BEA WebLogic Portal™

Production Operations User Guide

Version 8.1 with Service Pack 5
Document Revised: October 2005
Rev. 1.0

Copyright

Copyright © 2005 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

Trademarks and Service Marks

Contents

Introduction to Production Operations

Installing the Propagation Software

Managing a Team Development Environment

Configuring a Portal Cluster

Preparing and Deploying the EAR File

Developing a Propagation Strategy

Inside the Propagation Utility

Using the Propagation Utility

Using the Export/Import Utility

Using the Datasync Web Application

Index

Introduction to Production Operations

The life cycle of a 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 guide discusses the following major areas of Production Operations. The rest of this chapter provides a brief description of each area:

- Managing the Portal Life Cycle

- Setting Up a Team Development Environment

- Configuring the Portal Cluster

- Building and Deploying the EAR File

- Propagating a Portal Application

- Performing Round-Trip Development

Managing the Portal Life Cycle

Production operations addresses the tools, procedures, methodologies, and best practices that provide the backbone for managing the portal life cycle, from portal development to staging and testing to live production environments. As Figure 1-1 shows, portals are typically developed in a *team development environment* by developers using WebLogic Workshop. Portal components are then moved to a *staging environment*, where portal administrators use the WebLogic Administration Portal 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 utilities provided by BEA. Utilities such as the Propagation Utility and the Export/Import Utility allow you to easily and reliably move and merge changes between environments.

Figure 1-1 Typical WebLogic Portal Environments
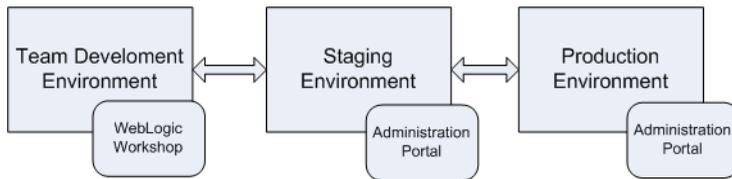
Like considering the architecture of a network or a software system, you should also consider and carefully plan how you will address production operations for your portal system. It is important to consider your particular portal system configuration, how your development team is organized, how you 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.

Setting Up a Team Development Environment

Team development of a WebLogic Portal Web site 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 3, "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 4, "Configuring a Portal Cluster" discusses how to set up a production database, choose a cluster architecture (single versus multi-cluster) and configure the domain.

Building and Deploying the EAR File

Deployment refers to preparing deployment descriptors and configuration files, building an Enterprise archive file (EAR), and deploying the EAR file to a destination server. Chapter 5, "Preparing and Deploying the EAR File" describes how to prepare a portal application's descriptor files 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 WebLogic Workshop to create portals and portal components, such as portlets.

- **Staging** – In a staging environment, administrators use the Administration Portal 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 Portal 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 Utility, described in Chapter 8, "Using the Propagation Utility"

- Datasync Web Application, described in Chapter 10, "Using the Datasync Web Application"

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 6, "Developing a Propagation Strategy"

- Chapter 7, "Inside the Propagation Utility"

Performing Round-Trip Development

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

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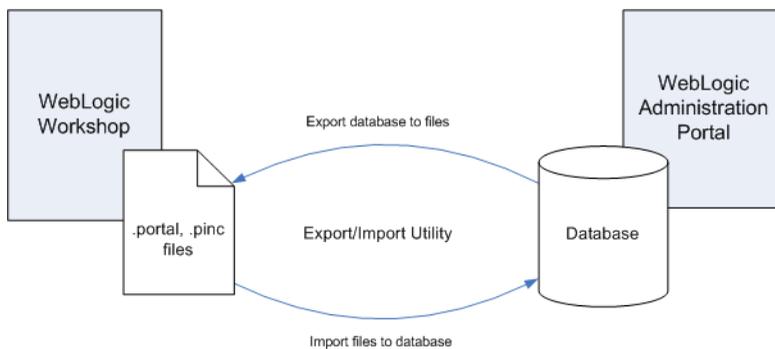
Tip: The Export/Import Utility is also known as the xip tool (pronounced "zip"). Typically, developers use this utility to move assets back and forth between a development and a staging environment.

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 9, "Using the Export/Import Utility".

Installing the Propagation Software

This chapter explains how to install the Propagation Utility and the Export/Import Utility.

This chapter includes the following sections:

- Updating the Database

- Installing the Propagation Utility

- Verifying the Propagation Utility Installation

- Installing the Export/Import Utility

- Running the Export/Import Utility

Updating the Database

Note: You only need to perform a database update if you are upgrading your WebLogic Portal installation from an SP4 or older version.

1. Run the `pf_update_system_data.sql` script using your database's SQL command tool. This script is located in:

   `BEA_HOME/weblogic81/portal/db/data/required`

Tip: The exact procedure for running this script depends on your database system. For example, if you use an Oracle database, start SQL*Plus and enter:

    SQL>@pf_update_system_data.sql

Note: This script updates the default markup in the database. The default markup for books, pages, and desktops is used when new desktops, books, and pages are created with the Administration Portal or Visitor Tools. The script includes the `$(markupName)` token in the database. If this token is not added, `SAXParseException` errors are thrown when you import desktops, pages, or books that were created with the Administration Portal or Visitor Tools.

Installing the Propagation Utility

The Propagation Utility is a web application that you deploy into an existing WebLogic Portal application. You need to install the Propagation Utility into any existing portal application that you want to propagate.

Note: If you are upgrading from SP4 or an older version of WebLogic Portal, you first need to run an SQL script to update your database, as explained in the previous section, Updating the Database.

1. Stop WebLogic Server if it is running.

2. Copy the `propagation.war` file from the `BEA_HOME/weblogic81/portal/lib` directory to the root directory of your Enterprise application.

3. In the root directory of your domain, open the `config.xml` file and add the following `WebAppComponent` tag to the `Application` tag for the Enterprise application you are propagating to or from:

```
<Application Name="myEnterpriseApp" ...>
    <WebAppComponent Name="propagation" Targets="portalServer" URI=
    "propagation.war"/>
</Application>
```

Note: You need to perform Step 3 for each new domain you create after the patch has been installed.

4. In the root directory of your Enterprise application, open the `META-INF\application.xml` file and add the following module if it is not already present:

```
<module>
    <web>
        <web-uri>propagation.war</web-uri>
        <context-root>propagation</context-root>
    </web>
</module>
```

Verifying the Propagation Utility Installation

1. Start WebLogic Server.

2. Open a Web browser.

3. Go to the following URL to start the Propagation Utility:

   ```
   http://host:port/propagation
   ```

   where *host* and *port* are the host name and port number that are configured for your WebLogic Server.

   The main page of the Propagation Utility appears.

4. Click the **View Configuration Details** link and verify that the JAR files were installed.

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 Propagation Utility, then the following procedure is unnecessary.

Before installing the Export/Import Utility, be sure you have Ant 1.5 in your PATH environment variable. Ant is part of the normal WebLogic Server installation. It is located in:

   ```
   WEBLOGIC_HOME\server\bin\ant
   ```

Note: If you are upgrading from SP4 or an older version of WebLogic Portal, you first need to run an SQL script to update your database, as explained in the previous section, "Updating the Database" on page 2-1.

1. Stop WebLogic Server if it is running.

2. Open the file BEA_HOME\weblogic81\portal\xip\build.xml, and edit the following properties in the **Installer** section to point to the appropriate locations:

Property	Description
bea.dir	Points to the BEA home directory
wlp.lib.dir	Points to the *WEBLOGIC_HOME*\portal\lib\netuix directory

3. Build the Export/Import Utility. To do this, run the following command from within the BEA_HOME\weblogic81\portal\xip directory:

```
ant jar
```

Running the Export/Import Utility

For detailed information about running and using the Export/Import Utility, see Chapter 9, "Using the Export/Import Utility".

Managing a Team Development Environment

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.

This section 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.

This document contains the following sections:

- Choosing a Source Control Vendor

- Creating a Shared Portal Domain

- Creating and Sharing the Portal Application

- WebLogic Portal Coding Practices

Choosing a Source Control Vendor

There are a number of source control providers, such as CVS, Perforce, Starteam, Visual Source Safe (VSS), and PVCS. This guide should assist 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. 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.

Note: Even if your source control management tool does not directly integrate with WebLogic Workshop (see also *Visual Studio Source Control Provider for WebLogic Workshop*), you can still use it to manage your WebLogic Server domains and applications.

Creating a Shared Portal Domain

Have the development team lead create the appropriate portal domain for the group. Before creating and storing domain assets, determine the BEA home directory—where the team will install WebLogic software.

The BEA Home Directory

The directory where WebLogic Platform software is installed is called the BEA home directory. WebLogic Workshop applications and domains each reference the BEA home directory. It is important to carefully consider where to put the BEA home directory. When installing WebLogic Platform, each developer can determine which drive and directory to install to.

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 any portal domain's `config.xml` file on each development machine and in domain batch scripts such as `startWeblogic.cmd`.

Understanding Why a Common BEA Home Matters

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 `config.xml` and the start scripts can be shared.

Note: When development environments require different domain configuration settings, and you want to use a configuration file template as a solution, do not store `config.xml` in source control. For more information see "Setting up a Configuration File Template (config-template.xml)" on page 3-17.

Following code fragments from `config.xml` and `startWebLogic.cmd` show hard-coded paths:

config.xml

```
<Application Name="JWSQueueTransport" Deployed="true"
    LoadOrder="1000" Path="C:\bea812\weblogic81\server\lib\" TwoPhase="true">
    <EJBComponent Name="QueueTransportEJB" Targets="portalServer"
```

```
    URI="QueueTransportEJB.jar"/>
</Application>
```

startWebLogic.cmd

```
set DOMAIN_HOME=C:\bea812\weblogic81\samples\domains\portal
```

If the `config.xml` and `startWeblogic.cmd` files are shared in source control, all team members must have installed WebLogic Server to the path shown in those files. However, there are circumstances when developers cannot install WebLogic Server to a particular drive, such as when they have multiple partitions with not enough space left on their C drive.

In addition, `pointbase.ini` in the domain root directory is a configuration file that may contain a `documentation.home` property that points to a hard-coded location on the filesystem.

Table 3-1 shows which files in a domain contain hard-coded paths.

Table 3-1 Domain Files with Hard-Coded Paths

```
backup_config.xml
config.xml
create_db.*
installService.cmd
set-dbenv.*
setDomainEnv.*
setDomainEnvQS.*
startManagedWebLogic.*
startManagedWebLogicQS.*
startPointBaseConsole.*
startPointBaseConsoleQS.*
startWebLogic.*
startWebLogicQS.*
stopManagedWebLogic.*
stopManagedWebLogicQS.*
stopWebLogic.*
stopWebLogicQS.*
uninstallService.*
webappCompile.*
webappCompileQS.*
domain-info.xml (in /_cfgwiz_donotdelete)
startscript.xml (in /_cfgwiz_donotdelete)
```

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 3-7.

Managing Multiple BEA Home Directory Locations for Your Team

There are a number of different techniques for sharing a common domain with team members with different BEA home directories, which are described in the following sections:

Option 1: Placing BEA Home in a Configuration File Template

Creating a configuration file template avoids many of the problems associated with other solutions, but it requires you to implement search and replace activities on your config.xml and other files. By creating a template for your config.xml which contains tokens that represent your BEA home directory, you can create a process to create a config.xml from a combination of the template and developer-specified token values.

A configuration file template can be used for much 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.

With a configuration file template, config.xml is not stored in source control. For information on setting up a configuration file template, see "Setting up a Configuration File Template (config-template.xml)" on page 3-17.

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

Windows-based developers may consider setting up a substitute drive letter to map to their BEA home directory.

From the command prompt, Windows lets you create a virtual drive and map it to an existing drive and path using the subst command. Your team can configure a common virtual drive letter not currently in use and use that drive for application and domain activities.

For example, you can create a drive letter P: that maps to a directory on the C: drive, such as C:\bea812, by executing the following command from a Windows Command prompt:

```
subst P: C:\bea812
```

Now, after creating a new domain, you can change all references to C:\bea812 to P: in the domain files listed in Table 3-1.

The previously listed config.xml and startWebLogic.cmd entries would now look like the following, with the changes highlighted in bold type:

config.xml

```
<Application Name="JWSQueueTransport" Deployed="true"
    LoadOrder="1000" Path="P:\weblogic81\server\lib\" TwoPhase="true">
    <EJBComponent Name="QueueTransportEJB" Targets="portalServer"
    URI="QueueTransportEJB.jar"/>
</Application>
```

startWebLogic.cmd

```
set DOMAIN_HOME=P:\weblogic81\samples\domains\portal
```

> **Note:** The new hard-coded paths do not contain C:\bea812, because C:\bea812 was mapped to P: using the subst command.

When you want to use the domain, switch to the P drive and go into the domain directory. If more developers install WebLogic Server to D:\bea, they can simply substitute that directory for P:\ by executing subst P: d:\bea and share the same config.xml and start scripts with ease.

This virtual drive option has a number of drawbacks:

- 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.

- UNIX development is not supported, though UNIX developers can use the link command.

Option 3: Using Relative Paths

If team members need to install different paths on the same drive, and the domain and application are located in a common relative path to the WebLogic Server directory, it is possible to change all file paths in config.xml and your start scripts to be relative. However, this solution is limited in its scope.

Assuming the domain is installed to C:\bea812\user_projects\mydomain, the previously listed config.xml and startWebLogic.cmd entries would now look like the following, with the changes highlighted in bold type:

config.xml

```
<Application Name="JWSQueueTransport" Deployed="true"
    LoadOrder="1000" Path="..\..\weblogic81\server\lib\" TwoPhase="true">
    <EJBComponent Name="QueueTransportEJB" Targets="portalServer"
    URI="QueueTransportEJB.jar"/>
</Application>
```

startWebLogic.cmd

```
set DOMAIN_HOME=..\..\weblogic81\samples\domains\portal
```

Problems with this solution include:

- No ability to span multiple drives.

- Project domain directory must always be in the exact same relative location to the server, even in a deployed production environment.

Creating and Sharing the Portal Domain

The first step the team lead needs to take is the creation of the new portal domain. There are several phases of domain creation, including creating your own domain template, performing the domain Configuration Wizard process, initially checking-in to source control, and configuring the domain.

Best Practices for Creating a Portal Domain to Share with a Team

- Create a custom domain template before creating a domain with the Configuration Wizard. A custom domain template determines what is in the domain you create, and you can store the template in source control. The domain Configuration Wizard builds the domain using your template. For instructions on using the Configuration Template Builder to create a domain template, see *Creating Configuration Templates Using the WebLogic Configuration Template Builder*.

- Select *Development Mode* when you use the Configuration Wizard to create a new domain for your team development environment. For instructions on running the Configuration Wizard to create a new portal domain, see *Creating WebLogic Configurations Using the Configuration Wizard*.

- Store the domain directory in a short path to avoid path-length exceptions, For example, *drive*:\ourDomain.

- Place the domain and application directories in a common parent directory to make sharing them with source control management systems easier to manage.

For example, install your domain to
WEBLOGIC_HOME\user_projects*PROJECT*\domain*DOMAIN*

And install your application to
WEBLOGIC_HOME\user_projects*PROJECT*\application*APP*

Note: Your paths can be shorter and outside the BEA installation hierarchy.

This approach lets you have a common root directory (%PROJECTNAME) in your source control system's project for both the domain and application.

- 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 3-2 lists the post-startup files to exclude.

- Use a shared parent directory for your domains and applications, and use that directory as the root directory for source control.

- Add an environment variable entry to the start script to enable file logging. File logging allows WebLogic Portal messages to be sent to a file instead of just to the screen. Open the startWebLogic script in a text editor and add a path for the WLS_REDIRECT_LOG file before the if "%WLS_REDIRECT_LOG%" line. For example:

  WLS_REDIRECT_LOG=D:\logs\admin

  if "%WLS_REDIRECT_LOG%"=="" (

- Check the domain back out of source control and start the domain using the startWeblogic command after establishing your initial baseline for the domain in source control. With the server running, you may want to configure the domain so it is ready to use for your project. Using the WebLogic Server Administration Console (http://*server*:*port*/console), you can set up the domain to support the development team, including the addition of needed data sources.

  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.

  For information on server configuration, see *WebLogic Server System Administration*.

  After you make changes to the server configuration, check in config.xml (or a hand-modified version of a templated config-template.xml).

- Have another developer validate the changes by checking out the domain and starting the server without error before configuring your portal application.

Excluding Domain Files From Source Control Management

Exclude the following domain files from source control:

Table 3-2 Domain Files to Exclude from Source Control

Path	Wildcard
/ (domain root)	`config.xml` (ONLY if you are using a configuration file template. See "Setting up a Configuration File Template (config-template.xml)" on page 17.)
/	`config.xml.booted`
/	`config.xml.original`
/	`*.log`
`/logs`	`*`
`/portalServer/pstore/` (persistent file store for session beans)	`*`
`/servername/`	`*.log`
`/servername/`	`.app_poller_lastrun`
`/servername/.wlnotdelete/`	`*`
`/servername/.internal/`	`*`
`/servername/ldap/`	`*LDAPBackup*.zip`
`/servername/ldap/log/`	`*`
`/servername/logs/`	`*`

Binary Files in Source Control Management

There are a number of binary files in the WebLogic Server domain that need to be checked into source control management for the domain to function properly. These binary files may change over time for user-initiated reasons, automatic growth of index files, and so on. For this reason, it is important that developers have a good understanding of what these files are, why they change, and when to check them in and out. The emphasis of this section is explaining how to determine when you need to update those files in source control management. Some kinds of

binary files you can update include: LDAP files, security-related files, and database configuration files.

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. Changes to binaries should be initiated by a single user, typically the team lead. This reduces the chances of merge conflicts over the project life cycle.

To modify domain binary files in source control:

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. Make your changes.

5. Stop the server.

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

7. Test a clean checkout from another machine.

Users, Groups, Roles, and Entitlements – Updating LDAP

A common activity in development is the creation of a base set of users that are used to test the system. By default, WebLogic Server stores user, group, role, and entitlement information in an embedded LDAP provided by BEA. This LDAP server persists its data store to the filesystem in the `domain`/ldap directory.

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

Because the LDAP server contains information 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 3-2). 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 Administration Portal and check in the updated LDAP files to source control.

For instructions on using the WebLogic Administration Portal, see *Getting Started with Portal Administration*.

Other Security Information

Other important security files located in the domain are the `SerializedSystemIni.dat`, `DefaultAuthenticatorInit.ldift`, `DefaultAuthorizerInit.ldift`, and `DefaultRoleMapperInit.ldift` files. 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 `boot.properties` file in the domain root 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.

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, it is recommended that an Enterprise-quality database be used for development efforts, because moving data between heterogeneous databases can be a labor-intensive manual effort. See "Developing Against an Enterprise-Quality Database" on page 12.

PointBase

When creating a new portal domain, an instance of a PointBase database is created that is persisted in the root directory of the domain. New domains can also create the database objects necessary for an Enterprise-quality database. For details on configuring an Oracle, SQL Server, DB2 or Sybase database, see the *Database Administration Guide*. For details on creating new domains, see *Creating WebLogic Configurations Using the Configuration Wizard*.

The database contains a number of tables that store base WebLogic Portal and WebLogic Workshop data. For a description of the database objects for each component of WebLogic Portal, see the *Data Dictionary*. In addition, WebLogic Workshop stores some of its internal state in the database. The topic *How Do I: Configure WebLogic Workshop to Use a Different Database for Internal State?* in the WebLogic Workshop help system addresses how to move that internal state store to another database.

PointBase Development Considerations

PointBase uses two files to persist the database to the filesystem: `workshop$1.wal` and `workshop.dbn`. Because the database is persisted to the filesystem, sharing copies of the database can easily be accomplished using source control management. However, PointBase files grows incrementally over time when PointBase is used, which means that the files always appear

to have been modified by the user. Over time, the PointBase files can grow from about 3 MB to 10 or more. Developers need to be aware that they should not check in the database unless they explicitly are making changes to the underlying data directly through the WebLogic Administration Portal or the PointBase console. When a change does need to be made, there is a process to follow to keep the size of the updates to a minimum, which is outlined in the following paragraph steps:

To make changes to the database:

1. Stop the servers (WebLogic Server and PointBase).

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

   This is especially important as your PointBase files may have grown significantly since the last checkout, so a new checkout will reduce the size of those files before making your additions.

3. To modify those files in source control, follow the procedure in "Working with Binary Files" on page 3-10.

Knowing When You Are Making Changes to PointBase

In general, most activities that are accomplished using the WebLogic Administration Portal are persisted to the database, with the exception of user data, group data, entitlements, and delegated administration policies, which are persisted to the embedded LDAP. However, there may be times when you want to develop with test users with user property, which are stored in the database.

Developing Against an Enterprise-Quality Database

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

The same Enterprise-quality DBMS used in production should be used for development. For example, if you plan to deploy your application on Oracle you should develop your application on Oracle as well.

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

This methodology allows greater performance and easier maintenance of a baseline of data (with proper support from a database administrator and scripts).

Each development machine is configured to use a specific database, contained in config.xml, which is a shared file in source control management. "Setting up a Configuration File Template (config-template.xml)" on page 3-17 can help provide some mechanisms for allowing developers to share config.xml while still pointing to their unique database instance.

Sharing Information Using Unique Enterprise Quality Database Instances

To share information, a database administrator sets up a process where a developer can save an instance of his database schema. This snapshot can then be applied to other developer schemas as part of a process that those developers can initiate. Snapshots of partial pieces of the database, or the storing of a common set of DDL scripts, are also common practices.

For a description of the database objects for each component of WebLogic Portal, see the *Data Dictionary*.

In addition, WebLogic Workshop stores some of its internal state in the database. The following topic in the WebLogic Workshop Help System, *How Do I: Configure WebLogic Workshop to Use a Different Database for Internal State* addresses how to move that internal state store to another database.

Creating and Sharing the Portal Application

After configuring the portal domain, the team lead needs to create a new portal application. There are several phases of application creation, including creating the application and any number of portal Web projects with WebLogic Workshop and initial check-in to source control.

For instructions on creating a new portal application and Web project, see *Creating a Portal Application and Portal Web Project*.

Be sure to install any services necessary to your application, such as Commerce and Pipeline, as well as any necessary tag libraries in each portal Web project, such as Commerce and Webflow (for compatibility with legacy portal web applications).

Creating the Domain and Application Directories

As mentioned in "Best Practices for Creating a Portal Domain to Share with a Team" on page 3-7, placing the domain and application directories in a common parent directory makes sharing them in source control management systems easier to manage.

For example, install your domain to: $drive$:\ourDomain and your application to: $drive$:\ourApp

This approach lets you have a common root directory (%$PROJECTNAME$) in your source control system's project for both the domain and application.

Note: To avoid path length exceptions, store the domain and application directories in a short path. For example, $drive$:\ourDomain.

Checking in the WebLogic Workshop Application

Once the WebLogic Workshop application has been constructed, the team lead should check the application into source control. This should be done before doing a build of the application, because a number of the files created during a build should not be checked into source control.

Excluding Portal Application Files From Source Control Management

Exclude the following application files from source control:

Table 3-3 Application Files to Exclude from Source Control

Path	Wildcard
/ (portal application root)	Each "EJB Project" contains a .jar file. These should be excluded.
/	.beabuild.txt
/APP-INF/lib/	Each "Java Project" contains a .jar file. These should be excluded.
/*project*/WEB-INF/ .pageflow-struts-generated/	*
/.workshop/	*

Files that are excluded from source control include compiled JARs, temporary configuration files, and the output directory for WebLogic Workshop.

Managing Checkouts of the WebLogic Workshop Application

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

When a build is initiated in WebLogic Workshop, a number of JAR files and temporary directories are created inside the WebLogic Workshop application itself. These files are listed in Table 3-3. In addition, WebLogic Workshop creates some additional files such as stateless session beans for controls in the `.workshop` directory of the application. These stateless beans often have names like `TimerControl_-1n8kn2z7skxv`.

When the application is deployed to the domain by WebLogic Workshop, it is registered in `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 3-1 shows the block added to `config.xml` for an application named `portalpm`.

Listing 3-1   Application Added to config.xml File

```
<Application Name="portalpm" Path="P:\user_projects\applications"
    StagingMode="nostage" TwoPhase="true">
    <WebAppComponent Name="portalpmAdmin" Targets="portalServer"
     URI="adminPortal.war"/>
    <EJBComponent Name="content.jar" Targets="portalServer"
     URI="content.jar"/>
    <EJBComponent Name="content_repo.jar" Targets="portalServer"
     URI="content_repo.jar"/>
    <WebAppComponent Name="portalpmDatasync" Targets="portalServer"
    URI="datasync.war"/>
    <EJBComponent Name="netuix.jar" Targets="portalServer"
     URI="netuix.jar"/>
    <EJBComponent Name="p13n_ejb.jar" Targets="portalServer"
     URI="p13n_ejb.jar"/>
    <EJBComponent Name="prefs.jar" Targets="portalServer"
     URI="prefs.jar"/>
    <WebServiceComponent Name="portalpmTool" Targets="portalServer"
     URI="wps-toolSupport.war"/>
    <EJBComponent Name="wps.jar" Targets="portalServer" URI="wps.jar"/>
    <ApplicationConfiguration Name="portalpm" Targets="portalServer"/>
</Application>
```

Because WebLogic Workshop updates the `config.xml` for the domain automatically, it is not necessary to check in a `config.xml` that contains the application name XML block. 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 deployed to `config.xml` with all the required references to the newly built application components. If the application name

XML block is checked in with `config.xml`, WebLogic Workshop automatically updates it if necessary to add or remove components.

Note: If you are using a configuration file template, as described in "Setting up a Configuration File Template (config-template.xml)" on page 3-17, the basic `config.xml` has already been created on each development machine using the configuration file template, and application modifications or additions are correctly added to each developer's `config.xml` file.

There are two other files that store the components for the application: `application.xml` and `weblogic-application.xml`, which are found in the application's `META-INF` directory. These files need to be shared in source control, and when new components are added to the application (such as a new EJB), the updated `application.xml` and `weblogic-application.xml` must be checked back in to source control.

WebLogic Portal Coding Practices

This section provides best practices guidance for storing portal application source code.

Java Projects

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

Cross-Platform Support

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

- Do not use spaces in filenames.

- Keep filenames short (older versions of TAR do not support long filenames).

- Use forward slashes (/) in paths when possible.

- Be aware of the difference between case-sensitive operating systems (UNIX) and case-preservative operating systems (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.

Definition Labels for Portal Components

As you add books, pages, and portlets to portal desktops in WebLogic Workshop, a Definition Label (unique ID) is automatically generated for each component (which appears in the Property Editor window). With multiple developers creating new portal components, it is possible that different components can have the same automatically generated Definition Labels. To avoid duplicate Definition Labels, manually change the Definition Label for each new component using your own naming conventions.

Cluster Configuration

Any code you write should be tested often in a clustered environment. Also important is keeping session data to a manageable size and configuring your web applications to support session sharing across the cluster. For clustering information, see Chapter 4, "Configuring a Portal Cluster".

Setting up a Configuration File Template (config-template.xml)

When working in a team development environment, your team members need to work with the same `config.xml` file so that all modifications to existing deployed applications, the addition of new applications, and other settings stored in `config.xml` file can be shared. However, there are configuration settings that may need to vary from user to user. The most common variations are when developers are using different BEA home directories or their own database instances. Developers typically have different database logins and need different settings for their JDBC connection pools.

A configuration file template addresses this problem by letting you distribute a templatized domain configuration file, `config-template.xml`. Developers share this `config-template.xml` file through source control.

Exclude config.xml From Source Control

When you need unique domain configurations in each developer environment, *exclude the config.xml file from source control*. Check in `config-template.xml` instead.

You can use Ant or another script language to set up a process that copies the `config-template.xml` over the `config.xml` file. Next, certain strings in the `config.xml` file are replaced with strings that each developer defines in a properties file.

For example, you can start with a `config-template.xml` file that contains the following:

```
<JDBCConnectionPool DriverName="weblogic.jdbc.oci.Driver"
    MaxCapacity="10" Name="Arcadia"
    Properties="user=ARCADIAUSER;password=ARCADIAPASSWORD;server=arcadia"
    RefreshMinutes="10" Targets="myserver"
    URL="jdbc:weblogic:oracle"/>
```

and a particular user has a `local.configtemplate.properties` file with the following two entries:

```
ARCADIAUSER=john
```

```
ARCADIAPASSWORD=mypassword
```

After running the replacement process, that user ends up with a `config.xml` file that reads:

```
<JDBCConnectionPool DriverName="weblogic.jdbc.oci.Driver"
    MaxCapacity="10" Name="Arcadia"
    Properties="user=john;password=mypassword;server=arcadia"
    RefreshMinutes="10" Targets="myserver"
    URL="jdbc:weblogic:oracle"/>
```

Each user need only keep his own copy of the `local.configtemplate.properties` file, which should *not* be checked into source control. A file named `configtemplate.properties` should be distributed in source control to serve as an example of a valid `local.configtemplate.properties` file.

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 3-1, 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 3-1 Multiple Enterprise Applications in a Single Domain
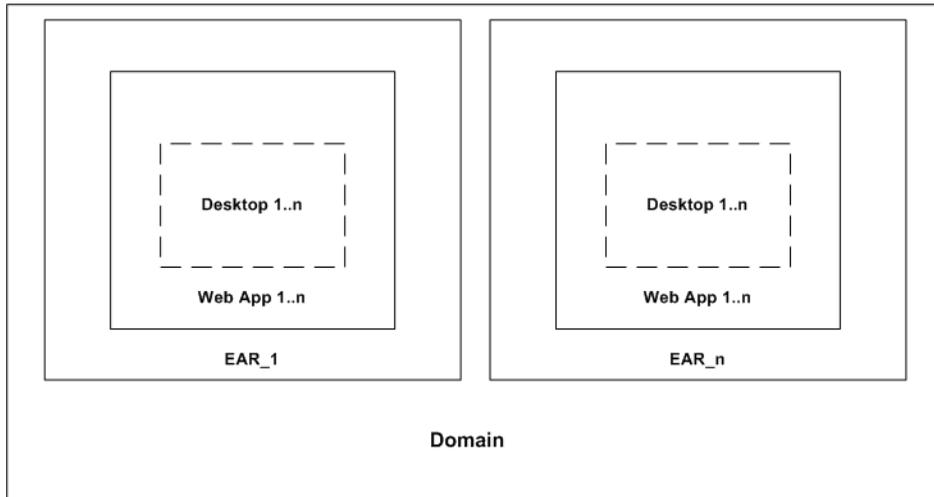
The following restrictions apply to this configuration of multiple enterprise applications in a single domain:

- **Database** – If you place multiple enterprise applications in a single domain, all applications must share the same database. The use of multiple databases in this configuration is not supported.

- **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 Administration Portal shows all of the portal web applications that are deployed to the server.

Note: Each enterprise application must be managed through its respective Administration Portal. Some domain-level resources, such as content, users, and groups, can be viewed and managed across enterprise applications from a single Administration Portal; however, be aware that data in one application may be cached, and updates to the same data made from another application's Administration Portal 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 repository datasource (one per repository) to be XA-enabled. For more

information regarding XA connections, see
http://edocs.bea.com/platform/docs81/confgwiz/examples.html#1074297.

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

● **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.

Additional Files Recommended in Source Control

For releases prior to SP4, BEA recommended not storing the class files in the path
`WEB-INF/classes` because these are built automatically by WebLogic Workshop. However, in
SP4 several new classes were added in the path:

```
WEB-INF/classes/com/bea/jsptools/portal
```

for which source files are not provided, specifically:

● `PortalManager.class`

● `PortalLogger.class`

● `PortalBeanManager.class`

● `PortalAdminLogger.class`

● `DefaultPortalLogger.class`

Because of this change, SP4 portal projects based on WebLogic Workshop will not build
successfully when extracted from their source control repository.

To ensure a successful build of your portal project while using source control, you can create a
`.jar` file containing the class files listed above, and check the `.jar` file into the source control
system repository.

Configuring a Portal Cluster

This chapter describes the steps necessary to set up a cluster across which your portal application is deployed. The topics discussed in this chapter include:

- Setting up a Production Database

- Reading the wlw-manifest.xml File

- Choosing a Cluster Architecture

- Configuring a Domain

- Understanding Portal Resources

- Zero-Downtime Architectures

Setting up a Production Database

To deploy a portal application into production, it is necessary to set up an Enterprise-quality database instance. PointBase is supported only for the design, development, and verification of applications. It is not supported for production server deployment.

Details on configuring your production database can be found in the Database Administration Guide.

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*. A simpler option, described in this chapter, is to create the DDL and DML from the domain Configuration Wizard when configuring your production environment.

Reading the wlw-manifest.xml File

When configuring your production servers or cluster with the domain Configuration Wizard, you need to deploy JMS queues that are required by WebLogic Workshop-generated components that are deployed at run time. To find the JMS queue names you need, open the `wlw-manifest.xml` file in the portal application's `/META-INF` directory.

In the file, find the JMS queue JNDI names that are the defined values in elements named `<con:async-request-queue>` and `<con:async-request-error-queue>`. Record the JNDI names of the JMS queues found in those definitions for use when configuring your production system.

You may also need to configure other settings in `wlw-manifest.xml`. For more information, see *How Do I: Deploy a WebLogic Workshop Application to a Production Server?*.

Choosing a Cluster Architecture

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.

Single Cluster

When setting up an environment to support a production instance of a portal application, the most common configuration is to use the *WebLogic Recommended Basic Architecture*.

Figure 4-1 shows a WebLogic Portal-specific version of the recommended basic architecture.

Figure 4-1 WebLogic Portal Single Cluster Architecture
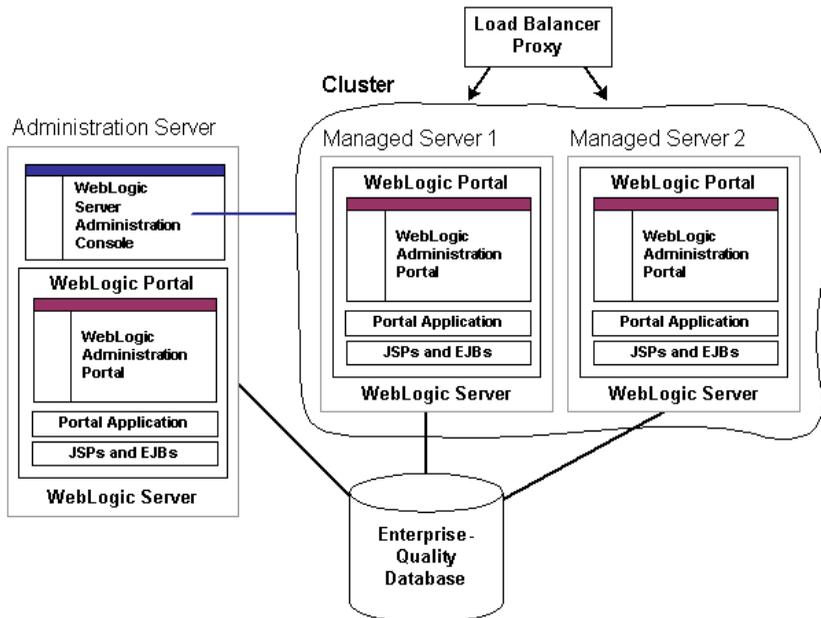
Note: WebLogic Portal does not support a split-configuration architecture where EJBs and JSPs are split onto different servers in a cluster. The basic architecture provides significant performance advantages over a split configuration for Portal.

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.

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 4-17. If this environment is of interest, you may want to review that section now.

Figure 4-2 Weblogic Portal Multi-Cluster Architecture
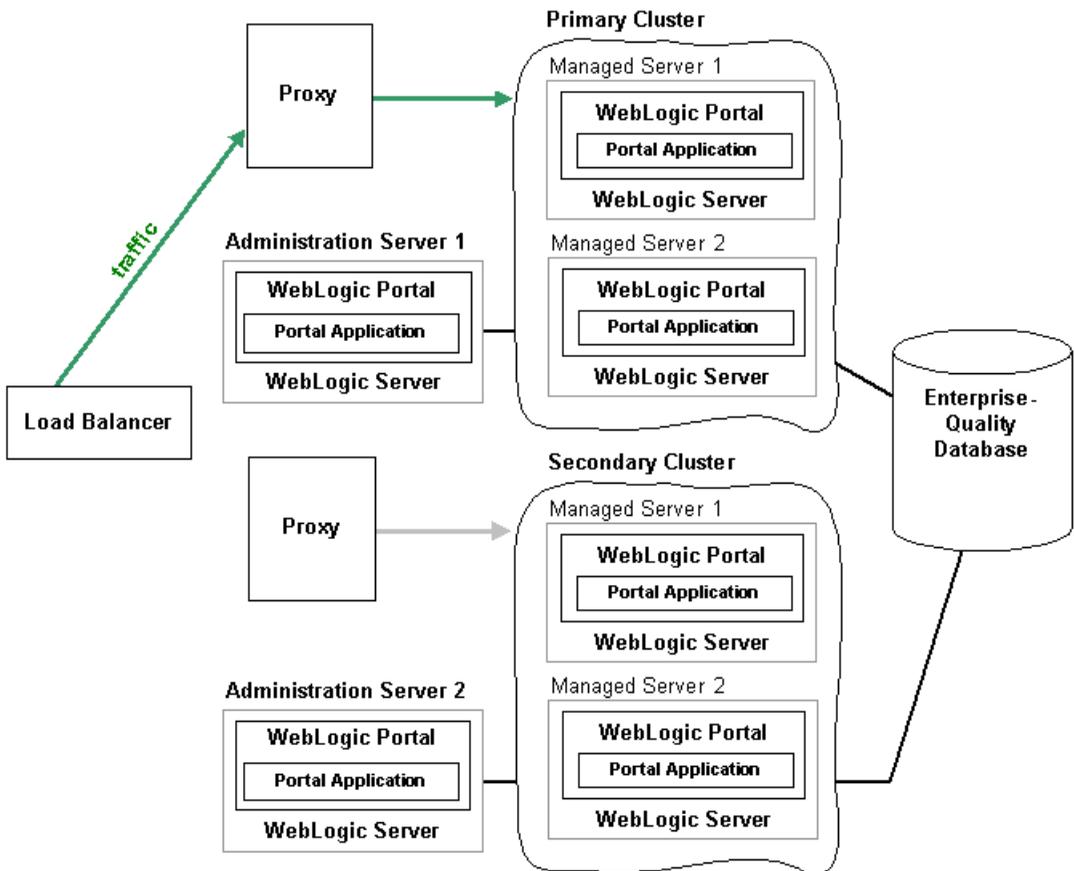
Configuring a Domain

You should determine the network layout of your domain before building your domain with the Configuration Wizard. Determine the number of managed servers you will have in your cluster—the machines they will run on, their listen ports, and their DNS addresses. Decide if you will use WebLogic Node Manager to start the servers. For information on Node Manager, see *Configuring and Managing WebLogic Server*.

WebLogic Portal must be installed on the cluster's administration server machine and on all managed server machines.

Using the Configuration Wizard

Create your new production environment with the domain Configuration Wizard. See *Creating WebLogic Configurations Using the Configuration Wizard*.

Creating a Production Cluster Environment with the Configuration Wizard

This section guides you through the creation of a production cluster environment for WebLogic Portal.

1. Start the Configuration Wizard. In Windows, choose **Start > Programs > BEA WebLogic Platform 8.1 > Configuration Wizard**.

2. In the Create or Extend a Configuration window, select **Create a new WebLogic configuration** and click **Next**.

3. In the Select a Configuration Template window, select **Basic WebLogic Portal Domain** and click **Next**.

4. In the Choose Express or Custom Configuration window, select **Custom** and click **Next**.

5. In the Configure the Administration Server window:

   a. Enter a name for your administration server.

   b. In the **Listen Address** field, leave the default "All Local Addresses" selection.

   c. Enter the listen port.

   d. If you want to use the Secure Sockets Layer (SSL) protocol for secure access to portal application resources, select the SSL enabled option and enter an SSL listen port.

e. Click **Next**.

6. In the Managed Servers, Clusters, and Machines Options window, select **Yes** to customize the configuration settings, and click **Next**.

7. In the Configure Managed Servers window, add your managed servers. The number of managed servers you want in your cluster(s) will vary depending on your choice of hardware.

   In the Listen Address field, enter the IP address for each managed server.

   Note: Do not leave the default "All Local Addresses" setting.

   When you are finished adding managed servers, click **Next**.

8. In the Configure Clusters window, add your cluster(s). Choose a multicast address that is not currently in use. Choose the Cluster addresses for the managed servers in this cluster. These take the form of a comma-separated list of the DNS alias names of the managed servers.

   See "Cluster Address" in *Using WebLogic Server Clusters* for more information.

   When you are finished, click Next.

9. In the Assign to Clusters window, choose all the managed servers you want to associate with each cluster by moving the server names from the left pane to the right pane. Click **Next**.

10. In the Configure Machines window, you can create logical representations of the systems that host your server instances. Host information is used for locality routing, especially during session replication. If you are running more than one managed server per machine, it is important that you configure host information so that WebLogic Server does not replicate a session on the same machine.

    Also, if you are using Node Manager to manage starting and stopping your servers, you should specify that information here.

    Click Next.

11. If you choose to Configure Machines, in the Assign Servers to Machines window, target the servers to the appropriate machines.

12. In the Database (JDBC) Options window, select **Yes** to define JDBC components, and click **Next**.

13. In the Configure JDBC Connection Pools window, there will be a cgPool tab. Change the cgPool Vendor to use your production database type, and then specify the information needed to connect to that database instance such as the host and port information.

    Make the same changes on the cgJMSPool-nonXA and portalPool tabs.

    For cgJMSPool-nonXA, in the Driver field, be sure to select the non-XA driver.

    Click **Next**.

14. In the Configure JDBC Multipools window, click **Next**.

15. In the Configure JDBC Data Sources window, you should see a list of JDBC Data Sources configured. Click **Next**.

16. In the Test JDBC Connection Pool and Setup JDBC Database window, select **cgPool** in the Available JDBC Connection Pools pane, and click **Test Connection**.

    If you have not already created the database objects for the portal application in your database instance, select your database version in the **DB Version** field, and click Load Database.

    Warning: Exercise caution when loading the database, as the scripts will delete any portal database objects from the database instance if they already exist. You will see a large number of SQL statements being executed. It is normal for the scripts to have a number of statements with errors on execution, because the script drops objects that may not have been created before.

    Click **Next**.

17. In the Messaging (JMS) Options window, select **Yes** to define JMS components, and click **Next**.

18. In the Configure JMS Connection Factories window, you should see the cgQueue. Its Default delivery mode should be set to Persistent. Click **Next**.

19. In the Configure JMS Destination Key(s) window, click **Next**.

20. In the Configure JMS Template(s) window, click **Next**.

21. In the Configure JMS File Stores window, validate that FileStore exists, and click **Next**.

22. In the Configure JMS JDBC Store window, validate that you have JMS stores for the administration server and for each of the managed servers, typically named `cgJMSStore_auto_1`, `cgJMSStore_auto_2`, and so on.

    To create a JMS store:

a. Click **Add**.

b. Specify a name, such as `cgJMSStore_auto_a` for the administration server.

c. In the Connection Pool field, select the same connection pool used for all stores (such as cgJMSPool-nonXA).

d. In the Prefix Name field, specify a unique JMS prefix name (such as `por_a` for the administration server).

   If you are using Oracle as a database, specify the Oracle schema name before the prefix name. For example, `OSCHEMA.por_a`.

e. Click **Next**.

23. In the Configure JMS Servers window, validate that you have servers (including the administration server) that correspond to the JMS stores created, typically named cgJMSServer_auto_1, cgJMSServer_auto_2, and so on. To create a JMS server:

a. Click **Add**.

b. Specify a name, such as `cgJMSServer_auto_a` for the administration server.

c. In the Store field, select the corresponding JMS JDBC store you created in the previous window.

d. Click **Next**.

24. In the Assign JMS Servers to WebLogic Servers window, assign the JMS Servers to the administration server and to each of the managed servers. Click **Next**.

25. In the Configure JMS Topics window, click **Next**.

26. In the Configure JMS Queues window, click **Next**.

27. In the Configure JMS Distributed Topics window, click **Next**.

28. In the Configure JMS Distributed Queues window, you need to add new queues that are required by WebLogic Workshop. The JNDI names for these queues can be found in your application's `/META-INF/wlw-manifest.xml` file, as described in "Reading the wlw-manifest.xml File" on page 4-2.

   These names will be similar to *WEB_APP*`.queue.AsyncDispacher` and *WEB_APP*`.queue.AsyncDispacher_error`. For each queue, add a new JMS Distributed Queue with the **Add** button. Set the Name entry and JNDI name entry to the name listed in

`wlw-manifest.xml`. Set the Load balancing policy and Forward delay as appropriate for your application.

A pair of queue entries exists for each web application (portal web project) in your portal application. When you are finished, you should have a distributed queue for each queue. In other words, if your Enterprise application has three web applications, you should have added six distributed queues—two for each web application.

You do not need to create queues for the WebLogic Administration Portal web application.

Note: If you are using multiple clusters, create an additional set of queues for each cluster. For example, if you have a web application called basicWebApp, and you are using a second cluster, create a unique basicWebApp queue for that cluster named something like `basicWebApp.queue.AsyncDispatcher.2`. When you do this, the Configuration Wizard does not let you enter the same JNDI name for multiple queues. In this example, enter a JNDI name that ends with a "`.2`". Later in the setup procedures you will make all JNDI names the same in the WebLogic Administration Console for each web application.

Click **Next**.

29. In the Assign JMS Distributed Destinations to Servers or Clusters window, target your newly defined queues to your cluster. In the right pane, select the cluster, in the left pane, select the queue(s), and click the right arrow icon. Click **Next**.

30. In the JMS Distributed Queue Members window, click Next. You will create distributed JMS queue members in "Setting up JMS Servers" on page 4-11.

31. In the Applications and Services Targeting Options window, select **Yes** and click Next.

32. In the Target Applications to Servers or Clusters window, target all applications to the administration server as well as to the cluster. Click **Next**.

33. In the Target Services to Servers or Clusters window, click **Next**.

34. In the Configure Administrative Username and Password window, enter a username and password for starting the administration server. You do not need to configure additional users, groups, and global roles, so make sure **No** is selected at the bottom of the window. Click **Next**.

35. If you are installing on Windows, in the Configure Windows Options window, select the options you want for adding a shortcut to the Windows menu and installing the administration server as Windows service. The wizard always creates a single default shortcut on the windows menu regardless of what you select for the Create Start Menu option. The option lets you create an additional shortcut with different settings.

If you chose to create a Windows menu shortcut for your domain, click **Next** in the Build Start Menu Entries window.

Click **Next**.

For information on starting WebLogic Server, see *Creating Startup Scripts*.

36. In the Configure Server Start Mode and Java SDK window, select **Production Mode** and select the SDK (JDK) you want to use. Click **Next**.

37. In the Create WebLogic Configuration window, browse to the directory where you want to install your administration server domain, and enter a Configuration Name.

   To avoid path length exceptions, use a short path for the domain, such as *drive*:/ourDomain.

38. Click **Create**.

39. When the domain is created, click **Done**.

Configuring the Administration Server

At this point your administration server domain has been configured using the domain Configuration Wizard. Before you start the administration server to do additional configuration work, you may want to perform one or both of the following setup tasks:

- increase the default memory size allocated to the administration server
- allow server startup without requiring authentication by creating a boot.properties file

Increasing the Default Memory Size

To increase the default memory size allocated to the administration server, you need to modify your startWebLogic script in the domain's root directory and change the memory arguments. For example:

In Windows, change:

```
set MEM_ARGS=-Xms256m %memmax% to set MEM_ARGS=-Xms512m -Xmx512m
```

In UNIX, change:

```
MEM_ARGS="-Xms256m ${memmax}" to MEM_ARGS="-Xms512m -Xmx512m"
```

The exact amount of memory you should allocate to the server will vary based on a number of factors such as the size of your portal application, but in general 512 Mb of memory is recommended as a default.

Allowing Server Startup Without Requiring Authentication

To allow server startup without requiring authentication, create a `boot.properties` file in your domain root directory that contains the username and password you want to log in with. For example:

```
username=weblogic
password=weblogic
```

After the server starts for the first time using this file, the username and password are encrypted in the file.

Setting up JMS Servers

In this procedure you finish configuration of the JMS servers.

1. Start the administration server and log in to the WebLogic Server Administration Console, found at `http://server:port/console`.

2. Configure the JMS distributed destinations.

   a. Expand **Services > JMS > Distributed Destinations**. Perform the following sub-steps for each queue you defined earlier for the WebLogic Workshop components.

      Note: You do not need to configure the `dist_cgJWSQueue_auto` queue.

   b. Select the queue name, and select the **Auto Deploy** tab.

   c. Click **Create members on the selected Servers (and JMS Servers)**.

   d. Select your cluster(s) to target the JMS queue to, and click **Next**.

   e. Select all the managed servers in the cluster to create members for the queue, and click **Next**.

   f. Select all the JMS Servers where members will be created, and click Next.

   g. Commit the changes by clicking **Apply**.

   h. If you are using multiple clusters, change the JNDI names you assigned to the cluster queues in Creating a Production Cluster Environment with the Configuration Wizard step 28. Select each cluster queue (for example, select `basicWebApp.queue.AsyncDispatcher.2`), click the **General** tab, and remove the "`.2`" suffix (or whatever unique identifier you used in the Configuration Wizard) so that all JNDI names are the same for each web application. For example, all JNDI names for

the basicWebApp should be `basicWebApp.queue.AsyncDispatcher`. After each change, click **Apply**.

3. Configure the JMS servers for the managed servers.

   a. Expand **Services > JMS > Servers >** *managed server JMS server name* **> Destinations**. Perform the following sub-steps for each member queue ending in `AsyncDispatcher` (but not for member queues ending in `AsyncDispatcher_error`).

   b. Select the queue, and select the **Redelivery** tab.

   c. In the **Error Destination** field, select the companion error queue for the queue.

   d. Click **Apply**.

4. Create JMS queues for the administration server.

   a. Select **Services > JMS > Servers >** *administration JMS server name* **> Destinations**.

   b. Click **Configure a New JMS Queue**.

   c. Create JMS queues and error queues for each web application. Use any name, but for convention you can make the Name the same as the JNDI Name.

      View your application's `META-INF/wlw-manifest.xml` file to see the queues you must create. See "Reading the wlw-manifest.xml File" on page 4-2.

      For example, if you have two web applications, basicWebApp and bigWebApp, create the following queues for the administration server JMS server:

      ```
      basicWebApp.queue.AsyncDispatcher
      basicWebApp.queue.AsyncDispatcher_error
      bigWebApp.queue.AsyncDispatcher
      bigWebApp.queue.AsyncDispatcher_error
      ```

      You must also create a single queue named `jws.queue` if it does not already exist.

      Click **Create** after you create each queue.

5. Retarget the JMS servers to "migratable" to support JMS failover.

   a. Expand **Services > JMS > Servers**. Perform the following sub-steps for each JMS server.

   b. Select the JMS server, and select the **Target and Deploy** tab.

   c. In the **Target** field, select the *target* (migratable) counterpart item for the previously
       selected target. For example, if the target was `managed1`, select
       `managed1 (migratable)`.

   d. Click **Apply**.

Creating Managed Server Directories

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).

- 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.

WebLogic Portal must be installed on all managed servers.

1. To create a new managed server, launch the Configuration Wizard.

2. Choose **Create a new WebLogic configuration**, and click **Next**.

3. In the Select a Configuration Template window, select Basic WebLogic Portal Domain, and
    click **Next**.

4. In the Choose Express or Custom Configuration window, select **Express**, and click **Next**.

5. In the Configure Administrative Username and Password window, enter a username and
    password for the server, and click **Next**.

    This information is not typically be used, because you bind this server to the administration
    server using the administration server's credentials.

6. In the Configure Server Start Mode and Java JDK, select **Production Mode** and the SDK (JDK) you will use with the domain. Click **Next**.

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

7. In the Create WebLogic Configuration window, choose the directory you want to install to, and in the **Configuration Name** field, enter a domain name to use. For best practices, choose a domain name like `managedServer1`, `managedServer2`, and so on.

8. Click **Create**.

9. When the domain is created, click **Done**.

10. If you want to allow server startup without requiring authentication on each managed server, create a `boot.properties` file in each managed server's domain directory (or one level above the server directory) that contains a username and password. For example:

    ```
    username=weblogic
    password=weblogic
    ```

    After the initial server startup using `boot.properties`, the username and password are encrypted in the file.

Once you have created a filesystem domain directory 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).

Increasing the Default Memory Size

To increase the default memory size allocated to a managed server (if you are not using Node Manager), you need to modify your `startManagedWebLogic` script in the managed server root directory and change the memory arguments. For example:

In Windows, change:

```
set MEM_ARGS=-Xms256m %memmax% to set MEM_ARGS=-Xms512m -Xmx512m
```

In UNIX, change:

```
MEM_ARGS="-Xms256m ${memmax}" to MEM_ARGS="-Xms512m -Xmx512m"
```

The exact amount of memory you should allocate to the server will vary based on a number of factors such as the size of your portal application, but in general 512 megabytes of memory is recommended as a default.

Understanding Portal Resources

The Portal Library contains books, pages, layouts, portlets, desktops, and other types of portal-specific resources. Using the WebLogic Administration Portal, these resources can be created, modified, entitled, and arranged to shape the portal desktops that end users access.

Figure 4-3 shows an image of the Portal Resources tree in the WebLogic Administration Portal. The tree contains two main nodes: Library and Portals. The Library node contains the global set of portlets and other resources, while the Portals node contains instances of those resources, such as the colorsSample desktop and its pages, books, and portlets.

Figure 4-3 PortAl Resources Library
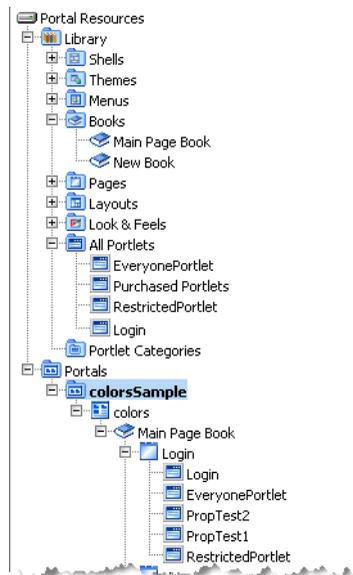
Each of these resources is defined partially in the portal database so they can be easily modified at run time. The majority of resources that exist are created by an administrator, either from scratch or by creating a new desktop from an existing `.portal` template file that was created in WebLogic Workshop.

However, portlets themselves are created by developers and exist initially as XML files. In production, any existing `.portlet` files in a portal application are automatically read into the database so they are available to the WebLogic Administration Portal.

The following section addresses the life cycle and storage mechanisms around portlets, because their deployment process is an important part of portal administration and management.

Understanding the Portlet Deployment Life Cycle

During development time, `.portlet` files are stored as XML in any existing portal web application in the Portal EAR. As a developer creates new `.portlet` files, a file poller thread monitors changes and loads the development database with the `.portlet` information.

In a production environment, `.portlet` files are loaded when the portal web application that contains them is redeployed on the administration server. This redeployment timing ensures that the content of the portlet, such as a JSP or Page Flow, is available at the same time as the `.portlet` file is available in the Portal Library. The administration server is the chosen master responsible for updating the database so that there are no issues around every server in the production cluster trying to write the new portlet information into the database at the same time. When deploying new portlets to a production environment, target the portal application for redeployment on the administration server.

Understanding the Database Structure for Storing Portlets

When a portlet is loaded into the database, the portlet XML is parsed and a number of tables are populated with information about the portlet, including `PF_PORTLET_DEFINITION`, `PF_MARKUP_DEFINITION`, `PF_PORTLET_INSTANCE`, `PF_PORTLET_PREFERENCE`, `L10N_RESOURCE`, and `L10N_INTERSECTION`.

`PF_PORTLET_DEFINITION` is the master record for the portlet and contains rows for properties that are defined for the portlet, such as the definition label, the forkable setting, edit URI, help URI, and so on. The definition label and web application name are the unique identifying records for the portlet. Portlet definitions refer to the rest of the actual XML for the portlet that is stored in `PF_MARKUP_DEF`.

`PF_MARKUP_DEF` contains stored tokenized XML for the `.portlet` file. This means that the `.portlet` XML is parsed into the database and properties are replaced with tokens. For example, the following code fragment shows a tokenized portlet:

```
<netuix:portlet $(definitionLabel) $(title) $(renderCacheable)
$(cacheExpires)>
```

These tokens are replaced by values from the master definition table in PF_PORTLET_DEFINITION, or by a customized instance of the portlet stored in PF_PORTLET_INSTANCE.

The following four types of portlet instances are recorded in the database for storing portlet properties:

- **Primary** – Properties defined in development and stored in the .portlet file.

- **Library** – Properties defined in the Portal Library, which may be changed using the WebLogic Administration Portal.

- **Admin** – A customized instance of the portlet in a desktop. This allows you to customize a portlet in a particular way for a desktop without affecting other instances of the portlet in other desktops.

- **User** – User-customized instances of the portlet defined in the Visitor Tools.

PF_PORTET_INSTANCE contains properties for the portlet for attributes such as DEFAULT_MINIMIZED, TITLE_BAR_ORIENTATION, and PORTLET_LABEL.

If a portlet has portlet preferences defined, those are stored in the PF_PORTLET_PREFERENCE table.

Finally, portlet titles can be internationalized. Those names are stored in the L10N_ RESOURCE table which is linked using L10N_INTERSECTION to PF_PORTLET_DEFINITION.

Removing Portlets from Production

If a portlet is removed from a newly deployed portal application, and it has already been defined in the production database, it is marked as IS_PORTLET_FILE_DELETED in the PF_PORTLET_DEFINITION table. It displays as grayed out in the WebLogic Administration Portal, and user requests for the portlet if it is still contained in a desktop instance return a message that says the portlet is unavailable.

Zero-Downtime Architectures

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 4-4. 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 4-4 During Normal Operations, Traffic Is Sent to the Primary Cluster
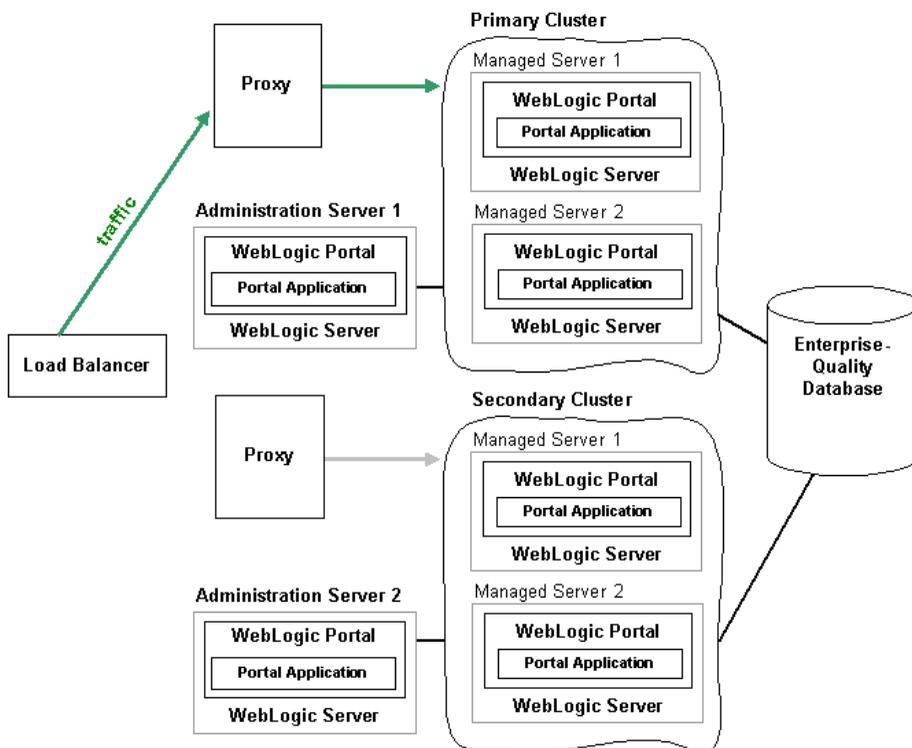
All traffic is routed to the secondary cluster, then the primary cluster is updated with a new Portal EAR, as shown in Figure 4-5. This EAR has a new portlet, which is loaded into the database. Routing requests to the secondary cluster is a gradual process. Existing requess to the primary

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 4-5 Traffic Is Routed to the Secondary Cluster; The Primary Cluster Is Updated
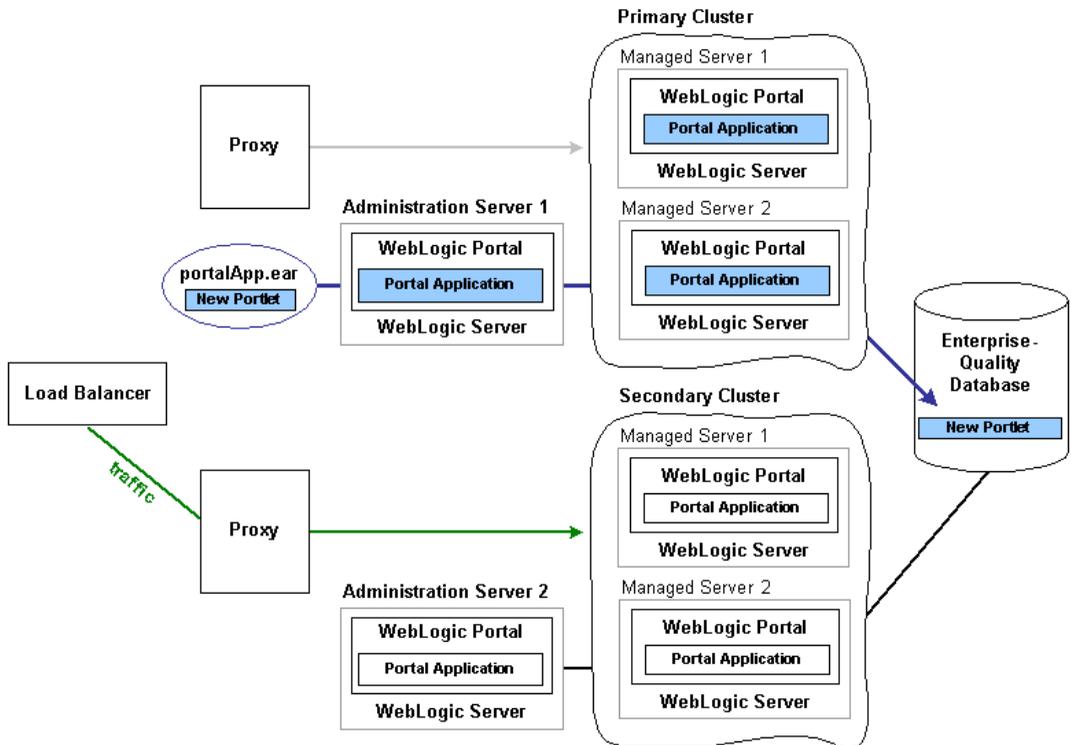
All traffic is routed back to the primary cluster, and the secondary cluster is updated with the new EAR, as shown in Figure 4-6. Because the database was updated when the primary cluster was updated, the database is not updated when the secondary cluster is updated.

Figure 4-6 Traffic Is Routed Back to the Primary Cluster; The Secondary Cluster Is Updated
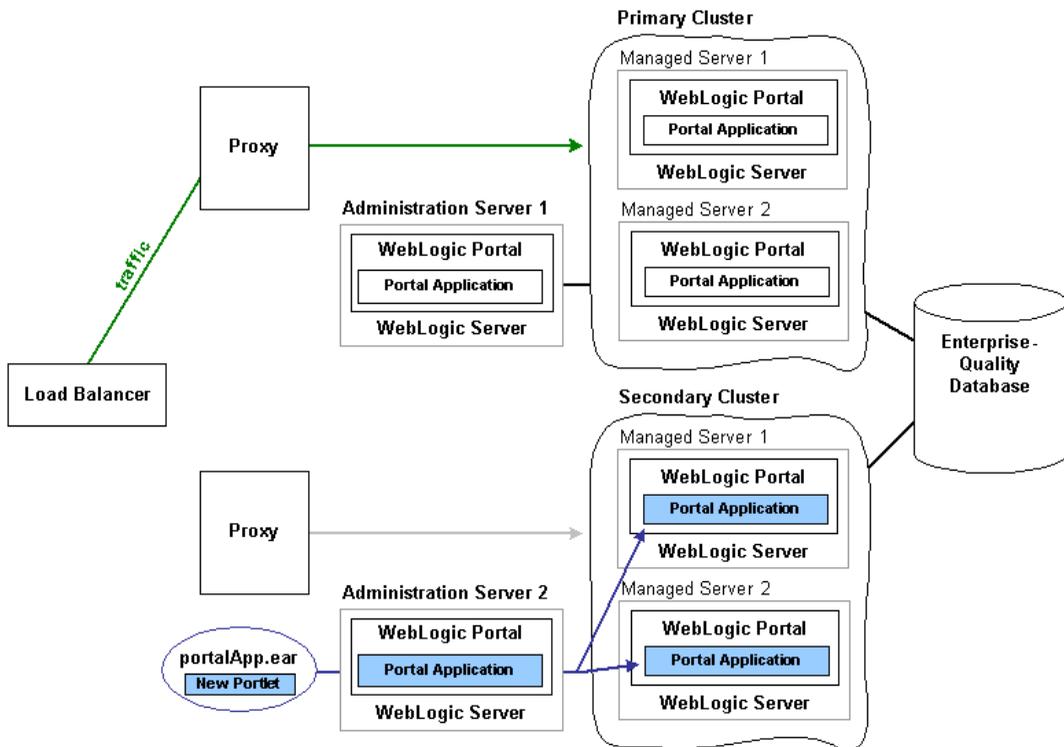
Even though the secondary cluster does not receive traffic under normal conditions, you must still update it with the current portal application. When you next update the portal application, the secondary cluster temporarily receives requests, and the current application must be available.

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 Administration Portal 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. First, you can delay adding the portlet to any desktop instances until all users are back on the primary cluster. Another option is to 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 simultaneoiusly 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. The portal cache is configured in the WebLogic Administration Console under

Configuration Settings > Service Administration > Cache Manager. 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.

Preparing and Deploying the EAR File

This chapter discusses the steps for preparing your application's Enterprise archive (EAR) file to be deployed. This chapter includes the following topics:

- Preparing the EAR File for Deployment

- Deploying the EAR for a New Application

- Deploying a Portal Application to the Cluster

Preparing the EAR File for Deployment

To bring your portal online in a production environment, it is first necessary to prepare your portal application. Typical preparation steps include modifying deployment descriptors for the product, building the Enterprise archive (EAR) with all its pre-compiled classes, and deciding if you want to compress that EAR into an archive or leave it exploded.

Configuring Portal Application Deployment Descriptors

Similar to any J2EE application, a portal application has a number of deployment descriptors that you may want to tune for your production environment.

Modifying Application Deployment Descriptors

Within the Portal application is the `/META-INF` directory that contains a number of deployment descriptors, including `application-config.xml`, a portal-specific deployment descriptor that contains cache configuration, behavior tracking, campaign, and commerce tax settings. If these

values are different for your production environment than for your existing development settings, modify this file appropriately before building the portal application.

Modifying Web Application Deployment Descriptors

Within any portal web application is a `/WEB-INF` directory that contains a number of deployment descriptors you may need to modify for your production environment.

- `web.xml` 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 *web.xml Deployment Descriptor Elements*.

  Note: If you are running your portal application on a managed server, you can improve the performance of the WebLogic Administration Portal by adding the following parameter to the `web.xml` file:

  ```
  <context-param>
     <param-name>portalFileDirectory</param-name>
     <param-value>/</param-value>
  </context-param>
  ```

  This parameter takes advantage of an optimized call to an mbean that returns EAR content information. Without this parameter, the mbean call recursively searches for `.portal` files. To avoid the overhead of this recursive call, you can place all of your `.portal` files in one directory and specify the directory with the `portalFileDirectory` parameter. In the example above, all `.portal` files reside in the web application's root directory (`/`).

  If you use this parameter, you must place all of the `.portal` files in the same directory under the portal web application. Use the `<param-value>` to specify the directory.

- `weblogic.xml` is a standard WebLogic Server deployment descriptor for web applications that has a number of important descriptor entries. For more details, see: *weblogic.xml Deployment Descriptor Elements*.

  Note: In a clustered production environment, it is important that you configure the `<session-param>` descriptor element in `weblogic.xml` to enable session replication to take place across the cluster. Without this setting, you will not have failover of a user's state information if a server in the cluster is stopped. You may need to add the following block to `weblogic.xml`.

  ```
  <session-descriptor>

  <session-param>

  <param-name>PersistentStoreType</param-name>
  ```

```
<param-value>replicated_if_clustered</param-value>
```

```
</session-param>
```

```
</session-descriptor>
```

By default if PersistentStoreType is not set, it defaults to disabling persistent session storage.

The other commonly modified element in weblogic.xml for production environments is the <jsp-descriptor>. For production, it is common to make these modifications:

- Turn off debugging by setting debug to false.

- Precompile the JSPs in the web application to reduce the time needed to display pages on their first invocation by setting precompile to true.

  Also set precompileContinue to true, because if any JSPs do not compile, deployment of the web application stops.

  You may need to add these elements to weblogic.xml inside the existing <jsp-descriptor> section. For example:

```
<jsp-param>
<param-name>precompile</param-name>
<param-value>true</param-value>
<param-name>precompileContinue</param-name>
<param-value>true</param-value>
</jsp-param>
```

- In a compressed EAR environment, set pageCheckSeconds to -1 to disable polling of JSP pages for changes.

Modifying WebLogic Workshop Deployment Descriptors

WebLogic Workshop has a number of additional deployment descriptors that are important if you are developing web services. You can find information on these descriptors in *WebLogic Workshop Internals*.

Creating Content Management Repositories

Before packaging your application into an EAR file, use the WebLogic Administration Portal to create any content management repositories you want to use in your application. This means creating only the root repositories, not the content nodes and content items. After you create repositories, they are registered in the application-config.xml deployment descriptor. When you create the application EAR, application-config.xml becomes read-only and cannot be

modified within the EAR. That is, you cannot add or remove repositories in the WebLogic Administration Portal when the application is in an EAR file.

For information on creating repositories, see *Add a New Repository Connection*.

Compiling with Your Runtime JVM

If you are going to use a particular Java Virtual Machine (JVM) in your production environment, it is a good idea to compile the EAR application with the JDK for that JVM. You can change the JVM for your WebLogic Workshop project by going to **Tools > Application Properties**, selecting **WebLogic Server**, and specifying the path to the JDK Home (root directory) you want to use.

Building a Portal Application with WebLogic Workshop

To deploy a portal application to a production environment, you must first build the application in WebLogic Workshop to compile necessary classes in the portal application. There are two options:

- **Build > Build EAR** – Most common. Use this option to compile the classes and build the portal application as a compressed EAR.

    or

- **Build > Build Application** – Use this option to compile the classes and build the portal application as an exploded (uncompressed) EAR. This option, however, many not invoke all the same processing as Build EAR. Review how WebLogic Workshop treats deployment descriptors when Build EAR is run and determine if there is an impact. See *WebLogic Workshop Internals* for more information.

    Alternatively, to generate an uncompressed EAR, you can run **Build > Build EAR** and then uncompress the generated EAR file.

    After you create an exploded EAR, 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 extension to the directory name prevents deployment exceptions for exploded EARs.

Building In the Command Line

You can build your portal application from the command line using the `wlwBuild` command. This can make it easier for you to automate the process of building your application. See *wlwBuild Command* for more information.

Deploying the EAR for a New Application

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

At this point you can deploy your portal application to the cluster. First, place the `.ear` file (or exploded EAR) on the filesystem of the administration server. To make it easier to redeploy changes to the application, place the file in a known location from which you will always deploy the application, such as the root directory of the administration domain.

The steps for initial application deployment to the administration server and managed servers, and the order with which you start the managed servers, differ in different scenarios. Table 5-1 describes different scenarios and the deployment/startup procedures that must be used for each.

Table 5-1 Initial Deployment/Startup Scenarios

Scenario	Deployment/Startup Procedures
Application not previously deployed, only administration server running – application does not use commerce (no `commerce*.jar` files in the application root directory)	1. Make sure the EAR (compressed or exploded) is in the location you want on the administration server. 2. Start the administration server. 3. Open the WebLogic Server Administration Console (`http://adminserver:port/console`) and select **Deployments > Applications**. 4. Click **Deploy a new Application**, and select the archive for the application from the filesystem. Click **Target Application**. 5. Target the application to the administration server and the cluster. 6. Deploy the application. 7. Once deployment has completed, start the managed servers however you like, such as in parallel (simultaneously).
Application not previously deployed, only administration server running – application uses commerce (has the `commerce*.jar` files in the application root directory)	Perform the previous deployment steps except the last step. When you start the managed servers, start a single managed server first. After this managed server has started, you can start the remaining managed servers in parallel.

Table 5-1 Initial Deployment/Startup Scenarios (Continued)

Scenario	Deployment/Startup Procedures
Administration server and cluster up and running (whether or not commerce is being used)	8. Make sure the EAR (compressed or exploded) is in the location you want on the administration server.
	9. In the WebLogic Server Administration Console on the administration server (`http://adminserver:port/console`), select **Deployments > Applications**.
	10. Click **Deploy a new Application**, and select the archive for the application from the filesystem. Click **Target Application**.
	11. Target the application to the administration server only.
	12. Deploy the application on the administration server.
	13. Once deployment has completed, select **Deployments > Applications > *appName***.
	14. On the **Targets** tab, target the administration server and the cluster, and click **Apply**.
	15. On the **Deploy** tab, redeploy the application.

Most components must be targeted to the administration server as well as the cluster. Here are the following exceptions:

- *AppName*Admin is not required on the administration server (though deploying *AppName*Admin on the administration server will not cause any problems).

- *AppName*Datasync is not required on the target server or cluster (though deploying *AppName*Datasync to the target server or cluster will not cause any problems).

This full level of deployment on the administration server and the cluster is required, and it is the only supported configuration. There are several application design challenges specific to clustering that WebLogic Portal solves to ensure that portal applications perform properly and optimally in a cluster environment. The full targeting scheme described is part of the solution to those design challenges.

If you want to reduce the number of modules deployed on the administration server and the cluster, click **Target Each Module** in the deployment steps and untarget *AppName*Admin from the administration server and *AppName*Datasync from the cluster.

While you need to deploy your portal application to the administration server, the administration server is not typically used to serve pages for portal applications.

Using Targeted Deployment

In the previous steps, you targeted the entire application for deployment rather than targeting individual modules to the administration server and to the cluster. This is the recommended deployment approach. Since almost all modules must be deployed to both the administration server and to the cluster, any performance or disk space benefits you want to achieve by targeting individual modules is not significant.

However, if for some reason you want to use targeted deployment, use the following recommendations, which are listed in order from easiest to most difficult.

- Avoid targeted deployment. Deploy the entire application to the administration server and to the cluster.

- Use Netscape or Mozilla browsers to perform targeted deployment in the WebLogic Administration Console.

- Perform multiple targeted deployments. Deploy half of the modules first, then deploy the rest.

- Run the `weblogic.Deployer` utility in Linux or UNIX environments. For information on using the `weblogic.Deployer` utility, see *Deployment Tools Reference*.

- Write Java code to call the `weblogic.Deployer main()` and pass in the module arguments.

- Use the WebLogic Server Scripting Tool (WLST). This option has not been tested on WebLogic Portal deployment. For information on WLST, see Code Library.

Starting Managed Servers

The sequence with which you start managed servers is important and depends upon your application deployment. See Table 5-1 for the startup sequences you must use.

If you are not deploying (a new application or an updated application), you can start all managed servers in parallel (simultaneously).

There are numerous ways to start a managed server and bind it to your administration server, including using Node Manager. For your initial setup, you may want to use the `startManagedWebLogic` script in the domain root directory. You can run this script by specifying the name of the managed server for this server instance and the URL of the administration server. Before starting the script, edit it and give the managed server more memory

than it is allocated by default. This can be done by specifying a new MEM_ARGS setting. For example, change the memory allocation to -Xms512m -Xmx512m.

After starting a managed server, you can browse your portal application by going to the appropriate URL on the managed server instance. To provide your users a single point of entry to your cluster, as well as support session failover, you will need to configure a proxy server.

Configuring Your Proxy Server

For instructions on configuring a proxy plugin for WebLogic Server, see *Configure Proxy Plugins*.

There are no WebLogic Portal-specific configuration tasks when setting up a proxy plug-in.

Troubleshooting Unresolved URLs

If you are using a proxy server or switching back and forth between non-secure and secure ports, you may find that URLs do not resolve if you use the {url:port} or {url:securePort} variables. This is because the variables for those values are read from the request. For example, if a user in a non-secure port (port 80) clicks a secure https link that was created with a URL template that uses the {url:securePort} variable, the port number of the request (80) is used for the {url:securePort} variable, which would create a secure request (https) on an non-secure port. The same could happen if a user on a proxy server (port 80) clicks a link to a resource outside the proxy server (port 443).

In both of those cases, you need to hard code port numbers in the URL templates to get URLs to resolve correctly.

For more information using URL templates, see *Creating URLs to Portal Resources* at http://e-docs.bea.com/workshop/docs81/doc/en/portal/buildportals/url.html

Deploying a Portal Application to the Cluster

This section contains instructions for redeployment, partial redeployment, and iterative deployment of datasync data, such as user profile properties, user segments, content selectors, campaigns, discounts, and other property sets.

Redeploying Applications

Use the following procedure to redeploy an application.

1. In the WebLogic Server Administration Console on the administration server
   (`http://adminserver:port/console`), select **Deployments > Applications**.

2. In the list of deployed applications, click the trash can icon next to the application you want
   to redeploy. This undeploys the current version of the application.

3. After undeployment (when the application name disappears from the list of applications),
   replace the EAR with the updated application on the filesystem.

4. Redeploy the application, targeting the administration server and the cluster.

5. Run the Propagation Utility if you have updated datasync files. For more details on what
   datasync files contain and how to use them, see Chapter 10, "Using the Datasync Web
   Application".

Redeploying a Portal Application with weblogic.Deployer

You can use the WebLogic Server Administration Console or `weblogic.Deployer` tool to
redeploy an updated portal application to your production server. See *weblogic.Deployer Utility*.

The following batch file shown in Listing 5-1 is an example of how to use `weblogic.Deployer`
to redeploy a portal application to production.

Listing 5-1 Batch File That Calls weblogic.Deployer

```
@echo off
echo Redeploys a Portal Web Project to a Server or Cluster
echo First Parameter is the name of the Server or Cluster
echo Second Parameter is the name of the Application
echo Third Parameter is the administrative username for the Portal Server
set SERVER=%1
set APPNAME=%2
set USERNAME=%3
echo server = %SERVER%
echo appname = %APPNAME%
echo username = %USERNAME%
java weblogic.Deployer -redeploy -username %USERNAME% -name %APPNAME%
-targets %SERVER%
```

Partial Redeployment with weblogic.Deployer

In certain situations you can reduce the time needed to redeploy individual pieces of a portal application by using the `weblogic.Deployer` tool.

If your updates are contained within a particular portal web application, you can redeploy just that web application and greatly reduce the time spent in redeployment. This is of use if you have new portlets and page flows, but no new EJBs, libraries, or modules (which are Enterprise application scoped).

Because a portal web application has a number of dependencies on WebLogic Workshop control classes, those needed to be redeployed as well. The following batch file can be used to help simplify that process. You need to have `weblogic.Deployer` in your classpath, which can be added by running *WEBLOGIC_HOME*/common/bin/commEnv script.

Listing 5-2   Batch File for Redeploying WebLogic Server Control Classes

```
@echo off
echo Redeploys a Portal Web Project to a Server or Cluster
echo First Parameter is the name of the Server or Cluster
echo Second Parameter is the name of the Application
echo Third Parameter is the name of the Portal Web Application
echo Fourth Parameter is the administrative username for the Portal Server
set SERVER=%1
set APPNAME=%2
set WEBAPPNAME=%3
set USERNAME=%4
echo server = %SERVER%
echo appname = %APPNAME%
echo webappname = %WEBAPPNAME%
echo username = %USERNAME%
set TARGETS=%APPNAME%@%SERVER%
set TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/TimerControl_-livsjc6qp6ws@
    %SERVER%
set TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/p13controls_k3cw9vg6497r@
    %SERVER%
set TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/MDBListener_-1x0154i4jz0he@
    %SERVER%
set TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/GenericStateless@%SERVER%
```

```
set TARGETS=%TARGETS%,.workshop/%APPNAME%/EJB/ProjectBeans@%SERVER%
java weblogic.Deployer -redeploy -username %USERNAME% -name %WEBAPPNAME%
-targets %TARGETS%
```

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 Kind of Data Can Be Propagated?

- What Tools Does BEA Provide to Assist with Propagation?

- Choosing the Right Propagation Tool

- Propagation Roadmap

- Assessing Your Portal System Configuration

- General Propagation Scenarios

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 a tool called the Propagation Utility, which is discussed in detail in Chapter 7, "Inside the Propagation Utility" and Chapter 8,

"Using the Propagation Utility". BEA also provides a tool, the Export/Import Utility, for moving portal assets from the database of a staging environment to files that can be imported back into WebLogic Workshop. The Export/Import Utility is described in Chapter 9, "Using the Export/Import Utility". This utility also lets you move and merge portal assets from a WebLogic Workshop environment to a staging environment's database.

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

Figure 6-1 Moving Portals Between Environments
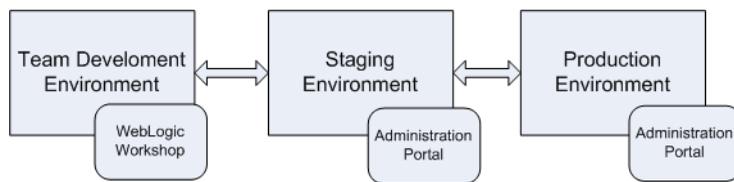
These three environments include:

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

- **Staging** – In the staging environment, administrators use the Administration Portal 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 Administration Portal to modify the production environment.

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.

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 Administration Portal

- **Datasync Data** – Refers to definition files, such as user profiles and content selectors that WebLogic Portal allows you to create. Within WebLogic Workshop, these definitions are exposed in the /data subdirectory. 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 security data, such as entitlements and role definitions. This data is generated by administrators using the WebLogic Administration Portal.

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

Table 6-1 lists the specific kinds of data that comprise each of these categories.

Table 6-1 General Categories of Data to Propagate

Portal Framework Data	Datasync Data	Security Data	EAR Data
• Desktops • Portals • Books • Pages • Portlets • Portlet Preferences	• Catalog data • Content selectors • Discounts • Events • Placeholders • Request properties • Segments • Session properties • User profiles • Campaigns (must be propagated with the Datasync Web Application)	• Policies (visitor entitlement and delegated administration) • Roles (global, visitor entitlement , and delegated administration) The following delegated administration policies: • portal resources (Library and Desktop level) • user group management • content selectors • campaigns • visitor roles • placeholders • segments • security providers Note that definitions pertaining to Content Management are not propagated.	• `.portal` • `.portlet` • `.pinc` • `.shell` • `.theme` • `.menu` • `.layout` • `.laf` (look and feel) • `.jsp` • `.class`

Note: EAR data consists of files that are generated by developers using WebLogic Workshop. If you make a change to EAR data in WebLogic Workshop (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 Administration Portal usually contain references to EAR data. For instance, an administrator can assign a specific theme to a page using the Administration Portal, and a reference to the actual theme file is maintained in the database. When you propagate a desktop to another environment using the Propagation Utility, those references are propagated, but the actual file the reference points to (for example, a .theme file) is not. EAR deployment and the Propagation Utility are described in the following sections.

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

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 WebLogic Workshop. 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 5, "Preparing and Deploying the EAR File".

Propagation Utility

The Propagation Utility guides 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, the Propagation Utility can play a role whenever you move a portal application from a staging environment to the production environment.

Tip: Before using the Propagation Utility to propagate a portal, always deploy the EAR file in the target environment first.

Features of the Propagation Utility include:

- **Exporting portal assets** – The tool lets you export the portal assets stored in the portal application's database into an XML file.

- **Differencing** – The tool lets you 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.

- **Policies** – Policies are well-defined conditions under 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. The Propagation Utility allows you to customize the policies that govern these merge operations. For more information on policies, see "Creating Policies for Merging Inventory Information" on page 8-16.

- **Scope** – You can set a scope for the propagation. Scopes include Enterprise application, web application, and desktop. For more information on scope, see "Setting the Scope" on page 7-6.

For detailed information on using the GUI-based Propagation Utility, see Chapter 8, "Using the Propagation Utility".

Datasync Web Application

Datasync data includes personalization components, such as campaigns, catalog definitions, content selectors, discounts, placeholders, user segments, and user profile definitions. You can find these components in your Enterprise application's /data directory.

Note: You can propagate datasync data using either the Propagation Utility or the Datasync web application; however, the Propagation Utility is usually recommended. The only scenario where you must use the Datasync web application is when you need to propagate campaign data. The Propagation Utility ignores campaign data.

To use the Datasync web application, you must first deploy the EAR file on the destination server. The Datasync web application synchronizes file-based datasync data with the application's

database. In other words, the web application pulls datasync data, such as content selector files (`.sel` files) out of the deployed EAR and merges that data into the database.

For detailed information on datasync data and using the Datasync web application, see Chapter 10, "Using the Datasync Web Application".

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 WebLogic Workshop by a developer, modified, and then merged back into the database using the Export/Import Utility. Therefore, 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 Administration Portal

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

- **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 9, "Using the Export/Import Utility".

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 propagated. Other cases involve known limitations in the tools themselves. For instance, propagation of campaign data and data from the content management system is currently not supported. Whenever possible, these cases are clearly documented and workarounds are discussed.

For detailed information on manual propagation steps, see "Making Manual Changes Prior to Propagation" on page 8-21.

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.

Choosing the Right Propagation Tool

Table 6-2 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 6-2, and the tools listed are summarized in "What Tools Does BEA Provide to Assist with Propagation?" on page 6-4.

For example, the first row of the table indicates that if you are making an initial deployment from a development to a staging environment, you simply deploy the EAR file.

Table 6-2 Choosing a Propagation Method

Source	Destination	Changes to Propagate	Propagation Method
Development environment	Staging environment • Development mode * • Initial deployment	• Portal Framework data • Datasync data • EAR data	Deploy EAR
Development environment	Staging environment • Production mode * • Redeployment	• Portal Framework data • Datasync data • EAR data	1. Deploy EAR 2. Datasync web application
Staging environment	Development environment	• Portal Framework data (only desktops, books, and pages)	Export/Import Utility
Staging environment • Production mode *	Development environment	• Datasync data	Datasync web application
Staging environment • Administration Portal changes	Production environment • Production mode * • Administration Portal changes	• Portal Framework data • Datasync data • Security data	1. Deploy EAR 2. Propagation Utility 3. Datasync web application (for campaigns)

Table 6-2 Choosing a Propagation Method (Continued)

Source	Destination	Changes to Propagate	Propagation Method
Staging Environment Administration Portal changes	Production Environment • Development mode *	• Portal Framework data • Datasync data • Security data	1. Deploy EAR 2. Propagation Utility
Development Environment • Administration Portal changes	Staging Environment • Administration Portal changes	• Portal Framework data	Propagation Utility

* See .

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 6-2, 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 6-2 Propagation Roadmap
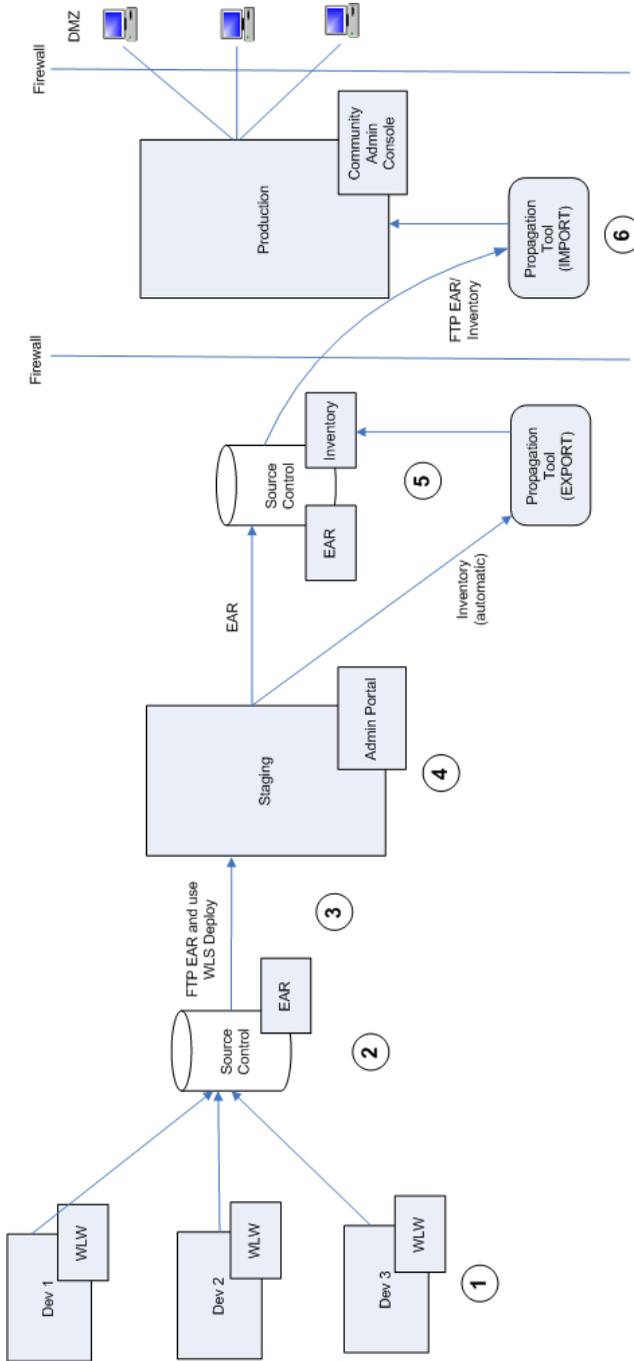
① Development Environments

Portal development is typically spread out among members of a team using WebLogic Workshop on individual client machines. Refer to Chapter 3, "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. WebLogic Workshop allows developers to create Java components that are used by portals, such as Java Pageflows, Controls, and JSPs. WebLogic Workshop 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`, `.pinc`, `.portlet`, `.laf`, and so on.

② Source Control

Development typically occurs in conjunction with a source control system. As explained in Chapter 3, "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.

Chapter 3, "Managing a Team Development Environment" discusses the use of source control in team development environments.

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 5, "Preparing and Deploying the EAR File" for more details.

Note: It is possible to run the Administration Portal from WebLogic Workshop. Whenever the Administration Portal is used, the output is stored in a database, not in a file. Therefore, if you are in development and use the Administration Portal 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 Administration Portal only in a staging or production environment. If you use the Administration Portal in a development environment, then you must use the Propagation Utility to move the database assets.

④ Staging Environment

In the staging environment the Administration Portal 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 Administration Portal to modify a portal, all portal assets from that point on are stored in the database: the connection to the `.portal` and `.pinc`, files created in WebLogic Workshop is lost.

Tip: It is possible to return portal 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 Propagation Utility.

For details on the Export/Import Utility, see Chapter 9, "Using the Export/Import Utility". For details on the Propagation Utility, see Chapter 8, "Using the Propagation Utility".

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*. The easiest way to extract the inventory from a web application is to use the Propagation Utility to export the inventory into a ZIP file.

For details on the Propagation Utility, see Chapter 8, "Using the Propagation Utility".

⑥ Moving to the Production Environment

The Propagation Utility allows you to move a staged portal application to a live production environment.

Note: The Propagation Utility reads the portal inventory ZIP file that was created in staging and reports the differences between the staging inventory and the current production inventory. At this point, you can view these differences 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 Utility 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 Utility 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).

For information on EAR deployment, see Chapter 5, "Preparing and Deploying the EAR File".

For details on the Propagation Utility, see Chapter 8, "Using the Propagation Utility".

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 WebLogic Workshop to create portal assets. In staging and production, administrators use the WebLogic Administration Portal 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 6-8

- **What *mode* is your production server in?** For more information on modes, see "Production Mode vs. Development Mode" on page 6-19.

- **Are you deploying your application for the first time, or are you redeploying it?** For details, see "General Propagation Scenarios" on page 6-13.

- **If you are redeploying an existing application, what kinds of changes were made to it, and how were they made?** Did developers use WebLogic Workshop 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 Portal? For details, see "General Propagation Scenarios" on page 6-13.

- **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 "Propagation Scope" on page 7-6.

- **Do you wish to push a portal from a staging or production environment back to a WebLogic Workshop environment?** For information on accomplishing this task, see Chapter 9, "Using the Export/Import Utility".

- **Do you wish to push a portal from a production environment back to a staging environment?** This can be accomplished with the Propagation Utility. See Chapter 8, "Using the Propagation Utility".

- **Have you defined entitlements for portal assets, such as portlets?** If so, you need to be aware that some security information is by design never propagated by existing BEA 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 7-14.

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 .

Figure 6-3 Example of a Portal Development and Production Environment
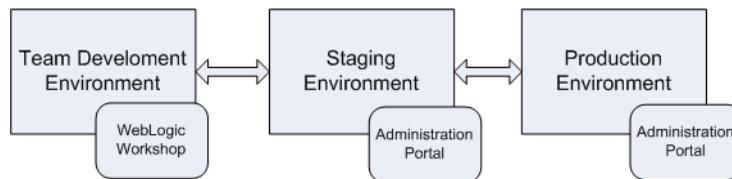
In a *development environment*, developers use WebLogic Workshop to create portals and portlets. In this environment, all portal-related data is file-based (stored in XML files, such as .portal and .portlet files). At the completion of development, these file-based assets, including Java and JSP files, 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 Portal 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 Portal 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 WebLogic Workshop 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 5, "Preparing and Deploying the EAR File".

Note: When you begin using the WebLogic Administration Portal, 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

When redeploying, it is assumed that the target server is a staging or production server. 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 Applications" on page 5-9.

If you are redeploying 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.

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 Portal.

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

If the Target Server is in Production Mode

- *Datasync data is ignored if the EAR is deployed to a server in production mode.* See the following section for information on moving the datasync data.

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

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

Moving the Datasync Data

When you redeploy an Enterprise application, you have a choice between deploying an EAR file or an exploded EAR file. The way in which datasync data is handled during deployment depends on (a) whether you deploy an exploded EAR or an EAR file and (b) whether the server is in development mode or production mode. (See also "Production Mode vs. Development Mode" on page 6-19.)

As Figure 6-4 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 later case, you must use either the Propagation Utility or the Datasync web application to move the datasync assets to the target server's database.

Figure 6-4 Where Datasync Data Is Located When You Deploy an Enterprise Application
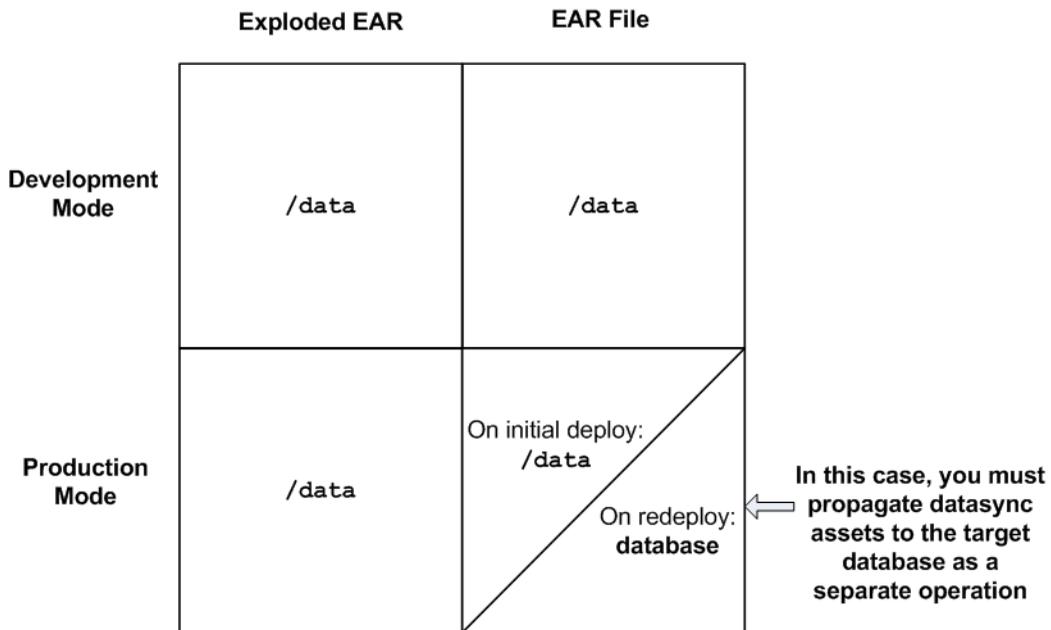
When you redeploy an EAR file (not an exploded EAR) to a server that is in production mode, you have to propagate the datasync data as a separate operation.

To do this, you have two options:

- Use the Propagation Utility to propagate the database from the source to the target server. The Propagation Utility lets you view the differences between the two environments and automatically handles merging those differences based on configurable policies.

  The Propagation Utility is the recommended method for propagating database assets.

- Use the Datasync Web Application to synchronize the contents of the EAR file with the database.

Tip: The best practice is to always use the Propagation Utility to move datasync data. There are situations, however, where the Datasync Web Application must be used. For example, if the scope of the Propagation Utility is set to the Desktop level, datasync data will not be propagated

(datasync data resides at the Enterprise Application level, and so would be out of scope in this case).

Scenario 3: Propagating from Staging to Production: Enterprise Scope

When propagating an Enterprise application from a staging to a production environment, it is assumed that the target server is in production mode. 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 deployed. 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 Utility 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 6-18.

3. If you are using a content management system, you must manually recreate the repository from the staging environment in the production environment.

Tip: Be aware that users and groups, by design, are not propagated by the Propagation Utility. 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 7-14.

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 Utility 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 Desktop scope. Therefore, you need a strategy for propagating datasync data in this circumstance.

The basic steps for propagating with Desktop scope are the same as Enterprise scope, described in the previous section.

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

To propagate the datasync data outside of the Desktop scope, use the Datasync web application. The recommended method for accomplishing this is to place the new datasync files in a JAR file and move the JAR file to the production server. Then, run the Datasync web application on that JAR file. The Datasync web application will insert, or bootstrap, the specified files into the production server's database. For more information on using the Datasync web application, see Chapter 10, "Using the Datasync Web Application".

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 Utility reports differences between the source and target environments. While the Propagation Utility 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. The Propagation Utility modifies or removes user customizations on the destination server only under certain circumstances, such as if another change causes a conflict. For example, if a user customizes a portlet, but the portlet is deleted, the user customizations that reference the deleted portlet will be removed. For more information see "User Customizations and Propagation" on page 7-15.

Tip: If you make a change to the staging environment using the Administration Portal, 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. You can

use the Propagation Utility or the Datasync Web Application to move datasync data. For more information, see and .

Scenario 6: Round-Trip Development

The Export/Import Utility allows you to propagate desktops, books, and pages back and forth between development (WebLogic Workshop) 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 and .pinc files that can be read into WebLogic Workshop. The utility also allows you to import .portal and .pinc 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 Portal, 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.

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 vs. Development Mode

When you configure a domain, you are given a choice between a development mode and a production mode. As a best practice, it makes sense for the production (live) environment to be in production mode because it runs more efficiently; however, some sites run their live production servers 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.

Inside the Propagation Utility

The Propagation Utility allows you to propagate the configuration of one portal environment to another. This chapter provides information that you need to know before using the Propagation Utility. For usage information about this utility, see "Using the Propagation Utility" on page 8-1. For a description of limitations in the current release of the Propagation Utility, see the *WebLogic Portal SP4 Propagation Patch Release Notes*.

The sections included in this chapter are:

- The Propagation Utility Sequence

- General Propagation Guidelines

- Before You Begin

- Propagation Scope

- Propagation Scope Reference Table

- Security Information and Propagation

- User Customizations and Propagation

- Datasync Assets and Propagation

- Conflict Resolution Using Policies

- Best Practices

The Propagation Utility Sequence

Propagation using the Propagation Utility consists of two primary tasks:

- First you *export* a system inventory from a source system, which is typically a staging system. This exported inventory list describes the entire Enterprise application configuration, as contained in the database for that system. The exported inventory is stored in the form of XML files, which are grouped into a single ZIP file.

- Then you *import* and commit the inventory to a destination system, which is typically a production system. The following text summarizes the steps that you perform as part of the import process:

  - **Set the scope** – You choose a scope for the propagation; scopes include Enterprise application, Portal web applications (as a group), individual Portal web application, portal, and desktop. For more information on scope, see "Setting the Scope" on page 7-6.

  - **Choose policies** – Policies are well-defined conditions under which source assets will (or will not) be merged into the destination database. For more information on policies, see "Conflict Resolution Using Policies" on page 7-16.

  - **Preview changes** – The Propagation Utility finds and displays the changes that will occur when you choose to commit the propagation, based on the scope and policies that you select. For more information on the types of data that can be propagated, see "Propagation Scope Reference Table" on page 7-8.

  - **Commit the changes** – When you commit the changes, the Propagation Utility performs the changes based on the previewed change manifest.

General Propagation Guidelines

The following sections describe some general considerations to keep in mind as you prepare to use the Propagation Utility.

Propagating in a Clustered Environment

In a clustered environment, propagate from the source server to the specific managed server on the destination.

Propagating Across Networked Systems

Keep in mind that the Propagation Utility is a Web-based application that runs on the server that you are importing to or exporting from. If you have a firewall between the source and the destination system that blocks access from a client browser, give access to the client machine or run the import process from the destination machine.

Before You Begin

The following sections describe some preparation tasks to perform before using the Propagation Utility. For additional information about the propagation planning process, see Chapter 6, "Developing a Propagation Strategy."

Perform a Data Backup

Back up your WebLogic LDAP repository using the steps in the "Recovering Failed Servers" section of the WebLogic Server documentation.

Back up your database using the 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

The Propagation Utility does not provide the ability to quiesce the system (that is, disable modification of portal data via the Administration Portal). You might want to prevent users from changing portal data during a propagation by redirecting WebLogic Administration Portal URLs to a maintenance page. It is very important that you make no changes to the production system after you load and validate the inventory listing during the import portion of the propagation. For more information, see "Acknowledging the Quiescence Requirement" on page 8-14.

Install the Patch

Before running the Propagation Utility you must have already installed appropriate software. For instructions, see "Installing the Propagation Software" on page 2-1.

Customize the Session Timeout and Inventory/Log File Location (Optional)

You can change the default location of the verbose log files and inventory files, and change the session timeout period.

To edit these values, uncompress the `propagation.war` file, which is located in the following directory:

    patchpath\8.1 SP4 Patch\Propagation Tool

where `patchpath` is the root directory where you unzipped the patch.

Edit the `web.xml` file to make any changes that you want; this file is located here:

    patchpath\8.1 SP4 Patch\Propagation Tooll\WEB-INF

When you finish editing the file, remember to compress the files to recreate the `propagation.war` file.

Session Timeout Value

To change the timeout period, expressed in minutes, edit the `web.xml` file to change the `session-timeout` value:

```
<session-config>
    <session-timeout>15</session-timeout>
</session-config>
```

Inventory and Log File Location

To change the default location of the generated inventories that the Propagation Utility produces and the default location for the Propagation Utility's verbose log files, edit the `web.xml` file to change `param-value`:

```
<context-param>
<param-name>oamDataFilesystemPath</param-name>
    <param-value>d:/propagation/81xDomain/inventories</param-value>
    <description>Base folder path for runtime data, such as
    inventory exports.</description>
</context-param>
```

For more information on the log files that the Propagation Utility produces, see "Reviewing Propagation Utility Export Log Files" on page 8-9 and "Reviewing the Import Log Files" on page 8-23.

Deploy the J2EE Application (EAR)

If you created new resources for your web application using WebLogic Workshop, 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 Utility. 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 6-13 and "Scenario 2: Redeploying an EAR file" on page 6-14. For detailed information on deploying EAR files, see "Preparing and Deploying the EAR File" on page 5-1.

Tip: It is not required that you deploy your J2EE application prior to *beginning* the Propagation Utility; the Propagation Utility Web-based interface contains a reminder page at the required stage in the propagation process. However, deploying the application beforehand is a good practice.

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

Because the Propagation Utility does not propagate the complete set of portal resources from the source to the destination system, there might be cases where propagated data depends on other data that is not propagated. For example:

- Visitor entitlement roles are propagated to the destination but any related users and groups are not, so you must manually add those related users and groups on the destination system.

- If you are propagating any data related to new content repositories, you must create the new content repositories on the destination system before using the Propagation Utility.

For a list of propagated and non-propagated resources, see "Propagation Scope Reference Table" on page 7-8.

The Propagation Utility Web-based interface includes a page that lists pending changes that require associated manual updates. Review this list of changes to verify that you have already performed all required manual changes.

To view an example of a manual change listing, see "Making Manual Changes Prior to Propagation" on page 8-21.

Propagation Scope

To understand how portal assets are propagated, it is crucial to understand propagation scope. The Propagation Utility detects differences that exist between the source and destination systems and creates a nested list of possible scopes. From this list, you may select a specific scope for the propagation. For instance, if you select web application scope, all assets in the web application are propagated. If you select desktop scope, only the assets associated with a specific desktop are propagated.

Setting the Scope

You specify the scope of a propagation during the import process. You can select to scope the propagation at several levels of granularity, depending on where differences exist between your two systems.

- **Enterprise application** – Propagates all detected changes for the Enterprise application to the destination system.

  Note: The Propagation Utility can be used to move data between different instances of the same enterprise application; it does not support propagating from one application to another.

- **Portal Webapps** – Propagates as a group all of the portal framework data for the web applications present in the Enterprise application.

- **Web application** – Propagates all portal framework data within a single web application.

- **Library** - Propagates portal framework Library data for a single web application.

- **Portal** – Propagates as a group all of the desktops in a single portal web application.

- **Desktop** – Propagates all differences within a single desktop. This scope maps to the Export/Import Utility's Admin scope, as described in Chapter 9, "Using the Export/Import Utility."

Tip: When using the Propagation Utility, the best practice is to scope to the highest level—Enterprise application. This ensures that the destination environment will exactly mirror the source environment. If you scope to a lower level, web application or desktop, the source and destination might be in different states. In this case, additional quality assurance testing may be required on the destination. For details, see "Set the Scope to the Enterprise Application Level" on page 7-19.

Scope and Library Inheritance

The WebLogic Administration Portal 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 WebLogic Workshop and stored in a `.portal` or `.portlet` file

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

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

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

Creating a New Desktop and Disassembling to the Library

When you create a new desktop using the Administration Portal, 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 WebLogic Workshop. (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

Administration Portal. 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 asset. On the other hand, a change to the desktop asset 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.

When you propagate a portal using either portal or desktop scope, new pages and books are disassembled to the Portal Library if they do not already exist there.

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 Administration Portal. When a portal is propagated, any assets that are decoupled in the source application will remain decoupled in the destination.

Propagation Scope Reference Table

This section lists the kinds of portal assets handled by the Propagation Utility and shows you the scoping levels that are available for each type of asset. Table 7-1 helps you choose the appropriate scope for the propagation you wish to perform, and to ensure that the assets you wish to propagate are propagated. For example, some assets, such as dataync data, are only propagated at the Enterprise scope.

This table includes data that can be propagated by the Propagation Utility as well as using the Datasync web application or by deploying the EAR. If you deploy the EAR prior to using the Propagation Utility, as recommended, fewer changes will actually be handled by the Propagation Utility. For details on the Datasync web application, see "Using the Datasync Web Application" on page 10-1. For details on EAR deployment, see "Preparing and Deploying the EAR File" on page 5-1.

For more details on manually propagating assets, see "Make Required Manual Changes" on page 7-5.

Data propagation depends on the scope that you set during the Import task. The following indicators in the table show the applicable propagation scopes for each type of data:

- **E** – Enterprise Application scope

- **PW** – Portal Webapps scope (all web applications that reside in the Enterprise application)

- **W** – Web application scope

- **L** - Web application Library scope

- **P** – Portal scope (all desktops that reside in an individual portal)

- **D** – Desktop scope

For more information about propagation scopes, see "Setting the Scope" on page 7-6.

Table 7-1 Data Propagation Reference Table

Data Category	Data	Propagation Scope and Description
Framework	Portals	E, PW, W, L, P – fully propagated. D – not applicable. Data at this scope is moved by deploying the EAR.
	Desktops	E, PW, W, L, P, D – fully propagated.
	Books	E, PW, W, L, P, D – fully propagated. **Note:** The description field content is not propagated; for details see the *WebLogic Portal 8.1 SP4 Propagation Patch Release Notes*.
	Pages	E, PW, W, L, P, D – fully propagated. In addition, Look and Feel references, such as a dependency between a page and its layout, are propagated. **Note:** The description field content is not propagated; for details see the *WebLogic Portal 8.1 SP4 Propagation Patch Release Notes*.
	Portlets	E, PW, W, L, P, D – fully propagated.
	Portlet Preferences	E, PW, W, L, P, D – fully propagated. **Note:** For the current release, portlet preferences are never deleted from the destination server; changed preference attributes are concatenated rather than updated. For details, see the *WebLogic Portal SP4 Propagation Patch Release Notes*.
	Portlet Categories	E, PW, W, L – fully propagated. P, D – not applicable. Data at this scope is moved by deploying the EAR.

Table 7-1 Data Propagation Reference Table (Continued)

Data Category	Data	Propagation Scope and Description
Framework, *continued*	Shells	Not applicable. Data at this scope is moved by deploying the EAR.
	Themes	Not applicable. Data at this scope is moved by deploying the EAR.
	Menus	Not applicable. Data at this scope is moved by deploying the EAR.
	Layouts	Not applicable. Data at this scope is moved by deploying the EAR.
	Look and Feels	Not applicable. Data at this scope is moved by deploying the EAR.
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 7-15.

Table 7-1 Data Propagation Reference Table (Continued)

Data Category	Data	Propagation Scope and Description
Datasync	Catalog Property Definitions	E – fully propagated. PW, W, L, P, D – not propagated.
	Content Selectors	E – fully propagated. PW, W, L, P, D – not propagated.
	Discount Definitions	E – fully propagated. PW, W, L, P, D – not propagated.
	Event Definitions	E – fully propagated. PW, W, L, P, D – not propagated.
	Placeholders	E – fully propagated. PW, W, L, P, D – not propagated.
	Request Properties	E – fully propagated. PW, W, L, P, D – not propagated.
	Segments	E – fully propagated. PW, W, L, P, D – not propagated.
	Session Properties	E – fully propagated. PW, W, L, P, D – not propagated.
	User Profile Definitions	E – fully propagated. PW, W, L, P, D – not propagated.
	Campaign Definitions	Not propagated.
Runtime Data	Behavior Tracking Events	Not propagated.
	Orders	Not propagated.
	User Profiles	Not propagated.

Table 7-1 Data Propagation Reference Table (Continued)

Data Category	Data	Propagation Scope and Description
Security	Global Roles (created using WebLogic Server)	E, PW, W, L, P, D – fully propagated, except definitions that you create using public entitlement APIs.
	Visitor Entitlement Policy Definitions	E, PW, W, L, P, D – fully propagated, except definitions that you create using your own custom code (APIs).
	Delegated Administration Policy Definitions	E – fully propagated. PW, W, L, P, D – 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 Note that definitions pertaining to Content Management are not propagated.
	Delegated Administration and Visitor Entitlement Roles	E – fully propagated. PW, W, L, P, D – only the roles that are required (that a given asset depends upon) are propagated. **Note:** **Whenever you propagate roles, do so at the Enterprise scope.** Otherwise, lower level dependencies and the role hierarchy might not be preserved on the destination. For details, see the *WebLogic Portal 8.1 SP4 Propagation Patch Release Notes*. Roles that you create using your own custom code (APIs) are not propagated. 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.

Table 7-1 Data Propagation Reference Table (Continued)

Data Category	Data	Propagation Scope and Description
WSRP	WSRP Registration Data	Not propagated. For more information on WSRP propagation, see "WSRP Propagation" on page 7-18.
Content Management	All Content Management information, items, types, and metadata	Not propagated.

Security Information and Propagation

Security information consists generally of authentication data and authorization data. As Table 7-2 shows, authorization data consists of roles and policies, while authentication consists of users and groups.

Table 7-2 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. The delegation hierarchy is stored in the database, but optionally can be stored in LDAP.
	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 provider configurations and audit trails.

For more details on the specific data propagated, see "Propagation Scope Reference Table" on page 7-8.

Authentication information (users and groups) is never propagated by the Propagation Utility. User and group information is often maintained in separate external authentication repositories over which WebLogic Portal has no control.

In addition to user and group information, the Propagation Utility does 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 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 7-5.

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.

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. As a result, the Propagation Utility is not designed to propagate user customizations.

User customizations are preserved on the destination, but when you propagate to a destination system, those customizations might be removed or modified on the destination server under certain conditions. For intstance, an administrator might make changes through the Administration Portal 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.

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 WebLogic Administration Portal.

Datasync Assets and Propagation

The Propagation Utility propagates datasync assets at the Enterprise Application scope only. For more information on datasync propagation, see "Moving the Datasync Data" on page 6-15.

Conflict Resolution Using Policies

The Propagation Utility merges source and destination data based on policy rules, also called merge rules, that you can configure for your requirements. You can choose to either activate or ignore the policy rules during a propagation. If the policy choices you make here lead to a conflict as the propagation proceeds, policies take precedence in the order shown here.

1. If an asset exists on the source, but not on the destination, add it to the destination.

2. If an asset exists on the destination, but not on the source, delete it from the destination.

3. If an asset exists on the source and the destination, update the destination with the asset's source attributes.

The following sections describe how the Propagation Utility prioritizes and implements policies, and describe some considerations that you must keep in mind as you decide how to set these propagation policies.

Identifying Differences Between Assets

The Propagation Utility creates unique identifiers for portal assets so that they can distinguish whether two assets are the same in the source and destination systems. A portion of this identifier is based on the definition label, or the instance label, which shown in the property editor pane in WebLogic Workshop and the page properties within the WebLogic Administration Portal. Figure 7-1 shows an example of the definition labels as displayed by the WebLogic Administration Portal:

Figure 7-1 Definition Labels

Definition labels assigned
using Administration Portal

Definition label assigned
using WebLogic Workshop

When you preview changes during the propagation process, you can view the planned changes based on the detected differences between the two systems.

Prioritizing Changes Based on Policies

If a conflict occurs while processing a propagation, the Propagation Utility attempts to handle conflicts according to the policies you created, in the priority order listed previously.

When the Propagation Utility resolves a conflict, the resolutions are listed in the log file that the utility creates during processing, and the web-based Propagation Utility interface will list any changes that occurred in order to resolve conflicts.

Handling Propagation Conflicts

When conflicts arise, policies take precedence in the order in which they appear in the Propagation Utility policies page. The following example describe how the Propagation Utility resolves a conflict when the same page is added to different books on the source and destination.

A user adds *Page1* to *Book2* on the source system, but another user adds the same *Page1* to *Book3* 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 Utility resolves the conflict by adding *Page1* to *Book2*, and removing it in *Book3* on the destination system.

Rolling Back an Import Process

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 7-3.

WSRP Propagation

One of the primary advantages of Web Services for Remote Portlets (WSRP) is that the WSRP model decouples the producers that host and maintain remote services and the consumers that use them. It is important to note that when you create a remote portlet on a consumer, registration information about the consumer is stored on the producer. Registration information includes the registration handle, WSDL URL, and other optional information. When you propagate the consumer from a staging to a production environment, the propagation utility is not directly aware of the producer or of the consumer registration information it maintains.

Two Models for WSRP Propagation

There are two models for working with WSRP propagation across staging and production environments: *shared registration* and *dual registration*.

Note: Currently, shared registration is the only model that is supported by WebLogic Portal.

- **Shared registration** — Both the staging and production servers use the same registration handle. After propagating the consumer, the producer cannot distinguish between the staging and production consumer, because they both share the same handle. In effect, both consumers "point" to the same producer. Note that any changes that are made to the producer's portlets will be seen by the consumers in both staging and production. In addition, if a producer is un-registered in the staging environment, it will be un-registered in the production environment as well.

- **Dual registration** — The staging and production environments each use their own registration handle. In this case, configuration changes made in one environment are not visible to the other.

Support for Shared Registration

WebLogic Portal supports only the shared registration model. Because the registration handle is shared by the staging and production environments, the same producer is visible to both environments; therefore, propagation of producer data is not necessary.

The drawback of this model is that it is not possible to "stage" the producer's portlets: any change to a producer portlet will automatically be reflected in both the staging and production systems.

In the future, WebLogic Portal will support WSRP 2.0, which provides support for propagating a producer's state with a dual registration model.

Best Practices

The following sections describe some practices that you should follow to achieve the most predictable and accurate results with the Propagation Utility.

Keep Portal Framework Definition Labels and Instance Labels

When you create new portals and portal resources in WebLogic Workshop, BEA recommends that you change the definition labels and instance labels at that time, to create meaningful names for these resources. Once you have used the Propagation Utility to propagate changes among your environments, it is very important that you do not change these resource names. The Propagation Utility uses definition labels and instance labels in order to identify differences between source and destination systems; inconsistent results might occur if you change these labels within WebLogic Workshop or the WebLogic Administration Portal.

Do Not Manually Replicate Changes Between Environments

Create artifacts in only one environment. If you make changes in both, even if they are identical changes, the utility cannot identify them as being the same changes—propagation is carried out as if they were two separate resources.

Set the Scope to the Enterprise Application Level

Choosing the highest level of scoping for the propagation ensures the that the destination environment will closely mirror the source environment. If you scope to a lower level, such as web application or desktop, the source and destination 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 utility 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 Utility 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 7-7.

Using the Propagation Utility

This chapter provides information about each page of the Propagation Utility web-based interface.

Before you use the Propagation Utility, familiarize yourself with how the Propagation Utility handles various types of data, and what you need to do before using it. See "Inside the Propagation Utility" on page 7-1.

The sections in this chapter are:

- Inventory Export Process
- Inventory Import Process

Inventory Export Process

You use the Propagation Utility to *export* a system inventory from a source system, which is typically a staging system. This exported inventory list describes the entire Enterprise application environment, as contained in the database, for that system. The exported inventory is stored in the form of XML files, which are grouped into a single ZIP file.

Figure 8-1 shows the overall process of exporting an inventory:

Figure 8-1 Exporting an Inventory
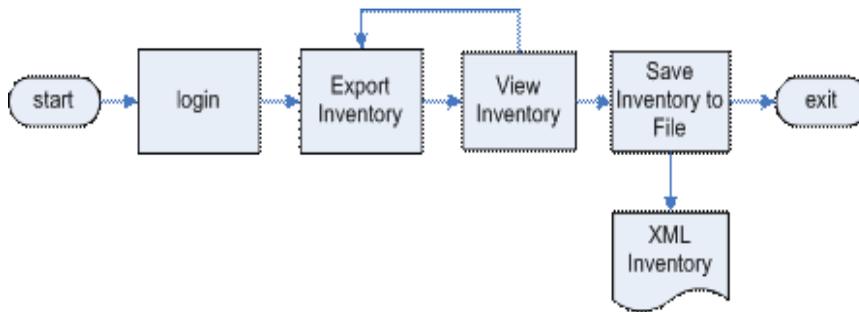
During the export process, you create an inventory report that describes the entire content of the source environment. Later, during the import process, you assign the scope of the information to propagate. For more information, see "Inventory Import Process" on page 8-9.

Warning: You must not edit the individual XML files generated during the export task.

Starting the Propagation Utility and Logging In

Before running the Propagation Utility you must have already completed the steps in "Before You Begin" on page 7-3.

To start the Propagation Utility, open your Web browser and enter a URL in the following format:

```
http://host:port/propagation/
```

Figure 8-2 displays:

Figure 8-2 Propagation Utility Start Page
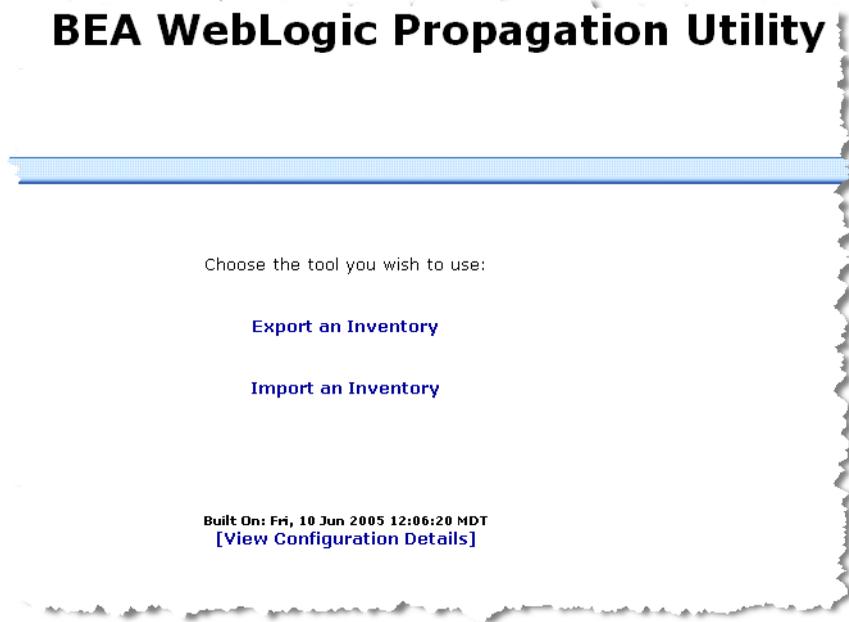
Select the task **Export an Inventory**. Then log in with the username and password of a valid user on the destination system; you must belong to the PortalSystemAdministrators group.

Figure 8-3 shows the login page:

Figure 8-3 Propagation Utility Login Page
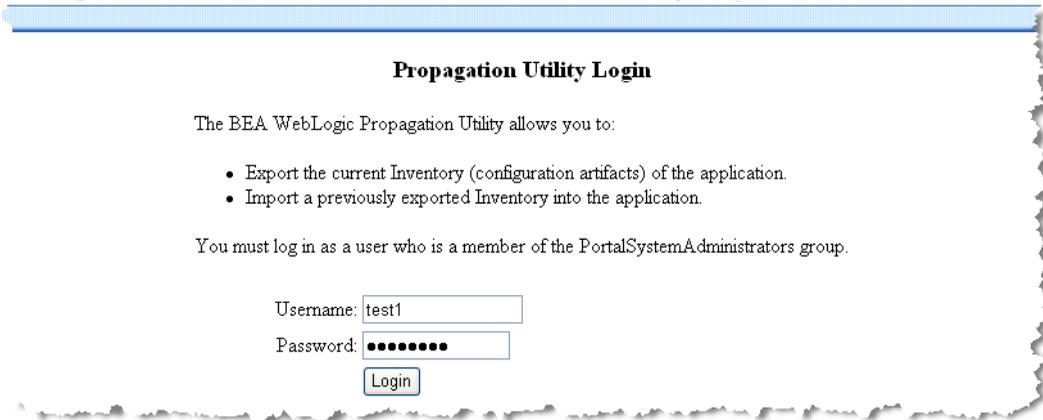
Viewing the Application Inventory

After you log in, the Propagation Utility creates an inventory tree and displays it, as shown in Figure 8-4. Options for accepting this inventory listing, regenerating an inventory listing, or restarting the Propagation Utility are also shown.

Figure 8-4 Propagation Utility View Application Inventory Page
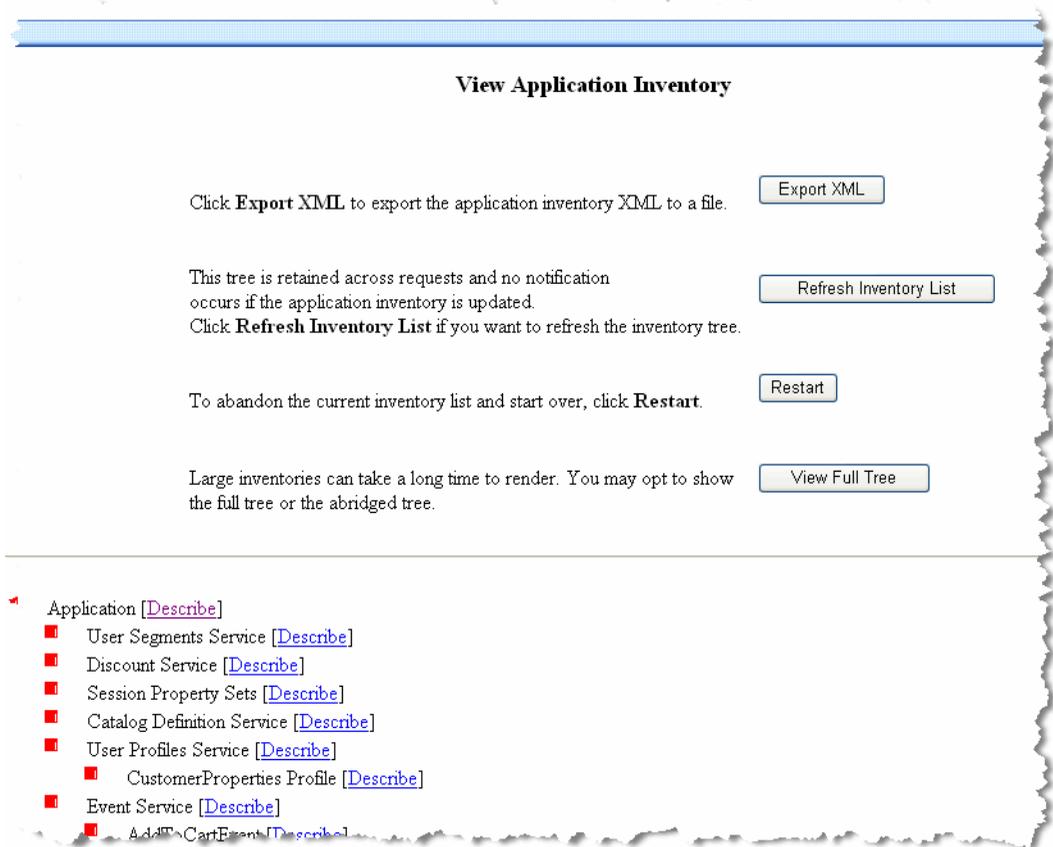
By default, the Propagation Utility displays a summarized, or "abridged" view of the inventory listing. If you want to view a more detailed listing of the inventory, click **View Full Tree**.

If you want to accept the inventory listing as shown and save it, click **Export XML**. If you know that changes were made using the WebLogic Administration Portal after you started the export process, you can click **Refresh Inventory List** to recreate the inventory listing. If you want to start the export process again from the beginning without using this inventory, click **Restart**.

The inventory list contains a hierarchical description of the entire content of the source environment that you want to propagate. Figure 8-5 shows a segment of an inventory listing:

Figure 8-5 Propagation Utility Inventory Listing Example

If you click the **Describe** link, detailed information about the particular asset (node) in the tree is displayed, including:

- **XML Description** – The XML for the asset. Note that the XML generated by the Propagation Utility must not be hand-edited; if you do so, propagation results might be corrupted.

- **Dependencies** – If applicable, displays a list of nodes that, if removed, will make this asset invalid.

- **Dependents** – If applicable, displays a list of nodes that will become invalid if this node is removed.

Exporting the Application Inventory to a File

You can choose to save the inventory file on your local machine, or save it on the machine hosting the associated inventory—the machine you are exporting from. For example, you might want to save the file to your local system for storage in a source control system.

Figure 8-6 shows the available locations for saving the exported inventory list:

Figure 8-6 Propagation Utility Export Application Inventory Page

If you prefer to export the file to your local system, click **Export Locally**. A file download window appears; follow the instructions to save the file to a desired location. The filename `inventory.zip` appears by default; create a meaningful name so that you can track this file, but keep the `.zip` file extension.

Note: If the file to be exported is larger than 5 MB in size, you *must* export it to the server that you are exporting from.

If you prefer to save the file on the machine you are exporting from, you can either use the automatically generated working directory path or enter a path and filename of your choice. The

Propagation Utility displays the working directory path as it is set up in the `web.xml` deployment descriptor file, and appends a *day_hour_minute* label.

Tip: BEA recommends that you store the saved file within a source control system for safe-keeping. At a minimum, keep track of your exported filenames and locations so that you can easily access them during the propagation Import process.

After you save the file, a final page describes the export results, as shown in Figure 8-7:

Figure 8-7 Propagation Utility Inventory Export Results Page

If the inventory file is saved successfully the results page will display a simple message indicating success. If a problem occurred, such as selecting an existing filename or inadequate disk space, an error description displays.

After a successful export, you can import the inventory from the source system into the destination system.

Warning: You must not edit the individual XML files generated during the export task.

Reviewing Propagation Utility Export Log Files

The Propagation Utility creates the following logging information during an export process:

- Overview information is placed in the default server log file. You control the detail level of this log using the WebLogic Server console. For details about server log files, see the topic "Server Log" in the WebLogic Server Administration Console Online Help.

- A verbose log is placed in the working directory—the same location where inventory files are stored. For details on the working directory and how to change its location, see "Customize the Session Timeout and Inventory/Log File Location (Optional)" on page 7-4.

You should review the overview log file to ensure that the import process completed as expected. The verbose log file is primarily for diagnostic purposes.

Inventory Import Process

Figure 8-8 shows the overall process of importing an inventory:

Figure 8-8 Inventory Import Process
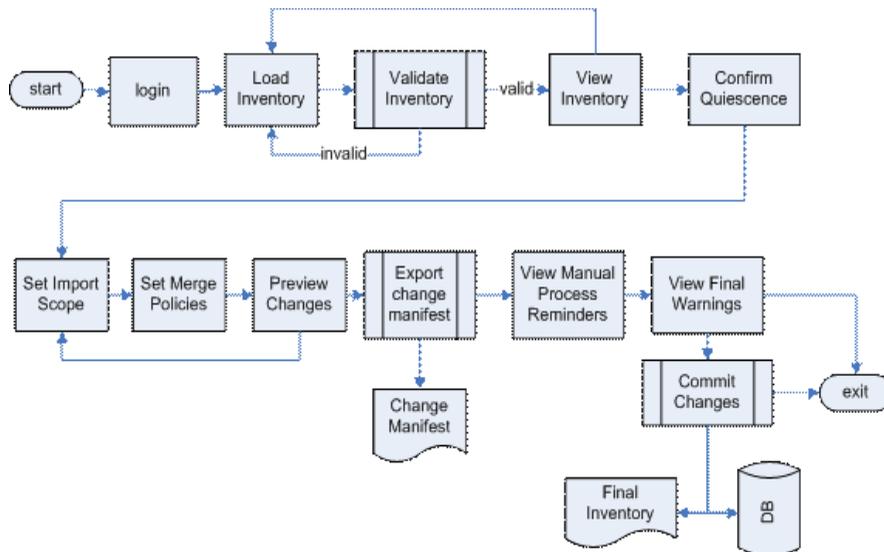
Starting the Utility

Follow the startup and login instructions as described in "Starting the Propagation Utility and Logging In" on page 8-2.

Selecting the Source Inventory to Import

Choose the source inventory (the ZIP file you exported in from the source system) from which you want to import changes, as shown in Figure 8-9:

Figure 8-9 Propagation Utility Select Import Source Page

Select a radio button to provide the source file using one of these methods:

- Select to specify a path and filename. The field automatically displays the working directory path, as it is set up in the web.xml deployment descriptor file.

- Use the dropdown menu to select a ZIP file in the working directory.

Click **Continue** to retrieve and validate the inventory.

Loading and Validating the Imported Inventory List

The Propagation Utility validates the imported inventory according to the XML schema. Validation errors that can occur include missing dependencies, for example.

The utility displays the results of the validation. Figure 8-10 shows the results of a successful validation:

Figure 8-10 Propagation Utility Review Inventory Validation Results Page
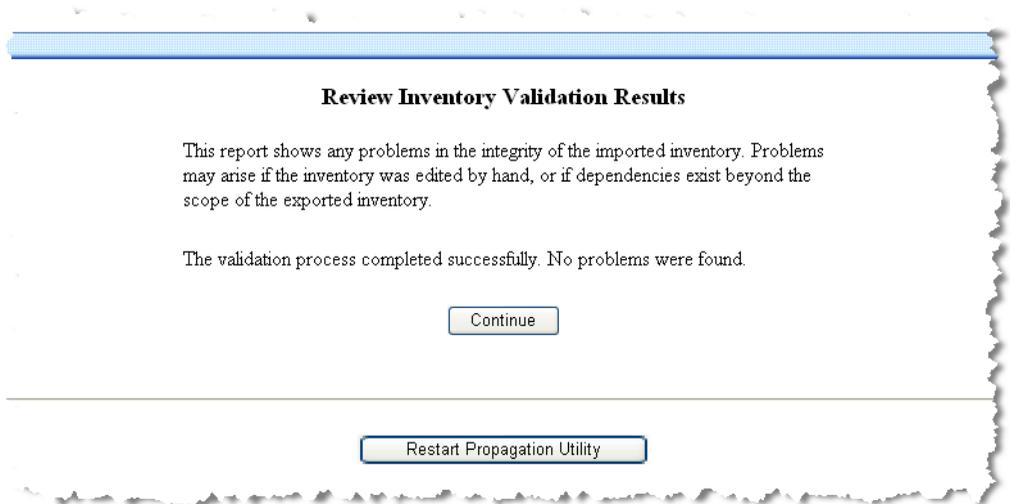
Click **Continue** to view the validated inventory listing on which the propagation will be based.

Viewing the Application Inventory

The Propagation Utility displays the validated inventory, as shown in Figure 8-11. Options for accepting this inventory listing or restarting the Propagation Utility are included.

Figure 8-11 Propagation Utility View Application Inventory Page
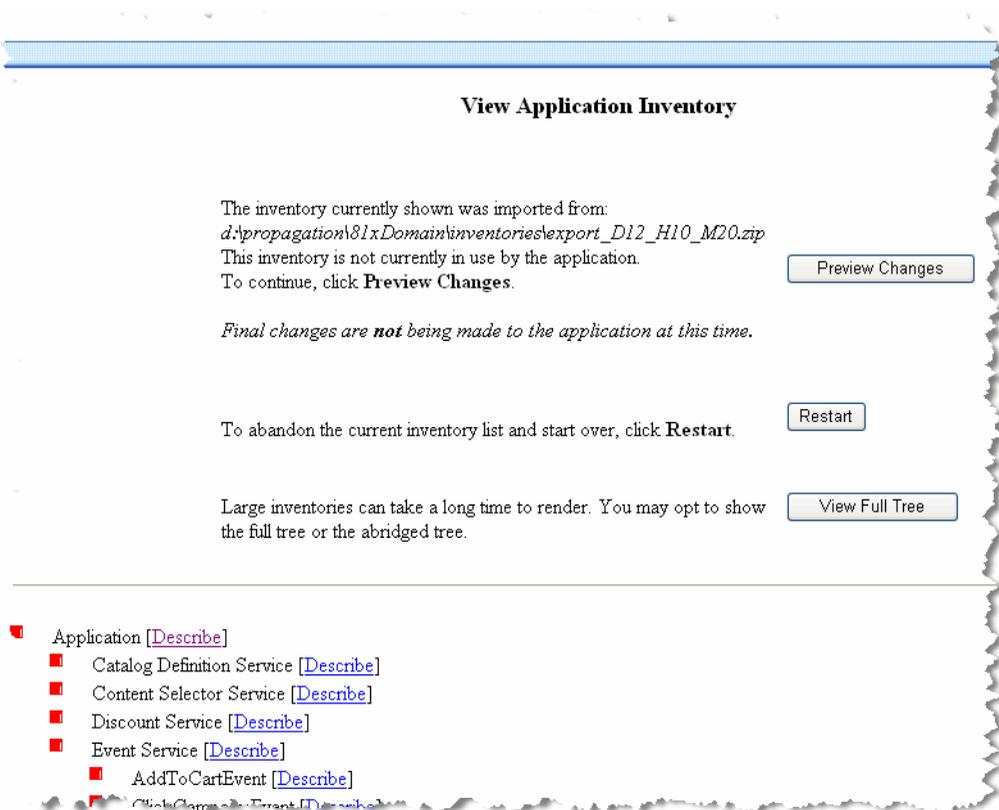
The Propagation Utility displays a hierarchical description of the content of the imported source environment that you want to propagate. By default, the Propagation Utility displays a summarized, or "abridged" view of the inventory listing. If you want to view a more detailed listing of the inventory, click **View Full Tree**.

Later, you can refine the set of changes by setting policies for making changes and by setting a level of scope for the propagation.

If you want to accept the inventory listing as shown and continue, click **Preview Import**. If you determine that you do not want to use this inventory and you want to start the Propagation Utility again from the beginning, scroll to the bottom of the page and click **Restart Propagation Utility**.

If you click the **Describe** link, detailed information about the particular asset (node) in the tree is displayed. This information is the same as that displayed as part of the export process, as described in "Viewing the Application Inventory" on page 8-4.

Figure 8-12 shows a segment of an inventory listing:

Figure 8-12 Propagation Utility Inventory Listing Example

Acknowledging the Quiescence Requirement

The Propagation Utility does not automatically quiesce the system to prevent users from changing portal data using the Administration Portal 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.

Warning: It is very important that you make no changes to the production system after you load and validate the inventory during the import process.

To ensure that you understand and agree to this requirement, a confirmation page appears, as shown in Figure 8-13. Select the checkbox and click **Continue** to proceed to the next step.

Figure 8-13 Propagation Utility Configure Changes Confirmation Page

Deploying the J2EE Application on the Destination

If you have not already done so, you must deploy the J2EE application (EAR file) from the source system to the destination system before committing changes that the Propagation Utility identified. For more information on this requirement, see "Deploy the J2EE Application (EAR)" on page 7-5.

Figure 8-14 shows the Deploy J2EE Application reminder page:

Figure 8-14 Propagation Utility Deploy J2EE Application Page
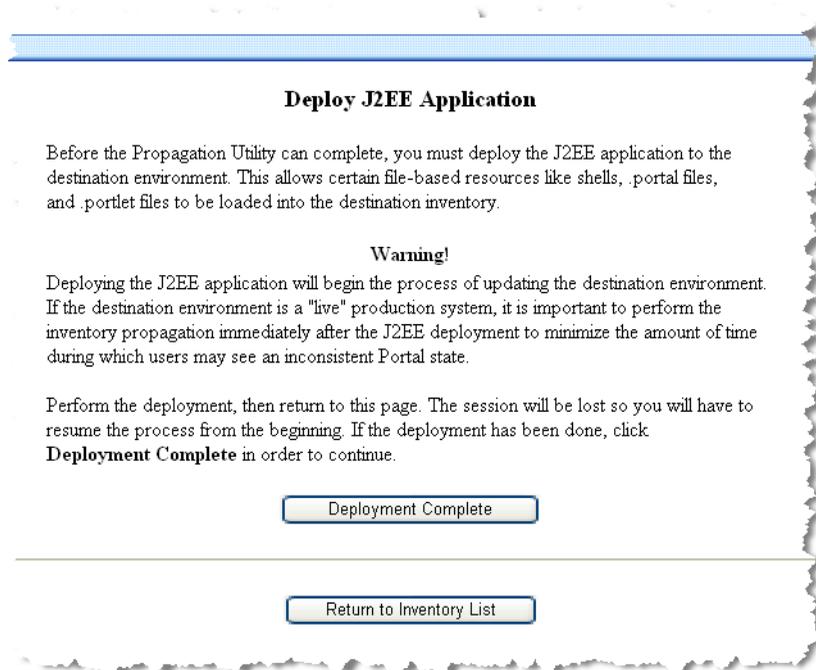
If you have not yet deployed the application, you can leave this window open while you do so, and then return to this page and click **Deployment Complete** to continue.

Note: If you leave the Propagation Utility window open while you perform other activities, the Propagation Utility might time out. If this occurs, your previous process and selections are lost. You must restart the Propagation Utility and perform the import again. For instructions on changing the session timeout period, see "Customize the Session Timeout and Inventory/Log File Location (Optional)" on page 7-4.

Setting the Scope

You specify the scope of a propagation during the import process. The Propagation Utility detects differences that exist between the source and destination systems and creates a nested list of possible scopes. A page similar to Figure 8-15 appears:

Figure 8-15 Propagation Utility Select Import Scope Page
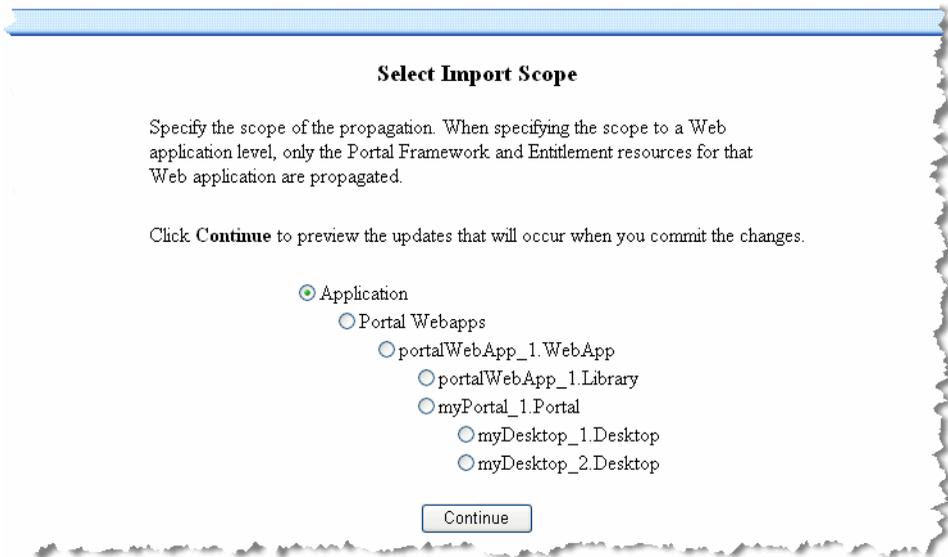
You can select to scope the propagation at several levels of granularity, depending on where differences exist between your two systems. For detailed information on scoping, see "Propagation Scope" on page 7-6.

Tip: When using the Propagation Utility, the best practice is to scope to the highest level—Enterprise application. This ensures the that the destination environment will closely mirror the source environment. If you scope to a lower level, web application or desktop, the source and destination might be in different states. In this case, additional quality assurance testing may be required on the destination. For details, see "Set the Scope to the Enterprise Application Level" on page 7-19.

Creating Policies for Merging Inventory Information

The Propagation Utility merges source and destination data based on policy rules, also called merge rules, that you can configure for your requirements.

Figure 8-16 shows the policy setting page in the Propagation Utility:

Figure 8-16 Propagation Utility Create Import Policy Page

For more details on how the Propagation Utility prioritizes and implements these policies, and some considerations that you must keep in mind, see "Conflict Resolution Using Policies" on page 7-16.

Previewing Changes

In the final step prior to committing the pending changes to the destination system, the Propagation Utility displays the changes that will be made to the destination if you continue. The Review Pending Changes page is similar to the example shown in Figure 8-17:

Figure 8-17 Propagation Utility Review Pending Changes Page

Review Pending Changes

This report shows the additions, updates, and deletions that will be made when you commit changes.

Review these changes carefully. If any change is unacceptable, modify the imported inventory.

Summary Statistics:

Adds	13
Deletes	1
Updates	4

Note that for longer item taxonomies, the character sequence "//" indicates a segment of text that was removed for display purposes. Click the **Details** link for more information about any change.

Legend:

Black: this change is pending, and has no special considerations.
Red: this change cannot be performed; see the Details page and documentation as needed.
Blue: this change requires that a manual change be performed.
Green: this change is implied by the acceptance of another change.
Brown: this change, although correct, should be reviewed carefully for unintended side effects.
Gray: this change has been excluded due to the policy in effect.

See the documentation for more details on each of these special considerations.

Change Manifest:

Index:	Action:	Details:	Application Resource:
1	Adds	Details	portalservices.::portalWebApp_1.WebApp.::myPortal_1.Portal.::myDesktop_2.Desktop
2	Adds	Details	portalservices.::portalWebApp_1.WebApp.::myPortal_1.Portal.::myDesktop_2.Desktop.::myDesktop_2.D

If you find that the changes displayed do not reflect your desired results, do not continue. Make changes as needed and return to this step later. Depending on the action that you want to change, you might be able to correct the propagation by changing the propagation scope or the propagation policies. If this does not solve the problem, exit from the Propagation Utility. Make changes within your source or destination environment, and run the export or import task again.

The Review Pending Changes page contains summary statistics, a detailed application resource change list, and a filename selection for saving the pending change manifest (pending changes) to a file. The following sections contain more details on each of these part of the page.

Summary Statistics

The totals in this section reflect the number and type of changes included in the **Action** column of the change list.

The Propagation Utility counts the total number of potential changes whether or not they will actually be carried out. For example, if 15 potential deletes were found, but 12 of them will be excluded because of policies that you specified earlier, the count will still include them.

Change Manifest Legend

The color-coding of the inventory list identifies the status of a pending change, indicating any items that might require special attention from you.

The following list describes each color and its meaning:

- **Black** – This change is pending, and has no special considerations.

- **Red** – This change cannot be performed; this status occurs very rarely, in a situation when a potential change violates a data integrity rule.

- **Blue** – This change requires that an associated manual change be performed; for example, entitlement changes are blue to indicate that any associated users or groups must be added manually on the destination system. Manual changes are also identified separately later; for details, see "Making Manual Changes Prior to Propagation" on page 8-21.

- **Green** – This change is implied by another change that the Propagation Utility will be making; for example, when you propagate using either portal or desktop scope, new pages and books are added to the Portal Library if they do not already exist there.

  Assets can have multiple dependencies, but depending on the type of dependency they might appear in the change manifest only once. Carefully review the listing for dependencies that result in multiple changes.

- **Brown** – This change, although correct, should be reviewed carefully for unintended side effects; for example, an item has this status if a change is required to the item based on the policy, but the item is not within the scope of the propagation.

- **Gray** – This change has been excluded due to the policy in effect; for example, if you chose to ignore (not delete) assets that exist on the destination but not on the source, all

assets that meet this criteria (they exist on the destination but not on the source) are shown in gray and will not be deleted.

For detailed information on any asset in the change manifest, click the **Details** link for that asset.

Pending Change List (Change Manifest)

The Pending Change List shows each change that will be made to the destination system by the Propagation Utility when you commit changes. To keep the display to a manageable size, long asset identifiers are shortened, with the notation "//" used to indicate a portion of the identifier that was removed.

Note: The Change Manifest includes only the updates that will be made explicitly by the Propagation Utility on the destination system, once you commit the changes. For example, changes that occur on the destination based on the EAR deployment that you perform prior to previewing changes are not included in the Change Manifest.

Figure 8-18 show an example portion of a Change Manifest:

Figure 8-18 Propagation Utility Change Manifest Listing Example

17	Deletes Details	portalservices.::em1WebProject1.WebApp.::em1WebProject1.Library.::em1Portal1_Demo1_book_4.Bo
18	Deletes Details	portalservices.::em1WebProject1.WebApp.::.//.::em1Portal1_Demo1_book_4.Book.::em1Portal1
19	Deletes Details	portalservices.::em1WebProject1.WebApp.::.//.::em1Portal1_Demo1_page_5.BookMember.::em1Portal1
20	Deletes Details	portalservices.::em1WebProject1.WebApp.::.//.::em1Portal1_Demo1_page_5.BookMember.::em1Portal1

You can click the **Details** link to see the details for a particular change; these details include the following information:

- **Identifier** – The internal identifier for the particular asset.

- **Taxonomy** – The complete taxonomy used internally by the Propagation Utility to track assets. For items displayed in the change manifest with shortened entries, you can view complete taxonomies here.

- **Item Description** – The description of the individual asset.

- **Explanation** – The reason for including this asset in the change manifest.

- **Has this change been implied by another change?** – Indicates whether or not the asset is an implied change based on another pending change.

- **Is this change impossible?** – Indicates whether this change can be performed by the Propagation Utility.

- **Does this change require manual work by the operator?** – Indicates whether or not the item requires that you perform an associated manual change.

- **Source XML** – The XML listing of the item from the source system, if it exists there.

- **Destination XML** – The XML listing of the item from the destination system, if it exists there.

Exporting the Change Manifest for the Inventory

If the pending change list correctly describes the propagation that you want to perform, save the change manifest. The Propagation Utility automatically displays a path and filename; you can accept this as shown or enter a different path and filename. Click **Continue** to save the file and continue to the next step.

Making Manual Changes Prior to Propagation

Because the Propagation Utility does not propagate the complete set of portal resources from the source to the destination system, 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.

If manual changes are needed to coordinate your two environments, the Propagation Utility displays these necessary changes on the Review Required Manual Changes page. The Review Required Manual Changes page is similar to the following example, as shown in Figure 8-19:

Figure 8-19 Propagation Utility Review Required Manual Changes Page
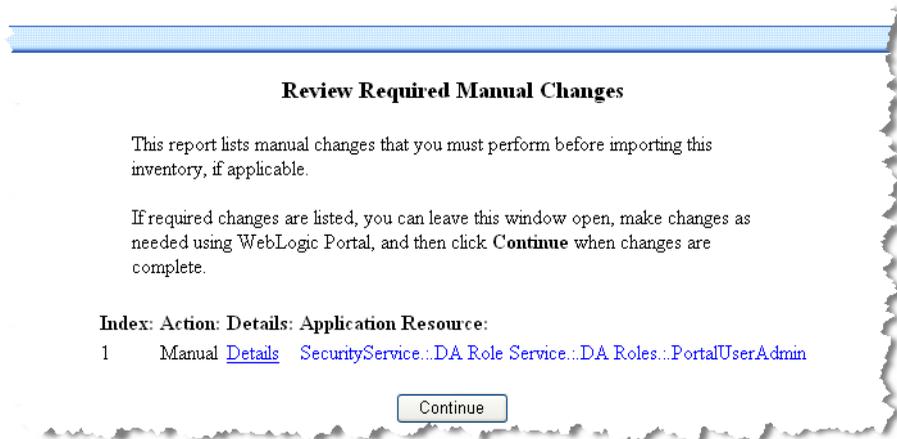
This page lists manual changes that you must perform before importing this inventory, if applicable. For more information, see "Make Required Manual Changes" on page 7-5.

If required changes are listed, you can leave the Propagation Utility window open, make changes as needed using WebLogic Portal, and then come back to this page and click **Continue** when changes are complete.

Note: If you leave the Propagation Utility window open while you perform other activities, the Propagation Utility might time out. If this occurs, your previous process and selections are lost. You must restart the Propagation Utility and perform the import again. For instructions on changing the session timeout period, see "Customize the Session Timeout and Inventory/Log File Location (Optional)" on page 7-4.

Committing the Inventory Import

The Propagate the Inventory page summarizes the policy selections you made previously, and provides an opportunity to return to the inventory listing if you decide you do not want to propagate this change. Figure 8-20 provides an example of the page:

Figure 8-20 Propagation Utility Propagate the Inventory Page

Propagate the Inventory

When you click the **Commit the Inventory** button, the proposed changes will be written to the inventory into the destination system. The inventory changes may update the Portal database and Portal LDAP repository.

Policy Summary:

- Items in the import inventory but not in the destination **will be** added.
- Items in the destination inventory but not in the import **will be** deleted.
- Items that exist in both but are different **will be** updated.

Commit the Inventory

When you click **Commit Inventory**, the Propagation Utility propagates the database assets from the staging server to the production server, according to the scope you assigned and the policies you selected previously.

When complete, a results page displays the status of the completed propagation and the time required to complete it. To save an inventory listing of the destination system after the propagation, click **Begin Export** to start an export process for the new destination inventory.

Reviewing the Import Log Files

The following sections describe log files that the Propagation Utility creates during an import process:

You should review the overview log file to ensure that the import process completed as expected. The verbose log files are primarily for diagnostic purposes.

Import Overview Log File

This log file contains errors and warnings, including:

- Warnings for propagated assets that depend on other non-propagated assets; for example, propagated roles dependent on non-propagated groups that do not exist on the destination.

- Warnings for "implied" changes

This information is placed in the default server log file. You control the detail level of this log using the WebLogic Server console. For details about server log files, see the topic "Server Log" in the WebLogic Server Administration Console Online Help.

Import Processing Verbose Log Files

The Propagation Utility creates two verbose log files for each import:

- Verbose output from the processes of reading in and validating the inventory on the filesystem

- Verbose output from the differencing and commiting process

Verbose logs are placed in the working directory—the same location where inventory files are stored. For details on the working directory and how to change its location, see "Customize the Session Timeout and Inventory/Log File Location (Optional)" on page 7-4.

Using the Export/Import Utility

The Export/Import Utility provides a round-trip feature that allows you to move books, pages, and desktops, back and forth between development and staging environments. The utility, also called the xip utility (eXport/Import Portal), lets you import `.portal` and `.pinc` (portal include) files into the database, and lets you export these files from the database. The exported files can be loaded back into WebLogic Workshop, or imported into another WebLogic Portal database. For more information on the specific elements the utility handles, see "Overview of the Export/Import Utility" on page 9-6.

Figure 9-1 shows how the Export/Import Utility moves files between environments.

Figure 9-1 Export/ImporT Utility Allows Round-Trip Development
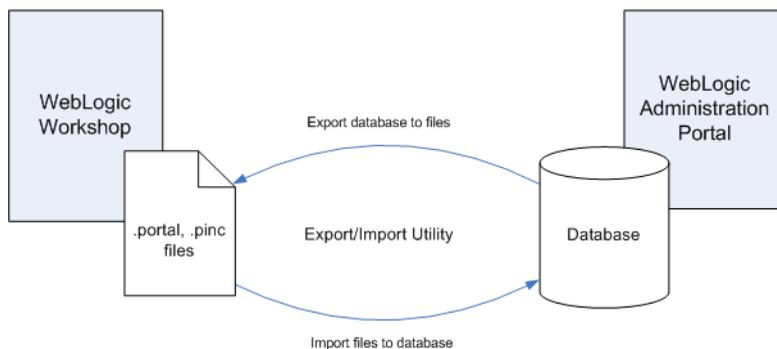
This chapter explains how to use the Export/Import Utility. The chapter includes background information on the utility and its purpose. In addition, detailed examples are provided that illustrate common use cases.

This chapter includes the following topics:

- Installing the Export/Import Utility

- Before You Use the Export/Import Utility

- Overview of the Export/Import Utility

- Understanding the Properties File

- Exporting a Desktop

- Importing a .portal File

- Exporting a Page

- Importing a Page

- Managing the Cache

Installing the Export/Import Utility

For information on installing the utility, see Chapter 2, "Installing the Propagation Software".

Before You Use the Export/Import Utility

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.

Understanding .portal Files vs. Desktops

The `.portal` file that you create in WebLogic Workshop 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 Administration Portal, 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 Administration Portal, 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.

Understanding .pinc Files

A `.pinc` file is a portal include file. Just like a JSP can include other JSPs, a `.portal` can include other `.pinc` files. The `.pinc` files help reduce the size of `.portal` files and prevent conflicts caused by many developers trying to access the same resources simultaneously. A `.pinc` file must begin with a page or book as its root element. The book or page may then contain other nested books and pages.

Understanding Export and Import Scope

Exports and imports can be scoped to the *library*, *admin*, or *visitor* levels. The first two levels correspond to the Library and Portals nodes in the Portal Resources tree of the WebLogic Administration Portal, as shown in Figure 9-2. The visitor level includes changes made by users to individual desktops using the Visitor Tools.

Figure 9-2 Portal Resources Tree of the WebLogic Administration Portal
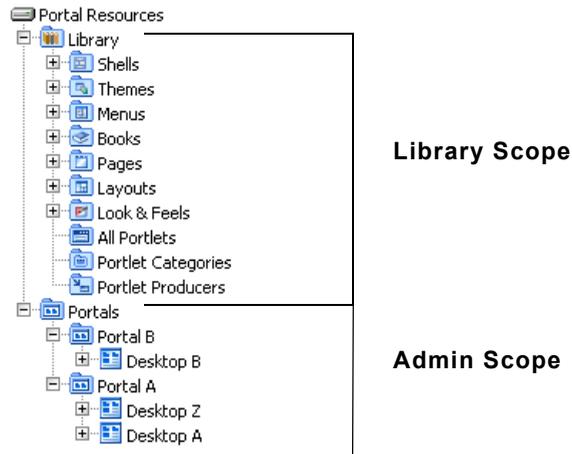
The relationshiop between library, admin, and visitor views is hierarchical, and properties from objects up the hierarchy can be inherited by objects down the hierarchy. For example, a change to a library element is inherited by desktops that use that element and by visitor views that use those desktops.

Figure 9-3 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 9-3 Scoping Hierarchy
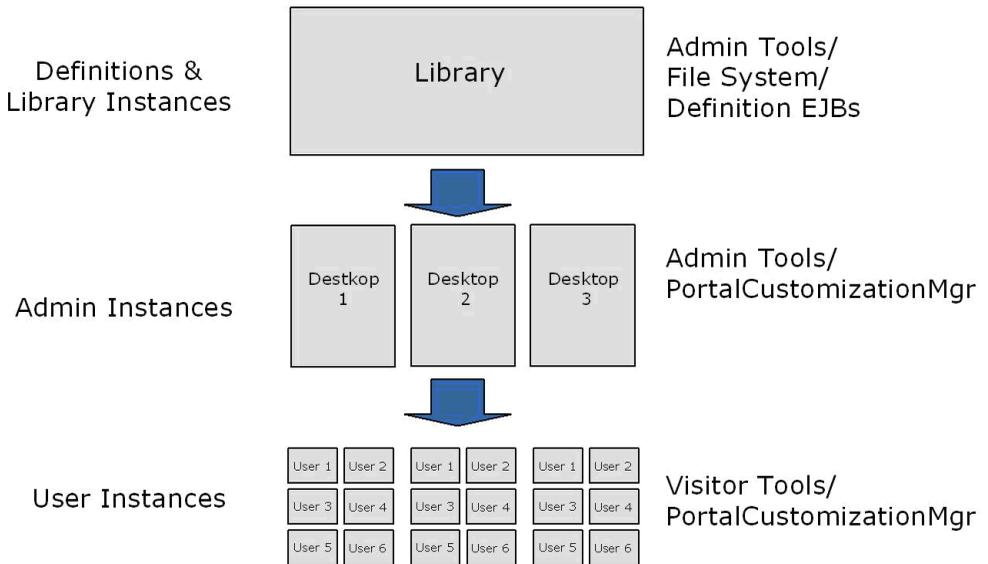
Library Scope

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 7-7.

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 7-7.

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 Administration Portal 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 (portlets, books, pages, menus, and so on).

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.

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 WebLogic Workshop environment and a staging or production environment, as shown in Figure 9-1. 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` and `.pinc` files from a database, and imports the contents of `.portal` and `.pinc` files into a database. These XML files describe fully functioning portals, and books and pages that are included in portals.

Tip: For detailed information on the portal framework, see the following documents:

    http://edocs/wlp/docs81/lookandfeel/lookandfeel.html

    http://edocs/wlp/docs81/whitepapers/netix/index.html

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 `.pinc` 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.

Merging and Scoping Rules

To refine and customize the export and import of `.portal` and `.pinc` 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 9-4, 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 7-7.

Figure 9-4 Import, Export, and Scoping Options Offered by the Export/Import Utility
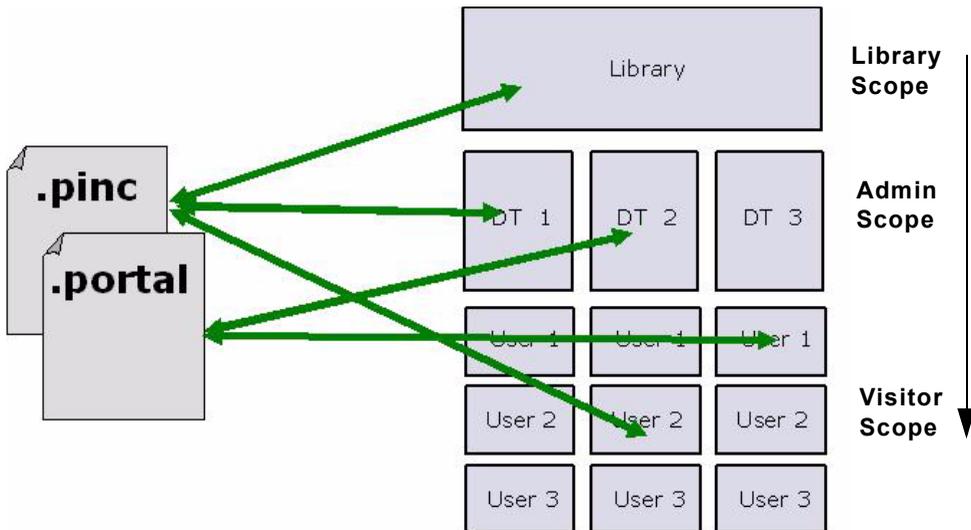
To summarize, the Export/Import Utility allows you to select an object (desktop, book or page) at any level (library, admin, visitor) and import it or export it, according to specified rules.

Using the Export/Import Utility

Before using the utility, you must install it. For installation instructions, see Chapter 2, "Installing the Propagation Software". The utility is a Java program that reads a properties file. 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 properites file. The following sections explain how to configure the properties file and present common use cases.

Tip: The Java client'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, if you wish.

Understanding the Properties File

By default the Export/Import Utility retrieves all of its parameters from a properties file. You must edit this properties file appropriately to configure the export or import parameters.

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. 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 9-1 shows the Ant target definition to edit:

Listing 9-1   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>
```

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.

Listing 9-2 shows the default `xip.properties` file.

Listing 9-2   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 (.pinc files). 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. These can be .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=
```

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 WebLogic Workshop for further development. To export a desktop, you need to edit the xip.properties file and run an Ant 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 WebLogic Workshop, this value corresponds to the Application name, as shown in the following figure:

Figure 9-5 Application Name Shown in the Administration Portal

`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 "Specifying Identifiers" on page 9-31 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 "Understanding Export and Import Scope" on page 9-3.

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 9-3. The build script's task writes status information, also shown below, to the console window.

Listing 9-3   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 WebLogic Workshop 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. As with other kinds of options, you specify merge options in the `xip.properties` file, as explained in the following sections.

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 WebLogic Workshop, this value corresponds to the Application name, as shown in the following figure:

Figure 9-6 Application Name

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 "Specifying Identifiers" on page 9-31.

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 "Understanding Export and Import Scope" on page 9-3.

`xip.import.context.deletes=false`

> You must specify how to handle *deletes*. See "Handling Deletes" on page 9-27 for detailed information.

`xip.import.context.moves=false`
`xip.import.context.outermoves=true`

> You must specify how to handle *moves*. See "Handling Moves" on page 9-29 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 7-7.

`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 scalablity 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 9-4. The build script's task writes status information, also shown below, to the console window.

Listing 9-4   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. For more information, see "Understanding .pinc Files" on page 9-3.

Note: If you export a page or a book, all children of that page or book are exported as well.

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 WebLogic Workshop, this value corresponds to the Application name, as shown in the following figure:

Figure 9-7 Application Name

```
xip.command=export
```
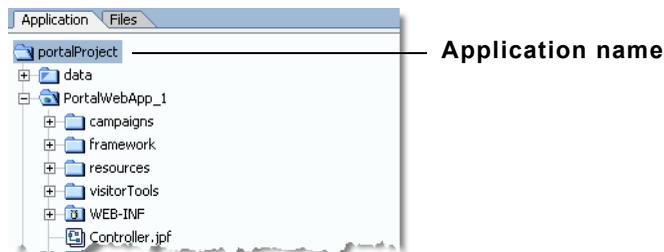
Specify that you wish to export, rather than import.

```
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 "Specifying Identifiers" on page 9-31.

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 WebLogic Workshop, but it could also have been automatically generated by the Administration Portal. See "Specifying Identifiers" on page 9-31 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 9-5.
The build script's task writes status information, also shown below, to the console window.

Listing 9-5   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 [P200134591113965077078]
[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 WebLogic Workshop 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.

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 WebLogic Workshop, this value corresponds to the Application name, as shown in the following figure:

Figure 9-8 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 "Specifying Identifiers" on page 9-31.

> 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 for more information on this property.

`xip.import.context.moves=false`

You need to specify how to handle moves. See 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 .

`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 synch, 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 scalablity 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 9-6. The build script's task writes status information, also shown below, to the console window.

Listing 9-6   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
```

Handling Deletes

The **xip.import.context.deletes** property lets you control how portal assets are merged during an import. As shown in Figure 9-10, 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 9-9 Merge Results with Deletes = false
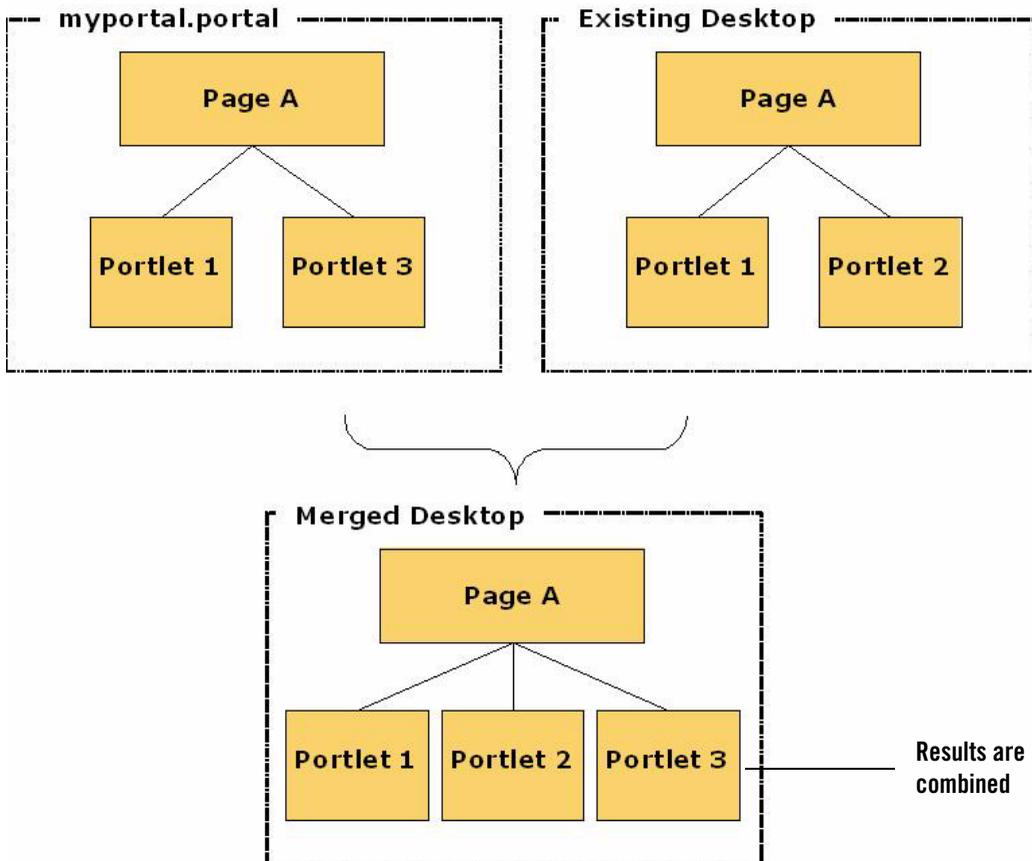
As shown in Figure 9-10, if you set this property to `true`, any portlets that do not exist in the `.portal` file but do exist in the desktop are deleted.

Note: This example refers to portlets; however, the same merge operations would apply as well for pages on a book.

Figure 9-10 Merge Result with Deletes = true
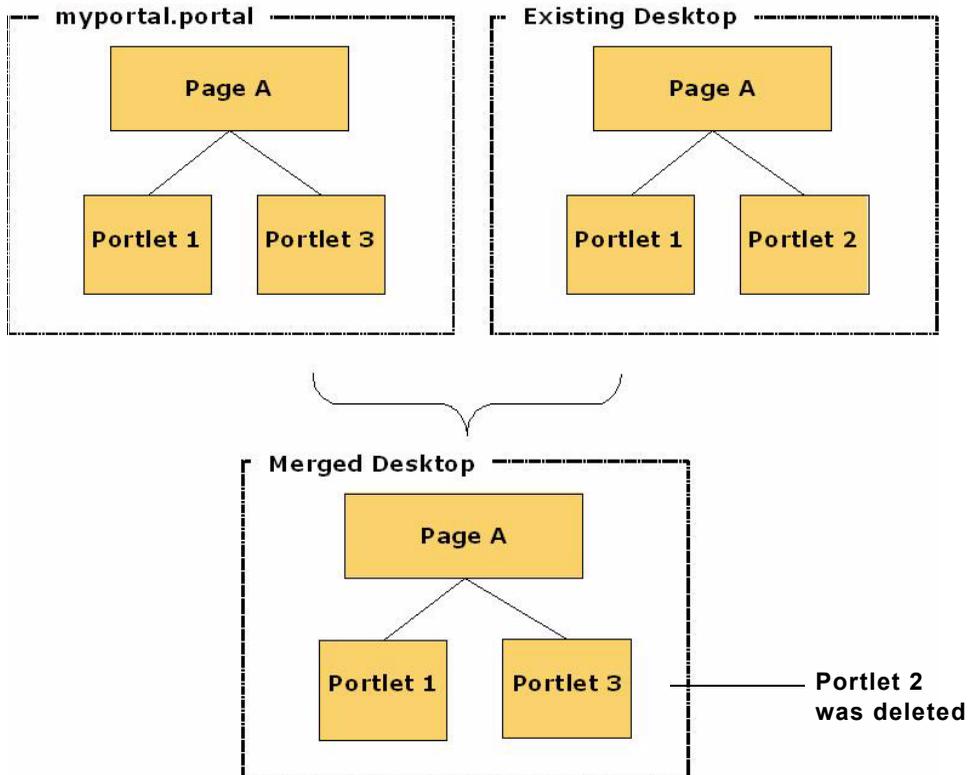
Handling Moves

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*.

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 9-11 Inner Move: Pages P1 and P2 Swap Positions Within the Same Book
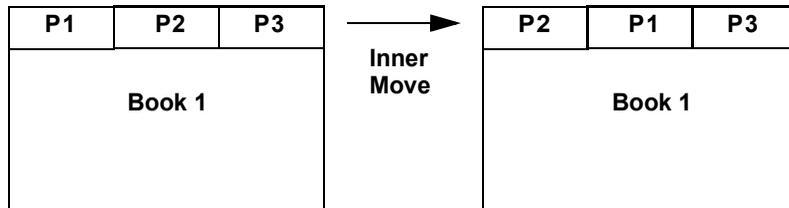
Set the `xip.import.context.moves` property to `true` to allow inner moves. If the property is set to `false`, the positions of assets will not change; however, the contents will be updated appropriately. If you want to move books, pages, or portlets across different parents, then use the `xip.import.context.outermoves` 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 9-12 Outer Move: Pages P1 and P4 Swap Positions Across Different Books
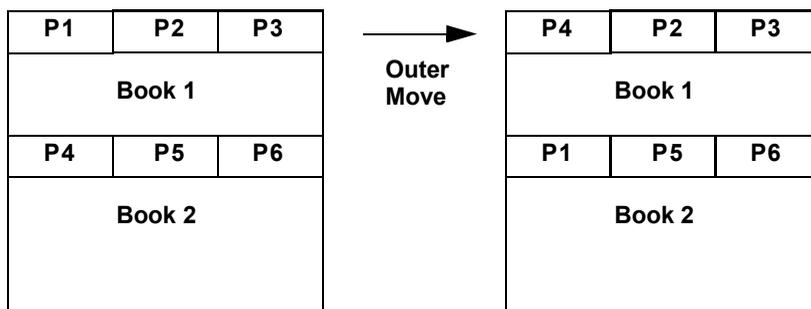
Set the `xip.import.context.outermoves` property to `true` to allow outer moves. If the property is set to `false`, then the operation handles the requested move as a deletion and addition operation.

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.

Specifying Identifiers

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. The `identifier` properties include:

```
xip.identifier.webapp=
xip.identifier.portal.path=
xip.identifier.desktop.path=
xip.identifier.book.label=
xip.identifier.page.label=
```

The web **webapp** property must always be specified. The web application name is listed in the WebLogic Administration Portal, as shown in the following figure:

Figure 9-13 The Web Application Name
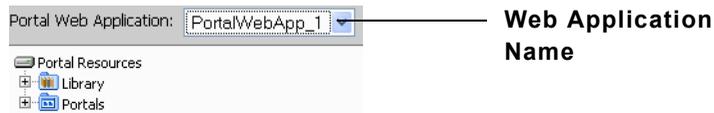
If you scope the export to the *admin* or *visitor* level, then you must specify the `portal.path` and `desktop.path` properties. You can find the correct value in the Administration Portal. The portal path is shown in the Portal Properties tab, and the desktop path is shown in the Desktop Properties tab. For example, the portal path is shown in the following figure:

Figure 9-14 The Portal Path Value As Shown in the Portal Properties Tab

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 WebLogic Workshop; however, they can also be created in the Administration Portal. If the page was created using the Administration Portal, 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 Portal on the Page Properties tab. Figure 9-15 shows the definition label for a page.

Figure 9-15 The Location of a Page's Definition Label in the Weblogic Administration Portal
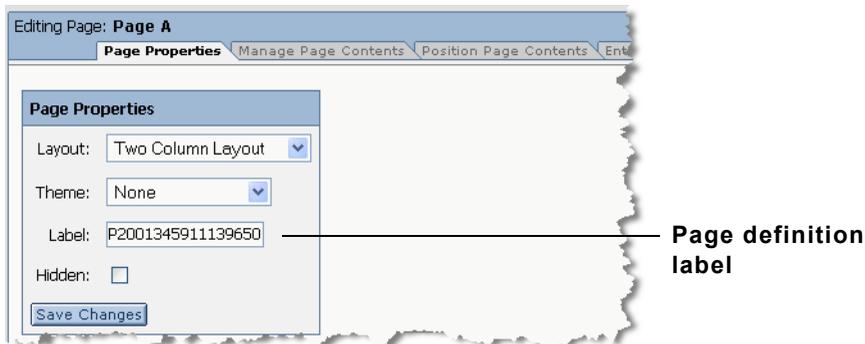
Locating the Definition Label for a Book

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 retreive 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
```

These properties are described below.

Note: 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.

xip.config.application

Specifies the name of the Enterprise application. Only portals under this Enterprise application will be affected. In WebLogic Workshop, this value corresponds to the Application name, as shown in the following figure:

Figure 9-16 Application Name

These properties are described below.

xip.identifier.webapp

Specifies the name of the web application. Only portals under this web application will be affected. See also "Specifying Identifiers" on page 9-31.

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 "Specifying Identifiers" on page 9-31.

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 7-7.

Using the Datasync Web Application

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 "Creating a Production Cluster Environment with the Configuration Wizard" on page 4-5.

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 WebLogic Workshop, portal definitions are created in a special datasync project, exposed in the WebLogic Workshop Application window as a /data subdirectory. (On the filesystem, the directory exists in the application's /META-INF/data directory.) 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 the META-INF/data directory of the portal application and exposed in WebLogic Workshop in the portalApplication/data directory. To provide optimim 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 WebLogic Workshop developers, but also by business users and portal administrators, who can modify user segments, campaigns, placeholders, and content selectors with the Weblogic Administration Portal. In the development environment, both WebLogic Workshop and the WebLogic Administration Portal write to the files in the META-INF/data directory.

Compressed Versus Uncompressed EAR

When deployed into a production system, portal definitions often need to be modifiable using the Weblogic Administration Portal. 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 10-1 shows how the /data directory from the updated portal application is put into a standalone JAR and bootstrapped to the database.

Figure 10-1 Loading Updated Datasync Files to the Database
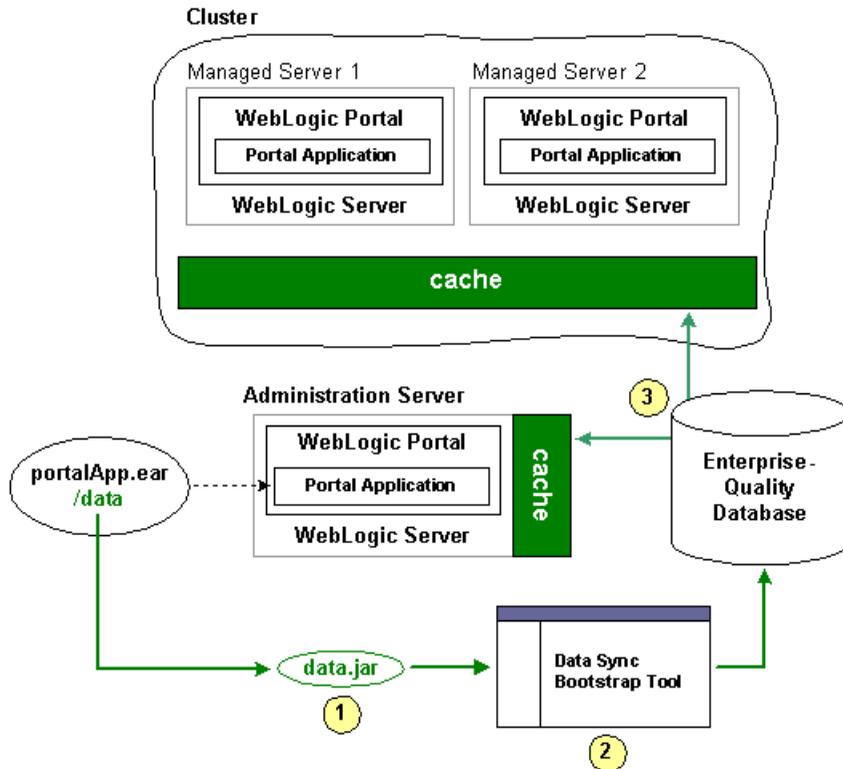
Alternatively, some production environments deploy their portal applications as uncompressed EARs. In this case, the deployed portal application on the administration server is the primary store of datasync definitions. Work done in the WebLogic Administration Portal on any managed server is automatically synchronized with the primary store.

For both compressed and uncompressed EAR files, you can view and update datasync definitions using the Datasync Web Application.

Datasync Web Application

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 10-2.

Figure 10-2 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 to bring up the window shown in .

Figure 10-3 Browsing the Datasync Repository

From this list, click on a particular data item to see its contents, as shown in .

Figure 10-4 Data Item Contents

As you can see in Figure 10-4, 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 .

Uploading new contents

When you click the ▦ icon, the following page appears, which lets you load data into the database.

Figure 10-5 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 10-8 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, 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=c:\bea\weblogic81\p13n\lib\p13n_system.jar;
c:\bea\weblogic81\p13n\lib\p13n_ejb.jar;
C:\bea\weblogic81\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
```

Working with an Uncompressed EAR File

When working with a production server with an uncompressed EAR, the only difference from development mode is that there is no poller thread.

When updating definition files using the WebLogic Administration Portal, the files are updated on the administration server in the deployed uncompressed EAR directory automatically. This means that the WebLogic Administration Portal can be used from any managed server in the cluster, but the primary store always resides on the administration server. If the deployable EAR directory is read-only, the WebLogic Administration Portal cannot be used to modify files.

Making sure you are not overwriting files – When working with an uncompressed EAR file in production, special care needs to be taken when working with definition files. When you redeploy your application to your production environment, the existing definition files are replaced. If you have administrators updating definitions using the WebLogic Administration Portal, their changes will be lost upon redeploying an updated application.

Copying back to development – To prevent overwriting any changes done by administrators to definition files when redeploying a new portal application, you must first copy all the definition files from the administration server back to development manually or using the Datasync web application.

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 Administration Portal 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 WebLogic Workshop at development time rather than in the WebLogic Administration Portal.

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.

Index

`wlw-manifext.xml` file 4-2
WSRP data, and Propagation Utility 7-14

X
xip
    see Export/Import Utility
`xip.properties file` 9-8

Z
zero-downtime architecture 4-17