is a `scope.properties` file. This file contains a set of scoping rules that you can then use when you combine two inventories to produce a final merged inventory.

The Ant tasks used in this workflow include:

- **OnlineDownloadTask** – This task is used to retrieve an inventory file from a WebLogic Portal application that is running on a server.

- **OfflineListScopesTask** – This task exports the valid scopes from an inventory file. You can specify the depth of the scope. The depth limits how far into the inventory tree the task traverses. This task produces a `scope.properties` file, which you can edit.

- **OfflineExtractTask** – This task exports files that are stored in the top level of an inventory ZIP file. One of these files is a `scope.properties` file.

Figure 8-3 illustrates the basic steps involved in scoping an inventory file. This procedure uses Ant tasks in a chain to produce the final output: a `scope.properties` file containing the scoping information needed to combine inventories.

Figure 8-3 Scoping a Source Inventory
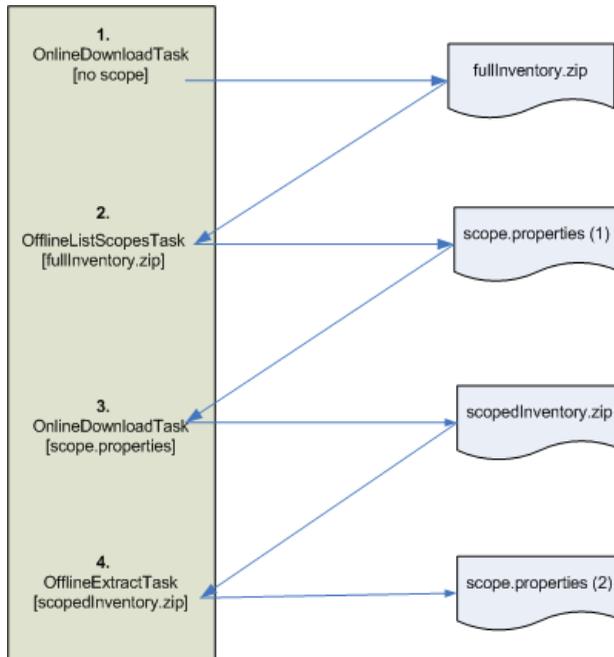
The steps shown in Figure 8-3 include the following:

1. Call the OnlineDownloadTask. The first time you call this task, do not specify the `scopeFile` attribute. If you do not specify this attribute, the task retrieves the entire inventory from the server. The retrieved inventory is stored in a ZIP file, which is shown in Figure 8-3 as `fullInventory.zip`.

2. Use OfflineListScopesTask to extract a `scope.properties` file from the inventory file. The `scope.properties` file specifies the scoping rules that were used to produce the inventory. See "Understanding a Scope Property File" on page 8-13 for more information on the contents of a `scope.properties` file. To produce a scoped inventory, edit the `scope.properties` file and use the edited file as input to the OnlineDownloadTask again.

3. Run the OnlineDownloadTask again with the edited `scope.properties` file as input. The result of this operation is a new inventory file that only includes the artifacts that were within the specified scope.

4. The final step in this workflow is optional. It is included in Figure 8-3 to show that you can extract the `scope.properties` file from the scoped inventory file. In fact, the properties file that is returned is identical to the one that was used as input to the OnlineDownloadTask in Step 3.

After you have obtained a scoped source inventory file, you can combine it with a destination inventory file to produce a merged inventory file.

Understanding a Scope Property File

The taxonomies listed in a `scope.properties` file indicate that the given taxonomy and all of its children will be included in the inventory. For example, Listing 8-3 shows a one-line `scope.properties` file, with only the taxonomy value `Application` specified. This taxonomy indicates that the entire application will be included in the inventory.

Listing 8-3 Excerpt from a scope.properties File

```
scope_0=Application
```

Listing 8-4 is another simple scope file instruction, that indicates that all content from every repository associated with the application will be included in the inventory.

Listing 8-4 Excerpt from a scope.properties File

```
scope_1=Application\:ContentServices
```

Listing 8-5 indicates that only the content node `node1` and its children and content type `typeA` and its children will be included.

Listing 8-5   Excerpt from a scope.properties File

```
scope_0=Application\:ContentServices\:Tools_Repository\:ContentNodes\:node1

scope_1=Application\:ContentServices\:Tools_Repository\:ContentTypes\:typeA
```

In summary, the taxonomies listed in the previous examples have the following meanings:

`Application`

– Application is the root of all taxonomies and includes everything. If you scope to this taxonomy value, the entire application will be included in the inventory.

`Application\:ContentServices`

– Includes all content from every repository.

`Application\:ContentServices\:Tools_Repository`

– Includes all content from the `Tools_Repository` content repository.

`Application\:ContentServices\:Tools_Repository\:ContentNodes`

– Includes all content nodes in the `Tools_Repository` content repository.

`Application\:ContentServices\:Tools_Repository\:ContentNodes\:node1`

– Includes `node1` and all its children in the `Tools_Repository` content repository.

Using Policies

Policies let you control how source and destination inventories are merged when they are combined into a final inventory file. To use policies, export a `policy.properties` file using the OfflineListPoliciesTask. This task lets you set global policies to apply to all assets in the inventory. For example, you can elect to accept all additions, but reject deletions, and updates. See "Understanding a Policies Property File" on page 8-15. You can then edit the `policy.properties` file to modify the policy elections for specific assets, if you want to.

Tip: Before continuing, we recommend you review the detailed discussion of policies in "Using Policies" on page 6-22.

Policies let you elect how to handle the following three merge cases. Each merge case can be set to true or false in the policy file.

- **Additions** – If an asset exists in the source inventory, but not in the destination inventory, add it to the destination.

- **Deletions** – If an asset exists in the destination inventory, but not in the source inventory, delete it from the destination.

- **Updates** – If an asset exists in the source inventory and in the destination inventory, update the destination with the artifact in the source inventory.

Understanding a Policies Property File

Listing 8-6 Shows an excerpt from a `policies.properties` file. As you can see, in every case, the policy for this propagation is to accept (Y) adds and deletes, but to ignore (N) updates. A policy is set on each asset of the portal. You can change the policy on an asset by editing this file.

Listing 8-6   Excerpt from a policies.properties File

```
policy_0_taxonomy=Application
policy_0_adds=Y
policy_0_updates=N
policy_0_deletes=Y
policy_1_taxonomy=Application:ContentServices
policy_1_adds=Y
policy_1_updates=N
policy_1_deletes=Y
policy_2_taxonomy=Application:ContentServices:Tools_Repository
policy_2_adds=Y
policy_2_updates=N
policy_2_deletes=Y
policy_3_taxonomy=Application:ContentServices:Tools_Repository:ContentNodes
policy_3_adds=Y
policy_3_updates=N
policy_3_deletes=Y
policy_4_taxonomy=Application:ContentServices:Tools_Repository:ContentTypes
policy_4_adds=Y
policy_4_updates=N
policy_4_deletes=Y
policy_5_taxonomy=Application:ContentServices:Tools_Repository:GlobalEntitlements
policy_5_adds=Y
policy_5_updates=N
policy_5_deletes=Y
```

Combining and Committing Inventories

You can think of propagation as a combining operation. When you propagate a portal, you combine the contents of a source inventory with a destination inventory. The combining is governed by scoping and policy rules. The previous section, "Scoping an Inventory" on page 8-10, explained how to create a `scope.properties` file, which contains the scoping rules for an inventory. You can use this file as input to the OfflineCombineTask, which uses scoping information to decide which artifacts to combine to produce the final inventory file.

This section explains the basic workflow for combining and committing inventories using propagation Ant tasks. The Ant tasks used in this workflow include:

- **OfflineCombineTask** – Combines a source with a destination inventory file. Optional task attributes let you specify a `scope.properties` and a `policy.properties` file. These files control how the two inventories are combined.

- **OnlineUploadTask** – This task moves an inventory file to a specified server, and is similar to an FTP transfer. The inventory file is stored in a temporary directory in the servlet container.

- **OnlineCommitTask** – Replaces the existing inventory in the running server with the new inventory.

Figure 8-4 shows the basic workflow for combining and committing inventories. The workflow assumes that you have already exported (using the OnlineDownloadTask) a source and a destination inventory file.

Figure 8-4 Combining and Committing an Inventory
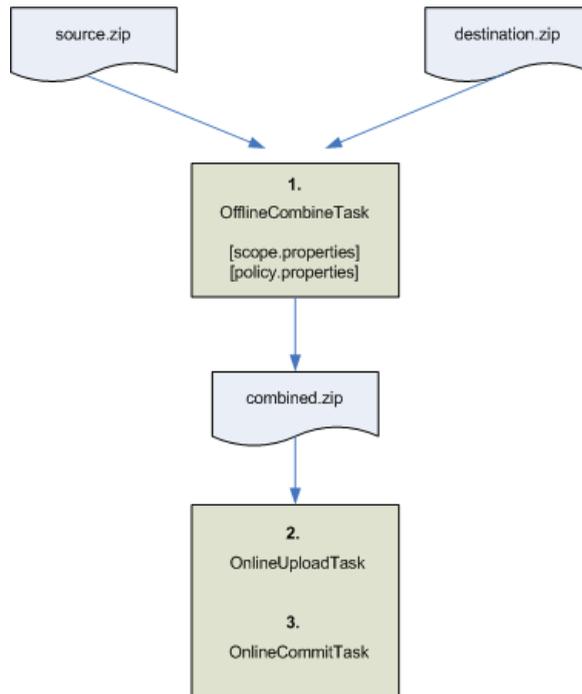
After you have a source and destination inventory file, you can combine them using the OfflineCombineTask. The result of this task is a combined inventory file. This file is a combination of the source and destination inventory files with scope and policy rules (if they were specified) applied.

Use the OnlineUploadTask to move the combined inventory to the destination server. Use the OnlineCommitTask to commit the combined inventory on the destination server.

The OfflineCombineTask, OnlineUploadTask, and OnlineCommitTask are explained in Chapter 9, "Propagation Ant Task Reference."

Propagation Ant Task Reference

This chapter describes the attributes, modifiers, and usage of the propagation Ant tasks. The Ant tasks are broken down into online and offline tasks. Online tasks interact directly with a WebLogic Portal application that is running on a server. Offline tasks interact with previously exported inventory files.

Note: WebLogic Portal inventories saved with WebLogic Portal 8.1 or 9.2 cannot be used with WebLogic Portal 10.0 propagation tools.

Tip: For a general overview of the Ant tasks, including use cases and examples, see Chapter 8, "Using the Propagation Ant Tasks."

This chapter includes the following sections:

- Online Tasks

- Offline Tasks

Online Tasks

This section describes each of the online propagation Ant tasks. For an introduction to online tasks, see "Overview of Online Tasks" on page 8-7. The online tasks described in this section include:

- OnlineCheckMutexTask

- OnlineCommitTask

- OnlineDownloadTask

- OnlinePingTask

- OnlineMaintenanceModeTask

- OnlineUploadTask

OnlineCheckMutexTask

Verify that the propagation servlet is not currently in use by another process. The mutex is scoped to the application, and is cluster aware. This means that if anyone is using the servlet in a particular application on any node in the cluster, the mutex is enforced: another user cannot use the servlet for that application on another node.

Tip: The propagation servlet executes propagation tasks on the server. This servlet performs operations on WebLogic Portal data, and therefore does not allow more than one thread of execution at a time. For more information on the propagation servlet, see "Deploying the Propagation Servlet" on page 8-3.

Attributes

- **allowHttp** – (optional) If set to `true`, allows the propagation management servlet specified by `servletURL` to use HTTP. If set to `false`, allows the URL to use HTTPS. Default: `false`. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **password** – (required) The user's password. The password can be specified either in plain text or 3DES encoded form.

- **retryTimes** – (optional) An integer that sets the number of times to retry obtaining the mutex. Default = 1. A value of –1 indicates to retry indefinitely. After the specified number of retries, the task fails, unless the `failOnError` attribute is set to `false`.

- **servletURL** – (required) The URL of the propagation management servlet. This URL must point to a specific managed node if in a cluster, not to a proxy. If the servlet is included in

your EAR project, it is automatically deployed when you start your server. The name of the deployed servlet is:

`earProjectName`Propagation

Where `earProjectName` is the name of the EAR project that contains the portal application that you are propagating. For example:

`myEARProjectPropagation`

See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **username** – (required) The name of a user with access to the destination server.

Ant Condition Property

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The servlet mutex is available for use.

- **failure** – The servlet mutex is locked by another thread. The task returns immediately without blocking.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if another thread has the mutex. The script must wait for the other thread to complete. See also "Troubleshooting Online Tasks" on page 8-9.

Usage

Listing 9-1 checks to see if the propagation servlet is in use. The script retries 10 times.
Listing 9-2 uses an Ant conditional to test for success. If the task succeeds, a message is printed.

Listing 9-1 OnlineCheckMutexTask Example

```
<target name="checkMutexDest"  description="checks to see if the mutex is available">
    <onlineCheckMutex
        servletURL="http://localhost:7001/myProjectPropagation/inventorymanagement"
        username="weblogic"
        password="weblogic"
        allowHttp="true"
        failOnError="false"
```

```
        retryTimes="10"
    />
</target>
```

Listing 9-2   OnlineCheckMutexTask with Ant Conditional

```
<target name="checkMutexDest" description="checks to see if the mutex is available">
    <condition property="mutex_success">
        <onlineMutex
        servletURL="http://localhost:7001/myProjectPropagation/inventorymanagement"
        username="weblogic"
        password="weblogic"
        allowHttp="true"
        failOnError="false"
        retryTimes="10"
    />

    </condition>
    <antcall target="mutex_success" />
</target>

<target name="mutex_success" if="mutex_success">
    <echo message="The mutex is available." />
</target>
```

OnlineCommitTask

Note: This task will fail if the embedded LDAP data and security data in the database are out
of sync. For example, if a role that exists in the database is deleted from the LDAP
repository, this task will fail. You can force the task to continue using the modifier
continueOnValidationError.

This task performs two operations:

- **Final Merge** – If the **differenceStrategy** attribute is set to **pessimistic**, a final merge of the
source and destination inventories is performed, based scope or policy settings. The merge
is performed in a temporary directory on the destination system.

- **Commit** – The task commits the final merged inventory to the destination server. Before
you commit an inventory, you must upload it to the destination server using the
OnlineUploadTask. The OnlineUploadTask places the inventory in a predetermined
location, and the OnlineCommitTask always looks in that location. For this reason, you do

not have to specify the name of the inventory file you are committing with this task. See also "OnlineUploadTask" on page 9-15.

WARNING: This task modifies the target application's configuration. Any users with active sessions may experience unwanted behavior depending on the changes that are being committed. The risk to users is similar to the behavior they would see if an administrator made the same changes with the WebLogic Portal Administration Console.

Attributes

- **allowHttp** – (optional) If set to `true`, allows the propagation management servlet specified by `servletURL` to use HTTP. If set to `false`, allows the URL to use HTTPS. Default: `false`. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **password** – (required) The user's password. The password can be specified either in plain text or 3DES encoded form.

- **servletURL** – (required) The URL of the propagation management servlet. This URL must point to a specific managed node in a cluster, not to a proxy. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **username** – (required) The username of a user with access to the destination server.

Modifiers

This section describes the `<modifier>` attributes available for this task. Each `<modifier>` attribute has a `name` and a `value`. See the *Usage* section for an example of using modifiers.

- **name = cm_checkinComment**

  Specifies a comment to use when new revisions of content are checked into a content management repository as a result of the propagation.

  value = A single line of text.

- **name = cm_checkoutPolicy**

Specifies what action to take when the task tries to update or delete a content item in a content management repository if that content item is already checked out.

value = abort

(default) Do not update or delete the content item and write a message to the log.

value = revert

Overrides the checked out content item, and adds or updates the content in the repository. This option reverts any changes made by the user who currently has the content item checked out and the changes in the change manifest are applied.

- **name = cm_nodeWorkflowPolicy**

Specifies what action to take if the name of a workflow on the source system does not match the expected workflow name on the destination. This modifier lets you choose how to handle the situations by letting you choose whether to retain the workflow on the destination, or to abort the propagation of the source node that includes the workflow.

Tip: Because WebLogic Portal does not propagate workflows, it is a best practice to ensure that the source and destination systems use exactly the same workflow. This practice ensures that nodes and types can be propagated with the least risk.

---

value = retain

(default) If the workflow name associated with a content node differs between the source and destination, retain the workflow on the destination.

value = abort

If the workflow name associated with a content node differs between the source and destination, do not propagate the node. Note that if any nodes on the source system depend on the aborted node, they cannot be propagated either.

- **name = cm_typeWorkflowPolicy**

Specifies what action to take if a workflow name on the source system does not match the expected workflow name on the destination. This modifier lets you choose how to handle the situations by letting you choose whether to retain the workflow on the destination, or to abort the propagation of the source content type that includes the workflow.

Tip: Because WebLogic Portal does not propagate workflows, it is a best practice to ensure that the source and destination systems use exactly the same workflow. This practice ensures that nodes and types can be propagated with the least risk.

value = retain

(default) If the workflow name associated with a content type differs between the source and destination, retain the workflow on the destination.

value = abort

If the workflow name associated with a content type differs between the source and destination, do not propagate the type. Note that if any types on the source system depend on the aborted type, they cannot be propagated either.

- **name = differenceStrategy**

Specifies how the propagation management servlet obtains the list of differences to process before the final inventory is committed to the server.

value = pessimistic

(default) The propagation servlet computes the differences between the uploaded inventory and the application inventory. It then applies those differences according to the policy and scope settings provided with the uploaded inventory. Pessimistic causes differences to be re-computed in real time based on the current state of the destination. Pessimistic differencing generally takes longer than optimistic. See also "Understanding a Scope Property File" on page 8-13 and "Understanding a Policies Property File" on page 8-15.

value = optimistic

The propagation servlet looks for the changemanifest.xml file in the inventory. If the change manifest exists, it applies the specified elections from the manifest without recomputing the differences. With this option, the servlet does not honor scope or policy settings provided with the inventory. If this option is specified, but the changemanifest.xml file does not exist, the servlet defaults to pessimistic.

- **name = continueOnValidationError**

If **value = true**, this modifier forces the task to continue in the event of a validation error. A validation error can occur if the embedded LDAP repository and the security information in the database are out of sync. (default = false).

Ant Conditional Support

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – This task succeeds if the inventory is successfully committed.

- **failure** – The inventory was not committed successfully. Check the log files for additional information.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if the inventory is not committed successfully. Check the log files for additional information. See also "Troubleshooting Online Tasks" on page 8-9.

Usage

The following example commits the inventory file uploaded to the destination server with the OnlineUploadTask, which places the file in a standard location.

Listing 9-3   OnlineCommitTask Example

```
<target name="commitOpt">
    <onlineCommit
        servletURL="http://localhost:7001/myProjectPropagation/inventorymanagement"
        username="weblogic"
        password="weblogic"
        allowHttp="true"
    >
        <modifier name="differenceStrategy" value="optimistic" />
        <modifier name="cm_checkinComment" value="My sample checkin comment." />
    </onlineCommit>
</target>
```

OnlineDownloadTask

Download the inventory from a currently running WebLogic Portal application to a specified ZIP file.

Note: This task will fail if the embedded LDAP data and security data in the database are out of sync. For example, if a role that exists in the database is deleted from the LDAP repository, this task will fail.

Note: This task extracts the portal inventory and attempts to write it to a ZIP file. If the ZIP file created exceeds 4 GB, this task fails and a message is written to the server log and the verbose log. If this occurs, try scoping your inventory to limit the size of the resulting archive file. See "Understanding Scope" on page 6-11 for more information.

Attributes

- **allowHttp** – (optional) If set to `true`, allows the propagation management servlet specified by `servletURL` to use HTTP. If set to `false`, allows the URL to use HTTPS. Default: `false`. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **outputInventoryFile** – (required) The file in which to write the inventory file. If `outputToServerFileSystem` is `false`, the file is saved locally, the name can include a relative or absolute path, and it must not already exist. If `outputToServerFileSystem` is `true`, the file is written to the destination server system, you must supply an absolute path on the server system, and the file must not already exist.

- **outputToServerFileSystem** – (optional) If true, the output file is written to the server file system. Use this attribute to avoid a lengthy HTTP download if the inventory file is large. Default: `false`.

- **password** – (required) The user's password. The password can be specified either in plain text or 3DES encoded form.

- **scopeFile** – (optional) The pathname of the `scope.properties` file. If specified, this file must exist or the task will fail. See "Scoping with Ant Tasks" on page 8-11.

- **servletURL** – (required) The URL of the propagation management servlet. This URL must point to a specific managed node in a cluster, not to a proxy. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **username** – (required) The username of a user with access to the destination server.

Modifiers

This section describes the `<modifier>` attributes available for this task. Each `<modifier>` attribute has a `name` and a `value`. See the *Usage* section for the OnlineCommitTask for an example of using modifiers.

- **name = continueOnValidationError**

  If **value = true**, this modifier forces the task to continue in the event of a validation error. A validation error can occur if the embedded LDAP repository and the security information in the database are out of sync. (default = false).

Ant Conditional Support

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The inventory downloaded successfully.

- **failure** – The download failed.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if the inventory is not downloaded successfully. Check the log files for more information. See also "Troubleshooting Online Tasks" on page 8-9.

Usage

The following example downloads an inventory file called `dest.zip`.

Listing 9-4 OnlineDownloadTask Example

```
<target name="downloadDest">
    <onlineDownload
    servletURL="http://localhost:7001/myProjectPropagation/inventorymanagement"
    username="weblogic"
    password="weblogic"
    allowHttp="true"
    outputInventoryFile="${portal.prop.home}/test/ant/dest.zip"
    />
</target>
```

OnlineMaintenanceModeTask

In order to maintain data integrity during a propagation session, the applications on the source and destination servers must be placed into maintenance mode. Maintenance mode prevents administrators from making changes to the portal through the WebLogic Portal Administration Console. Maintenance mode takes effect for the entire enterprise application (not just a single web application) and it takes effect for all nodes in a cluster. After the export of the finalized inventory is complete, you can turn maintenance mode off to enable the applications on the server for modifications.

This task toggles the state of maintenance mode on the server. This Ant task causes delegated administration and entitlements to deny access to all resources with policies within the application regardless of requested resource or capability.

Tip: You can also set Maintenance Mode in the WebLogic Portal Administration Console, select **Configurations > Service Administration**. For more information, refer to the online help.

Attributes

- **allowHttp** – (optional) If set to `true`, allows the propagation management servlet specified by `servletURL` to use HTTP. If set to `false`, allows the URL to use HTTPS. Default: `false`. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **enable** – (required) If set to `true`, turns maintenance mode on. To turn maintenance mode off, and thereby allow users to modify portal resources, set this value to `false`.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **password** – (required) The user's password. The password can be specified either in plain text or 3DES encoded form.

- **servletURL** – (required) The URL of the propagation management servlet. This URL must point to a specific managed node if in a cluster, not to a proxy. For information on how to

form this URL, see "Deploying the Propagation Servlet" on page 8-3. See also "Using Online Tasks with HTTPS" on page 8-8.

- **username** – (required) The username of a user with access to the destination server.

Ant Condition Property

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – Maintenance mode state was toggled successfully.

- **fails** – Maintenance mode state could not be toggled. Check the maintenance mode state in the WebLogic Portal Administration Console.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if maintenance mode could not be toggled. Check the Maintenance mode state in the WebLogic Portal Administration Console. See also "Troubleshooting Online Tasks" on page 8-9.

Usage

Listing 9-5 places the server into maintenance mode. Listing 9-6 uses an Ant conditional to place the server in maintenance mode and print a message if the operation is successful.

Listing 9-5   OnlineMaintenanceModeTask Example

```
<target name="lockDest" description="lock the server">
    <onlineMaintenanceMode
        servletURL="http://localhost:7001/propagation/inventorymanagement"
        username="weblogic"
        password="weblogic"
        allowHttp="true"
        enable="true"
    />
</target>
```

Listing 9-6   Using OnlineMaintenanceModeTask with an Ant Conditional

```
<target name="lockDestC1" description="lock the server">
    <condition property="lock_success">
```

```
        <onlineMaintenanceMode
            servletURL="http://localhost:7001/propagation/inventorymanagement"
            username="weblogic"
            password="weblogic"
            allowHttp="true"
            enable="true"
        />
    </condition>
    <antcall target="lock_success" />
</target>

<target name="locksuccess"
        if="lock_success">
            <echo message="Maintenance mode has been toggled."/>
</target>
```

OnlinePingTask

Tests if the propagation management servlet is running on the designated server. This task
verifies that the propagation servlet can be contacted. For information on the propagation servlet,
see "Deploying the Propagation Servlet" on page 8-3.

Attributes

- **allowHttp** – (optional) If set to `true`, allows the propagation management servlet specified
  by `servletURL` to use HTTP. If set to `false`, allows the URL to use HTTPS. Default:
  `false`. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the
  Propagation Servlet" on page 8-3.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the
  task ends in the event of an error. If set to `false`, the task does not terminate. Typically,
  you set this attribute to `false` in cases where you use Ant conditionals to control the flow
  in a series of propagation tasks. Default: `true`.

- **password** – (required) The user's password. The password can be specified either in plain
  text or 3DES encoded form.

- **servletURL** – (required) The URL of the propagation management servlet. This URL must
  point to a specific managed node if in a cluster, not to a proxy. See also "Using Online
  Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **username** – (required) The username of a user with access to the destination server.

Ant Conditional Support

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The propagation management servlet was contacted successfully.

- **failure** – The servlet did not return a successful reply. See also "Troubleshooting Online Tasks" on page 8-9.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if the servlet does not return a successful reply. See also "Troubleshooting Online Tasks" on page 8-9.

Usage

Listing 9-7 tests to see if the propagation servlet is running. Listing 9-8 uses an Ant conditional to print a message if the task succeeds.

Listing 9-7   OnlinePingTask Example

```
<target name="pingDest" description="ping the server">
    <onlinePing
        servletURL="http://localhost:7001/propagation/inventorymanagement"
        username="weblogic"
        password="weblogic"
        allowHttp="true"
    />
</target>
```

Listing 9-8   OnlinePingTask Example with Ant Conditional

```
<target name="pingDest" description="ping the server">
    <condition property="ping_success">
        <onlinePing
            servletURL="http://localhost:7001/propagation/inventorymanagement"
            username="weblogic"
            password="weblogic"
            allowHttp="true"
        />

    </condition>
```

```
    <antcall target="ping_success" />
</target>

<target name="ping_success" if="ping_success">
    <echo message="The server is available." />
</target>
```

OnlineUploadTask

This task uploads an inventory file to a temporary location associated with a running WebLogic Portal application. You must execute this task before you execute the OnlineCommitTask. Only one inventory can be uploaded at a time.

This task overwrites an existing inventory file that exists in the upload location.

Note: This task only moves a file. It does not affect the configuration of the running application. This operation is safe to do even with active users on the system.

Note: The propagation management servlet has a configuration setting to help mitigate "denial of service" attacks. The servlet is configured with a maximum size allowed for uploaded files (files uploaded over HTTP). By default, this is set to 1 MB. If any given file inside the inventory ZIP file is larger than this value, it will be rejected. The simplest way to work around this limit is to physically copy the inventory to the destination server, and then use the readFromServerFileSystem attribute of this task. For information on changing the servlet configuration to allow larger files, see "Increasing the Default Upload File Size" on page 6-35.

Attributes

- **allowHttp** – (optional) If set to true, allows the propagation management servlet specified by servletURL to use HTTP. If set to false, allows the URL to use HTTPS. Default: false. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to true, the task ends in the event of an error. If set to false, the task does not terminate. Typically, you set this attribute to false in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: true.

- **password** – (required) The user's password. The password can be specified either in plain text or 3DES encoded form.

- **readFromServerFileSystem** – (optional) If `true`, the inventory file is read from the destination server file system rather than from the source system. Use this attribute to eliminate the HTTP upload overhead for large files. Default: `false`.

- **servletURL** – (required) The URL of the propagation management servlet. This URL must point to a specific managed node if in a cluster, not to a proxy. See also "Using Online Tasks with HTTPS" on page 8-8 and "Deploying the Propagation Servlet" on page 8-3.

- **sourceFile** – (required) The file to read the inventory file (relative or absolute path), must exist. If `readFromServerFileSystem` is `true`, this file location is for the server file system.

- **username** – (required) The username of a user with access to the destination server.

Ant Condition Property

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The inventory uploaded successfully.

- **failure** – The upload failed. Check the log files for more information.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if the upload fails. Check log files for more information. See also "Troubleshooting Online Tasks" on page 8-9.

Usage

The following example uploads a file called `combined.zip` to the server.

Listing 9-9 OnlineUploadTask Example

```
<target name="upload">
    <onlineUpload
        servletURL="http://localhost:7001/propagation/inventorymanagement"
        username="weblogic"
        password="weblogic"
        allowHttp="true"
        sourceFile="${portal.prop.home}/test/ant/combined.zip"
    />
</target>
```

Offline Tasks

This section describes each of the offline propagation Ant tasks. For an introduction to offline tasks, see . The offline tasks described in this section include:

- OfflineCheckManualElectionsTask

- OfflineCombineTask

- OfflineDiffTask

- OfflineElectionAlgebraTask

- OfflineExtractTask

- OfflineInsertTask

- OfflineListPoliciesTask

- OfflineListScopesTask

- OfflineSearchTask

- OfflineValidateTask

OfflineCheckManualElectionsTask

Tests for the presence of manual elections (changes). The task can check either an inventory file (containing a `changemanifest.xml`) or a `changemanifest.xml` specified directly. You can use this task to stop an automated propagation if any required manual changes are detected.

Attributes

Note: You must specify *either* the **changeManifestFile** *or* the **sourceFile** attribute.

- **changeManifestFile** – An XML file in which to write the differences found between the source and destination inventory files. The change manifest file must not already exist.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **sourceFile** – A valid inventory file.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Condition Property

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – At least one manual election was found.

- **failure** – No manual elections were found.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if at least one manual election is detected. If failOnError is set to false, the task will not fail, but will report a message indicating a manual election was detected. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

The following example combines two inventories. Note that some of the attributes are taken from the OfflineDiffTask.

Listing 9-10 OfflineCombineTask Example

```
<target name="checkManual" description="Checks for manual elections (changes)">
    <offlineCheckManualElections
        changeManifestFile="${portal.prop.home}/test/ant/combine_cm.xml""
        logFile="${portal.prop.home}/test/ant/combine_log.txt"
        verboseLogFile="${portal.prop.home}/test/ant/combine_verboselog.txt"
    />
</target>
```

OfflineCombineTask

Combines two inventories and reports in a change manifest file the differences as a set of adds, updates, and deletes.

Note: This task extracts the portal inventory and attempts to write it to a ZIP file. If the ZIP file created exceeds 4 GB, this task fails and a message is written to the server log and the verbose log. If this occurs, try scoping your inventory to limit the size of the resulting archive file. See "Understanding Scope" on page 6-11 for more information. For information on the verbose log, see "Reviewing Log Files" on page 6-29.

Note: This task extends the OfflineDiffTask task, so all attributes available for OfflineDiffTask can be used with this task too.

Attributes

- **combinedInventoryFile** – (required) The ZIP file in which to write the combined inventory. The file must not already exist.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **sourceFile** – (required) A valid inventory file.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Condition Property

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The inventory combination completed successfully and differences were found.

- **failure** – Something failed during the operation or there were no differences found.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if no differences were found between the two inventories. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

The following example combines two inventories. Note that some of the attributes are taken from the OfflineDiffTask.

Listing 9-11 OfflineCombineTask Example

```
<target name="combine" description="Combine two inventories with logging">
    <offlineCombine
        sourceFile="${portal.prop.home}/test/ant/src.zip"
        destFile="${portal.prop.home}/test/ant/dest.zip"
        combinedInventoryFile="${portal.prop.home}/test/ant/combined.zip"
        changeManifestFile="${portal.prop.home}/test/ant/combine_cm.xml"
        logFile="${portal.prop.home}/test/ant/combine_log.txt"
        verboseLogFile="${portal.prop.home}/test/ant/combine_verboselog.txt"
    />
</target>
```

OfflineDiffTask

Compares two inventories and reports the differences in a change manifest file as a set of adds, updates, and deletes.

Attributes

- **changeManifestFile** – (required) An XML file in which to write the differences found between the source and destination inventory files. The change manifest file must not already exist.

- **destFile** – (required) A valid inventory file. This file contains the inventory of the destination portal application: the application you are propagating to.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **globalAddFlag** – (optional) The flag to indicate the default policy for Adds unless overridden by the policies property file. Default: `true`. See also "Understanding a Policies Property File" on page 8-15.

- **globalDeleteFlag** – (optional) The flag to indicate the default policy for Deletes unless overridden by the policies property file. Default: `true`. See also "Understanding a Policies Property File" on page 8-15.

- **globalUpdateFlag** – (optional) The flag to indicate the default policy for Updates unless overidden by the policies property file. Default: `true`. See also "Understanding a Policies Property File" on page 8-15.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **policiesFile** – (optional) A file with a `.properties` extension that contains the policies to use when comparing or combining inventories. This file must exist if specified. See also "Using Policies" on page 8-14.

- **scopeFile** – (optional) A file with a `.properties` extension that contains the scoping information to use when comparing or combining inventories. This file must exist if specified. See also "Scoping an Inventory" on page 8-10.

- **sourceFile** – (required) A valid inventory file. This file contains the inventory of the source portal application: the application you are propagating from.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Conditional Support

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The diff opertation completed successfully and one or more differences were found.

- **failure** – The diff operation failed or no differences were found.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if no differences are found. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

Listing 9-12 writes the differences between two files to a change manifest file.

Listing 9-12 OfflineDiffTask Example

```
<target name="diff" description="compare two inventories with logging">
    <offlineDiff
        sourceFile="${portal.prop.home}/test/ant/src.zip"
        destFile="${portal.prop.home}/test/ant/dest.zip"
        logFile="${portal.prop.home}/test/ant/diff_log.txt"
        verboseLogFile="${portal.prop.home}/test/ant/diff_verboselog.txt"
    />
</target>
```

OfflineElectionAlgebraTask

Allows for algebraic operations on two change manifest files. This task is not commonly used.

- **Add Operation** – Combines the two specified election lists. For any node that appears in both lists, the entry from election list 1 is used.

- **Subtract Operation** – Removes any entry for a node in election list 1 if there exists an entry for that node in election list 2.

Attributes

- **electionList1File** – (required) The XML file that contains change manifest file 1. Use the OfflineDiffTask to produce a change manifest file, or the OfflineExtractTask to extract a change manifest file from an existing inventory.

- **electionList2File** – (required) The XML file that contains change manifest file 2.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **operation** – (required) The operation to apply; valid values are: `add` or `subtract`.

- **outputFile** – (required) The file in which to write the result elections.

Ant Condition Property

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The algebraic operations completed successfully and one or more differences were found.

- **failure** – The operation failed or no differences were found.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if the algebraic operation failed. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

Listing 9-13 adds the contents of two change manifests. Listing 9-14 subtracts the contents of two change manifests.

Listing 9-13 OfflineElectionAlgebraTask Example: Add

```
<target name="electionsAdd" description="add the two election lists">
    <offlineAlgebra
        electionList1File="${portal.prop.home}/test/ant/elections1.xml"
        electionList2File="${portal.prop.home}/test/ant/elections2.xml"
        operation="add"
        outputFile="${portal.prop.home}/test/ant/addedElections.xml"
    />
</target>
```

Listing 9-14 OfflineElectionAlgebraTask Example: Subtract

```
<target name="electionsSubtract" description="subtract the two election lists">
/>
    <offlineAlgebra
        electionList1File="${portal.prop.home}/test/ant/elections1.xml"
        electionList2File="${portal.prop.home}/test/ant/elections2.xml"
```

```
        operation="subtract"
        outputFile="${portal.prop.home}/test/ant/subtractedElections.xml"
    />

</target>
```

OfflineExtractTask

Extracts top level files from an inventory file. These files are stored in an inventory ZIP file at the top level of the file, and can include a change manifest file, export file, manual changes file, policy file, and scope file.

Attributes

- **changeManifestfile** – (optional) The file in which to write the `changemanifest.xml` file from the inventory (if it exists). This XML file contains the change manifest file that describes all of the changes (adds, deletes, updates) made to the inventory. See also "OfflineDiffTask" on page 9-20.

- **exportFile** – (optional) The file in which to write the `export.properties` file from the inventory. This file contains summary information about the inventory; including who exported it, when it was exported, how many nodes are in the export, and other information.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Default: `true`.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **manualChangesFile** – (optional) The file in which to write the `manualchanges.xml` file from the inventory (if it exists). This file is provided for convenience, and describes the manual changes that are required to propagate the inventory. See also "Make Required Manual Changes" on page 6-5.

- **policyFile** – (optional) The file in which to write the `policy.properties` file from the inventory (if it exists). This file contains the policy rules that were used to produce the inventory. See also "Using Policies" on page 8-14.

- **sourceFile** – (required) A valid inventory file.

- **scopeFile** – (optional) The file in which to write the `scope.properties` file from the inventory (if it exists). This file contains the scoping rules that were used to produce the inventory. See also "Scoping an Inventory" on page 8-10.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Conditional Support

This task does not support the Ant condition task.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if any problems were encountered during the operation. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

The following example extracts specified files from an inventory file.

Listing 9-15 OfflineExtractTask Example

```
<target name="extractCombined" description="gather resources from the combined
inventory">
    <offlineExtract
        sourceFile="${portal.prop.home}/test/ant/combined.zip"
        exportFile="${portal.prop.home}/test/ant/extractCombined_export.properties"
        scopeFile="${portal.prop.home}/test/ant/extractCombined_scope.properties"
        policyFile="${portal.prop.home}/test/ant/extractCombined_policy.properties"
        changemanifestfile="${portal.prop.home}/test/ant/extractCombined_change.xml"
        logFile="${portal.prop.home}/test/ant/extractCombined_log.txt"
        verboseLogFile="${portal.prop.home}/test/ant/extractCombined_verboselog.txt"
    />
</target>
```

OfflineInsertTask

Inserts top level files into an inventory file. These files are stored in an inventory ZIP file at the top level of the file, and can include a change manifest file, export file, manual changes file, policy file, and scope file.

Attributes

- **changeManifestFile** – (optional) The file to insert as the `changemanifest.xml` file from the inventory. This XML file contains the change manifest file that describes all of the changes (adds, deletes, updates) made to the inventory. See also "OfflineDiffTask" on page 9-20.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **outputFile** – (required) The ZIP file in which to write the new inventory. This file must not already exist

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **policyFile** – (optional) The file to insert as the `policy.properties` file from the inventory. See also "Using Policies" on page 8-14.

- **scopeFile** – (optional) The file to insert as the `scope.properties` file into the inventory. See also "Scoping an Inventory" on page 8-10.

- **sourceFile** – (required) A valid inventory file.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Conditional Support

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – If the insert succeeds.

- **failure** – If the operation fails.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if the operation fails. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

The following example inserts specified files into an inventory.

Listing 9-16 OfflineInsertTask Example

```
<target name="insertCombined" description="insert resources into combined inventory">
    <offlineInsert
        sourceFile="${portal.prop.home}/test/ant/combined.zip"
        policyFile="${portal.prop.home}/test/ant/extractCombined_policy.properties"
        changemanifestfile="${portal.prop.home}/test/ant/extractCombined_change.xml"
        outputFile="${portal.prop.home}/test/ant/newCombined.zip"
    />
</target>
```

OfflineListPoliciesTask

Exports the valid policies from an inventory file. These policies are written into a `.properties` text file. You can later edit this property file and use it as an input into other tasks, such as OfflineCombineTask. See also "Using Policies" on page 8-14.

Attributes

- **depth** – (optional) Specifies the number of levels into the inventory tree to examine for policies. This attribute defaults to 3. This is a 0 based number, so a depth of 3 will examine the inventory tree 4 levels deep.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **globalAddFlag** – (optional) The flag to indicate the default policy for Adds. Default: `true`. See also "Understanding a Policies Property File" on page 8-15.

- **globalDeleteFlag** – (optional) The flag to indicate the default policy for Deletes. Default: `true`. See also "Understanding a Policies Property File" on page 8-15.

- **globalUpdateFlag** – (optional) The flag to indicate the default policy for Updates. Default: `true`. See also "Understanding a Policies Property File" on page 8-15.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **policyFile** – (required) The file in which to write the valid policies. Give this file a `.properties` extension. This file must not already exist.

- **sourceFile** – (required) A valid inventory file.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Conditional Support

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The policy file is written successfully.

- **failure** – The operation failed.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if the operation fails. The script can fail if there is a problem walking the inventory tree. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

The following example writes the policy file for the portal application.

Listing 9-17 OfflineListPoliciesTask Example

```
<offlineListPolicies
    sourceFile="${portal.prop.home}/test/ant/src.zip"
    policyFile="${portal.prop.home}/test/ant/listPolicies_policies.properties"
    globalAddFlag="true"
    globalUpdateFlag="false"
    globalDeleteFlag="true"
    logFile="${portal.prop.home}/test/ant/listPolicies_log.txt"
    verboseLogFile="${portal.prop.home}/test/ant/listPolicies_verboselog.txt"
/>
```

OfflineListScopesTask

Exports the valid scoping information from an inventory file. Scopes are written into a `.properties` text file. You can later edit this property file and use it as an input into other tasks, such as OfflineCombineTask. See also "Using Policies" on page 8-14.

Attributes

- **depth** – (optional) Specifies the number of levels into the inventory tree to examine for scopes. This attribute defaults to 3. This is a 0 based number, so a depth of 3 will examine the inventory tree 4 levels deep.

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **scopeFile** – (required) The file in which to write the scope information. Give this file a `.properties` extension. This file must not already exist.

- **sourceFile** – (required) A valid inventory file.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Conditional Support

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The scope file was created successfully.

- **failure** – The operation failed.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if there was a problem traversing the inventory tree. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

The following example exports the scoped artifacts to a property file, using the default depth value of 3.

Listing 9-18 OfflineListScopesTask Example

```
<target name="listScopes" description="lists the scopes found in the source inventory">
    <offlineListScopes
        sourceFile="${portal.prop.home}/test/ant/src.zip"
        scopeFile="${portal.prop.home}/test/ant/listScopes_scopes.properties"
        logFile="${portal.prop.home}/test/ant/listScopes_log.txt"
        verboseLogFile="${portal.prop.home}/test/ant/listScopes_verboselog.txt"
    />
</target>
```

OfflineSearchTask

Searches an inventory for node names that match the specified string. The search string is only matched if it occurs in the node name (the last term in the taxonomy).

For example, consider the following inventory:

1. Application

2. Application:PersonalizationService

3. Application:PersonalizationService:EventService:redEvent.evt

4. Application:PersonalizationService:EventService:coloredEvent.evt

5. Application:PersonalizationService:EventService:blueEvent.evt

If you search this inventory for the string *red*, the task returns nodes 3 and 4. If you search for *PersonalizationService*, the task returns just node 2.

Attributes

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **listFile** – (required) The file in which to write the search results (the nodes that matched the search string). This file must not already exist.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **searchString –** (required) The task matches nodes that contain all or part of this string.

- **sourceFile** – (required) A valid inventory file.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Conditional Support

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

success – One or more matches are found.

failure – No matches are found.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if no matches are found. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

Listing 9-19 searches the specified inventory for the string `global`. Listing 9-20 uses an Ant conditional to print a message if the search succeeds.

Listing 9-19 OfflineSearchTask Example

```
<target name="search">
    <offlineSearch|
        searchString="global"
        sourceFile="${portal.prop.home}/test/ant/src.zip"
        listFile="${portal.prop.home}/test/ant/search.txt"
        logFile="${portal.prop.home}/test/ant/search_log.txt"
        verboseLogFile="${portal.prop.home}/test/ant/search_verboselog.txt"
    />
</target>
```

Listing 9-20 OfflineSearchTask Example with Ant Conditional

```
<target name="searchC1" description="finds the nodes that contain the search string in
the name">
        <condition property="search_success">
            <offlineSearch
                searchString="esktop"
                sourceFile="${portal.prop.home}/test/ant/src.zip"
            />
        </condition>

    <antcall target="search_success" />
</target>
<target name="search_success" if="search_success"><echo message="The search
    succeeded." />
</target>
```

OfflineValidateTask

Verifies that the source ZIP file contains a valid portal application inventory. Use this task after moving or downloading an inventory to ensure that it was transferred successfully. This task ensures that the ZIP file's internal structure adheres to an exported inventory. It does not validate the XML of every node in the inventory tree.

Attributes

- **failOnError** – (optional) Specifies the task behavior if the task fails. If set to `true`, the task ends in the event of an error. If set to `false`, the task does not terminate. Typically, you set this attribute to `false` in cases where you use Ant conditionals to control the flow in a series of propagation tasks. Default: `true`.

- **logFile** – (optional) The name of a file. The task writes log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

- **sourceFile** – (required) A valid inventory file.

- **verboseLogFile** – (optional) The name of a file. The task writes verbose log messages to this file. If specified the file must not already exist. By default, the file is placed in the directory from which the task is run.

Ant Condition Property

This task supports the Ant condition task. The condition task's property is set if the Ant task succeeds.

- **success** – The file is a valid inventory file.

- **failure** – The file is not a valid inventory file.

On Failure

If the `failOnError` attribute is set to `true` (the default), the script terminates if the file is not a valid inventory file. See also "Troubleshooting Offline Tasks" on page 8-10.

Usage

The following example validates the inventory file called `src.zip`.

Listing 9-21 OfflineValidateTask Example

```
<target name="validateSrc" description="valid inventory, with logging">
    <offlineValidate
    sourceFile="${portal.prop.home}/test/ant/src.zip"
    logFile="${portal.prop.home}/test/ant/validateSrc_log.txt"
    verboseLogFile="${portal.prop.home}/test/ant/validateSrc_verboselog.txt"
    />
</target>
```

Propagation Tips and Best Practices

This chapter includes tips and best practices for using the propagation tools. This chapter includes the following sections:

- Best Practices

- Troubleshooting Common Problems

Best Practices

The following sections describe best practices to follow to achieve the most predictable and accurate results with the propagation tools. This section includes these topics:

- Do Not Change Definition Labels and Instance Labels

- Do Not Manually Replicate Changes Between Environments

- Scope to the Enterprise Application Level

- Use Default Scoping and Policy Options

- Use Workshop for WebLogic to Develop a Propagation Process

- Use Ant Tasks for Import and Export

- Avoid Propagating Across a Proxy Server or Load Balancer

- Ensure That the Cluster Administration Server is Running

- Interpreting Error Messages

- Restarting Export and OnlineCommit Operations

- Improving the Speed of the Online Operations

- Using a Microsoft Windows File System

- Choosing the Inventory File Transport Protocol

- Note and Configure the Manual Changes

- Ensure Appropriate Delegated Administration Rights

- Avoid LDAP and Database Synchronization Problems

Consult the Sample Ant Script

A common practice is to automate the propagation processes with a custom Ant script, as explained in Chapter 8, "Using the Propagation Ant Tasks."

Before beginning to write your own Ant script, be sure to review and consult the sample that has been provided. The sample contains a comprehensive set of steps for a propagation. You will likely need a reduced set to perform your propagation. Using the sample Ant script as a start is a good practice.

You may find the sample in the install at this location:

`BEA_HOME/wlserver10.0/portal/bin/propagation/propagation_ant.xml`

Maintain a Store of Historical Inventories

Periodically download and retain a series of historical inventories for your environment. It is highly recommended that you place historical inventories in your source code control system (if size allows) or on a back-up disk drive (for very large inventories). We recommend that you automate a script to periodically take an inventory snapshot of the environment and store that inventory. The same care in maintaining historical versions of your code should be applied to inventories.

Note: Inventory snapshots do not replace the need to perform database and file system backups.

Do Not Change Definition Labels and Instance Labels

When you create new portals and portal resources in Workshop for WebLogic, BEA recommends that you change the definition labels and instance labels at that time to create meaningful names for these resources. After you have used the propagation tools to propagate changes between

environments, it is very important that you do not change these resource labels. The propagation tools use definition labels and instance labels to identify differences between source and destination systems; inconsistent results might occur if you change these labels within Workshop for WebLogic or the Administration Console.

Tip: Definition labels are described in the section "Portlet Properties in the Portlet Properties View" in the *Portlet Development Guide*. Instance labels are described in the section "Portlet Properties in the Portal Properties View" in the same guide.

Do Not Manually Replicate Changes Between Environments

Create portal framework and security assets in only one environment. In other words, do not create an asset in staging and then manually create the same asset in production. If you make identical changes in both the source and target environment, the propagation tools cannot identify them as being the same changes—propagation is carried out as if they were two separate resources. This advice does not apply to datasync and content management assets; however, it is still a best practice to create assets in one environment only.

Scope to the Enterprise Application Level

Choose the highest level of scoping for the propagation to ensure that the destination environment closely mirrors the source environment. If you scope to a lower level, such as web application or desktop, the source and destination systems will inherently be in different states. In this case, additional quality assurance testing may be required on the destination.

Although you can restrict the scope of a propagation, sometimes the propagation tools must implement a higher-level change if a change at that narrower scope depends on a change that occurs at a higher level. For example, if you propagate a desktop that depends on assets in the library, and changes occur for those library assets, the propagation tools can cause changes to other desktops even though you set the scope at a desktop level. For more information on the relationships among Portal resources, see "Scope and Library Inheritance" on page 6-20.

Tip: An exception to this practice is using scope to include or exclude a content management system. A common practice is to propagate only the content management system or to propagate everything but the content management system.

Use Default Scoping and Policy Options

The scoping and policy features offer powerful control over propagation. However, with this power can come some complexity. When first starting out with the propagation tools it is recommended that you retain the default scope (the full application) and the default Policy (accept all Adds, Updates, and Deletes). Once you are comfortable with the concepts and process of propagation, it is appropriate to start using the advanced features. For more information on scope, see "Understanding Scope" on page 6-11. For more information on policies, see "Using Policies" on page 6-22.

Use Workshop for WebLogic to Develop a Propagation Process

Workshop for WebLogic provides many features for defining how assets are to be propagated, including a user interface for visualizing the combined inventory. The Workshop for WebLogic interface is the best choice when propagating an application for the first few times from one environment to another. However, over time, the propagation process generally becomes routine with the same options chosen every time. The Ant tasks are more appropriate for supporting a repeatable process when the propagation scope and policy choices are well understood and held constant. For more information on using Workshop for WebLogic to propagation portals, see Chapter 7, "Using Workshop for WebLogic Propagation Tools." The Ant tasks are discussed in Chapter 8, "Using the Propagation Ant Tasks."

Use Ant Tasks for Import and Export

Workshop for WebLogic allows you to Import and Export inventories from running WebLogic Portal applications. However, the Workshop for WebLogic does not provide access to all of the options available for these operations.

It is important to understand that the Workshop for WebLogic is actually using the Ant tasks when invoking Import and Export. An Ant script is created and run whenever these features are used from Workshop for WebLogic. You can save the Ant script that is used by Workshop for WebLogic. By saving the script, you can examine it and begin to understand how it works. The OnlineDownload, OnlineUpload, and OnlineCommit tasks have numerous options that can be used when using the Ant tasks directly.

For information on saving propagation Ant scripts, refer to Workshop for WebLogic online help. For information on specific Ant tasks, see Chapter 9, "Propagation Ant Task Reference."

Avoid Propagating Across a Proxy Server or Load Balancer

The Export Propagation Inventory to Server and Import Propagation Inventory From Server operations in Workshop for WebLogic, and the OnlineDownload, OnlineUpload, and OnlineCommit Ant tasks open an HTTP connection to a servlet designed to assist with the propagation. The servlet runs in the WebLogic Portal application running on WebLogic Server. The URL specified for these commands indicate how the servlet is addressed.

For WebLogic Portal applications deployed in a cluster, a proxy server or load balancer sits in front of the cluster to marshal traffic to the cluster nodes. While it is possible to target a propagation to the proxy, a better approach is to identify a single cluster managed node and to address it directly. This approach has these advantages:

- The `readFromServerFilesystem` option for the OnlineUpload task can work reliably without a shared file system. See "OnlineUploadTask" on page 9-15.

- Propagation verbose log files are stored only on the managed server that serviced the request. They will be easier to locate if the managed server is known. For information, see "Verbose Log File Location" on page 6-40.

- Propagation operations can take minutes to complete; the more network devices between the client and servlet increases the chances that communication will be disrupted.

Ensure That the Cluster Administration Server is Running

The propagation tools primarily commit configuration changes into the WebLogic Portal application's database schema. Portal desktops, portlet preferences, content, and most other artifacts are persisted into the database. However, some security information such as entitlements and delegated administration policies are persisted into the embedded LDAP server intrinsic to every WebLogic Server instance.

In a cluster, the Administration Server is responsible for distributing updates to the embedded LDAP server to all nodes in the cluster. If the Administration Server is down, the nodes in the cluster can get out of sync if updates are made to entitlements or delegated administration. Therefore, it is strongly recommended to have the Administration Server running when the Propagation Tool is updating an application. This advice applies specifically to uploading a final inventory file to the server using Workshop for WebLogic, or a call to the OnlineCommit Ant task. See also "Uploading the Final Inventory to the Server" on page 7-19 and "OnlineCommitTask" on page 9-4.

Interpreting Error Messages

Unlike any other activity in a WebLogic Portal application, the propagation tools iterate through the configuration of the entire WebLogic Portal application. Therefore, if there is a latent problem in the application configuration, the propagation tools are apt to encounter it. While these problems adversely affect the propagation, the true source of the error is usually at a deeper level.

To troubleshoot such a problem, attempt test the area in question using another means such as the WebLogic Portal Administration Console. This process can help you to discover and fix the actual problem or provide a simpler, reproducible test case.

Some cases have arisen where an unsupported database driver or corrupt security repository have negatively impacted the propagation tools. While these problems affect the propagation, the solution lies in addressing the root cause.

Restarting Export and OnlineCommit Operations

Note that during an Export Propagation Inventory to Server operation (Workshop for WebLogic) or OnlineCommit (Ant) operation, the propagation tools processes a list of changes to make on the destination system. At times, this list can be enormous, with thousands of necessary changes. This operation can take a significant amount of time. The propagation tools do not create an encompassing transaction for all changes for several reasons:

- Performance concerns as the transaction grows larger and larger

- Concerns for transaction deadlock in the database

- The embedded LDAP repository is not transaction aware

During propagation, each change to the destination system is executed in its own transaction. If the propagation operation is interrupted in the middle, the configuration will be left in an incomplete state. For example, this situation can occur if the managed server hardware fails during the operation. In such a case, you can simply rerun the propagation tools. The tools will detect the new, shorter, list of remaining changes that need to be applied and will start again where the previous propagation failed.

Improving the Speed of the Online Operations

The Export Propagation Inventory to Server and Import Propagation Inventory From Server operations in Workshop for WebLogic, and the OnlineDownload, OnlineUpload, and OnlineCommit Ant tasks are considered the *online* operations because they communicate with the propagation servlet running in the WebLogic Portal application. These operations read and

write large amounts of database records while working with the WebLogic Portal application configuration. These operations are notable in that they tend to take the longest amounts of time to execute.

An effective way to improve the speed of these operations is to improve the performance of the database. Allocating more powerful hardware is an effective but expensive solution. Often, having a database administrator tune the database is effective.

Using a Microsoft Windows File System

The Microsoft Windows file system is more restrictive than those of other operating systems. Specifically, there are file path length limitations that are problematic when working with deeply nested folder structures.

The propagation tools make use of the file system extensively when doing certain operations. The tools use a designated working folder to write large numbers of files for the artifacts that they are exporting, importing, differencing, and committing. This ordinarily is not a problem because the propagation tools use the working folder judiciously and do not create deeply nested folders. However, if the working folder itself is set to be a deeply nested folder, the propagation tools may fail while trying to create, read, delete or update some of the files because the file path may exceed the limit.

By default, the propagation servlet will use the temporary folder provided by the WebLogic Server servlet container as the working folder. This path is a nested folder under the domain directory, like:

```
C:\bea100\user_projects\domains\wlpHome_domain\servers\AdminServer\
tmp\_WL_user\wlpBEA\c0b5ju\public
```

This path can grow very long, especially if the domain folder path is lengthy. In some cases, this will cause the propagation tools to fail because the file path exceeds the limit.

A best practice for Windows systems is to not allow the propagation servlet to use the servlet container temporary space for its working directory. Give the working folder a small folder path length, like `C:\bea100\propagation`. You can set the working folder in `web.xml` for the Propagation Tool web application, as described in "Configuring the Propagation Servlet" on page 6-38.

Choosing the Inventory File Transport Protocol

The Export Propagation Inventory to Server and Import Propagation Inventory From Server or OnlineDownload/OnlineUpload operations retrieve or push inventory files to a remote server that

is hosting a WebLogic Portal application. Because these operations use a servlet, this transfer uses HTTP. When firewalls are an issue, this can be a convenient approach to move the inventory files. It is also the only approach supported by the Export Propagation Inventory to Server and Import Propagation Inventory From Server operations in the Workshop for WebLogic.

To ensure that the Export Propagation Inventory to Server/OnlineUpload operation succeeds, it is sometimes necessary to increase the allowed upload size of the servlet. Because upload capabilities have historically been used in Denial of Service attacks on web applications, the default limit for any uploaded inventory is one megabyte. This can be increased in web.xml by following the instructions in "Configuring the Propagation Servlet" on page 6-38.

Additionally, the user must be granted upload rights in the WebLogic Server console. Only users in the Administrator and Deployer roles are granted this right by default. Other users can be given this right in the WebLogic Server Console. For more information see the WebLogic Server document "Resource Types You Can Secure with Policies."

If you want, you can use other approaches to move files to and from the destination, such as FTP, a source code control client, or a physical medium such as a DVD. In these cases, the OnlineDownload and OnlineUpload tasks support an option to avoid using HTTP for the file transfer. They allow the administrator to position the inventory file on the server's local file system, and reference the file location in the Ant task.

For example:

```
<target name="downloadProductionInventory">
    <onlineDownload
        servletURL="http://myhost/propagation/inventorymanagement"
        username="weblogic" password="weblogic" allowHttp="true"
        outputInventoryFile="/opt/inventories/prod.zip"
        outputToServerFileSystem="true"
    />
</target>
```

In this example, inventory file is not downloaded to the machine running the Ant task, rather, the file is written to the file system of the server machine, at location /opt/inventories/prod.zip.

The OnlineUpload task has a similar flag, called *readFromServerFileSystem*. For more information on these Ant tasks, see Chapter 9, "Propagation Ant Task Reference."

Note and Configure the Manual Changes

The propagation tools can commit nearly all of the changes to configuration artifacts that it detects. However, there are a few types of changes that the propagation tools can detect, but cannot automatically commit. These types of changes are called Manual Changes. The administrator must manually make these changes using the WebLogic Portal Administration Console or the WebLogic Server Console.

The following is a list of changes that are detected, but always manual:

- Global roles

- BEA content repository definitions

- WSRP Producer registrations (if both consumer and producer applications are at WebLogic Portal 10.0, then no manual steps are required)

- Any out of scope dependency

To view the required Manual Changes in the Workshop for WebLogic, right-click on the Merged Inventory view and filter for **Manual Changes Only**. When using the Ant tasks, the OfflineExtractTask produces a file that lists the Manual Changes.

Tip: You can use the OfflineCheckManualElectionsTask to halt an automated propagation if any manual changes are detected. For information on this task, see "OfflineCheckManualElectionsTask" on page 9-17.

Ensure Appropriate Delegated Administration Rights

You must be a member of the PortalSystemAdministrators role to run any online propagation tool tasks. These tasks include the online Ant tasks and the Export Propagation Inventory to Server and Import Propagation Inventory From Server operations in Workshop for WebLogic. However, unless you are also a member of the Administrators role, you will not be able to view or update any configuration artifacts in the WebLogic Portal application by default. You must be granted delegated administration rights to the artifacts you wish to propagate.

In most cases, it is recommended that you be a member of the Administrators role when you propagate. This allows you to propagate any artifact in the application and is the safest approach to propagation because the propagation tools have full visibility into the application configuration.

Note, however, that use cases exist in which the propagation is targeted to a specific area of the application and the previous advice does not apply. For example, a content administrator may want to propagate a single folder of content items. This user will likely not have delegated administration rights to any Portal Framework assets, but that is not a problem in this use case.

In such cases it is acceptable for you to perform a propagation without full delegated administration rights to the application. It is important to understand that this may cause errors in cases where the you are attempting to add items that exist in the destination, but are not visible to you because of the absence of delegated administration rights. You will see errors on the console about the conflicted artifacts failing to add because they already exist.

For more information on delegated administration, see the *WebLogic Portal Security Guide*.

Avoid LDAP and Database Synchronization Problems

If the embedded LDAP repository and database become out of sync, it is possible that some policy data will be lost during propagation. To prevent this situation from occurring, the propagation tools automatically detect when LDAP and the database are out of sync and, by default, prevent download and commit operations. See "OnlineCommitTask" on page 9-4 and "OnlineDownloadTask" on page 9-8 for more information.

Possible reasons for these two stores to become out of sync are:

- Resetting the database tables or changing to a new database schema without also resetting the embedded LDAP store.

- Resetting the embedded LDAP store without also resetting the database tables.

- Updating delegated administration or entitlement data via the WebLogic Portal Administration Console or propagation tools without a running the Administration Server.

To avoid the synchronization problem:

- Understand and honor the coupling between the embedded LDAP store and database tables; do not reset one without the other.

- Ensure the cluster Administration Server is running while doing propagations or manually administer delegated administration and entitlements.

Troubleshooting Common Problems

This section contains a list of common problems that you might encounter while using the propagation tools. None of these issues represent product problems; typically they result from general misuse or misunderstanding of the propagation tools.

This section includes these topics:

- Propagation fails trying to add an artifact because the artifact already exists

- Propagation OnlineUpload task will not work if the user is not given Upload permission in WLS Console

- Cannot propagate between applications that do not have identical J2EE structures

- Propagation download fails when propagating a content repository; could not read InputStream for a binary property.

Table 10-1 Propagation fails trying to add an artifact because the artifact already exists

Symptom: Errors that appear on the console indicate that there are problems adding an artifact in some area of the inventory (Content, Portal, Security, etc.)
Cause: The user running the propagation has more delegated administration rights to the source application than the destination application. Because of this, the source inventory contains more artifacts than the destination inventory. The propagation tools therefore believe that those artifacts need to be added to the destination. Upon doing so, the Add operation fails because that artifact already exists in the destination.
Solution: Works as designed. In general, it is best to propagate with a user that has full delegated administration rights to an application. See "Ensure Appropriate Delegated Administration Rights" on page 10-9.

Table 10-2 Propagation OnlineUpload task will not work if the user is not given Upload permission in WLS Console

Symptom: Errors will appear on the console indicating that there are problems reading the uploaded inventory.

```
<Error> <InventoryServices> <000000> <FAILURE: There was no uploaded
inventory in the request.>
```

```
<Error> <InventoryServices> <000000> <The files in the upload directory
could not be read.>
```

Cause: WebLogic Server restricts the right to uploading files to a server to the Administrator and Deployer roles. This is a security mechanism.

Solution: Works as designed. The user running the propagation must be granted sufficient rights, such as the Administrator or Deployer roles, in the WebLogic Administration Console. See the WebLogic Server document "Resource Types You Can Secure with Policies."

Table 10-3 Cannot propagate between applications that do not have identical J2EE structures

Symptom: Errors will appear on the console indicating nodes cannot be added because their parent does not exist. The parent is usually a web application name, or a resource that comes from files within the web application, like .portlet, .shell, .laf files.

Cause: The propagation tools assume that the .EAR file that was used on the source system will be moved to the destination system and the core J2EE structure will not be modified by deployment plans. This includes maintaining the same name for the application, and context paths for all web applications.

Solution: Works as designed.

Table 10-4 Propagation download fails when propagating a content repository; could not read InputStream for a binary property.

Symptom: Errors will appear on the console indicating that there are problems reading or writing InputStreams for content items.

```
<Error> <ContentManagement> <000000>
<com.bea.content.RepositoryException: java.io.IOException: Stream closed
    at com.bea.content.internal.server.dao.

    DaoBase.doDatabaseAction(DaoBase.java:248)
    at com.bea.content.internal.server.dao.

    DaoBase.doExecuteActionAndCloseConnection(DaoBase.java:204)
```

Cause: This has only been seen by Oracle database users using an unsupported Oracle 9i Thin database driver.

Solution: Works as designed. Use a supported database driver, such as 10G Thin. See the following WebLogic Server document "Supported Database Configurations."

Using the Export/Import Utility

The Export/Import Utility allows a full round-trip development life cycle, where you can easily move portals between a Workshop for WebLogic environment and a staging or production environment, as shown in Figure 11-1.

This chapter explains how to use the Export/Import Utility. The chapter includes background information on the utility and its purpose. In addition, detailed examples are provided that illustrate common use cases.

This chapter includes the following topics:

- Installing the Export/Import Utility

- Overview of the Export/Import Utility

- Basic Concepts and Terminology

- The Export/Import Utility Client Program

- Configuring the Export/Import Utility Properties File

- Exporting a Desktop

- Importing a .portal File

- Exporting a Page

- Importing a Page

- Controlling How Portal Assets are Merged When Imported

- Controlling How Portal Assets are Moved When Imported

- Locating and Specifying Identifier Properties

- Managing the Cache

Installing the Export/Import Utility

You only need to perform the following procedure if you intend to run the Export/Import Utility as a stand-alone application. If you only want to run the Workshop for WebLogic propagation tools, then the following procedure is unnecessary.

1. Before installing the Export/Import Utility, be sure you have Ant 1.6.5 in your `PATH` environment variable. Ant is part of the normal WebLogic Server installation. It is located in:

   `BEA_HOME/modules/org.apache.ant_1.6.5/bin`

2. Stop WebLogic Server if it is running.

3. Open the file `BEA_HOME/wlserver_10.0/portal/bin/xip/build.xml`, and edit the following properties in the **Installer** section to point to the appropriate locations:

Property	Description
`bea.dir`	Points to `BEA_HOME/wlserver_10.0`.
`wlp.lib.dir`	Points to `${bea.dir}/portal/lib`.

4. Using a utility such as WinZip, open the following WAR file:

   `BEA_HOME/wlserver_10.0/portal/lib/modules/wlp-propagation-web-lib.war`

5. Extract the file `propagation.jar` from the WAR file and save it in `BEA_HOME/wlserver_10.0/portal/lib`.

   Tip: If you place the `propagation.jar` file in `BEA_HOME/wlserver_10.0/portal/lib`, you do not need to add it to the `wlp.classpath` in the `build.xml file`. If it is not in this directory, you must add it to the `wlp.classpath` in `build.xml`.

6. Create a directory called `ejb` under `${wlp.lib.dir}/netuix`, and extract `netuix.jar` from the following EAR and place it in that directory:

```
BEA_HOME/wlserver_10.0/portal/lib/modules/wlp-framework-full-app-lib.ear
```

Note: The file `${wlp.lib.dir}/netuix/ejb/netuix.jar` is referenced in the `wlp.classpath` in `build.xml`.

7. Build the Export/Import Utility. To do this, run the following command from within the `BEA_HOME/wlserver_10.0/portal/bin/xip` directory:

```
ant
```

Overview of the Export/Import Utility

The Export/Import Utility allows a full round-trip development life cycle, where you can easily move portals between a Workshop for WebLogic environment and a staging or production environment, as shown in Figure 11-1.

Figure 11-1 Export/Import Utility Allows Round-Trip Development
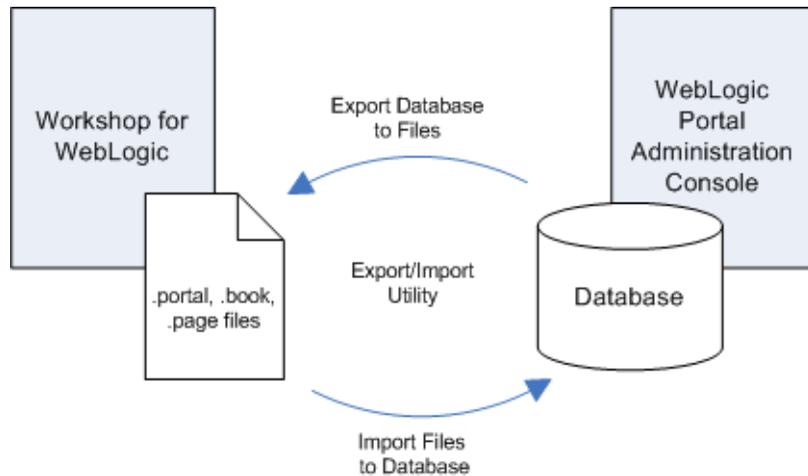
This utility lets you import `.portal`, `.pinc`, and other portal framework files into the database, and lets you export these files from the database. The exported files can be loaded back into Workshop for WebLogic, or imported into another WebLogic Portal database.

The utility performs its work in a single database transaction. If the utility fails for some reason, the database is not affected.

What the Utility Moves

The Export/Import Utility moves desktops, portlet references, books, pages, and localization definitions. In other words, the utility exports `.portal`, `.pinc`, and other portal framework files from a database, and imports the contents of those files back into a database.

Tip: For detailed information on the portal framework, see the *Portal Development Guide* .

Note: The actual definitions for portlets, look and feels, shells, menus, layouts, themes, JSPs, and other code are contained in the EAR file. These files are stored in directories in portal web applications, such as the `framework/markup` directory. If any of these file-based elements change, you must rebuild and redeploy the EAR. The `.portal` and other portal framework files simply refer to the definition files.

What the Utility Does Not Move

The Export/Import Utility does not handle the following items: campaigns, behavior tracking events, content management assets, entitlements, WSRP producer registration, portlet categories, localization resources, user profiles, and commerce data.

Refining Rules for Exporting and Importing

The Export/Import Utility allows you to select an object (desktop, book or page) at any level (library, admin, visitor) and import it or export it, according to specified rules.

To refine and customize the export and import of `.portal`, `.pinc`, and other portal framework files to and from the database, you can:

- Specify rules to determine how portal elements are merged. For instance, in a manner similar to that of a source code control mechanism, changes in a `.portal` file can be merged with changes in the database.

- Specify scoping rules. Scoping rules determine how new books and pages will be merged into the new environment. Note that user and administrator customizations are preserved when assets are merged.

As shown in Figure 11-2, the Export/Import Utility offers flexibility with respect to importing, exporting, and scoping. You can scope changes to the *library*, *admin* (desktop), or *visitor* (individual user) level. For instance, if you import a desktop at the *admin* scope, the imported changes will be applied only to the specified desktop. If a user has customized that particular desktop, then the changes will also be inherited by the user desktop. Note, however, that changes

are never inherited up the hierarchy. Elements in the library will not inherit changes made to a desktop.

Tip: For a more in depth discussion of the relationship between the library, desktops, and user views, see "Scope and Library Inheritance" on page 6-20.

Figure 11-2 Import, Export, and Scoping Options Offered by the Export/Import Utility
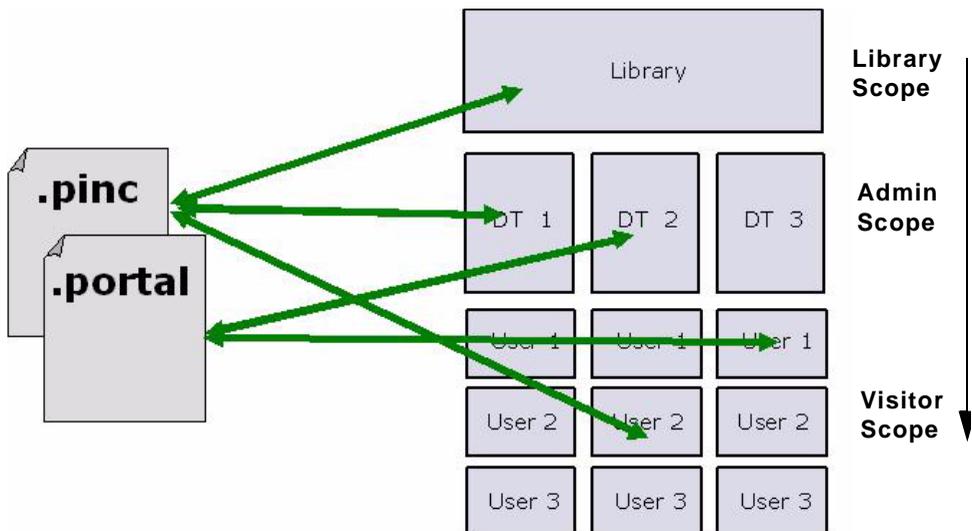
Basic Concepts and Terminology

Before you use the Export/Import Utility, it is important to understand some basic portal concepts and terms. If you review this section, the explanations and examples provided in the rest of this chapter will be more clear.

The concepts and terms described in this section include:

- .portal Files Versus Desktops

- Export and Import Scope

- Customization

.portal Files Versus Desktops

The `.portal` file that you create in Workshop for WebLogic is a fully functioning portal. However, it can also be used as a template to create a *desktop*. In this template you create books, pages and references to portlets. When you view the `.portal` file with your browser, the portal is rendered in "single file mode," meaning that you are viewing the portal from your filesystem as opposed to a database; the `.portal` file's XML is parsed and the rendered portal is returned to the browser. The creation and use of a `.portal` is intended for development purposes and for static portals (portals that are not customized by the end user or administrator). Because there is no database involved you cannot take advantage of functionality such as user customization or entitlements.

Once you have created a `.portal` file you can use it as a template to create desktops for a staging or production environment. A desktop is a particular view of a portal that visitors access. When you create a desktop based on the `.portal` file in the WebLogic Portal Administration Console, a desktop and its books and pages are placed into the database. The desktop, books, and pages reference shells, menus, look and feels, and portlets. The settings in the `.portal` file template, such as the look and feel, serve as defaults to the desktop. Once a new desktop is created from a `.portal` template, the desktop is decoupled from the template, and modifications to the `.portal` file do not affect the desktop, and vice versa. For example, when you change a desktop's look and feel in the WebLogic Portal Administration Console, the change is made only to the desktop, not to the original `.portal` file. When you view a desktop with a browser it is rendered in "streaming mode" (from the database). Now that a database is involved, desktop customizations can be saved and delegated administration policies and entitlements can be set on portal resources.

Export and Import Scope

Exports and imports can be scoped to the following levels:

- Library Scope
- Admin Scope
- Visitor Scope

The first two levels correspond to the Library and Portals nodes in the Portal Resources tree of the WebLogic Portal Administration Console, as shown in Figure 11-3. The visitor level includes changes made by users to individual desktops using the Visitor Tools.

Figure 11-3 Portal Resources Tree of the WebLogic Portal Administration Console
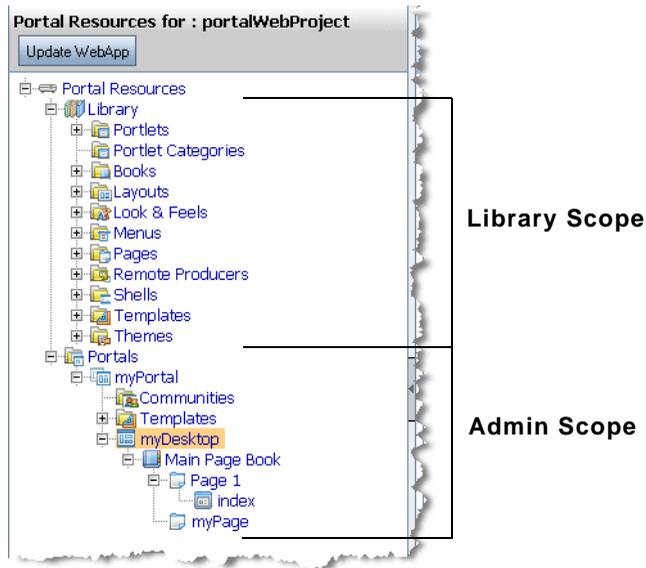
The relationship between library, admin, and visitor views is hierarchical, and properties from objects up the hierarchy can be inherited by objects down the hierarchy. For example, a change to a library element is inherited by desktops that use that element and by visitor views that use those desktops.

Figure 11-4 shows the hierarchy in which library, admin, and visitor instances are organized, the direction in which changes are inherited, and the tools that are typically used to make modifications at the different levels in the hierarchy.

Figure 11-4 Scoping Hierarchy
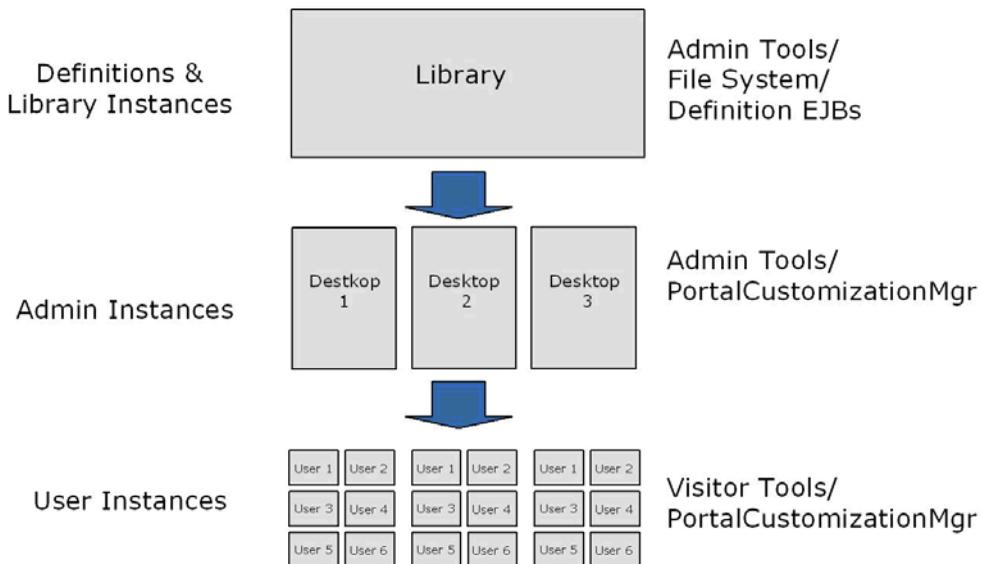
Library Scope

An object (book, page, or portlet) can exist in the library and not be referenced by any desktop. A developer or administrator may create a page or book in the library only, outside the context of a desktop. This page or book can then be placed on a desktop at a later point in time. Changes made to objects in the library cascade down through all desktops that reference those objects. For more information on library inheritance, see "Scope and Library Inheritance" on page 6-20.

Admin Scope

The admin scope represents the "default desktop." This is where an administrator creates and modifies individual desktops. Changes made by an administrator to a desktop may use elements from the library, but desktop changes are never reflected back up into the library. Furthermore, changes made to individual desktops do not affect other desktops. However, changes made to desktops are cascaded down to the visitor views. See also "Scope and Library Inheritance" on page 6-20.

Visitor Scope

Visitors (users who access desktops) may be permitted to customize their desktop. Changes made to a visitor's desktop are restricted to that visitor's view. The changes do not show up in either

the admin-level desktop, which the visitor view references, or in the library level. The changes also do not show up in other visitor's views.

Customization

Customization refers to modifying a portal through an API. This API is typically called from the WebLogic Portal Administration Console and Visitor Tools, but it is also exposed to developers who wish to modify desktops. The API provides all the create, read, update, and delete (CRUD) operations needed to modify a desktop and all of its components (for example, portlets, books, pages, and menus).

Note: Customization and personalization are two distinctly different features. With customization, someone is making a conscious decision to change the makeup of a desktop. With personalization, desktops are modified based on rules and user behavior.

The Export/Import Utility Client Program

The Export/Import Utility client consists of a simple command-driven Java program. This program reads a local properties file, which you must manually configure, as explained in . After parameters are read from the properties file, the Java program calls an API that executes the appropriate actions on the server. For instance, if you wish to import a page and scope the change to the *admin* level, you must specify this in the properties file.

Tip: The Java program's source code is freely available in the installation directory of the Export/Import Utility. You are free to use this source code to develop your own client interface.

Configuring the Export/Import Utility Properties File

As previously mentioned, the Export/Import Utility is a Java program that reads a properties file. To use the utility, you must manually configure this properties file.

Specifying Parameters in the Properties File

In the properties file you can specify such things as server configuration information, export import commands, objects, scoping rules, and propagation rules. The default properties file is fully commented to help guide you in editing it. In addition, the examples discussed later in this

chapter, such as "Exporting a Desktop" on page 11-11, contain detailed information specific properties.

An excerpt from the default `xip.properties` file is shown in Figure 11-5. You can find this file in `WEBLOGIC_HOME/portal/bin/xip`.

Figure 11-5 Excerpt from Default Properties File

```
# Export/Import Properties file. The properties in this file are read by the Xip
# (pronounced zip) utility. You may specify an alternate properties file via the
# -properties command line argument.
#
# Server confifiguration information
#
xip.config.url=t3://localhost:7003
xip.config.username=weblogic
xip.config.password=weblogic
xip.config.application=myEnterpriseApp
#
# command - Are we exporting or importing. Valid values are: "export", "import"
#
#xip.command=export
xip.command=import
#
# object - The "thing" you want to export/import (desktop, book, page)
xip.object=desktop
#xip.object=book
#xip.object=page
#
# Identifier properties, tells the import export utility how to identify the
#
```

Tip: The complete default properties file is shown in Appendix A, "Export/Import Utility Files."

Specifying the Properties File Location

By default, the properties file that is used to configure the Export/Import Utility is called `xip.properties`. It is located in the installation directory of the Export/Import Utility (`WEBLOGIC_HOME/portal/bin/xip`). You can specify an alternate properties file by editing the Ant build file `build.xml`, which is also included in the installation directory of the utility. Listing 11-1 shows the Ant target definition to edit:

Listing 11-1   Modifying the Location of the Properties File in the Ant Script

```
<target name="run" depends="jar" description="Run the Xip utility">
    <java classname="com.bea.wlp.xip.Xip" fork="true" failonerror="true">
        <arg value="-verbose"/>
        <arg value="-properties=my.properties"/>
        <classpath>
            <pathelement path="${wls.classpath}"/>
            <pathelement path="${wlp.classpath}"/>
            <pathelement path="${jarfile}"/>
        </classpath>
    </java>
</target>
```

Exporting a Desktop

This section explains how to export a desktop using the Export/Import Utility. Exporting a desktop means retrieving the attributes of a desktop from the database and restoring them in a `.portal` file. The exported `.portal` file can then be loaded into Workshop for WebLogic for further development.

The basic steps for exporting a desktop include editing the `xip.properties` file and running an Ant build script.

These steps are explained in the following sections:

1. Editing the Properties File

2. Running the Build Script

Editing the Properties File

To export an existing desktop as a `.portal` file you need to specify attributes in the `xip.properties` file. This section highlights the required changes.

`xip.config.application=portalProject`

You must specify the name of Enterprise application from which you are exporting. In Workshop for WebLogic, this value corresponds to the Portal EAR Project name, as shown in the following figure:

Figure 11-6 Application Name Shown in the WebLogic Portal Administration Console


```
xip.command=export
```
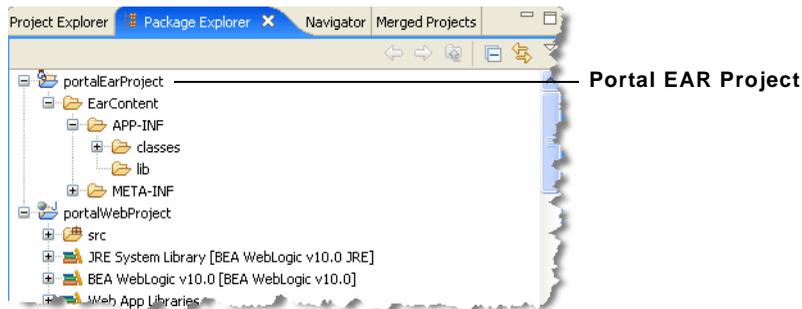
Specify that you wish to export, rather than import.

```
xip.object=desktop
```

Specify the object you wish to export—in this example, a desktop.

```
xip.identifier.webapp=PortalWebApp_1
xip.identifier.portal.path=yourPortalPath
xip.identifier.desktop.path=yourDesktopPath
```

Note: These lines identify the objects you wish to export. For information on where to find the values for these properties, if you do not know what they are, see "Locating and Specifying Identifier Properties" on page 11-31 for more information.

The web **webapp** property must always be specified.

If you scope the export to the *admin* or *visitor* level, then you must specify the `portal.path` and `desktop.path` properties.

```
xip.output.file=myportal.portal
```

Specify where you wish to save the result (the exported .portal file). In this example, the resulting file is called `myportal.portal`, and it is placed in a location relative to where the utility was run.

Note: You do not need to specify the encoding as this information is in the database.

```
xip.export.context.scope=admin
```

Specify the scope of the export. In this example, the **admin** scope is specified. For more information on scope, see "Export and Import Scope" on page 11-6.

```
xip.export.context.locale.language=en
```

Specify the locale of the exported desktop (in this example, English). Only one locale can be exported or imported at a time.

Running the Build Script

Once the property attributes are defined, you can run the Ant build script, as shown in Listing 11-2. The build script's task writes status information, also shown below, to the console window.

Listing 11-2   Running the Ant Build Script

```
C:\dev\xip>ant run
Buildfile: build.xml
init:
compile:
jar:
run:

[java] Using: Properties from file [xip.properties]
[java]      Name [xip.config.url] Value [t3://localhost:7003]
[java]      Name [xip.config.username] Value [weblogic]
[java]      Name [xip.config.password] Value [********]
[java]      Name [xip.config.application] Value [portalProject]
[java]      Name [xip.command] Value [export]
[java]      Name [xip.object] Value [desktop]
[java]      Name [xip.identifier.webapp] Value [PortalWebApp_1]
[java]      Name [xip.identifier.portal.path] Value [yourPortalPath]
[java]      Name [xip.identifier.desktop.path] Value [yourPortalDesktop]
[java]      Name [xip.identifier.book.label] Value [mainBook]
[java]      Name [xip.identifier.page.label] Value []
[java]      Name [xip.input.file] Value [yourPortal.portal]
[java]      Name [xip.output.file] Value [myportal.portal]
[java]      Name [xip.import.context.deletes] Value [false]
[java]      Name [xip.import.context.moves] Value [false]
[java]      Name [xip.import.context.outermoves] Value [false]
[java]      Name [xip.import.context.updates] Value [true]
[java]      Name [xip.import.context.abort.on.collisions] Value [null]
[java]      Name [xip.import.context.abort.if.portlets.missing] Value [false]
[java]      Name [xip.import.context.scope] Value [admin]
[java]      Name [xip.import.context.modify.definitions] Value [false]
[java]      Name [xip.import.context.propagate.changes] Value [sync]
[java]      Name [xip.import.context.create.portal] Value [true]
[java]      Name [xip.import.context.portal.title] Value [My Portal]
[java]      Name [xip.import.context.locale.language] Value [en]
[java]      Name [xip.import.context.locale.country] Value []
```

```
[java]      Name [xip.import.context.locale.variant] Value []
[java]      Name [xip.export.context.scope] Value [admin]
[java]      Name [xip.export.context.locale.language] Value [en]
[java]      Name [xip.export.context.locale.country] Value []
[java]      Name [xip.export.context.locale.variant] Value []


[java] Executing command: export
[java] Exporting desktop [Webapp: [PortalWebApp_1] PortalPath: [yourPortalPath]
DesktopPath: [yourDesktopPath]] to file [myportal.portal]
[java] Connection to host: t3://localhost:7003
[java] Saving changes to: myportal.portal
[java] Done. time taken 6 sec.

BUILD SUCCESSFUL

Total time: 8 seconds

C:\dev\xip>
```

The file `myportal.portal` can now be reloaded into Workshop for WebLogic or imported into another staging or production (database) environment.

Importing a .portal File

This section explains how to import a `.portal` file into a desktop using the Export/Import Utility. When importing a `.portal` file into a desktop, there are two possible cases: the desktop already exists in the database or it does not. If the desktop does not already exist in the database, then the Export/Import Utility automatically creates it. If, however, the desktop does already exist in the database, you need to specify merging options.

The basic steps for importing a `.portal` file include editing the `xip.properties` file and running an Ant build script.

These steps are explained in the following sections:

1. Editing the Properties File

2. Running the Build Script

Editing the Properties File

To import an existing `.portal` file into a desktop, you need to specify attributes in the `xip.properties` file. This section highlights the required changes.

xip.config.application=portalProject

> You must specify the name of Enterprise application you are importing to. In Workshop for WebLogic, this value corresponds to the Portal EAR Project name, as shown in the following figure:

Figure 11-7 Application Name
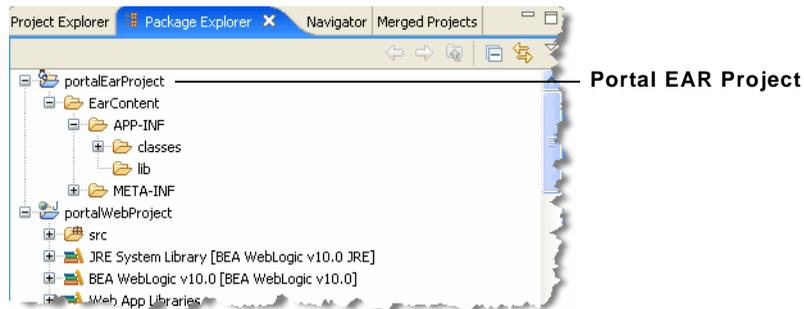
xip.command=import

> Specify that you wish to import, rather than export.

xip.object=desktop

> The object you are interested in importing – in this example a desktop.

xip.identifier.webapp=PortalWebApp_1
xip.identifier.portal.path=yourPortalPath
xip.identifier.desktop.path=yourDesktopPath

Note: These lines identify the objects you wish to export. For information on where to find the values for these properties, if you do not know what they are, see "Locating and Specifying Identifier Properties" on page 11-31.

> The web `webapp` property must always be specified.

> If you scope the export to the *admin* or *visitor* level, then you must specify the `portal.path` and `desktop.path` properties.

xip.input.file=myportal.portal

> Specify the `.portal` file that you wish to import from. The utility looks for the file in a location relative to where the utility is installed.

Note: You do not need to specify an encoding as that is defined in the `.portal` file itself.

xip.import.context.scope=admin

Specify the scope of the import. In this example, you wish to import a desktop at the **admin** level. For detailed information on scope, see "Export and Import Scope" on page 11-6.

xip.import.context.deletes=false

You must specify how to handle *deletes*. See "Controlling How Portal Assets are Merged When Imported" on page 11-27 for detailed information.

xip.import.context.moves=false
xip.import.context.outermoves=true

You must specify how to handle *moves*. See "Controlling How Portal Assets are Moved When Imported" on page 11-29 for detailed information.

xip.import.context.updates=true

If **xip.import.context.updates** is **true** then new portlets, books, and pages that do not exist in the desktop but exist in the .portal will be added to the new desktop. Also instance level resources will be updated.

xip.import.context.abort.if.portlets.missing=true

If **xip.import.context.abort.if.portlets.missing** is **true** then if a portlet is referenced in the .portal file but does not exist in the web application then the import is halted. If this flag is **false** a warning is logged and the import continues.

xip.import.context.modify.definitions=true

If **xip.import.context.modify.definitions** is **true**, when updating books and pages the utility also updates the library definition. A library definition consists of markup, which includes backing files, activate, deactivate, and rollover images. Also titles and descriptions for books and pages, and the is_hidden flag are stored in these definitions.

Note: Be aware that if you modify a library definition, desktop instances that share that library definition could be affected. Therefore, if you scope to a particular desktop, you may inadvertently affect other desktops if this property is set to **true**. For a detailed discussion about the relationship between library definitions and desktop instances, see "Scope and Library Inheritance" on page 6-20.

xip.import.context.propagate.changes=sync

The **xip.import.context.propagate.changes** property lets you specify whether or not to propagate changes down the hierarchy of portal elements. Valid values for this property are **sync** or **off**.

When adding, removing, or moving portlets, pages or books, the changes typically are propagated down the hierarchy. In other words, changes made in the library are seen in all desktops, and changes made to the admin view (default desktop) are seen by all users.

A user's desktop view initially points to (inherits its properties from) the default desktop (admin view). If the default desktop changes, the changes are propagated downward to the user view.

When users customize their views, each user's view receives a copy of the customized portions of their view, and the rest of the portal continues to reference the default, or parent, desktop.

This property gives you the ability to control how changes are propagated down the hierarchy when users have customized their portals. If a user's or admin's portal is never customized, their views will always inherit changes from up the hierarchy, even if this property is set to **off**.

If, however, a user has customized a portion of a portal, and the same portion is modified up the hierarchy, this property allows you to control whether or not the change is propagated down the hierarchy. In this case, if you set this property to **off**, admin and user views will not inherit updates made to parent components. If set to **synch**, changes will be propagated downward, even if the user customized his or her view.

Tip: You may want to turn this property **off** if you do not want to modify the pages or books that the user or administrator considers to be privately owned.

Tip: It is considered good practice to allow your users to modify only a certain section of the portal and lock down the rest (for instance, one page that they have full control over), instead of allowing them free control over the entire portal. This promotes better scalability and makes portals more manageable when a large number of users make customizations.

xip.import.context.create.portal=true

If **xip.import.context.create.portal** is set to true and a portal does not exist in the database, one will be created for you. In addition, you must specify a title for the new portal using the next property.

xip.import.context.portal.title=My Portal

If **xip.import.context.create.portal** (described previously) is set to true, then you must specify a title using this property.

```
xip.import.context.locale.language=en
xip.import.context.locale.country=
xip.import.context.locale.variant=
```

These properties define the locale for the `.portal` file's titles and descriptions.

Running the Build Script

Once the property attributes are defined, you can run the build script, as shown in Listing 11-3. The build script's task writes status information, also shown below, to the console window.

Listing 11-3   Running the Ant Build Script

```
C:\dev\xip>ant run

[java] Using: Properties from file [xip.properties]
[java]      Name [xip.config.url] Value [t3://localhost:7003]
[java]      Name [xip.config.username] Value [weblogic]
[java]      Name [xip.config.password] Value [weblogic]
[java]      Name [xip.config.application] Value [portalProject]
[java]      Name [xip.command] Value [import]
[java]      Name [xip.object] Value [desktop]
[java]      Name [xip.identifier.webapp] Value [PortalWebApp_1]
[java]      Name [xip.identifier.portal.path] Value [yourPortal]
[java]      Name [xip.identifier.desktop.path] Value [yourDesktop]
[java]      Name [xip.identifier.book.label] Value []
[java]      Name [xip.identifier.page.label] Value []
[java]      Name [xip.input.file] Value [myportal.portal]
[java]      Name [xip.output.file] Value []
[java]      Name [xip.import.context.deletes] Value [false]
[java]      Name [xip.import.context.moves] Value [false]
[java]      Name [xip.import.context.outermoves] Value [false]
[java]      Name [xip.import.context.updates] Value [true]
[java]      Name [xip.import.context.abort.on.collisions] Value [null]
[java]      Name [xip.import.context.abort.if.portlets.missing] Value [false]
[java]      Name [xip.import.context.scope] Value [admin]
[java]      Name [xip.import.context.modify.definitions] Value [false]
[java]      Name [xip.import.context.propagate.changes] Value [sync]
[java]      Name [xip.import.context.create.portal] Value [true]
[java]      Name [xip.import.context.portal.title] Value [My Portal]
[java]      Name [xip.import.context.locale.language] Value [en]
[java]      Name [xip.import.context.locale.country] Value []
[java]      Name [xip.import.context.locale.variant] Value []
[java]      Name [xip.export.context.scope] Value [admin]
[java]      Name [xip.export.context.locale.language] Value [en]
```

```
[java]      Name [xip.export.context.locale.country] Value []
[java]      Name [xip.export.context.locale.variant] Value []


[java] Executing command: import
[java] Importing desktop: Webapp: [PortalWebApp_1] PortalPath: [yourPortalPath]
DesktopPath: [yourDesktopPath]
[java] Connection to host: t3://localhost:7003
[java] Uploading file: myportal.portal
[java] Done. time taken 40 sec.

BUILD SUCCESSFUL

Total time: 52 seconds
```

Exporting a Page

If you do not wish to export an entire desktop, you can configure the utility to export a single page.

Tip: While this section explicitly deals with exporting pages, the same basic procedure applies to exporting books.

This section explains how to export a page from a desktop. Exporting a page is another common use case. When you export a page or a book, the result is a `.pinc` file.

Note: If you export a page or a book, all children of that page or book are exported as well.

The basic steps for exporting a page include editing the `xip.properties` file and running an Ant build script.

These steps are explained in the following sections:

1. Editing the Properties File

2. Running the Build Script

Editing the Properties File

To export an existing page as a `.pinc` file you will need to specify attributes in the `xip.properties` file. This section highlights the required changes.

`xip.config.application=portalProject`

You must specify the name of Enterprise application you are exporting from. In Workshop for WebLogic, this value corresponds to the Portal EAR Project name, as shown in the following figure:

Figure 11-8 Application Name


```
xip.command=export
```
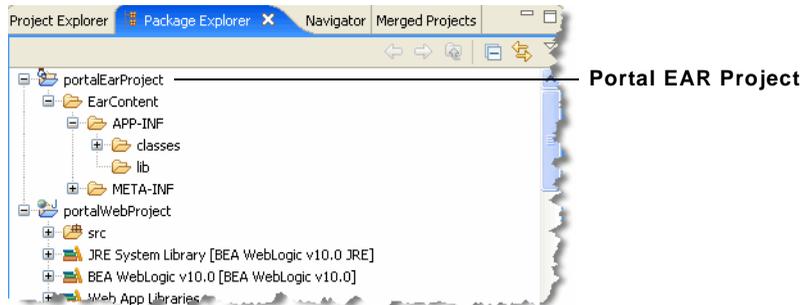
Specify that you wish to export, rather than import.

```
xip.object=page
```

The object you are interested in exporting – in this example a page.

```
xip.identifier.webapp=PortalWebApp_1
xip.identifier.portal.path=yourPortalPath
xip.identifier.desktop.path=yourDesktopPath
```

Note: These lines identify the objects you wish to export. For information on where to find the values for these properties (if you do not know what they are), see "Locating and Specifying Identifier Properties" on page 11-31.

The web `webapp` property must always be specified.

If you scope the export to the *admin* or *visitor* level, then you must specify the `portal.path` and `desktop.path` properties. If you are exporting a book or a page, then the `book.label` or `page.label` properties must be specified.

If you scope the export to the *library* then you do not need to supply a `portal.path` and `desktop.path`, but you still need to supply a `webapp` name.

```
xip.identifier.book.label=
xip.identifier.page.label=P200134591113965077078
```

You need to identify the page or book that you wish to export. To do this, specify the *definition label* of the page or book using the `xip.identifier.page.label` property.

The definition label is typically supplied by the developer in Workshop for WebLogic, but it could also have been automatically generated by the WebLogic Portal Administration Console. See "Locating and Specifying Identifier Properties" on page 11-31 for information on finding the definition label.

xip.output.file=mypage.pinc

Use this property to specify a file in which to save the result. In this example, the file mypage.pinc is saved in the directory in which the utility is run.

Note: You do not need to specify the encoding as this information is in the database.

xip.export.context.scope=admin

Specify the scope of the export. In this example, the page is exported from within a desktop (*admin* scope). If you want to export a page from the library, set the scope to **library**.

xip.export.context.locale.language=en

Specify the locale of the exported page.

Running the Build Script

Once the property attributes are defined, you can run the Ant build script, as shown in Listing 11-4. The build script's task writes status information, also shown below, to the console window.

Listing 11-4   Running the Ant Build Script

```
C:\dev\xip>ant run
Buildfile: build.xml
init:
compile:
jar:
run:

[java] Using: Properties from file [xip.properties]
[java]      Name [xip.config.url] Value [t3://localhost:7003]
[java]      Name [xip.config.username] Value [weblogic]
[java]      Name [xip.config.password] Value [********]
[java]      Name [xip.config.application] Value [portalProject]
[java]      Name [xip.command] Value [export]
[java]      Name [xip.object] Value [page]
[java]      Name [xip.identifier.webapp] Value [yourPortal]
[java]      Name [xip.identifier.portal.path] Value [yourPortalPath]
```

```
[java]       Name [xip.identifier.desktop.path] Value [yourDesktopPath]
[java]       Name [xip.identifier.book.label] Value []
[java]       Name [xip.identifier.page.label] Value [P2001345911113965077078]
[java]       Name [xip.input.file] Value []
[java]       Name [xip.output.file] Value [mypage.pinc]
[java]       Name [xip.import.context.deletes] Value [false]
[java]       Name [xip.import.context.moves] Value [false]
[java]       Name [xip.import.context.outermoves] Value [false]
[java]       Name [xip.import.context.updates] Value [true]
[java]       Name [xip.import.context.abort.on.collisions] Value [null]
[java]       Name [xip.import.context.abort.if.portlets.missing] Value [false]
[java]       Name [xip.import.context.scope] Value [admin]
[java]       Name [xip.import.context.modify.definitions] Value [false]
[java]       Name [xip.import.context.propagate.changes] Value [sync]
[java]       Name [xip.import.context.create.portal] Value [true]
[java]       Name [xip.import.context.portal.title] Value [My Portal]
[java]       Name [xip.import.context.locale.language] Value [en]
[java]       Name [xip.import.context.locale.country] Value []
[java]       Name [xip.import.context.locale.variant] Value []
[java]       Name [xip.export.context.scope] Value [admin]
[java]       Name [xip.export.context.locale.language] Value [en]
[java]       Name [xip.export.context.locale.country] Value []
[java]       Name [xip.export.context.locale.variant] Value []

[java] Executing command: export

[java] Exporting page [mypage] scoped to desktop: Webapp: [PortalWebApp_1]
PortalPath: [yourPortalPath] DesktopPath: [yourDesktopPath]]
[java] Connection to host: t3://localhost:7003
[java] Saving changes to: mypage.pinc
[java] Done. time taken 6 sec.

BUILD SUCCESSFUL

Total time: 8 seconds
```

Importing a Page

This section explains how to import a single page.

Tip: While this section explicitly deals with pages, the same procedure applies to importing a book.

The imported page could have been created in Workshop for WebLogic or exported from another staging or production environment. Either way you are importing a `.pinc` file into the database.

Note: When you import or export a page or book, all the children are imported as well.

The basic steps for importing a page include editing the `xip.properties` file and running an Ant build script.

These steps are explained in the following sections:

1. Editing the Properties File

2. Running the Build Script

Editing the Properties File

To import an existing `.pinc` file you must specify attributes in the `xip.properties` file. This section highlights the required changes you must make to the properties file to import a page.

`xip.config.application=portalProject`

> You must specify the name of Enterprise application you are importing to. In Workshop for WebLogic, this value corresponds to the Portal EAR Project folder, as shown in the following figure:

Figure 11-9 Application Name
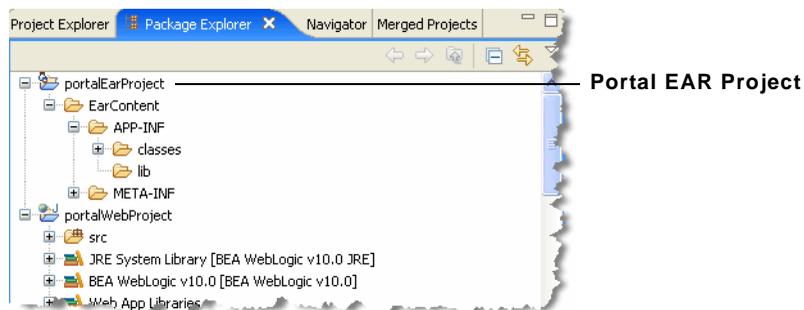
`xip.command=import`

> Specify that you wish to import, rather than export.

`xip.object=page`

> The object you are interested in importing – in this example a page.

```
xip.identifier.webapp=PortalWebApp_1
xip.identifier.portal.path=yourPortalPath
xip.identifier.desktop.path=yourDesktopPath
xip.identifier.book.label=
xip.identifier.page.label=
```

Note: These lines identify the objects you wish to export. For information on where to find the values for these properties (if you do not know what they are), see "Locating and Specifying Identifier Properties" on page 11-31.

If you scope the import to the *admin* or *visitor* level, then you must specify the `portal.path` and `desktop.path` properties. If you are importing a book or a page, then the `book.label` or `page.label` properties must be specified.

The web **webapp** property must always be specified.

Note: The **page.label** property is not needed on imports. This is because the page label is defined in the `.pinc` file as an attribute of the `<netuix:page ... />` element.

```
xip.input.file=mypage.pinc
```

Specify the `.pinc` file to import from. By default, the utility locates the file in a directory relative to where you run the utility.

```
xip.import.context.scope=admin
```

Identify the scope of the import. In this example you are importing a page at the admin level.

```
xip.import.context.deletes=false
```

You need to specify how to handle *deletes*. See "Controlling How Portal Assets are Merged When Imported" on page 11-27 for more information on this property.

```
xip.import.context.moves=false
```

You need to specify how to handle moves. See "Controlling How Portal Assets are Moved When Imported" on page 11-29 for more information on this property.

```
xip.import.context.updates=true
```

If **xip.import.context.updates** is true then new portlets, book and pages that don't exist in the desktop but exist in the `.portal` will be added to the new desktop. Also instance level resources will be updated.

```
xip.import.context.abort.if.portlets.missing=true
```

If **xip.import.context.abort.if.portlets.missing** is **true** then if a portlet is referenced in the .portal file but does not exist in the web application then the utility will halt the import. If this flag is **false** then the utility logs a warning and continues.

xip.import.context.modify.definitions=true

If **xip.import.context.modify.definitions** is **true**, when updating books and pages the utility also updates the library definition. A library definition consists of markup, which includes backing files, activate, deactivate, and rollover images. Also titles and descriptions for books and pages, and the is_hidden flag are stored in these definitions.

Note: Be aware that if you modify a library definition, desktop instances that share that library definition could be affected. Therefore, if you scope to a particular desktop, you may inadvertently affect other desktops if this property is set to true. For a detailed discussion about the relationship between library definitions and desktop instances, see "Scope and Library Inheritance" on page 6-20.

xip.import.context.propagate.changes=sync

The **xip.import.context.propagate.changes** property lets you specify whether or not to propagate changes down the hierarchy of portal elements. Valid values for this property are **sync** or **off**.

When adding, removing, or moving portlets, pages or books, the changes typically are propagated down the hierarchy. In other words changes made in the library are seen in all desktops. And changes made to the admin view (desktop) are seen by all users.

A user's desktop view inherits its properties from the default desktop (admin view). If the default desktop changes, the changes are propagated downward to the user view. When a users customize their views, each user's view receives a copy of the customized portions of their view, and the rest of the portal continues to reference the default, or parent, desktop.

This property gives you the ability to control how changes are propagated down the hierarchy when users have customized their portals. If a user's or admin's portal is never customized, their views will always inherit changes from up the hierarchy, even if this property is set to **off**.

If, however, they have customized a portion of a portal, *and the same portion is modified up the hierarchy*, this property allows you to control whether or not the change is propagated down the hierarchy. In this case, if you set this property to **off**, admin and user views will not inherit updates made to parent components. If set to **sync**, changes will be propagated downward, even if the user customized his or her view.

Tip: You may wish to turn this property **off** if you do not want to modify the pages or books that the user or administrator considers to be privately owned.

Tip: It is considered good practice to only allow your users to modify a certain section of the portal and lock down the rest (for instance, one page that they have full control over), instead of allowing them free control over the entire portal. This promotes better scalability and makes portals more manageable when a large number of users make customizations.

```
xip.import.context.locale.language=en
xip.import.context.locale.country=
xip.import.context.locale.variant=
```

These properties define the locale for the `.pinc` file's titles and descriptions.

Running the Build Script

Once the property attributes are defined, you can run the build script, as shown in Listing 11-5. The build script's task writes status information, also shown below, to the console window.

Listing 11-5 Running the Ant Build Script

```
C:\dev\xip>ant run

[java] Using: Properties from file [xip.properties]
[java]      Name [xip.config.url] Value [t3://localhost:7003]
[java]      Name [xip.config.username] Value [weblogic]
[java]      Name [xip.config.password] Value [weblogic]
[java]      Name [xip.config.application] Value [portalProject]
[java]      Name [xip.command] Value [import]
[java]      Name [xip.object] Value [page]
[java]      Name [xip.identifier.webapp] Value [PortalWebApp_1]
[java]      Name [xip.identifier.portal.path] Value [yourPortalPath]
[java]      Name [xip.identifier.desktop.path] Value [yourDesktopPath]
[java]      Name [xip.identifier.book.label] Value []
[java]      Name [xip.identifier.page.label] Value []
[java]      Name [xip.input.file] Value [mypage.pinc]
[java]      Name [xip.output.file] Value []
[java]      Name [xip.import.context.deletes] Value [false]
[java]      Name [xip.import.context.moves] Value [false]
[java]      Name [xip.import.context.outermoves] Value [false]
```

```
[java]      Name [xip.import.context.updates] Value [true]
[java]      Name [xip.import.context.abort.on.collisions] Value [null]
[java]      Name [xip.import.context.abort.if.portlets.missing] Value [false]
[java]      Name [xip.import.context.scope] Value [admin]
[java]      Name [xip.import.context.modify.definitions] Value [false]
[java]      Name [xip.import.context.propagate.changes] Value [sync]
[java]      Name [xip.import.context.create.portal] Value [true]
[java]      Name [xip.import.context.portal.title] Value [My Portal]
[java]      Name [xip.import.context.locale.language] Value [en]
[java]      Name [xip.import.context.locale.country] Value []
[java]      Name [xip.import.context.locale.variant] Value []
[java]      Name [xip.export.context.scope] Value [admin]
[java]      Name [xip.export.context.locale.language] Value [en]
[java]      Name [xip.export.context.locale.country] Value []
[java]      Name [xip.export.context.locale.variant] Value []

[java] Executing command: import

[java] Importing page scoped to desktop: Webapp: [PortalWebApp_1] PortalPath:
[yourPortalPath] DesktopPath: [yourDesktopPath]

[java] Connection to host: t3://localhost:7003
[java] Uploading file: mypage.pinc
[java] Done. time taken 20 sec.

BUILD SUCCESSFUL

Total time: 25 seconds
```

Controlling How Portal Assets are Merged When Imported

The **xip.import.context.deletes** property lets you control how portal assets are merged during an import. As shown in Figure 11-10, if you set this property to false, the contents of the imported pages are combined with the existing pages.

Note: This example refers to portlets; however, the same merge operations would apply as well for pages on a book.

Figure 11-10 Merge Results with Deletes = false
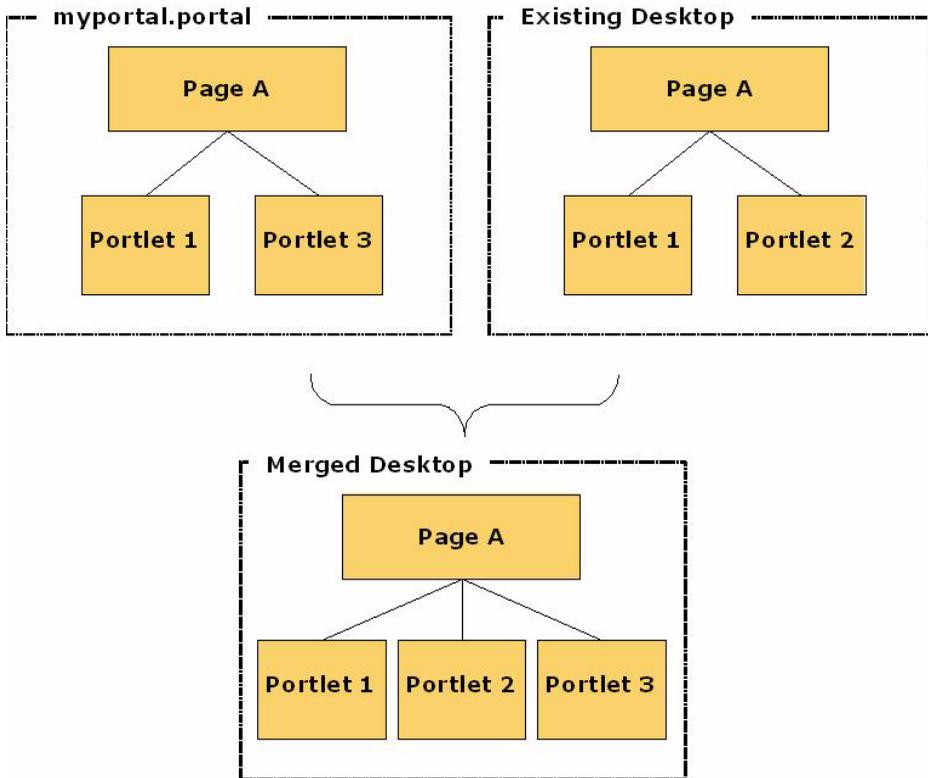
As shown in Figure 11-11, if you set this property to `true`, any portlets that do not exist in the `.portal` file but do exist in the desktop are deleted.

Figure 11-11 Merge Result with Deletes = true
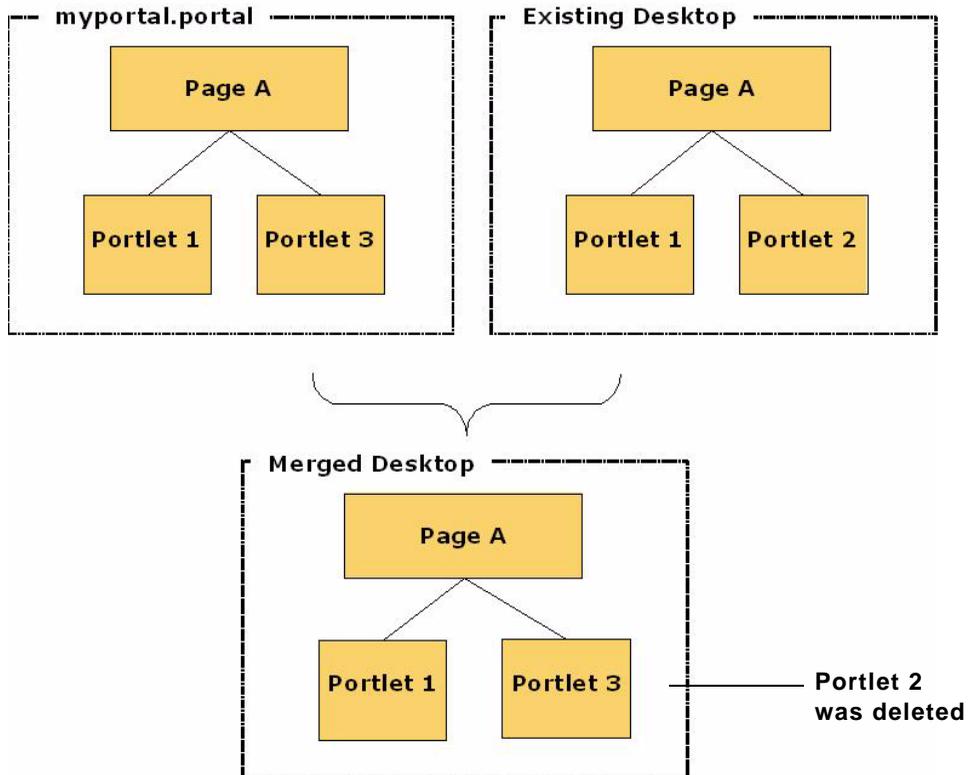
Controlling How Portal Assets are Moved When Imported

The `xip.import.context.moves` and `xip.import.context.outermoves` properties let you specify how moves are handled when portal assets are imported. To understand how moves operate, you need to understand the meaning of *inner moves* and *outer moves*. These terms are explained in the following sections.

Inner Moves

When an asset such as a portlet or a page is moved within the context of a single parent, the move is called an inner move. For example, if you move a portlet to a different placeholder within a page, it is an inner move because the parent (the page) remains the same. Likewise, if you move

a page to a different position in a book, it is an inner move because the parent (the book) remains the same. The following figure illustrates this concept:

Figure 11-12 Inner Move: Pages P1 and P2 Swap Positions Within the Same Book
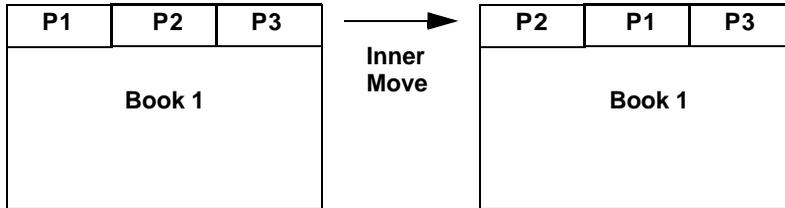
Set the `xip.import.context.moves` property to `true` to allow inner moves. If the property is set to `false`, the positions of assets will not change; however, the contents will be updated appropriately. If you want to move books, pages, or portlets across different parents, then use the `xip.import.context.outermoves` property, described in the next section.

Outer Moves

When an asset such as a portlet or a page is moved from one parent to another, the move is called an outer move. For example, if you move a page from one book to a different book, that is an outer move because the parent (the book) is different. The following figure illustrates this concept:

Figure 11-13 Outer Move: Pages P1 and P4 Swap Positions Across Different Books
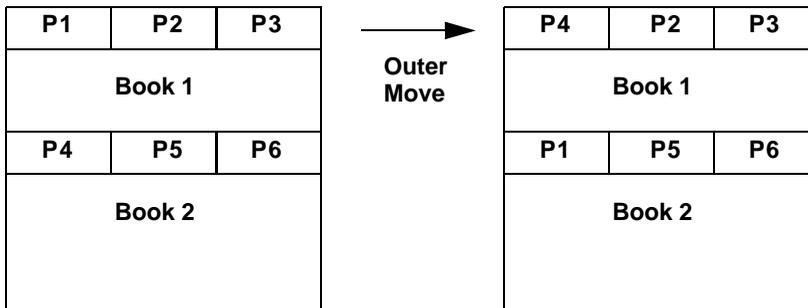
Set the `xip.import.context.outermoves` property to `true` to allow outer moves. If the property is set to `false`, then the operation handles the requested move as a deletion and addition operation.

Note: Move operations preserve customizations. When a deletion/addition operation is performed, some customizations are lost.

If you want to move books, pages, or portlets across the same parent, then use the `xip.import.context.moves` property, described in the previous section.

Locating and Specifying Identifier Properties

This section explains how to find the values for the `identifier` properties in the `xip.properties` file. These properties help to identify the portal element to export or import. This section discusses the following properties:

The webapp Property

The portal.path and desktop.path Properties

The page.label and book.label Properties

The webapp Property

The web `xip.identifier.webapp` property must always be specified. The web application name is listed in the WebLogic Portal Administration Console, as shown in the following figure:

Figure 11-14 The Web Application Name
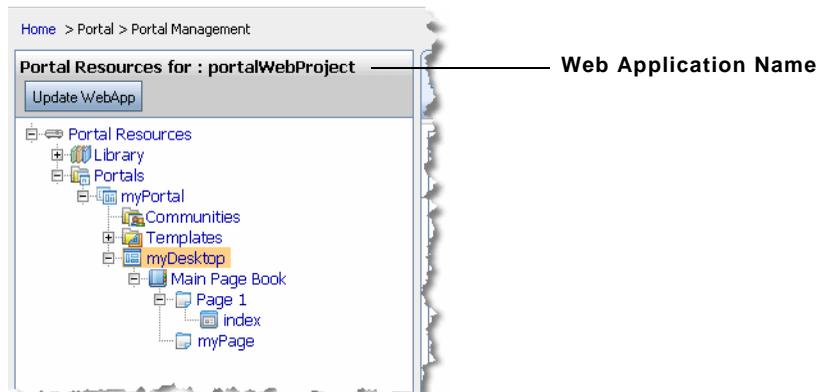
The portal.path and desktop.path Properties

If you scope the export to the *admin* or *visitor* level, then you must specify the `xip.identifier.portal.path` and `xip.identifier.desktop.path` properties. You can

find the correct value in the WebLogic Portal Administration Console. The portal path is shown in the Portal Details tab, and the desktop path is shown in the Desktop Details tab. For example, the portal path is shown in the following figure:

Figure 11-15 The Portal Path Value As Shown in the Portal Details Tab

The page.label and book.label Properties

If you are exporting or importing a page or book, you need to specify the *definition label* of the page or book using the `xip.identifier.page.label` or `xip.identifier.book.label` property. Definition labels are typically supplied by the developer in Workshop for WebLogic; however, they can also be created in the WebLogic Portal Administration Console. If the page was created using the Administration Console, a definition label is assigned automatically.

Note: If you are exporting a book or a page, then the `book.label` or `page.label` properties must be specified. If you are importing a book or a page, the `book.label` or `page.label` properties are not needed, because these values are obtained directly from the `.pinc` files.

Locating the Definition Label for a Page

You can locate the definition label for a page in the Administration Console on the Details tab of the page window (click on the page in the Library tree to display the page window). Figure 11-16 shows the definition label for a page.

Figure 11-16 The Location of a Page's Definition Label in the WebLogic Portal Administration Console
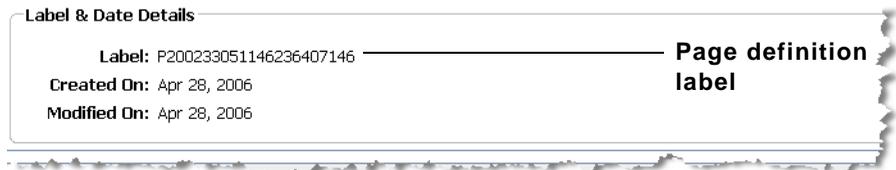
Locating the Definition Label for a Book

To find the definition label of a book, you need to look at the Manage Book Contents tab of the book's parent book. You will find the labels of all of the child books listed there.

Managing the Cache

Proper cache management greatly improves portal performance. Whenever a cache is invalidated, the portal API must retrieve fresh data from the database. It is inefficient when the API retrieves data that has not changed. The Export/Import Utility invalidates only the caches (specifically, the `portalControlTreeCache`) for portal resources that have changed. The invalidation of the cache is governed by the following properties:

```
xip.config.application
xip.identifier.webapp
xip.identifier.portal.path
xip.identifier.desktop.path
xip.import.context.scope
xip.import.context.modify.definitions
```

Be sure to understand these properties and how they affect the invalidation of the cache. Understanding your changes, scoping them correctly, and tuning these properties correctly will greatly improve your production system's performance.

These properties are described below.

xip.config.application

Specifies the name of the Enterprise application. Only portals under this Enterprise application will be affected. In Workshop for WebLogic, this value corresponds to the name of an Portal EAR Project folder, as shown in the following figure:

Figure 11-17 Application Name

Portal EAR Project
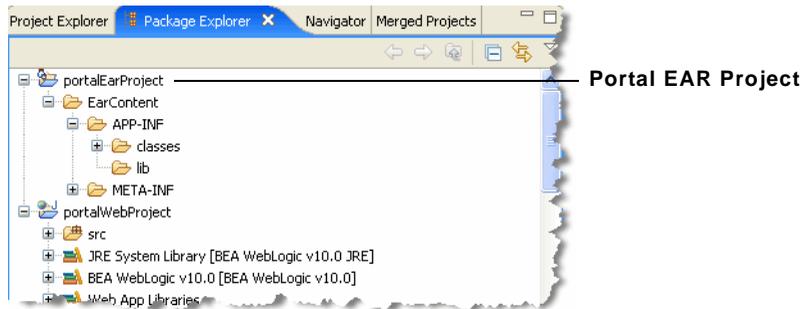
`xip.identifier.webapp`

> Specifies the name of the web application. Only portals under this web application will be affected. See also "Locating and Specifying Identifier Properties" on page 11-31.

`xip.import.context.scope`

> Specifies the scope of the change. If the scope is set to `library` then all `portalControlTree` caches for the entire web application are invalidated. If the scope is set to `admin` then only the cache for the desktop defined by `xip.identifier.portal.path` and `xip.identifier.desktop.path` is affected, leaving the other desktop caches intact. If the scope is set to `visitor` then only the cache of an individual user's view is invalidated. See also "Locating and Specifying Identifier Properties" on page 11-31.

`xip.import.context.modify.definitions`

> If set to `true`, library definitions may be modified. Since definitions are shared across all instances of library resources, the entire web application cache must be invalidated, even if the scope is set to `admin`. If you are just adding pages or portlets to the books and pages there is no reason to update the definitions.

Note: A library definition consists of markup, which includes backing files, activate, deactivate, and rollover images. Also titles and descriptions for books and pages, and the `is_hidden` flag are stored in these definitions.

WARNING: Be aware that if you modify a library definition, desktop instances that share that library definition could be affected. Therefore, if you scope to a particular desktop, you may inadvertently affect other desktops if this property is set to true. For a detailed discussion about the relationship between library definitions and desktop instances, see "Scope and Library Inheritance" on page 6-20.

Using the Datasync Web Application

Note: **The Datasync Web Application is deprecated.** It is recommended that you use the propagation tools to propagate datasync data. If you want to use the Datasync Web Application in a browser, you need to add the following WAR file to your application:

```
WEBLOGIC_HOME/common/p13n/lib/deprecated/datasync.war
```

This chaper provides instructions for updating portal application datasync data, such as user profile properties, user segments, content selectors, campaigns, discounts, and other property sets, which must be bootstrapped to the database in a separate deployment process.

Note: In a WebLogic Portal cluster where the Managed Servers are running on different computers than the Administration Server, the ListenAddress attribute of each Managed Server must be set to its own IP address or DNS name; this allows datasync to propagate updates throughout the cluster. Setting the cluster addresses to DNS addresses is covered in the WebLogic Server document "Setting Up WebLogic Clusters."

This chapter includes the following topics:

- Portal Datasync Definitions
- Datasync Definition Usage During Development
- Compressed Versus Uncompressed EAR
- Rules for Deploying Datasync Definitions

Portal Datasync Definitions

WebLogic Portal allows you to author a number of definition files, such as user profiles and content selectors, that must be managed carefully when moving from development to production and back.

Within Workshop for WebLogic, portal definitions are created in a special datasync project. This project can contain user profile property sets, user segments, content selectors, campaigns, discounts, catalog property sets, event property sets, and session and request property sets.

Datasync Definition Usage During Development

During development, all files created in the datasync project are stored in a datasync project. To provide optimum access from runtime components to the definitions, a datasync facility provides an in-memory cache of the files. This cache intelligently polls for changes to definitions, loads new contents into memory, and provides listener-based notification services when content changes, letting developers preview datasync functionality in the development environment.

Datasync definition modifications are made not only by Workshop for WebLogic developers, but also by business users and portal administrators, who can modify user segments, campaigns, placeholders, and content selectors with the WebLogic Portal Administration Console. In the development environment, both Workshop for WebLogic and the WebLogic Portal Administration Console write to the files in the datasync project directory.

Compressed Versus Uncompressed EAR

When deployed into a production system, portal definitions often need to be modifiable using the WebLogic Portal Administration Console. In most production environments, the portal application will be deployed as a compressed EAR file, which limits the ability to write modifications to these files. In a production environment, all datasync assets must be loaded from the filesystem into the database so the application can be updated.

Figure 12-1 shows how the /data directory from the updated portal application is put into a standalone JAR and bootstrapped to the database.

Figure 12-1 Loading Updated Datasync Files to the Database
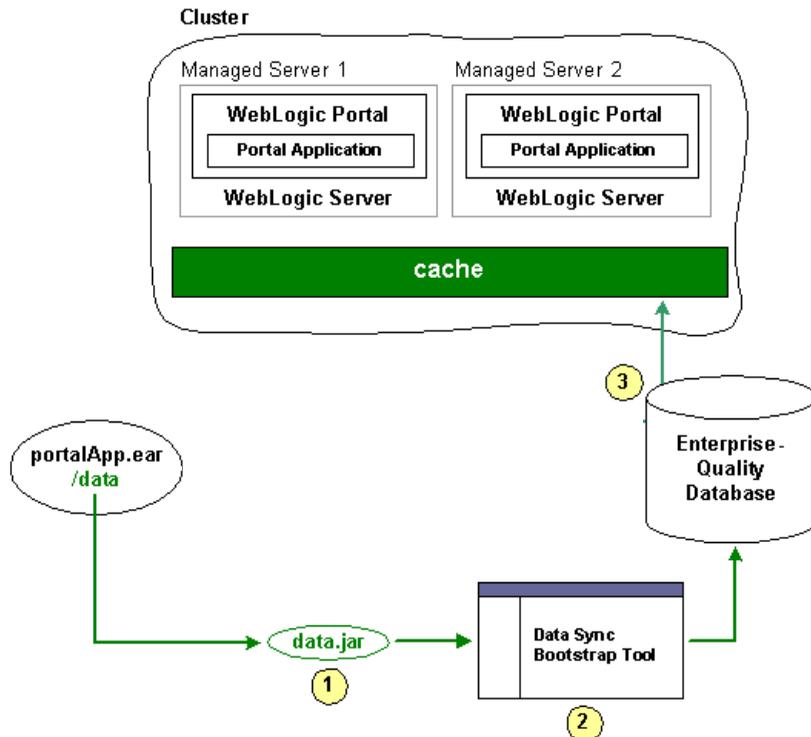
Alternatively, some production environments deploy their portal applications as uncompressed EARs.

For both compressed and uncompressed EAR files, you can view and update datasync definitions using the Datasync Web Application.

Datasync Web Application

Note: If you want to use the Datasync Web Application in a browser, you need to add the following WAR file to your application:

```
WEBLOGIC_HOME/common/p13n/lib/deprecated/datasync.war
```

Each portal application contains a Datasync Web Application located in `datasync.war` in the application root directory. Typically, the URL to the Datasync web application is `http://`*`server`*`:`*`port`*`/`*`appName`*`DataSync`. For example,

http://localhost:7001/portalAppDataSync. You can also find the URL to your Datasync web application by loading the WebLogic Server Administration Console and selecting **Deployments > Applications >** *appName* **> *DataSync** and clicking the **Testing** tab to view the URL.

The Datasync web application allows you to view the contents of the repository and upload new content, as shown in .

Figure 12-2 Datasync Web Application Home Page

Working with the Repository Browser – When working with the Data Repository Browser, you have the option to work with all the files in the repository using the icons on the left hand side of the page, or drill down into a particular sub-repository, such as the repository that contains all Property Set definitions.

View Contents – To view the contents of a repository, click on the ![icon] icon to bring up the window shown in Figure 12-3.

Figure 12-3 Browsing the Datasync Repository

From this list, click on a particular data item to see its contents, as shown in Figure 12-4.

Figure 12-4 Data Item Contents

As you can see in Figure 12-4, you can view the XML data for a particular content item.

Removing Content

To remove content from a repository, click the trash can icon on the left side of the page.

Working with a Compressed EAR File

When the application is deployed, if the JDBC Repository is empty (contains no data), then the files in the EAR will be used to bootstrap (initialize) the database. The Datasync assets are stored in the following tables: DATA_SYNC_APPLICATION, DATA_SYNC_ITEM, DATA_SYNC_SCHEMA_URI, and DATA_SYNC_VERSION. The bootstrap operation by default

happens only if the database is empty. When you want to do incremental updates, the Datasync web application provides the ability to load new definitions directly into the database. This can be done as part of redeploying a portal application, or independently using a special JAR file that contains your definitions, as shown in Figure 12-4, "Data Item Contents," on page 12-6.

Uploading new contents

When you click the icon, the following page appears, which lets you load data into the database.

Figure 12-5 Uploading New Datasync Data
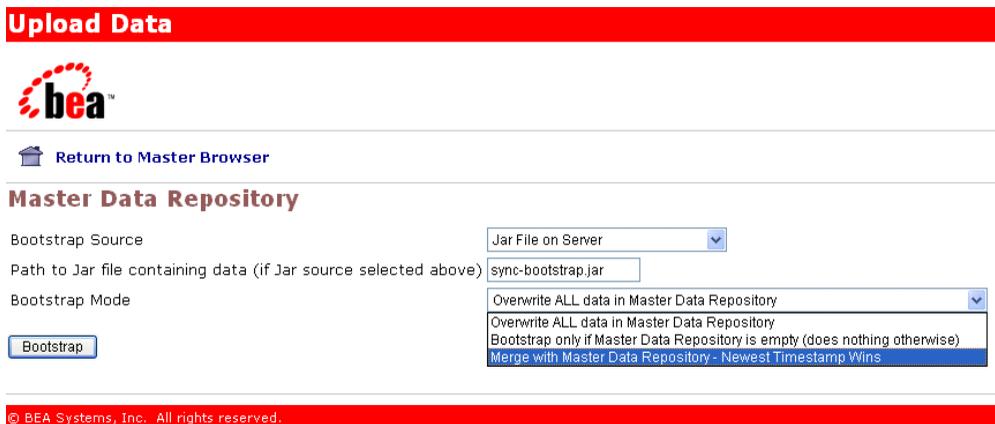
When you bootstrap, you choose a bootstrap source, which is either your deployed portal application or a stand-alone JAR file. For example, if you have an updated portal application that you have redeployed to your production environment, you can add any new definitions it contains to your portal. Alternatively, if you have authored new definitions that you want to load independently, you can create a JAR file with only those definitions and load them at any point.

Either way, when you update the data repository, you can choose to "Overwrite ALL data in the Master Data Repository," "Bootstrap only if the Mastery Repository is empty," or "Merge with Master Data Repository–latest timestamp wins."

Bootstrapping from an EAR

If you are redeploy an existing EAR application and want to load any new definitions into the database, choose the **Application Data (META-INF/data)** as your bootstrap source, and then choose the appropriate Bootstrap Mode. To ensure that you do not lose any information, you may

want to follow the instructions in the section entitled, "Pulling Definitions from Production" on page 12-8 to create a backup first. It is not possible to bootstrap definition data from an EAR file that is not deployed.

Creating a JAR file

To bootstrap new definition files independently of updates to your portal application, you can create a JAR file that is loaded onto the server that contains the files (content selectors, campaigns, user segments, and so on) that you want to add to the production system.

To do this, you can use the jar command from your `META-INF/data` directory. For example:

```
jar -cvf myfiles.jar *
```

This example will create a JAR file called `myfiles.jar` that contains all the files in your data directory, in the root of the JAR file. Then, you can bootstrap information from this JAR file by choosing **Jar File on Server** as your data source, specifying the full physical path to the JAR file and choosing the appropriate bootstrap mode. By running this process you can upgrade all the files that are packaged in your JAR. Controlling the contents of your JAR allows you to be selective in what pieces you want to update.

When creating the JAR file, the contents of the `META-INF/data` directory should be in the root of the JAR file. Do not jar the files into a `META-INF/data` directory in the JAR itself.

Validating Contents

After bootstrapping data, it is a good idea to validate the contents of what you loaded by using the View functionality of the Datasync web application.

Pulling Definitions from Production

Developers and testers may be interested in bringing datasync definitions that are being modified in a production environment back into their development domains. As the modified files are stored in the database, WebLogic Portal provides a mechanism for exporting XML from the database back into files.

One approach is to use the browse capability of the Datasync web application to view all XML stored in the database in a web browser. This information can then be cut and pasted into a file.

A better alternative is to use the DataRepositoryQuery Command Line Tool, which allows you to fetch particular files from the database using an FTP-like interface.

The DataRepositoryQuery Command Line Tool supports a basic, FTP-style interface for querying the data repository on a server.

The command line class is `com.bea.p13n.management.data.DataRepositoryQuery`. In order to use it, you must have the following in your CLASSPATH: `p13n_ejb.jar`, `p13n_system.jar`, and `weblogic.jar`.

Run the class with the argument `help` to print basic usage help.

For example:

```
set classpath=WEBLOGIC_HOME\p13n\lib\p13n_system.jar;
WEBLOGIC_HOME\p13n\lib\p13n_ejb.jar;
WEBLOGIC_HOME\server\lib\weblogic.jar

java com.bea.p13n.management.data.DataRepositoryQuery help
```

Options for Connecting to the Server

Several optional command arguments are used for connecting to the server. The default values are probably adequate for samples provided by BEA. In real deployments, the options will be necessary.

`-username userid`	Username of a privileged user (an administrator)	Default = weblogic
`-password password`	Password for the privileged user	Default = weblogic
`-app appName@host:port`	Application to manage	Default = @7001
`-url url`	URL to `DataRepositoryQuery` servlet	

Only one of `-app` or `-url` may be used, as they represent two alternate ways to describe how to connect to a server.

The URL is the most direct route to the server, but it must point to the `DataRepositoryQuery` servlet in the Datasync web application. This servlet should have the URI of `DataRepositoryQuery`, but you also need to know the hostname, port, and the `context-root` used to deploy `datasync.war` in your application. So the URL might be something like `http://localhost:7001/datasync/DataRepositoryQuery` if `datasync.war` was deployed with a `context-root` of `datasync`.

The `-app` option allows you to be a bit less specific. All you need to know is the hostname, port number, and the name of the application. If there is only one `datasync.war` deployed, you do not even need to know the application name. The form of the `-app` description is `appname@host:port`, but you can leave out some pieces if they match the default of a single application on localhost port 7001.

The `-app` option can be slow, as it has to perform many queries to the server, but it will print the URL that it finds, so you can use that with the `-url` option on future invocations.

Examples

This section lists examples of using the `DataRepositoryQuery` command.

Assuming `CLASSPATH` is set as previously described, and the default username/password of `weblogic`/`weblogic` is valid):

Find the application named p13nBase running on localhost port 7001:

`java com.bea.p13n.management.data.DataRepositoryQuery -app p13nBase`

Find the application named p13nBase running on snidely port 7501:

`java com.bea.p13n.management.data.DataRepositoryQuery -app`
`p13nBase@snidely:7501`

Find the single application running on localhost port 7101:

`java com.bea.p13n.management.data.DataRepositoryQuery -app @7101`

Find the single application running on snidely port 7001:

`java com.bea.p13n.management.data.DataRepositoryQuery -app @snidely`

Find the single application running on snidely port 7501:

`java com.bea.p13n.management.data.DataRepositoryQuery -app @snidely:7501`

In each of the examples, the first line of output will be something like this:
`Using url: http://snidely:7001/myApp/datasync/DataRepositoryQuery`

Usage

The easiest way to use the tool is in shell mode. To use this mode, invoke `DataRepositoryQuery` without any arguments (other than those needed to connect as described previously).

In this mode, the tool starts a command shell (you will see a `drq>` prompt) where you can interactively type commands, similar to how you would use something like `ftp`.

Alternatively, you can supply a single command (and its arguments), and `DataRepositoryQuery` will run that command and exit.

Commands

The `HELP` command gives you help on the commands you can use. Or use `HELP` *command* to get help on a specific command.

The available commands are:

Command	Description
HELP	Basic Help
HELP OPTIONS	Help on command line options
HELP command	Help on a specific command
HELP WILDCARDS	Help on wildcards that can be used with URI arguments
LIST [-l] [uri(s)]	List available data items
INFO [-l] [-d]	Print repository info
PRINT uri	Print a data item (the xml)
GET [-f] uri [filename]	Retrieve a data item to a file
MGET [-f] [uri(s)]	Retrieve multiple data items as files. Not specifying a URI retrieves all files.
EXIT or QUIT	Exit the shell (shell only)

Commands are not case-sensitive, but arguments such as filenames and data item URIs may be. More help than what is listed above can be obtained by using `HELP` *command* for the command you are interested in.

Where multiple URIs are allowed (indicated by `uri(s)` in the help), you can use simple wildcards, or you can specify multiple URIs. The result of the command includes all matching data items.

Options in square brackets (`[ ]`) are optional and have the following meanings:

`-l`	Output a longer, more detailed listing
`-d`	Include URIs of data items contained in each repository
`-f`	Force overwrite of existing files

The following example retrieves all assets from the repository as files:

```
java com.bea.p13n.management.data.DataRepositoryQuery -app mget
```

Rules for Deploying Datasync Definitions

There are a number of general concepts to think about when iteratively deploying datasync definitions into a production system. In general, adding new datasync definitions to a production system is a routine process that you can do at any time. However, removing or making destructive modifications to datasync definitions can have unintended consequences if you are not careful.

When removing or making destructive modifications to datasync definitions, you should first consider whether there are other components that are linked to those components. There are several types of bindings that might exist between definitions.

Note: For some of these bindings, it is very important to understand that they may have been defined on the production server using the WebLogic Portal Administration Console and may not be known by the developers.

One example of bindings is that you may have two datasync definitions bound together. An example of this is a campaign that is based on a user property defined in a user property set. If you remove the property set or the specify property, that campaign will no longer execute properly. In this case, you should update any associated datasync definitions before removing the property set or property.

A second scenario is that you have defined an entitlement rule that is bound to a datasync definition. For example, you might have locked down a portlet based on a dynamic role that checks if a user has a particular user property value. In this case, you should update that dynamic role before removing the property set or property.

A third scenario is that there are in-page bindings between datasync items and Portal JSP tags. An example is a `<pz:contentSelector>` tag that references a content selector. Update the content selector tag in the production environment before you remove the content selector. This

is one type of binding that is only configured in Workshop for WebLogic at development time rather than in the WebLogic Portal Administration Console.

A good guideline for developers is not to remove or make significant changes to existing datasync definitions that are in production. Instead, create new definitions with the changes that are needed. This can be accomplished by creating new versions of, for example, campaigns where there is no chance that they are being used in unanticipated ways. Additionally, perform datasync bootstraps of the production system's existing datasync definitions back into development on a regular basis.

Removing Property Sets

When you remove a property set, any existing values being stored locally by WebLogic Portal in the database will NOT be removed automatically. You need to examine the `PROPERTY_KEY` and `PROPERTY_VALUE` tables to clean up the data if desired.

Export/Import Utility Files

This appendix shows the default properties file for the Export/Import Utility.

Listing A-1 shows the default `xip.properties` file.

Listing A-1   The Default xip.properties File

```
# Export/Import Properties file. The properties in this file are read by the Xip
# (pronounced zip) utility. You may specify an alternate properties file via the
# -properties command line argument.
#
# Server connfiguration information
#
xip.config.url=t3://localhost:7003
xip.config.username=weblogic
xip.config.password=weblogic
xip.config.application=myEnterpriseApp
#
# command - Are we exporting or importing. Valid values are: "export", "import"
#
#xip.command=export
xip.command=import
#
# object - The "thing" you want to export/import (desktop, book, page)
xip.object=desktop
#xip.object=book
#xip.object=page
#
# Identifier properties, tells the import export utility how to identify the
# artifacts to be retrieved or updated. When importing and exporting books and
```

```
# pages. If scoping changes to the admin desktop (default desktop)
# or visitor desktop then "portal.path" and "desktop.path" must be specified. If
# you are exporting a book or page then the book.label or page.label need to be
# specified.
#
#page.label and book.label are not used on import as the labels are pulled from
#the .pinc files themselves.
#
# The webapp must always be specified. This is the webapp name not necessarily
# the directory name. If you are export or importing this is where you are export
# from or importing to respectively.
#
xip.identifier.webapp=myPortal
xip.identifier.portal.path=myPortal
xip.identifier.desktop.path=myDesktop
xip.identifier.book.label=
xip.identifier.page.label=
#
# Input and output files -- .pinc or.portal files. These files are
# relative to the "xip" directory
#
#xip.input.file=Book1.pinc
xip.input.file=myPortal.portal
xip.output.file=myPortal1.portal
xip.output.encoding=UTF-8
#
# Import options - these options are used as rules to the export/import utility
#
# scope - Changes can be scoped to the "library", "admin", or "visitor" when
# importing a .pinc file, and "admin" or "visitor" when
# exporting a .portal file.
# If this property # has a value of "admin" or "visitor" then a
# xip.identifier.portal.path and xip.identifier.desktop.path # must be specified
# above. Of course to scope exports to the "library" or "admin" you must be in
# the Admin or PortalSystemAdministrator Role.
#

#
# deletes - If true, then books, pages and portlets that are currently on the
# existing desktop in the database but not in the new import file (.portal or
# .pinc) will be removed from exiting desktop.
#
# moves - (innerMoves) If true, then existing books, pages and portlets that are
# in different locations on the same parent will be moved to the correct location.
# If you want to move books, pages and portlets across different parents
# then see outermoves
#
# outermoves - If true, then existing books, pages and portlets that are moved
# from different parents will be moved to the new parent. If this is not set then
```

```
# it will be handled as a remove and add (different customizations are lost)
#
# updates - If true, then books, pages and portlets that are currently not on
# the existing desktop will be added, and any instance attributes on the books,
# page, and portlet will be updated in the database.
#
# abort.if.portlets.missing - if true, then if the new .portal or .pinc
# file references a portlet that is not in the current webapp then
# abort, otherwise skip the portlet and continue on.
#
# modify.definitions - If this flag is set to true then any changes in the import
# file will effect the defintions and not just the instances. These include
# things like markup (backing files, rollover images, isHidden, ... for a more|
# complete list refer to the database schema). It is important to note that these
# changes may effect other desktops outside the one you are scoping it to.
#
# propagate.changes - Typically all changes that are made to Library artifacts
# are cascaded down to the admin's desktop and subsequently cascaded down to the
# visitor'ss view. If this property is set to "sync" then
# these changes will occur synchronously as part of this transaction. If this
# property is set to "off" then changes will not get cascaded for the artifacts
# which have been modified. For books, pages and portlets that have not
# been modified at the admin or visitor level, then these will always receive
# the changes as they point to the default.
#
# create.portal - If this flag is set then when importing a desktop and the given
# portal is not already create then one will be created for you.
#
# portal.title - If the above flag is set and a new portal is being created it
# needs a title. This property value will be the new portal's title.
#
# locale - the locale of the titles and descriptions in the .portal
# or .pinc file. Note the encoding is defined in the file itself.
#
xip.import.context.scope=admin
xip.import.context.deletes=false
xip.import.context.moves=false
xip.import.context.outermoves=false
xip.import.context.updates=true
xip.import.context.abort.if.portlets.missing=false
xip.import.context.modify.definitions=true
xip.import.context.propagate.changes=off
xip.import.context.create.portal=true
xip.import.context.portal.title=My Green Portal
xip.import.context.locale.language=en
xip.import.context.locale.country=
xip.import.context.locale.variant=
#
# Export Options
```

```
#
# scope - Changes can be scoped to the "library", "admin", or "visitor" when
# importing a .pinc file, and "admin" or "visitor" when exporting a
# .portal file.
# If this property has a value of "admin" or "visitor" then a
# xip.identifier.portal.path and xip.identifier.desktop.path
# must be specified above. Of course to scope exports to the "library" or "admin"
# you must be in the Admin or PortalSystemAdministrator Role.
#
# locale - the locale of the titles and descriptions in the .portal or.pinc
# file.
#
xip.export.context.scope=admin
xip.export.context.locale.language=en
xip.export.context.locale.country=
xip.export.context.locale.variant=
```