BEA WebLogic Portal

Development Guide

Release 7.0 Service Pack 4
Document Date: August 2003

WebLogic Portal Development Guides

Part Number	Date	Software Version
N/A	April 2004	7.0 Service Pack 5

Contents

Preface

Welcome to the BEA WebLogic Portal Development Guide. In addtion to this document, we encourage you to use the following resources, as well.

Finding documentation online BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "e-docs" Product Documentation page at **http://e-docs.bea.com**.

Providing documentation feedback Your feedback on the BEA WebLogic Portal documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Please indicate that you are using the documentation for the WebLogic Portal version **Product Version:**

Contacting BEA WebSUPPORT If you have any questions about this version of WebLogic Portal, or if you have problems installing and running WebLogic Portal, contact BEA Customer Support through BEA WebSUPPORT at **http://www.bea.com** or by using the contact information provided on the Customer Support Card in the product package.

CHAPTER

1 Introduction to WebLogic Portal Development

Welcome to the WebLogic Portal *Development Guide*. This guide shows you how to develop and deploy portals and portlets and create the resources necessary to extend their capabilities. The portal development activities described in this guide comprise the initial phase of a portal's lifecycle: creating a portal and the resources used to extend that portal. Once portal development is complete, portal administration becomes the primary concern. Administrative tasks are described in the WebLogic Portal *Administration Guide*.

This section includes information on the following subjects:

- A Developer's Portal Primer
- Portal Component File Locations
- Roadmap for Building a Portal
- How Do I Get Started?

A Developer's Portal Primer

A portal is a feature-rich Web site. It provides a single point of access to enterprise data and applications, presenting a unified and personalized view of that information to employees, customers, and business partners.

Portals allow you to have multiple Web applications within a single Web interface. In addition to regular Web content that appears in a portal, portals provide the ability to display portlets—self-contained applications or content—all in a single Web interface.

Portals support multiple pages with tab-based navigation, with each page containing its own content and portlets.

Portal Features

While a fully-functioning portal offers many features to the portal user that enhance their experience when using WebLogic Portal many development features likewise enhance your experience when developing portals and portal resources. This section describes some of these features.

Personalization and Authorization

Because WebLogic Portal comes with robust authentication and personalization features, administrators can determine what content a visitor can interact with and how that information will appear to the specific visitor. Visitors themselves can leverage WebLogic Portal's personalization features to select their own content and create their own look and feel. A major component of the Portal development process is to create the resources that make such authorization and personalization possible.

Group Portals

Portals are designed either for single users or for groups. With group portals you can set up delegated administration for portals and restrict portal access to specific users. You can create multiple group portals within a portal Web Application. The group portals can share portal resources, such as layouts and portlets, but can be configured differently to satisfy the needs of each group separately. Because users are designated individually as members of a group, the group portal uses a static form of personalization.

JSPs and JSP tags

As a portal developer, you can use JavaServer Pages (JSPs) to rapidly develop and easily maintain dynamic web pages that leverage existing business systems. By using JSPs, you can quickly develop web-based applications that are platform independent. By separating the user interface from content generation, JSPs allow you to change the overall page layout without altering the underlying dynamic content.

Key components of a JSP are the JSP tags, simple code that allows you to easily develop JSPs without using any Java code. JSP tags are XML-like tags and scriptlets written in Java that encapsulate the logic that generates the content for the page. WebLogic Portal ships with a vast library of JSP tags for use in such tasks as creating webflows and pipelines, building a product catalog, developing campaigns, and integrating content management systems.

EJBs

Enterprise Java Beans (EJBs) allow you to write software components that execute business logic that runs on the server. With EJB transaction and state management, multithreading, and resource pooling are left to the server implementation. In WebLogic Portal, EJBs comprise the enterprise application layer shown in Figure 1-2 and perform such functions are loading pipelines into Web applications.

Unified User Profile

In WebLogic Portal, users are represented by user profiles. A user profile employs a user's ID to access such properties for that user as age or e-mail address. A Unified User Profile incorporates user data from external data sources in addition to or instead of LDAP servers, such as a legacy system or database, so that the user can access data from many different sources through a single profile. During portal development, you will create this profile so that your Portal application can retrieve data from multiple external sources.

Other Useful Features

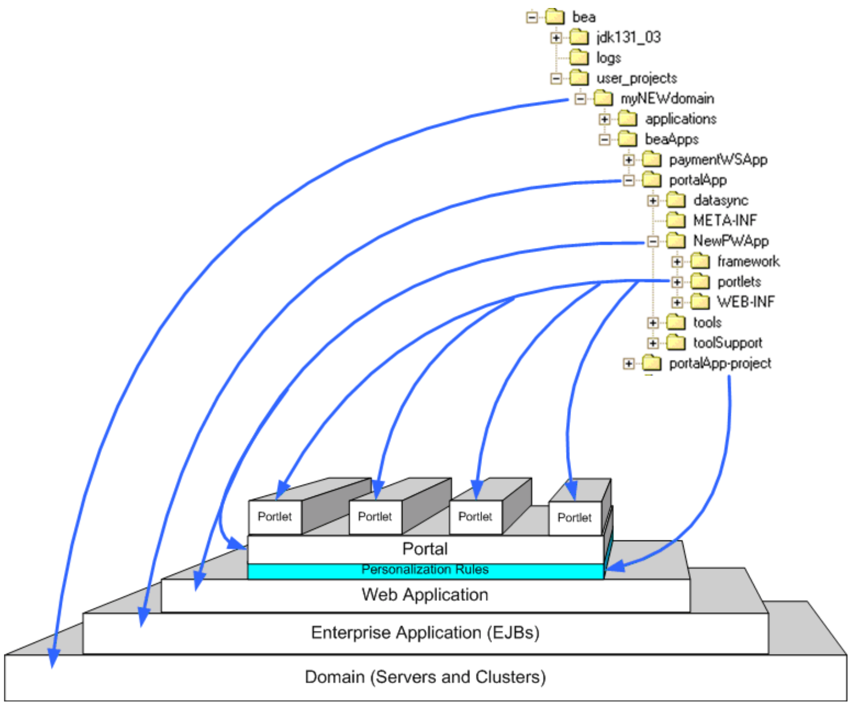
WebLogic Portal also provides these other features that facilitate portal development:

- A layout paradigm for dynamic, interactive, personalized content
- Content modules called portlets, laid out in rectangular grids
- The ability to personalize portlets at many levels
- The ability to save customized layout settings made by a visitor to your Web site
- The ability to define multiple views for the same group portal
- The ability to designate and customize color schemes called skins
- Delegated administration, which enables complex, distributed security implementations

Portal Component File Locations

When you install WebLogic Portal, you automatically create a file structure that represents your development environment. Figure 1-1 shows the relationship between generic portal architecture and where the files that compose the various levels reside in the file structure.

Figure 1-1 Portal Component File Locations



- Everything is built upon a `domain`, which is composed of an application server (for example WebLogic Server) and clustered servers. All files and directories that make up the full portal architecture are contained in the domain folder (**myNEWdomain**).
- The *enterprise application* sits on top of the domain. A domain can have multiple enterprise applications, which are characterized by a set of EJBs you can implement to create functionality in your Web applications and portals. The files and directories that make up the enterprise application are contained in the **portalApp** folder.
- Each enterprise application can host multiple *Web applications*, the basis for your portals. The files that comprise a Web application include deployment descriptors, configuration files, and the Java Archive (`.jar`) files that contain the logic and formatting instructions for the web application and the portal it supports. Files and subdirectories that comprise a Web application are contained in the Web application folder (**newPWApp**).

- The Web application also contains the logic that administers and enforces the *personalization rules* that help to channel the desired content to a specific visitor. These rules determine what content a visitor can interact with and how that information will appear to the specific visitor. The files that define the personalization layer of the WebLogic Portal architecture are contained in subdirectories of the **portalApp-project** file. These files are used in the E-Business Control Center and include configuration files for Webflow and Pipeline, events, portals and portlets, properties, and skins layouts.
- A Web application can have only one *portal*. This layer is the user interface of WebLogic Portal: it contains the components the visitor actually sees and interacts with when using WebLogic Portal. The files that comprise the Portal application include deployment descriptors, configuration files, and the Java Archive (`.jar`) files that contain the logic and formatting instructions for the Portal application. Files and subdirectories that comprise a Portal application are contained in the same folder as the Web application (**newPWApp**).
- The top layer of the WebLogic Portal architecture are the *portlets*, subsets of portals. The files that make up portlets include `.jsp` files and image files, such as `.gif` files. These files are contained in the **portlets** folder.

Roadmap for Building a Portal

This section describes the tasks required to create a portal and portal ressources and shows you how WebLogic Portal helps you complete those tasks. It answers these questions:

- How do I Build a Portal?
- How Can I Extend these Portals?

How do I Build a Portal?

WebLogic Portal makes developing portals and portal applications easy, whether you are building a portal for a new domain or for an existing domain. BEA provides "Wizards"— GUIs in to which you enter portal configuration and setup information—

that enable you to create and configure portals and portlets without having to know Java, XML, or HTML. You simply complete the data requested by these wizards and the portal, with its requisite domain, enterprise application, Web application, and portlet, are created.

For example, if you are creating a portal for a new domain with the Domain Configuration wizard, you:

1. Create an enterprise application that will support your Web application.
2. Create a new domain for your portal application

Then, by using the Portal wizard, you:

3. Create the Web application to support your portal.
4. Create a new portal and add a portlet to that portal.
5. Deploy the portal and portlet.

Additionally, by using the Portlet wizard, you can add more portlets to your portal.

By using these wizards, you can build a functional portal in less than an hour.

Part 1 of this guide, "Portal Development Tutorial" walks you through the steps outlined above to build a new portal with a new domain and then deploy that portal. In addition, it will show you how to enable an existing domain to host a new portal.

How Can I Extend these Portals?

Once you have a portal in place, you can extend it by adding features and functionality to increase its value to your enterprise. Among the ways you can extend a portal are:

- Adding more portlets
- Adding static or dynamic content
- Personalizing it for a specific user
- Creating a navigation scheme called a webflow and enhancing the functionality of the webflow by adding a pipeline to it
- Integrating it with third party systems and services, such as LDAP servers and search engines

- Integrating it with commerce services such as payment and taxation services
- Modifying its default look-and-feel by adding skins or changing the layout.

These and many other ways of extending a portal are described in Part II, "Extending Portals" on page -15

How Do I Get Started?

With the basic background on portals and portlets presented in this section, you can now begin building portals. This guide is structured to allow you to both develop a portal and extend its functionality.

While the procedures contained in this guide will show you what you need to know to develop portals and portlets, you should also do some advanced planning to enable your portal to fully support your enterprise.

The following list suggests some activities you need to consider before building your portal. This list is not a comprehensive planning guide for a new portal, however it should provide sufficient guidance for getting you started.

Before actually developing a portal, you should:

- Determine the tools you will use to create, test, debug, and deploy your portals.
- Determine which business needs the portal will address and whether you need to build a new portal or modify an existing one.
- Identify the portal audience by defining users and groups.
- Identify the portal components; that is, what will be available in the portal.
- Identify portal management roles and responsibilities; that is, who are the administrators (SA, PA, GA) and what are they required to do?
- If you are developing a new portal, build the wireframes for the portal and its portlets.
- Create the HTML mock-ups of the portal and portlets to model the desired look-and-feel.

- Collect or identify the specific content and determine what processes will be required to make it available in the portal and portlets.

You are now ready to begin building your portal.

Part I Developing Portals – Tutorials

Part I, "Developing Portals – Tutorials," shows you how to build and deploy portals. You will learn how to create and deploy a portal both for a new domain and for an existing domain, alongside another Web application, for example. By the time you are finished using these tutorials, you will be able to quickly create and deploy a portal, with portlets, using BEA-supplied resources.

This section includes information on the following subjects:

- Creating a New Portal in a New Domain
- Adding Portal to an Existing Domain
- Deploying Portals

After you have created and deployed portals, you can extend and add functionality to those portals. For those procedures, see the Extending Portals section of this guide.

CHAPTER

2 Creating a New Portal in a New Domain

Step 1: Create the New Domain

This section shows how to run the Domain Configuration Wizard to create a new complete set of enterprise applications which include all the administration, commerce, personalization and portal functionality offered by the WebLogic Portal platform.

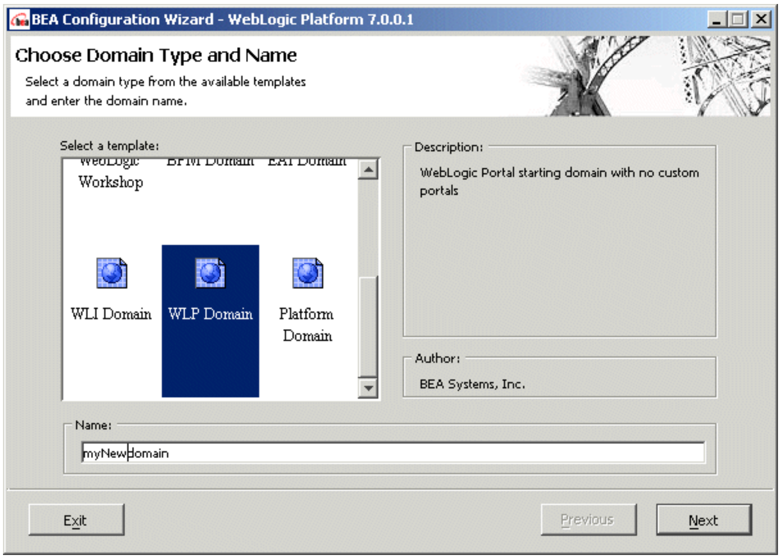
1. On the Windows platform, select Start →Programs →BEA WebLogic Platform 7.0 →Domain Configuration Wizard.

   You can also launch the Domain Configuration Wizard by executing `dmwiz.cmd` (or `dmwiz.sh` on UNIX) from the following directory:

   ```
   <BEA_HOME>\weblogic700\common\bin
   ```

2. From the list of domain templates on the left, select **WLP Domain** as shown in Figure 2-1. Name the domain; examples in this procedure use the name **`myNewDomain`**. Click **Next**.

Figure 2-1 Select the WLP Domain



BEA Configuration Wizard - WebLogic Platform 7.0.0.1

Choose Domain Type and Name

Select a domain type from the available templates and enter the domain name.

Select a template:

Description:

WebLogic Portal starting domain with no custom portals

Author:

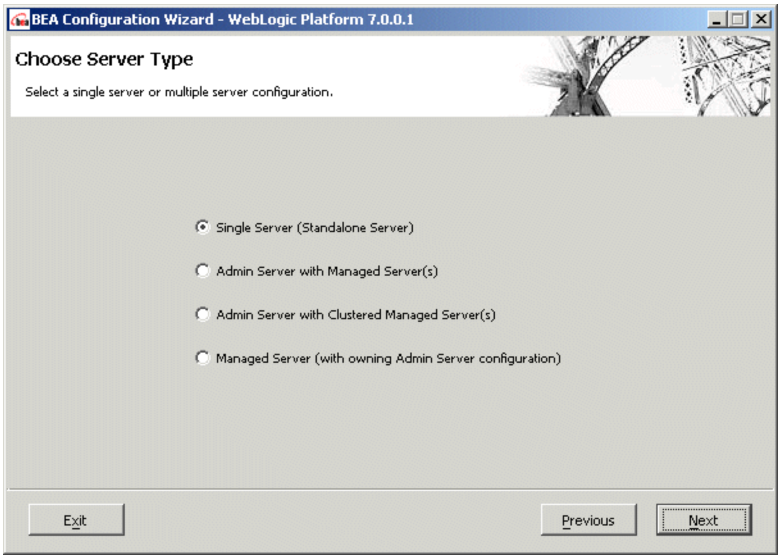
BEA Systems, Inc.

Name:

myNewDomain

3. In the Choose Server Type page, verify that **Single Server (StandAlone Server)** is selected, as shown in Figure 2-2, and click **Next**.

Figure 2-2 Choose Server Type



Choose Server Type

Select a single server or multiple server configuration.

(●) Single Server (Standalone Server)

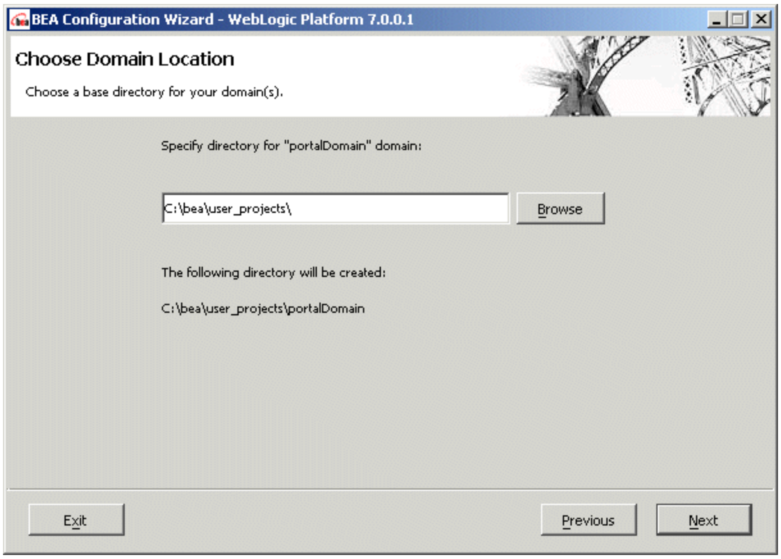
( ) Admin Server with Managed Server(s)

( ) Admin Server with Clustered Managed Server(s)

( ) Managed Server (with owning Admin Server configuration)

4. Verify that the domain location is correct in the Choose Domain Location page. For this example, it should be `<BEA_HOME>\user_projects`, as shown in Figure 2-3. Click **Next**.

Figure 2-3 Choose Domain Location


BEA Configuration Wizard - WebLogic Platform 7.0.0.1
Choose Domain Location
Choose a base directory for your domain(s).
Browse
Exit
Previous
Next

Specify directory for "portalDomain" domain:

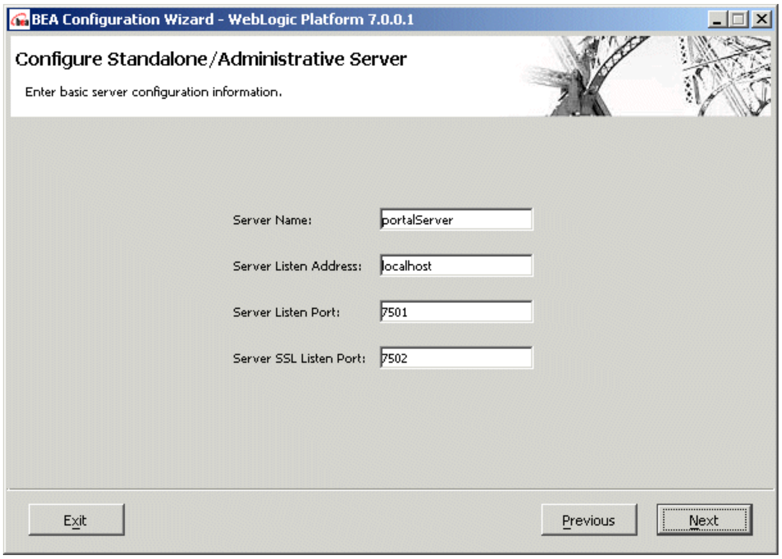
C:\bea\user_projects\

The following directory will be created:

C:\bea\user_projects\portalDomain

5. Check the displayed server information in the Configure Standalone/Administrative Server page. If you are running WebLogic Portal locally, the information should be as shown in Figure 2-4. Click **Next**.

Figure 2-4 Configure Server

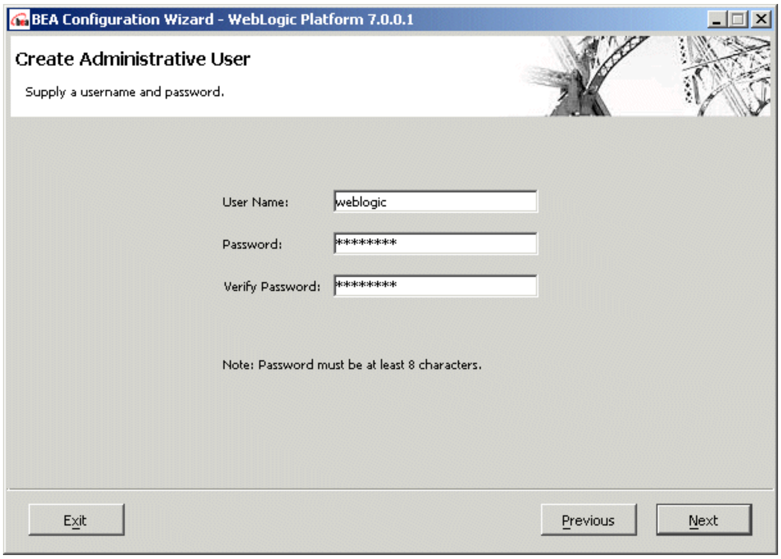

BEA Configuration Wizard - WebLogic Platform 7.0.0.1
Configure Standalone/Administrative Server
Enter basic server configuration information.
Exit
Previous
Next

Server Name: portalServer
Server Listen Address: localhost
Server Listen Port: 7501
Server SSL Listen Port: 7502

Note: For more information on options in the Domain Configuration Wizard, consult “Using the Configuration Wizard” at http://edocs.bea.com/platform/docs70/confgwiz/index.html.

6. Create the administrative user by entering an administrator user name and password. A typical choice is `weblogic/weblogic`, as shown in Figure 2-5. Click Next.

Figure 2-5 Create Administrative User



Create Administrative User

Supply a username and password.

User Name: weblogic

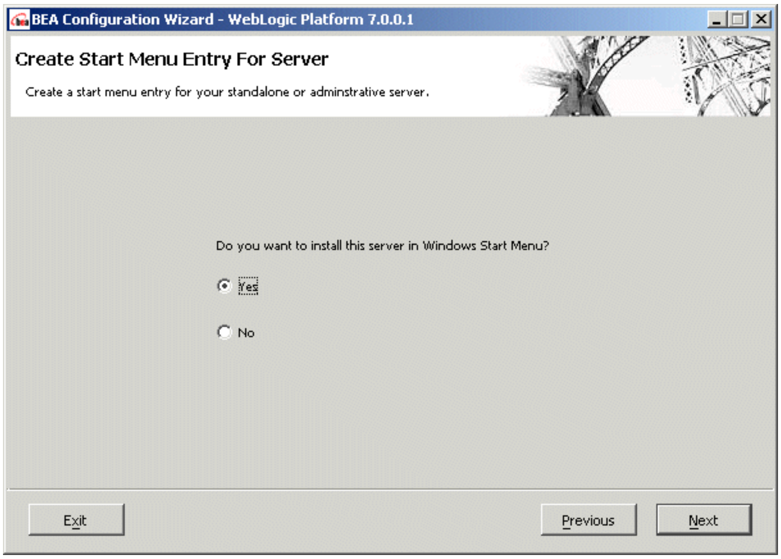
Password: ********

Verify Password: ********

Note: Password must be at least 8 characters.

7. Select **Yes** on the Create Start Menu Entry for Server page, as shown in Figure 2-6, then click Next.

Figure 2-6 Create Start Menu Entry


BEA Configuration Wizard - WebLogic Platform 7.0.0.1

Create Start Menu Entry For Server

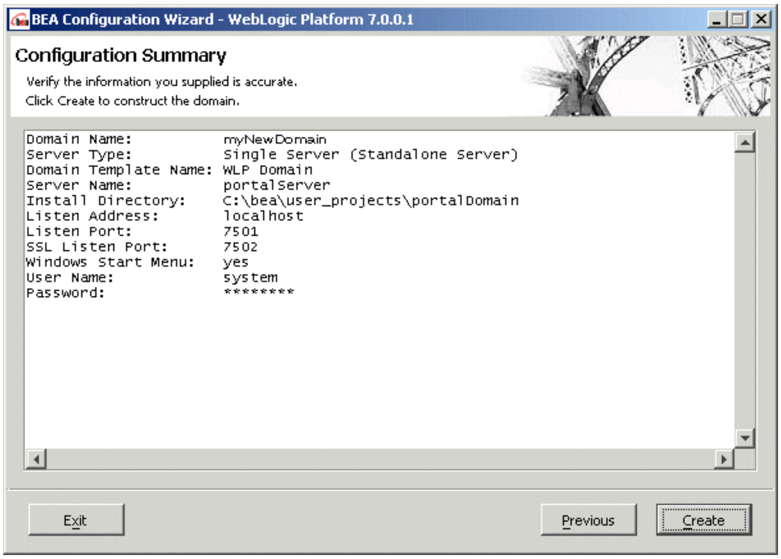
Create a start menu entry for your standalone or adminstrative server.

Do you want to install this server in Windows Start Menu?

[✓ Yes] No

8. Verify the settings in the Configuration Settings window, as shown in Figure 2-7, and click **Create**. The Wizard runs for a moment, processing and creating files.

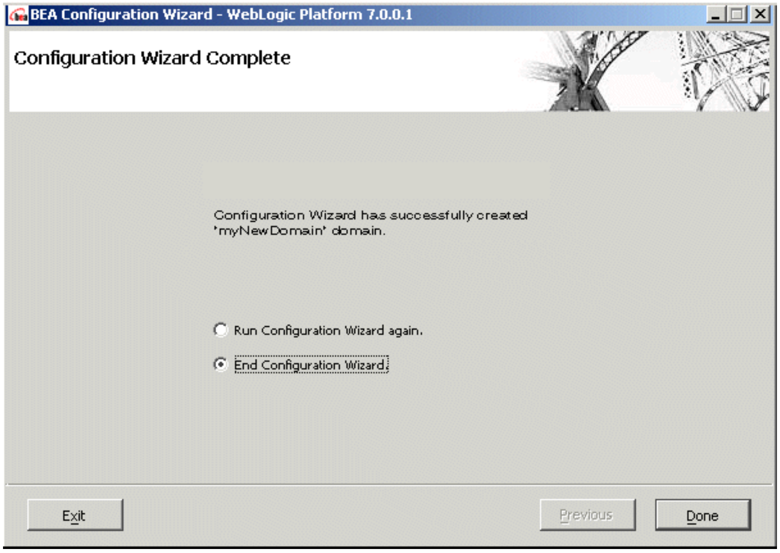
Figure 2-7 Verify Configuration Summary


BEA Configuration Wizard - WebLogic Platform 7.0.0.1
Configuration Summary
Verify the information you supplied is accurate.
Click Create to construct the domain.
Exit
Previous
Create

```
Domain Name:           myNewDomain
Server Type:           Single Server (Standalone Server)
Domain Template Name:  WLP Domain
Server Name:           portalServer
Install Directory:     C:\bea\user_projects\portalDomain
Listen Address:        localhost
Listen Port:           7501
SSL Listen Port:       7502
Windows Start Menu:    yes
User Name:             system
Password:              ********
```

9. The Configuration Wizard Complete page appears. Make sure **End Configuration Wizard** is selected and click **Done**, as shown in Figure 2-8.

Figure 2-8 End Configuration Wizard



Configuration Wizard Complete

Configuration Wizard has successfully created 'myNewDomain' domain.

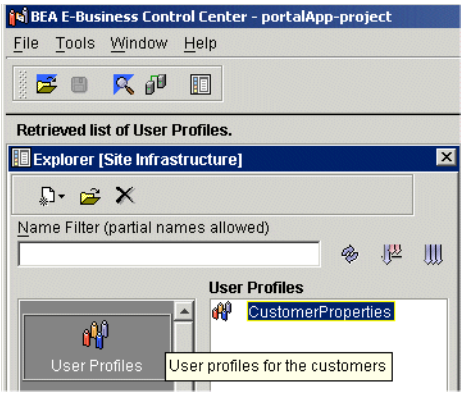
( ) Run Configuration Wizard again.
(●) End Configuration Wizard.

You are now ready to create a new portal and an associated portal Web application.

Notes on the Resources You Just Created

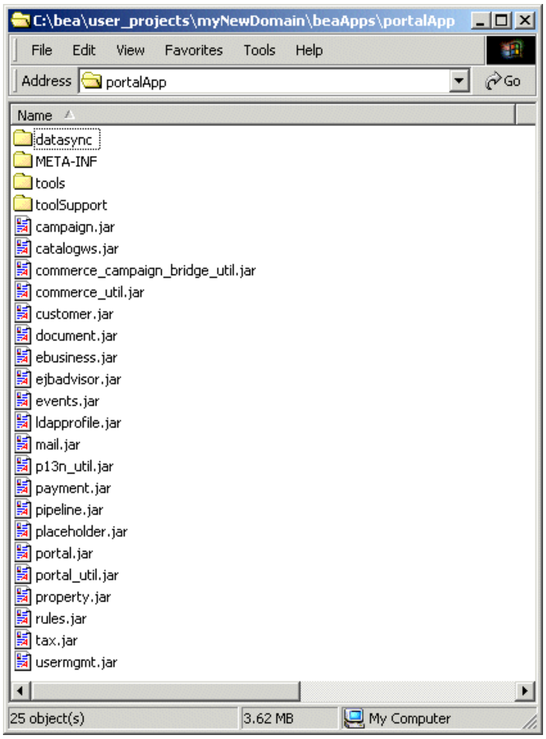
Site Infrastructure Provided by the Configuration Wizard: As part of creating the new domain, the domain wizard also creates a complete enterprise application called `portalApp`. With the `portalApp-project` file open in the E-Business Control Center, click the **Site Infrastructure** tab and select **User Profiles**, as shown in Figure 2-9. Notice that a user property set called `CustomerProperties` is already in place.

Figure 2-9 User Profiles


BEA E-Business Control Center - portalApp-project
File Tools Window Help
Retrieved list of User Profiles.
Explorer [Site Infrastructure]
Name Filter (partial names allowed)
User Profiles
CustomerProperties
User Profiles
User profiles for the customers

J2EE Resources Provided by the Configuration Wizard: The new enterprise application, as yet a blank template, includes built-in support for foundation services, personalization, interaction management, intelligent administration, and integration services. A look into the newly created directory shows the JARs used to implement these services, as shown in Figure 2-10.

Figure 2-10 JARs in the New Enterprise Application Directory


C:\bea\user_projects\myNewDomain\beaApps\portalApp
File Edit View Favorites Tools Help
Address portalApp
Go
Name

```
datasync
META-INF
tools
toolSupport
campaign.jar
catalogws.jar
commerce_campaign_bridge_util.jar
commerce_util.jar
customer.jar
document.jar
ebusiness.jar
ejbadvisor.jar
events.jar
ldapprofile.jar
mail.jar
p13n_util.jar
payment.jar
pipeline.jar
placeholder.jar
portal.jar
portal_util.jar
property.jar
rules.jar
tax.jar
usermgmt.jar
```

25 object(s) 3.62 MB My Computer

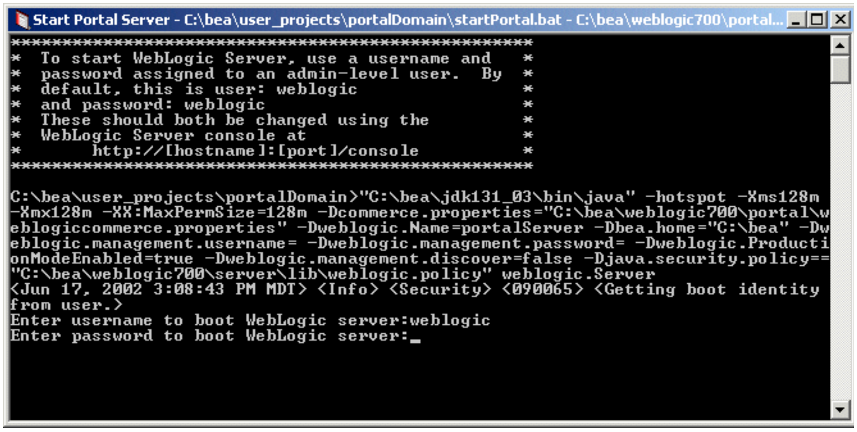
Step 2: Create the New Portal

With the supporting enterprise application resources in place, take the following steps to create and deploy a new portal.

1. Start the server by selecting Start →Programs →BEA WebLogic Platform 7.0 → User Projects →<new domain name> Start Portal Server. (If you followed the examples in this procedure, the new domain name is `myNewDomain`.)

2. When prompted for credentials, enter the username and password you created in Figure 2-5, such as `weblogic/weblogic`. The login is shown in Figure 2-11.

Figure 2-11 Entering the Username and Password You Created for the Domain



Start Portal Server - C:\bea\user_projects\portalDomain\startPortal.bat - C:\bea\weblogic700\portal...

```
*****************************************************
*  To start WebLogic Server, use a username and    *
*  password assigned to an admin-level user.  By   *
*  default, this is user: weblogic                 *
*  and password: weblogic                          *
*  These should both be changed using the          *
*  WebLogic Server console at                      *
*       http://[hostname]:[port]/console           *
*****************************************************

C:\bea\user_projects\portalDomain>"C:\bea\jdk131_03\bin\java" -hotspot -Xms128m
-Xmx128m -XX:MaxPermSize=128m -Dcommerce.properties="C:\bea\weblogic700\portal\w
eblogiccommerce.properties" -Dweblogic.Name=portalServer -Dbea.home="C:\bea" -Dw
eblogic.management.username= -Dweblogic.management.password= -Dweblogic.Producti
onModeEnabled=true -Dweblogic.management.discover=false -Djava.security.policy==
"C:\bea\weblogic700\server\lib\weblogic.policy" weblogic.Server
<Jun 17, 2002 3:08:43 PM MDT> <Info> <Security> <090065> <Getting boot identity
from user.>
Enter username to boot WebLogic server:weblogic
Enter password to boot WebLogic server:_
```

Note: **For UNIX platform only:** By default, none of the new domain scripts have executable privileges, so they must be granted these privileges by a system administrator.

3. Launch the E-Business Control Center by selecting Start →Programs →BEA WebLogic Platform 7.0 →WebLogic Portal 7.0 →E-Business Control Center.

4. When it has started, choose File →Open and open the `portalApp-project.eaprj` project file inside the `<BEA_HOME>` `user_projects\`<new domain name>`\beaApps\portalApp-project` directory, as shown in Figure 2-12 and Figure 2-13.

Figure 2-12 A: Opening a Project File

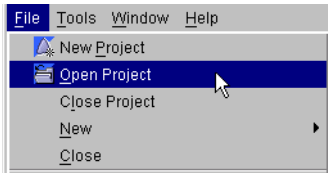
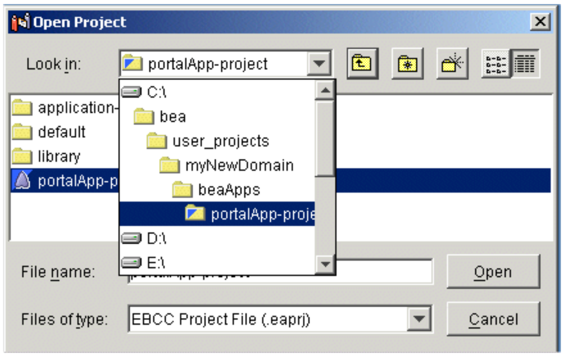
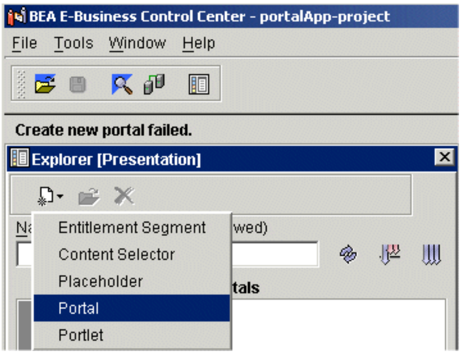


Figure 2-13 B: Opening a Project File



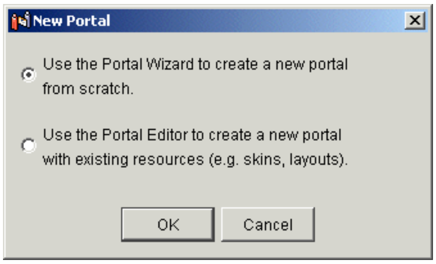
5. Click the **Presentation** tab of the E-Business Control Center.
6. Click the **New** icon in the Explorer toolbar and select **Portal**, as shown in Figure 2-14.

Figure 2-14 Opening the New Portal dialog



7. Be sure the **Use the Portal Wizard** option is selected, as shown in Figure 2-15. Click OK.

Figure 2-15 Portal Wizard Screen

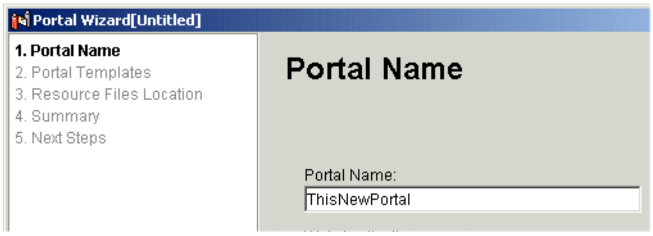

New Portal

[✓ Use the Portal Wizard to create a new portal from scratch.]

Use the Portal Editor to create a new portal with existing resources (e.g. skins, layouts).

8. Name the new portal, as shown in Figure 2-16. Examples in this procedure use the portal name **ThisNewPortal**.

Figure 2-16 Naming the New Portal


Portal Wizard[Untitled]

1. Portal Name
2. Portal Templates
3. Resource Files Location
4. Summary
5. Next Steps

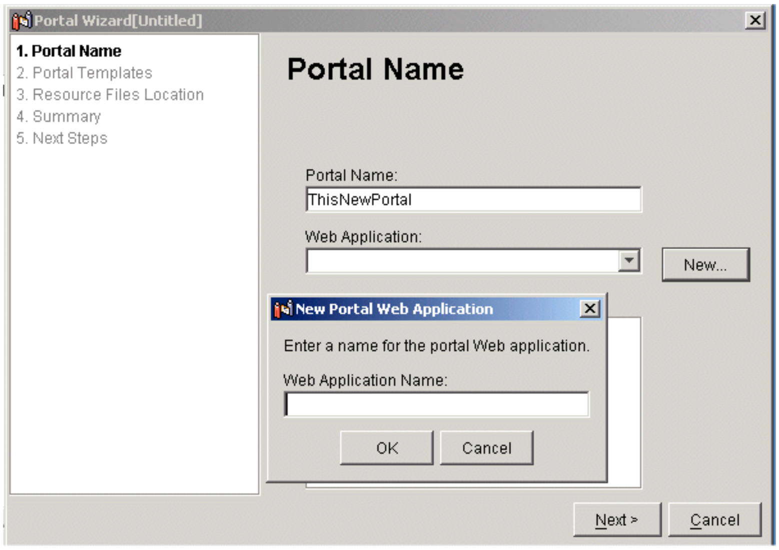
Portal Name

Portal Name: ThisNewPortal

9. Click the **New** button shown in Figure 2-17 to create a new portal Web application.

10. Enter the portal name; in this procedure, the example name **NewPortalWebApp** is used. Click **OK**.

Figure 2-17 Naming the New Portal Web Application



Portal Name: ThisNewPortal

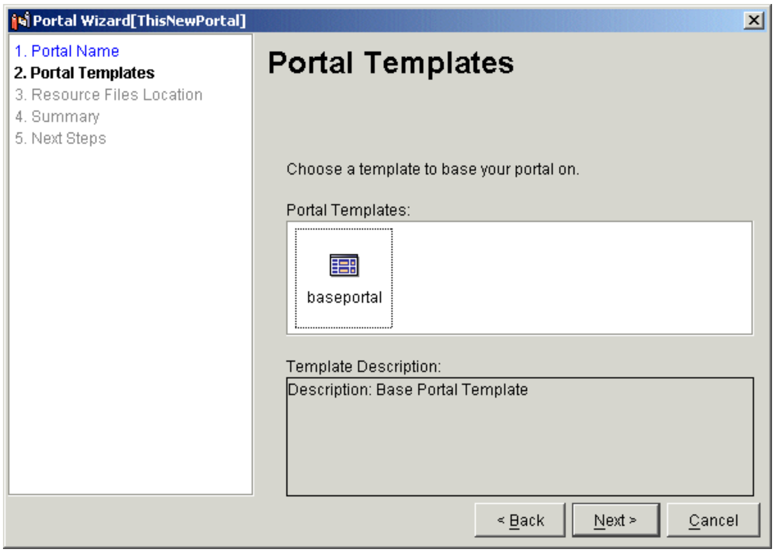
Web Application:

Enter a name for the portal Web application.

Web Application Name:

11. Select a portal template, as shown in Figure 2-18. Click **Next**.

Figure 2-18 Select a Portal Template



Portal Wizard[ThisNewPortal]

1. Portal Name
2. Portal Templates
3. Resource Files Location
4. Summary
5. Next Steps

Portal Templates

Choose a template to base your portal on.

Portal Templates:

baseportal

Template Description:

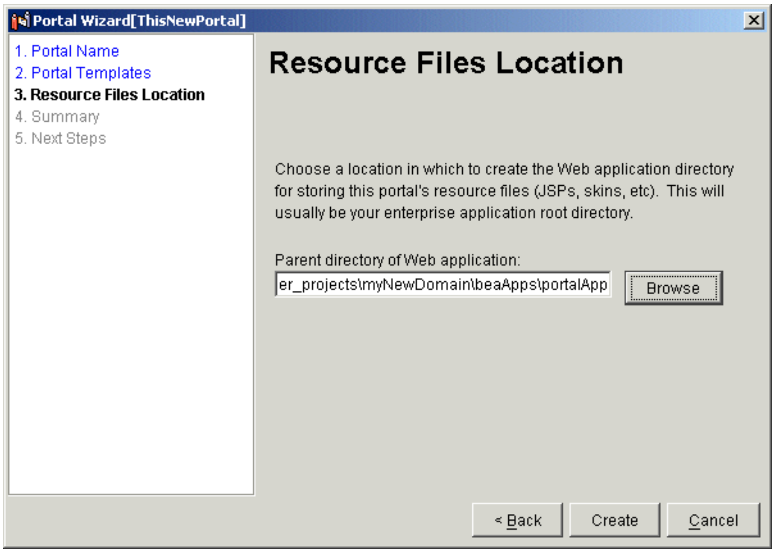
Description: Base Portal Template

12. In the Resource Files Location window (Figure 2-19), verify that the location for J2EE resources for the new Portal Web Application is correct.

Note: If you are running the E-Business Control Center on the server machine, the default location should be correct.

13. Click **Create**.

Figure 2-19 Select a Location for Resource Files



Resource Files Location

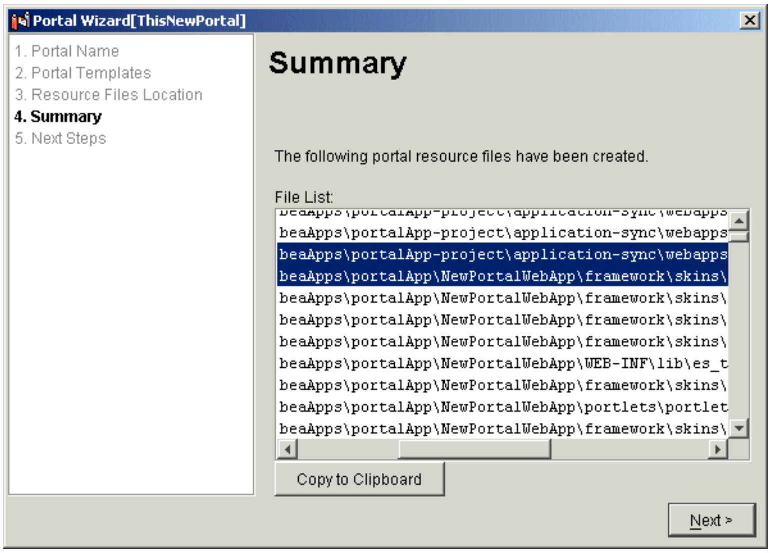
Choose a location in which to create the Web application directory for storing this portal's resource files (JSPs, skins, etc). This will usually be your enterprise application root directory.

Parent directory of Web application:

er_projects\myNewDomain\beaApps\portalApp

14. The Portal Wizard creates files and lists them in the Summary page shown in Figure 2-20. Click Next.

Figure 2-20 Summary



Portal Wizard[ThisNewPortal]

Summary

The following portal resource files have been created.

File List:

```
beaApps\portalApp-project\application-sync\webapps
beaApps\portalApp-project\application-sync\webapps
beaApps\portalApp-project\application-sync\webapps
beaApps\portalApp\NewPortalWebApp\framework\skins\
beaApps\portalApp\NewPortalWebApp\framework\skins\
beaApps\portalApp\NewPortalWebApp\framework\skins\
beaApps\portalApp\NewPortalWebApp\framework\skins\
beaApps\portalApp\NewPortalWebApp\WEB-INF\lib\es_t
beaApps\portalApp\NewPortalWebApp\framework\skins\
beaApps\portalApp\NewPortalWebApp\portlets\portlet
beaApps\portalApp\NewPortalWebApp\framework\skins\
```

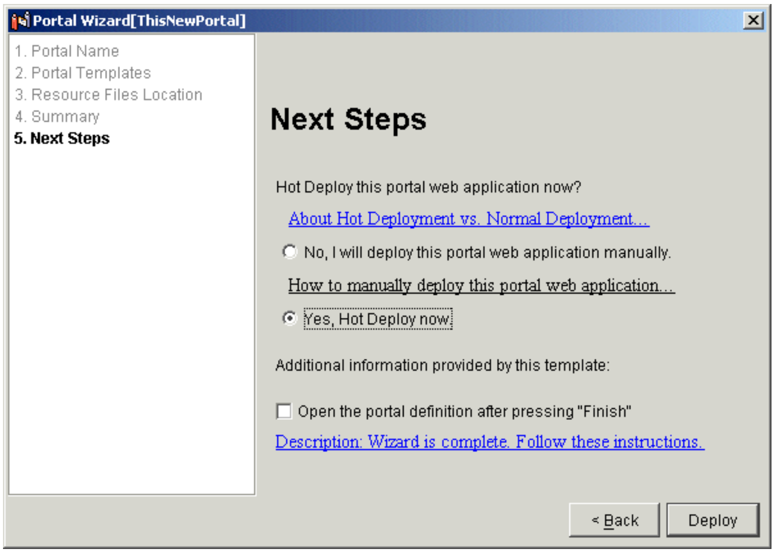
Note: Listing 2-1 includes examples of two types of files are created automatically by the Portal Wizard; J2EE resources such as `select_page_view.gif`, and metadata used by the portal framework, such as `security.wf`. Notice the different directories used to store these files.

Listing 2-1 Different Types of Portal Resources

```
\portalApp\NewPortalWebApp\framework\skins\futurism\images\select_page_view.gif

\portalApp-project\application-sync\webapps\NewPortalWebApp\security.wf
```

15. Deploy the new portal: Select the **Yes, Hot Deploy Now** radio button and click **Deploy**, as shown in Figure 2-21.

Figure 2-21 Hot Deploying the New Portal



Next Steps

Hot Deploy this portal web application now?

About Hot Deployment vs. Normal Deployment...

No, I will deploy this portal web application manually.

How to manually deploy this portal web application...

[✓ Yes, Hot Deploy now]

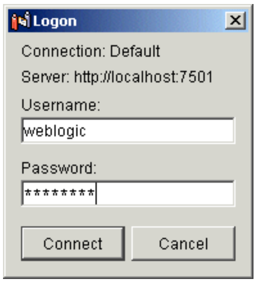
Additional information provided by this template:

☐ Open the portal definition after pressing "Finish"

Description: Wizard is complete. Follow these instructions.

16. When prompted, enter the administrator username and password you created in Figure 2-5, **`weblogic/weblogic`**.

Figure 2-22 Enter the Administrator Username and Password



Connection: Default

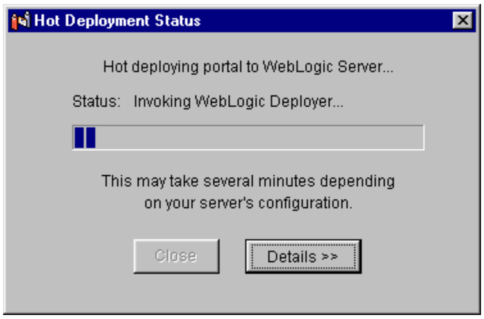
Server: http://localhost:7501

Username: weblogic

Password: ********

17. The deployment process runs for a moment, displaying the window in Figure 2-23. Click Details to view the deployment log.

Figure 2-23 Processing Message During Hot Deployment



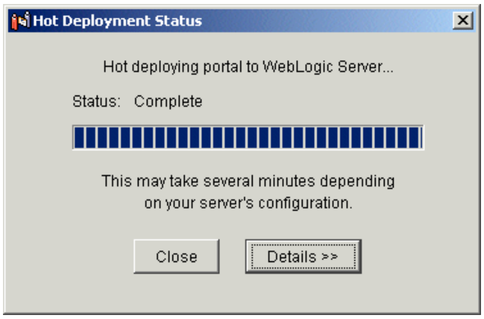
Hot deploying portal to WebLogic Server...

Status: Invoking WebLogic Deployer...

This may take several minutes depending on your server's configuration.

18. When the new portal has been deployed successfully, click **Close**, as shown in Figure 2-24.

Figure 2-24 Hot Deploy Success Message



Hot deploying portal to WebLogic Server...

Status: Complete

This may take several minutes depending on your server's configuration.

19. Confirm that the new portal is visible from within the WebLogic Portal Administration Tools, as shown in Figure 2-27, by navigating to the following URL:

```
http://<hostname>:<port>/portalAppTools/
```

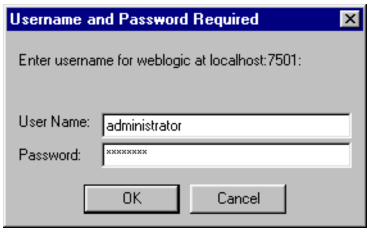
If you are running the tools locally, the URL should be:

```
http://localhost:7501/portalAppTools/
```

20. You will be prompted to log in, as shown in Figure 2-25. Log in as `administrator/password`.

Note: Do not use `weblogic/weblogic`.

Figure 2-25 Logging Into the Administration Tools



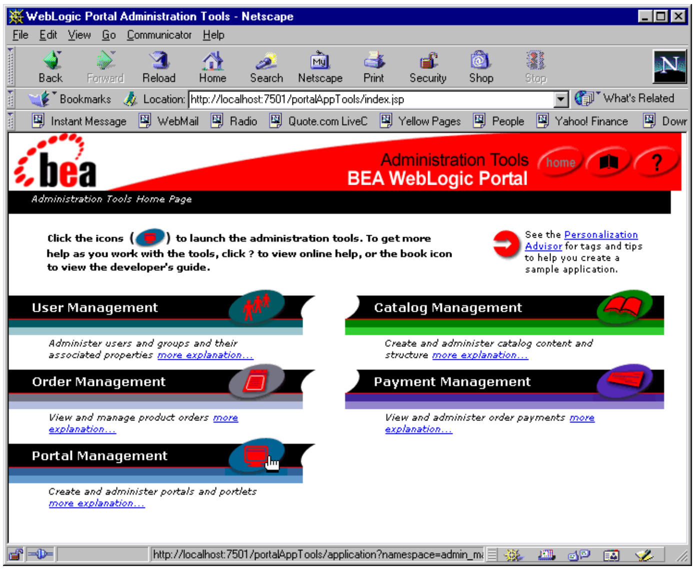
Enter username for weblogic at localhost:7501:

User Name: administrator

Password: ********

21. The main Administration Tools window appears, as shown in Figure 2-26. Click the icon to the right of the **Portal Management** heading.

Figure 2-26 Clicking the Portal Management Icon


WebLogic Portal Administration Tools - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks
What's Related
Instant Message WebMail Radio Quote.com LiveC Yellow Pages People Yahoo! Finance Dowr
bea
Administration Tools
BEA WebLogic Portal
home
?
Administration Tools Home Page
User Management
Catalog Management
Order Management
Payment Management
Portal Management

Location: http://localhost:7501/portalAppTools/index.jsp

Click the icons ( ) to launch the administration tools. To get more help as you work with the tools, click ? to view online help, or the book icon to view the developer's guide.

See the Personalization Advisor for tags and tips to help you create a sample application.

User Management: Administer users and groups and their associated properties more explanation...

Catalog Management: Create and administer catalog content and structure more explanation...

Order Management: View and manage product orders more explanation...

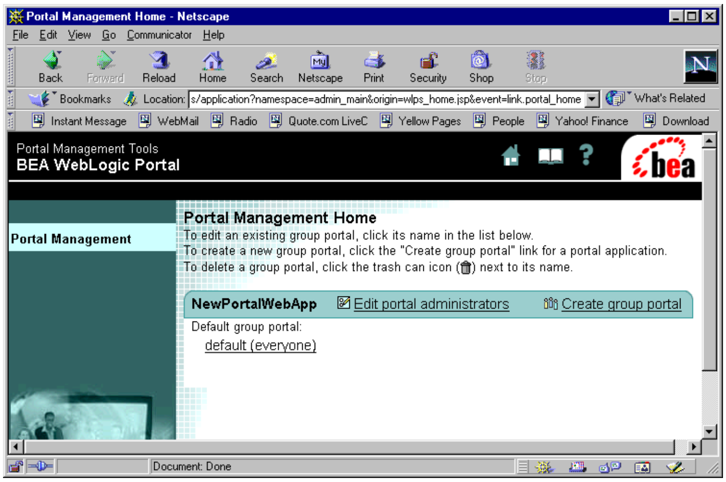
Payment Management: View and administer order payments more explanation...

Portal Management: Create and administer portals and portlets more explanation...

http://localhost:7501/portalAppTools/application?namespace=admin_m

22. The Portal Management page appears, displaying the name of your portal Web application, as shown in Figure 2-27.

Figure 2-27 Information About the Portal You Created in the Wizard, Viewed in the Administrator Tools


Portal Management Home - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: s/application?namespace=admin_main&origin=wlps_home.jsp&event=link.portal_home
What's Related
Instant Message WebMail Radio Quote.com LiveC Yellow Pages People Yahoo! Finance Download
Portal Management Tools
BEA WebLogic Portal
bea
Portal Management
Document: Done

Portal Management Home

To edit an existing group portal, click its name in the list below.
To create a new group portal, click the "Create group portal" link for a portal application.
To delete a group portal, click the trash can icon (🗑) next to its name.

NewPortalWebApp   Edit portal administrators   Create group portal

Default group portal:
default (everyone)

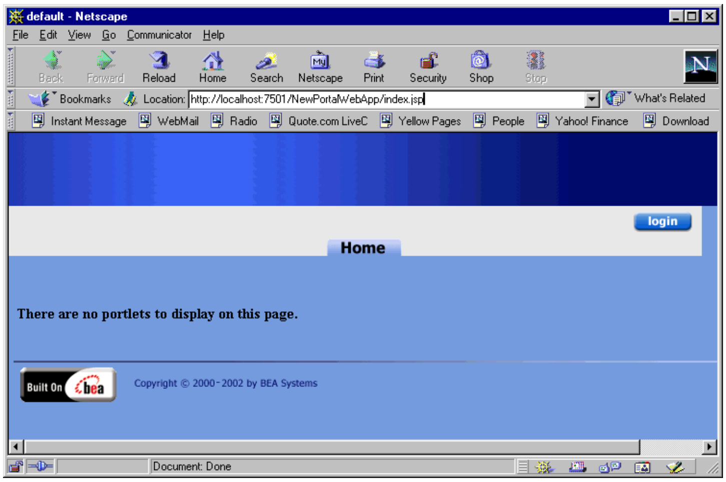
23. The new empty portal should be visible through your browser, though no content has been placed inside this portal. View the portal by entering the following in your browser.

```
http://<hostname>:<port>/<newportalwebappname/index.jsp
```

If you are running the tools locally and used the sample names provided, the URL should be:

```
http://localhost:7501/NewPortalWebApp/index.jsp
```

Figure 2-28 New Portal Viewed Through Your Browser


default - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://localhost:7501/NewPortalWebApp/index.jsp
What's Related
Instant Message WebMail Radio Quote.com LiveC Yellow Pages People Yahoo! Finance Download
login
Home
Built On bea
Document: Done

There are no portlets to display on this page.

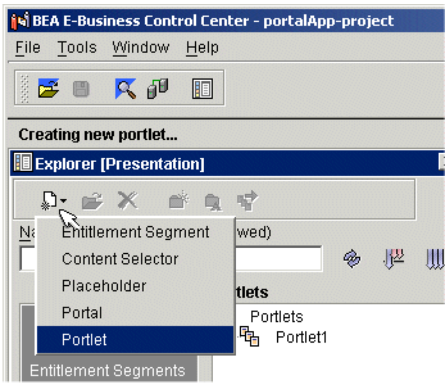
Copyright © 2000-2002 by BEA Systems

Step 3: Add a Portlet

Now that the portal is deployed and running, use the Portlet Wizard to add a new portlet to the portal.

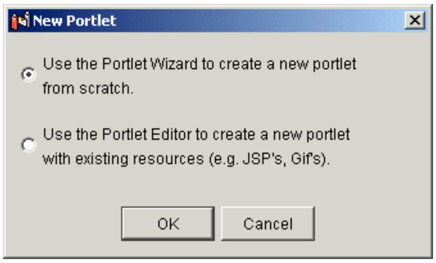
1. From the Presentation Tab in the E-Business Control Center, click the **New** icon in the Explorer and select **Portlet**, as shown in Figure 2-29.

Figure 2-29 Choosing to Create a new Portlet



2. Be sure **Use the Portlet Wizard** is selected, as shown in Figure 2-30, and click OK.

Figure 2-30 Portlet Wizard Screen



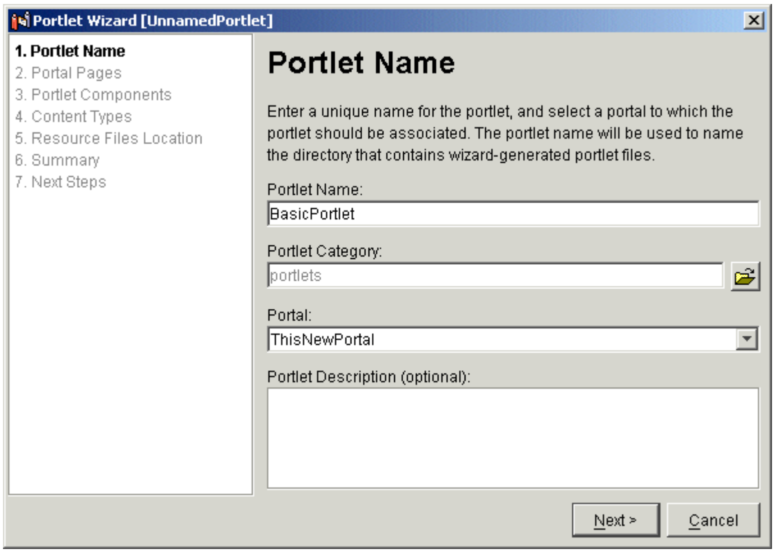
New Portlet

[✓ Use the Portlet Wizard to create a new portlet from scratch.]

Use the Portlet Editor to create a new portlet with existing resources (e.g. JSP's, Gif's).

3. Name the new portlet, as shown in Figure 2-31. Examples in this procedure use the name **BasicPortlet**. Click **Next**.

Figure 2-31 Name the Portlet



Portlet Name

Enter a unique name for the portlet, and select a portal to which the portlet should be associated. The portlet name will be used to name the directory that contains wizard-generated portlet files.

Portlet Name: BasicPortlet

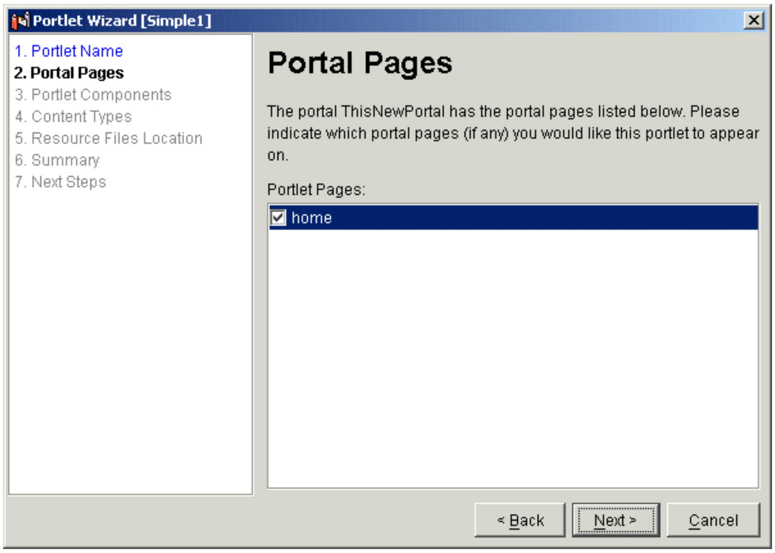
Portlet Category: portlets

Portal: ThisNewPortal

Portlet Description (optional):

4. Associate the portlet with a portal page, as shown in Figure 2-32. In this example, select the only page displayed, the **`home`** portal page.

Figure 2-32 Associate the Portlet with a Portal Page



Portal Pages

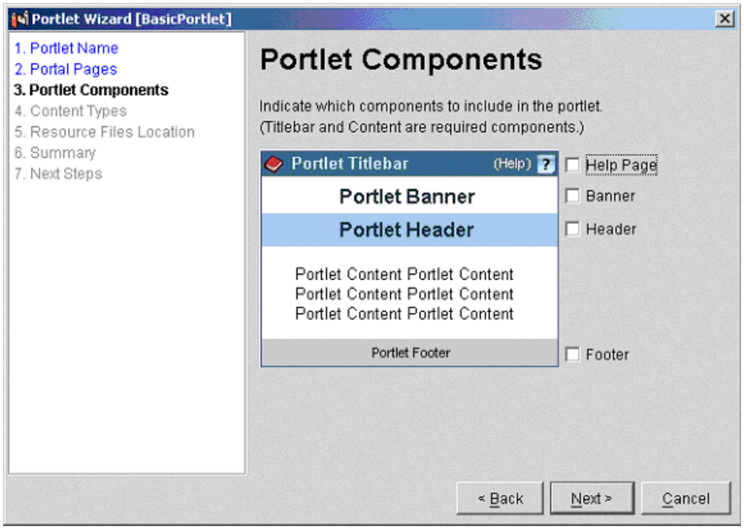
The portal ThisNewPortal has the portal pages listed below. Please indicate which portal pages (if any) you would like this portlet to appear on.

Portlet Pages:

☑ home

5. The Portlet Components page in Figure 2-33 shows components that can be added to the portlet, such as headers, banners, help and footers. For this example, do not select any additional components, and click **Next**.

Figure 2-33 Select Portlet Components

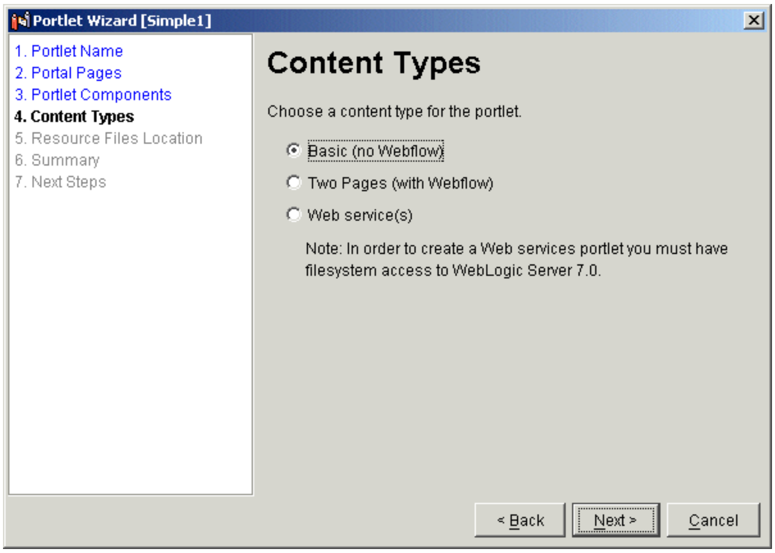


Indicate which components to include in the portlet.
(Titlebar and Content are required components.)

☐ Help Page
☐ Banner
☐ Header
☐ Footer

6. Select a portlet content type, as shown in Figure 2-34. For this example, select **Basic (no Webflow)** and click **Next**.

Figure 2-34 Select a Content Type



Content Types

Choose a content type for the portlet.

[✓ Basic (no Webflow)] Two Pages (with Webflow) Web service(s)

Note: In order to create a Web services portlet you must have filesystem access to WebLogic Server 7.0.

7. A default location for portlet resources appears; verify that it is correct. It should be something like:

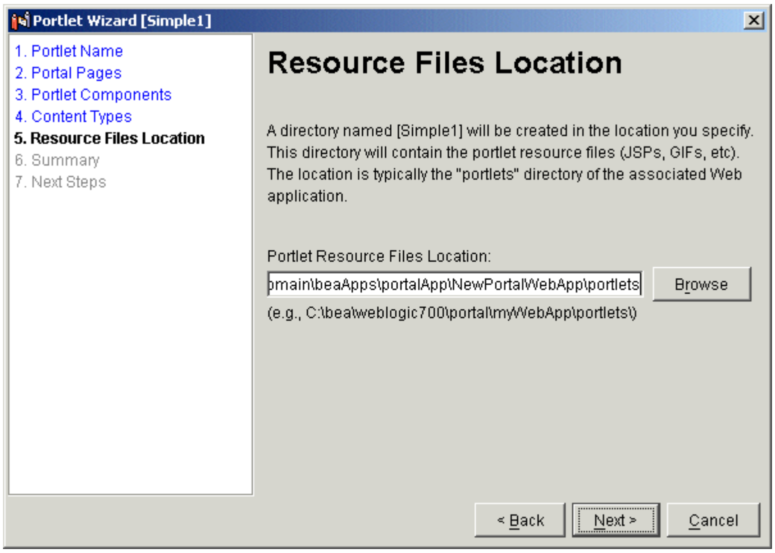
   ```
   <BEA_HOME>\user_projects\<domainname>\beaApps\
   portalApp\<portalwebappname>\portlets
   ```

   For this example, the location shown in Figure 2-35 should be correct:

   ```
   C:\bea\user_projects\myNewdomain\beaApps\
   portalApp\NewPortalWebApp\portlets
   ```

   Click **Next**.

Figure 2-35 Select Resource Files Location



Resource Files Location

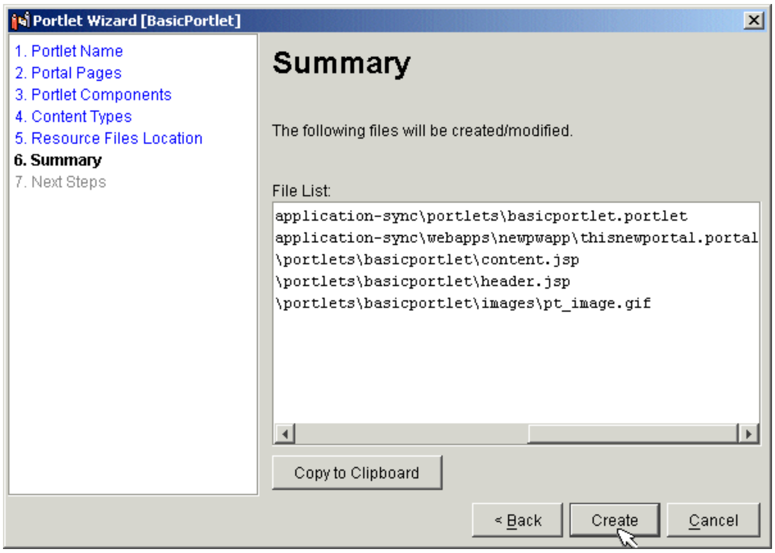
A directory named [Simple1] will be created in the location you specify. This directory will contain the portlet resource files (JSPs, GIFs, etc). The location is typically the "portlets" directory of the associated Web application.

Portlet Resource Files Location: omain\beaApps\portalApp\NewPortalWebApp\portlets

(e.g., C:\bea\weblogic700\portal\myWebApp\portlets\)

8. The Summary page in Figure 2-36 shows the files created by the Portlet Wizard. Click **Create**.

Figure 2-36 View Summary



Summary

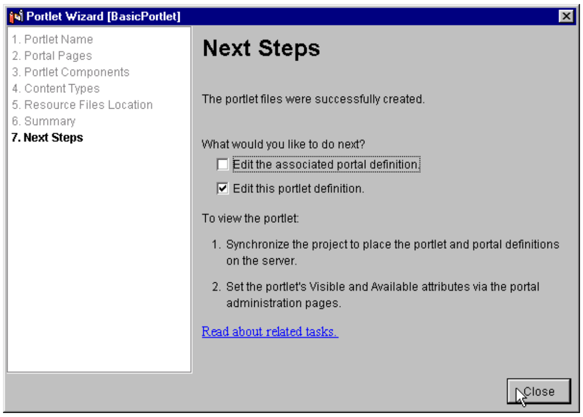
The following files will be created/modified.

File List:

```
application-sync\portlets\basicportlet.portlet
application-sync\webapps\newpwapp\thisnewportal.portal
\portlets\basicportlet\content.jsp
\portlets\basicportlet\header.jsp
\portlets\basicportlet\images\pt_image.gif
```

9. When the Next Steps page appears as shown in Figure 2-37, uncheck both boxes and click **Close**.

Figure 2-37 Unmark Options and Close the Next Steps Window



Next Steps

The portlet files were successfully created.

What would you like to do next?

- [ ] Edit the associated portal definition.
- [✓] Edit this portlet definition.

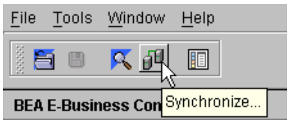
To view the portlet:

1. Synchronize the project to place the portlet and portal definitions on the server.
2. Set the portlet's Visible and Available attributes via the portal administration pages.

Read about related tasks.

10. Sync the portal project: Click the Synchronize button on the E-Business Control Center toolbar, shown in Figure 2-38.

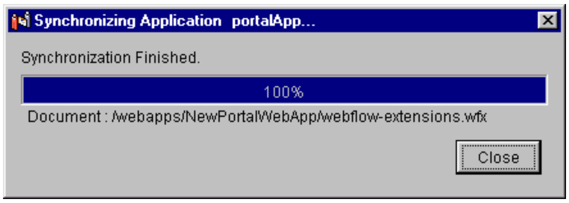
Figure 2-38 The Syncronize Button in the E-Business Control Center



Note: For this example, the Default connection settings, `localhost:7501`, should work.

11. The E-Business Control Center synchronizes the data you created in the Portlet Wizard, then displays the message in Figure 2-39. Click **Close**.

Figure 2-39 Synchronization Is Complete


Synchronizing Application portalApp...

Synchronization Finished.

100%

Document : /webapps/NewPortalWebApp/webflow-extensions.wfx

Close

12. If a window appears prompting you to reset compaign states, click **Cancel**.

To see this portlet from the browser, it must be designated as visible and available, as shown in Step 4: Make New Portlet Visible.

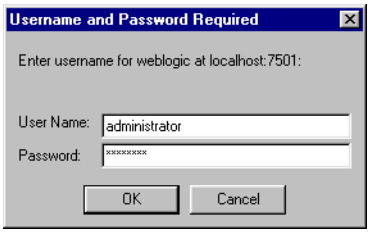
Step 4: Make New Portlet Visible

The new portlet is now on the server, but must be made available using the WebLogic Portal Administration Tools.

1. In your Web browser, navigate to the following URL: `http://<hostname>:<port>/portalAppTools`. For this example, use the URL `http://localhost:7501/portalAppTools`.

2. Log in as **`administrator`**/**`password`** as shown in Figure 2-40.

Figure 2-40 Logging Into the Administration Tools



Username and Password Required

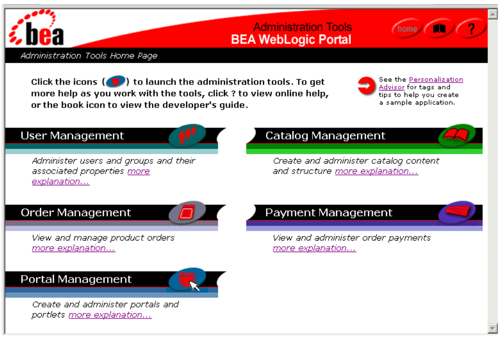
Enter username for weblogic at localhost:7501:

User Name: administrator

Password: ********

3. When the Administration Tools main page apears, click **Portal Management**, as shown in Figure 2-41.

Figure 2-41 Go to Portal Management



Administration Tools Home Page

Click the icons ( ) to launch the administration tools. To get more help as you work with the tools, click ? to view online help, or the book icon to view the developer's guide.

See the Personalization Advisor for tags and tips to help you create a sample application.

User Management

Administer users and groups and their associated properties more explanation...

Catalog Management

Create and administer catalog content and structure more explanation...

Order Management

View and manage product orders more explanation...

Payment Management

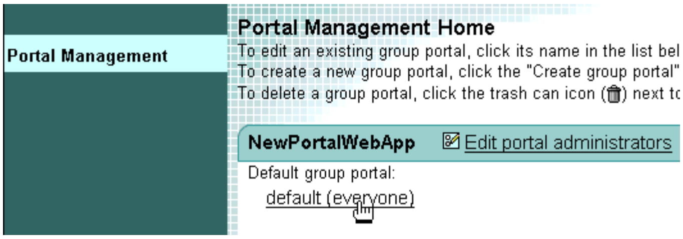
View and administer order payments more explanation...

Portal Management

Create and administer portals and portlets more explanation...

4. From the Portal Management Home page, click the Default Portal, as shown in Figure 2-42.

Figure 2-42 Select Default Group Portal



Portal Management Home

To edit an existing group portal, click its name in the list bel

To create a new group portal, click the "Create group portal"

To delete a group portal, click the trash can icon ( ) next tc

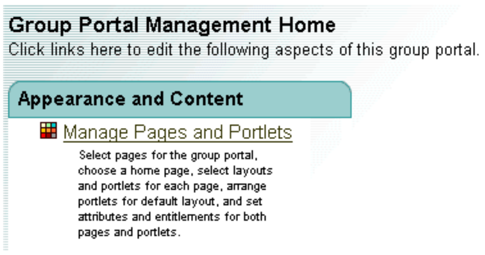
NewPortalWebApp

Default group portal:

default (everyone)

5. From the Group Portal Management Home page, click **Manage Pages and Portlets** as shown in Figure 2-43.

Figure 2-43 Manage Pages and Portlets


Group Portal Management Home

Click links here to edit the following aspects of this group portal.

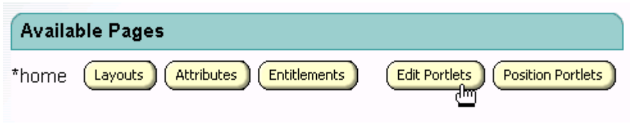
Appearance and Content

Manage Pages and Portlets

Select pages for the group portal, choose a home page, select layouts and portlets for each page, arrange portlets for default layout, and set attributes and entitlements for both pages and portlets.

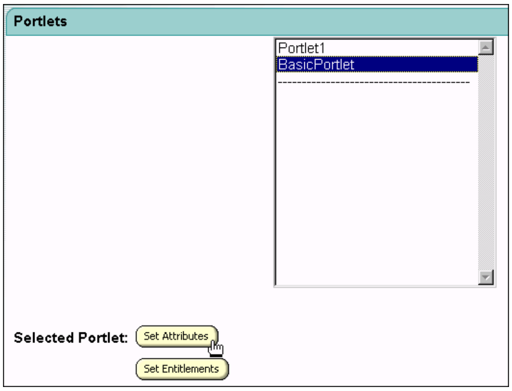
6. Next, in the Pages and Portlets page, click **Edit Portlets**, as shown in Figure 2-44.

Figure 2-44 Click Edit Portlets



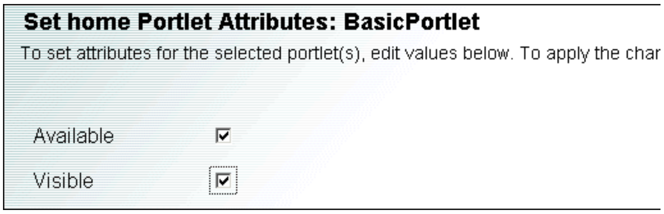
7. In the Edit Portlet Entitlements and Attributes page, select the portlet (BasicPortlet in this example), then click the **Set Attributes** button as shown in Figure 2-45.

Figure 2-45 Choose to Set Attributes



8. Set the Portlet attributes to **Visible** and **Available**, as shown in Figure 2-46.

Figure 2-46 Set Portlet Attributes



Set home Portlet Attributes: BasicPortlet

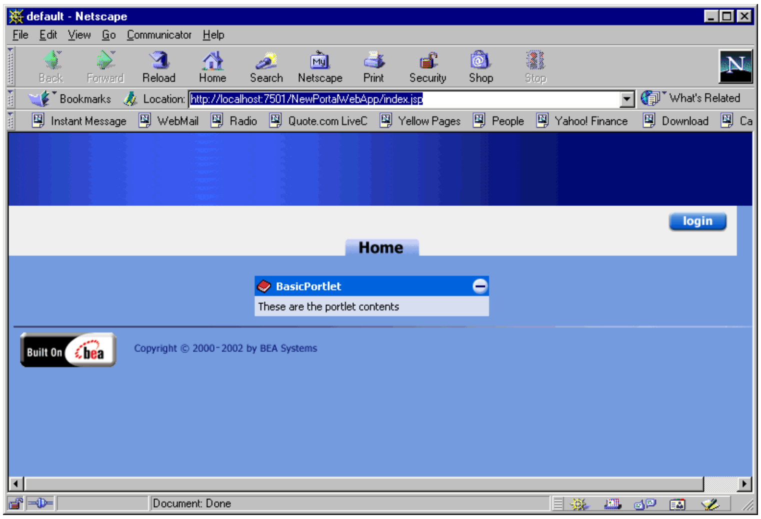
To set attributes for the selected portlet(s), edit values below. To apply the char

Available	✓
Visible	✓

9. Click **Save**. The attributes of the portlet, in this example BasicPortlet, are now set such that it can be seen.

10. View the results by navigating to `http://<hostname>:<port>/<webappname>/index.jsp;` for this example, go to `http://localhost:7501/NewPortalWebApp/index.jsp`. The result should resemble that shown in Figure 2-47.

Figure 2-47 Viewing the New Portlet



default - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://localhost:7501/NewPortalWebApp/index.jsp
What's Related
Instant Message WebMail Radio Quote.com LiveC Yellow Pages People Yahoo! Finance Download Ca
login
Home
BasicPortlet
These are the portlet contents
Built On bea
Copyright © 2000-2002 by BEA Systems
Document: Done

CHAPTER

3 Adding Portal to an Existing Domain

This section explains how to add portal functionality to an existing domain. These procedures and tasks are highly technical and require a good deal of familiarity with WebLogic Server and J2EE architecture.

About Your Domain

The overall procedure outlined in this section is based on the following assumptions about your existing domain:

- The existing domain has been backed up.
- The existing domain is installed in `<BEA_HOME>\user_projects\` directory.
- The existing domain contains a J2EE application and at least one Web application, but does not include Portal functionality.
- The existing domain, referred to in this section as `yourDomain`, must be preserved. That is, it is not being replaced with `portalDomain`.
- Those responsible for performing these tasks are familiar with the WebLogic Server platform, and with J2EE architecture in general.

Before You Begin

Arrive at the following decisions before moving on:

- Preserve or Replace the Existing Domain
- Use or Replace Existing Database
- Locate or Install Enterprise Application

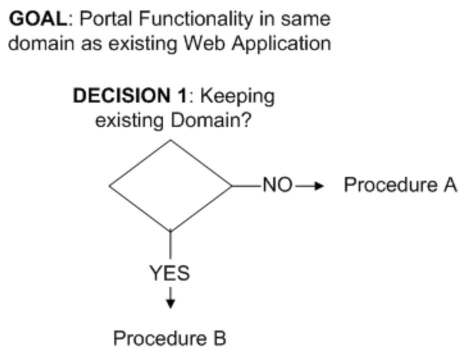
After these decisions are made, select which procedures to perform and get started.

Preserve or Replace the Existing Domain

The first decision to be made is illustrated in Figure 3-1, and concerns the domain containing your existing Web application.

- If the existing domain can be replaced with a new portal domain, follow "Procedure A" on page 3-3 to move your Web application components into a new portal domain.
- If the existing domain must be preserved, follow "Procedure B" on page 3-3 to install portal functionality within your existing domain.

Figure 3-1 Decision 1


GOAL: Portal Functionality in same domain as existing Web Application
DECISION 1: Keeping existing Domain?
NO
Procedure A
YES
Procedure B

Procedure A

1. Use the Domain Configuration Wizard to create a new portalDomain, as described in "Step 1: Create the New Domain" on page 2-1.
2. Import the objects that constitute the existing Web application into the new portal domain.
3. Use the Portal Wizard to create a new portal and portal Web application, as described in "Step 2: Create the New Portal" on page 2-11.

Once these steps have been completed, you will have a single domain containing your existing portal Web application and a complete portal Web application.

Procedure B

1. Use the Domain Configuration Wizard to create a new WLP domain named partsDomain, as described in "Step 1: Create the New Domain" on page 2-1.

Note: This `partsDomain` will be used as a template, but will never be started. Therefore, do not add a link to it on the Start menu.

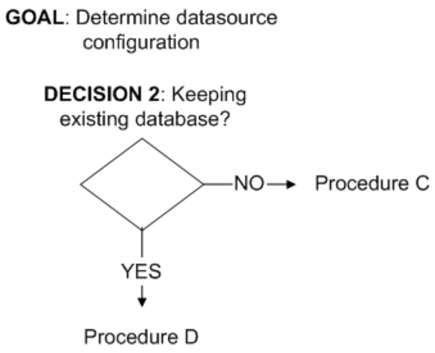
2. Continue with the remainder of the steps in this section, using the partsDomain to copy resources into the existing domain, as directed within the procedure.

Use or Replace Existing Database

Another decision, illustrated in Figure 3-2, concerns what database to use.

- If you don't need to keep the existing database, follow "Procedure C" on page 3-4 to move your Web application components into a new portal domain.
- If the existing domain must be preserved, follow "Procedure D" on page 3-4 to install portal functionality within your existing domain.

Figure 3-2 Decision 2


GOAL: Determine datasource configuration
DECISION 2: Keeping existing database?
NO
Procedure C
YES
Procedure D

Procedure C

If the existing Web application can be supported by a new database, load the database objects that support the existing Web application inside the database inside the new domain.

Procedure D

To retain the database associated with the existing Web application, the following entries must be put in place:

JDBC Entries

The following JDBC entries must be added to the `config.xml` file from the existing domain:

- CommercePool: A portal web application requires two connection pools and two data sources for each pool, as shown in Listing 3-1.

Listing 3-1 Commerce Pool Datasource entry

```
<JDBCDataSource

  JNDIName="weblogic.jdbc.pool.commercePool"

  Name="commercePool"
  PoolName="commercePool"
  Targets="portalServer"
/>
```

- CommercePool Datasource: Commerce functionality requires CommercePool datasource entries like those shown in Listing 3-2.

Listing 3-2 CommercePool DataSource entries

```
<JDBCTxDataSource
  EnableTwoPhaseCommit="false"
  JNDIName="weblogic.jdbc.jts.commercePool"
  Name="commercePool"
  PoolName="commercePool"
  Targets="portalServer"
/>
<JDBCDataSource
  JNDIName="weblogic.jdbc.pool.commercePool"
  Name="commercePool"
  PoolName="commercePool"
  Targets="portalServer"
/>
```

- DataSync Pool: add a DataSync Pool entry, such as the one shown in Listing 3-3.

Listing 3-3 DataSync Pool entry

```
<JDBCTxDataSource
  EnableTwoPhaseCommit="false"
  JNDIName="weblogic.jdbc.jts.dataSyncPool"
  Name="dataSyncPool"
  PoolName="dataSyncPool"
  Targets="portalServer"
/>
```

- DataSync Data: add a DataSync Data entry, such as the one shown in Listing 3-4.

Listing 3-4 DataSync DataSource entry

```
<JDBCTxDataSource
  EnableTwoPhaseCommit="false"
  JNDIName="weblogic.jdbc.jts.dataSyncPool"
  Name="dataSyncPool"
  PoolName="dataSyncPool"
  Targets="portalServer"
/>
<WebAppComponent
  Name="datasync"
  ServletReloadCheckSecs="300"
  Targets="portalServer"
  URI="datasync"
/>
```

Select a Realm (Optional)

- To use the existing database realm, no special configuration is required.
- To use a new RDBMS realm, one must be configured, and then referenced in `config.xml`.

Key Bootstrap (if using commerce):

Using commerce credit card functionality with the portal application requires a reference to the KeyBootstrap class, as shown in (shown in Listing 3-5):

Listing 3-5 Reference to the KeyBootstrap class

```
<StartupClass
  ClassName="com.beasys.commerce.ebusiness.security.KeyBootstrap"
  FailureIsFatal="false"
  Name="KeyBootstrap"
  Targets="portalServer"
/>
```

P13N Console

3. Deploy the Personalization Console by adding an entry to the Config.xml file, as shown in Listing 3-6:

Listing 3-6 Personalization Console Deployment Entry

```
<Application
   Deployed="true"
   TwoPhase="true"
   StagedTargets="portalServer"
   Name="p13nConsoleApp"
   Path="<BEA_HOME>/weblogic700/portal/lib"
>
<WebAppComponent
   Name="p13nConsole"
   ServletReloadCheckSecs="300"
   Targets="portalServer"
   URI="p13nConsole.war"
/>
</Application>
```

WLPDocs Services

- WLPSDocs services are required to link WebLogic Portal Administration Tools to the online help.

Listing 3-7 WLPSDocs Services entries

```
<Application
   TwoPhase="true"
   StagedTargets="portalServer"
   Deployed="true"
   Name="wlpDocsApp"
   Notes=""
   Path="<BEA_HOME>/weblogic700/portal/lib"
>
   <WebAppComponent
     IndexDirectoryEnabled="false"
```

```
      Name="wlpDocs"

      Targets="portalServer"

      URI="wlpDocs.war"

      ServletReloadCheckSecs="300"

    />

</Application>
```

Tax and Payment Services (optional)

To add sample payment and tax services to the existing domain requires the entries shown in Listing 3-8. For information on adding these services to your portal application, consult "Setting Up Commerce Services" in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/commerce.htm.

Listing 3-8 Tax and Payment Services entry

```
<Application

  Deployed="true"

  TwoPhase="true"

  StagedTargets="portalServer"

  Name="paymentWSApp"

  Path="<BEA_HOME>/user_projects
/myPARTSdomain/beaApps/paymentWSApp"

>

 <EJBComponent

  Name="payment"

  URI="payment.jar"

  Targets="portalServer"

 />

 <WebAppComponent
```

```
  Name="payment-webservice"
  URI="pay-ws"
  Targets="portalServer"
/>
</Application>
<Application
  TwoPhase="true"
  StagedTargets="portalServer"
  Deployed="true"
  Name="taxWSApp"
  Path="<BEA_HOME>/user_projects/myPARTSdomain/beaApps/taxWSApp"
>
  <EJBComponent
    Name="tax"
    URI="tax.jar"
    Targets="portalServer"
  />
  <WebAppComponent
    Name="tax-webservice"
    URI="tax-ws"
    Targets="portalServer"
  />
</Application>
```

Verify Server References

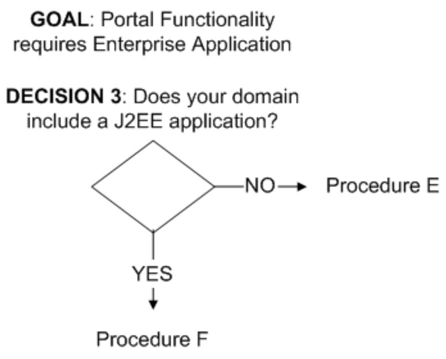
Examine the `config.xml` file after these edits. Search and replace any reference to portalServer with the name of the server targeted by the existing domain.

The `config.xml` file in your existing domain now has the entries required so that the WebLogic Portal server can connect to all the components that make up your J2EE enterprise Portal application. The next step is to identify the enterprise application in which these components can run on the server.

Locate or Install Enterprise Application

Within a BEA WebLogic Platform 7.0 domain, portal requires a complete J2EE application. At Decision 3, illustrated in Figure 3-3, an enterprise application is designated: Either it is added to the existing domain by following "Procedure E" on page 3-11, or portal functionality is merged with an enterprise application already present in the existing domain, "Procedure F" on page 3-12.

Figure 3-3 Decision 3


GOAL: Portal Functionality requires Enterprise Application
DECISION 3: Does your domain include a J2EE application?
NO
Procedure E
YES
Procedure F

Procedure E

1. Copy the entire contents of the <BEA_HOME>\user_projects\portalDomain\beaApps\portalApp directory into the existing domain.
2. Move in the objects that make up the existing Web application into this enterprise application.

Procedure F

1. Begin with the complete enterprise application from the existing domain.
2. Copy the following directories, including all contents and subfolders, into this application directory structure:
   ```
   tools/
   DataSync/
   toolSupport/
   ```
3. Copy the `portalApp-project.eaprj` file from within the following directory:

```
bea\user_projects\portalDomain\beaApps\portalApp-project
```

4. Using the BEA XML Editor, merge the `META-INF/weblogic-application.xml` files from the existing domain and the partsDomain. Merge these files by copying the entries in Listing 3-9 with the following changes: replace the <portalApp> string with the name of the J2EE application in the existing domain. For example, if your existing enterprise application were named existingApp, the entries would read existingAppTools, existingAppDataSync and existingAppTool.

Listing 3-9 Listings for weblogic-application.xml

```
<module>

  <web>

    <web-uri>tools</web-uri>

    <context-root>portalAppTools</context-root>

  </web>

</module>

<module>

  <web>

    <web-uri>datasync</web-uri>

    <context-root>portalAppDataSync</context-root>

  </web>

</module>
```

```
<module>
  <web>
    <web-uri>toolSupport</web-uri>
    <context-root>portalAppTool</context-root>
  </web>
</module>
```

5. Copy `application-config.xml` into the existing enterprise application.
6. Insert references to the portal enterprise application into `config.xml`, or deploy these modules using the WebLogic Server console.
7. Use the Portal Wizard to create a new portal and portal Web application, as described in "Step 2: Create the New Portal" on page 2-11.

   After following these procedures, you should have a single domain capable of running both your existing Web application and a complete instance of WebLogic Portal.

CHAPTER

4 Deploying Portals

This section outlines the steps required to deploy portal applications in the development environment.

This chapter includes information on the following topics:

- Hot Deploying With the Portal Wizard
- Deploying Without the Portal Wizard
- Deploying a Portal without Hot Deploy
- Best Practice Guidelines for Deploying Your Portal

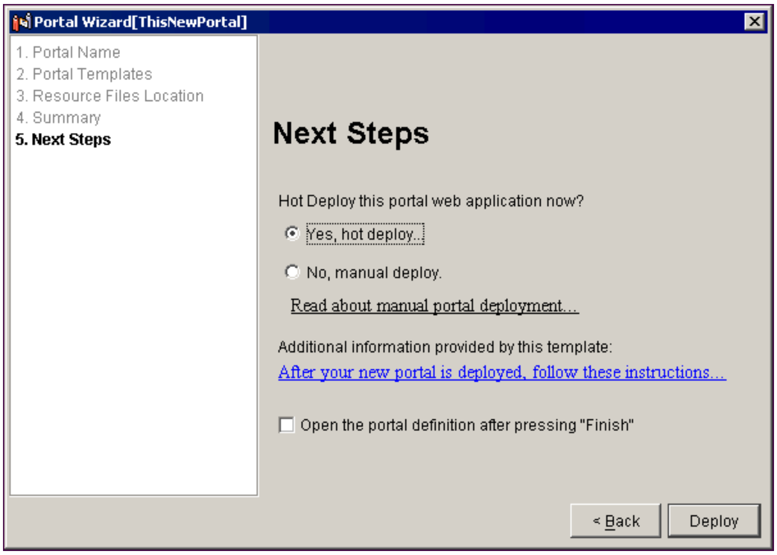
For more information, see the deployment guides at http://e-docs.bea.com//wls/docs70/deployment.html. For instructions on deploying to production environments, including clusters and managed servers, consult *Using the Domain Configuration Wizard*.at http://edocs.bea.com/platform/docs70/confgwiz/index.html.

Hot Deploying With the Portal Wizard

When you use the Portal Wizard to create a portal, you have an opportunity at the end of the wizard to hot deploy the portal Web application immediately. This is the simplest way to deploy.

Once you have deployed the portal, you can create additional portlets for it without deploying again. You only need to synchronize the E-Business Control Center to add the new portlets to the deployed portal.

Figure 4-1 Hot Deploy Option in the Next Steps Window of the Portal Wizard


Portal Wizard[ThisNewPortal]

1. Portal Name
2. Portal Templates
3. Resource Files Location
4. Summary
5. **Next Steps**

Next Steps

Hot Deploy this portal web application now?

[✓ Yes, hot deploy...]

No, manual deploy.

Read about manual portal deployment...

Additional information provided by this template:

After your new portal is deployed, follow these instructions...

☐ Open the portal definition after pressing "Finish"

< Back   Deploy

Deploying Without the Portal Wizard

Deploying without the Portal Wizard requires some manual steps.

Note: It can be very helpful to create a dummy portal and hot deploy it, as a model of what you need to do when you deploy manually. To create and deploy a dummy domain, portal, portal Web application, and associated J2EE resources, see Chapter 2, "Creating a New Portal in a New Domain."

To deploy manually, use the WebLogic Server deployment guides at http://e-docs.bea.com/wls/docs70/deployment.html, in particular "WebLogic Server Deployment" at http://e-docs.bea.com/wls/docs70/programming/deploying.html.

As you follow those deployment instructions and use the dummy portal as a reference, ensure that you accomplish the following:

- Place the J2EE resources for the portal on the server

- Use the E-Business Control Center to edit metadata (so the server knows where to look for your J2EE resources)
- Add the portal Web application to the WebLogic server using the console
- Start the WebLogic Portal server from your user domain
- Use the E-Business Control Center to synchronize the metadata to the WebLogic Server

Deploying a Portal without Hot Deploy

If you create a new portal and a new portal Web application using the E-Business Control Center, but for some reason choose not to hot deploy it to the server, the J2EE resources for that application will not be automatically deployed. This presents two scenarios for deploying the new portal Web application:

- If the E-Business Control Center project file is on a machine separate from the target server, the J2EE resources will have to be moved to the server before they can be deployed using the WebLogic console. If this is the case, perform steps 1 - 3 in this section.
- If you are running the E-Business Control Center on the target server, the J2EE files will already be in place, so the only thing you need to do is deploy the application components in the WebLogic console. If this is the case, skip the first step in this section.

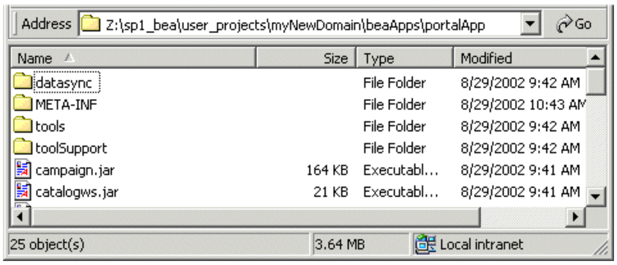
Manually Deploying a Portal Web Application

Deploying a portal that was not deployed using the hot sync function of the Portal Wizard requires the following steps:

Step 1: Move J2EE Resources

If the E-Business Control Center is running on a server remote from the target server, J2EE resources need to be moved. A remote server must have a complete enterprise portal application, as shown in Figure 4-2.

Figure 4-2 An Enterprise Portal domain without a Portal

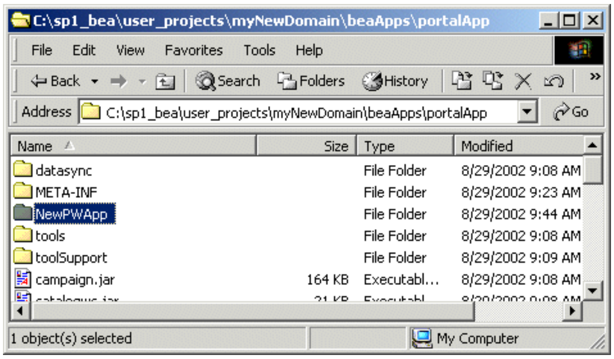


```
Address  Z:\sp1_bea\user_projects\myNewDomain\beaApps\portalApp      Go

Name              Size    Type          Modified
datasync                  File Folder   8/29/2002 9:42 AM
META-INF                  File Folder   8/29/2002 10:43 AM
tools                     File Folder   8/29/2002 9:42 AM
toolSupport               File Folder   8/29/2002 9:42 AM
campaign.jar      164 KB  Executabl...  8/29/2002 9:41 AM
catalogws.jar      21 KB  Executabl...  8/29/2002 9:41 AM

25 object(s)      3.64 MB     Local intranet
```

To move J2EE resource from the local server to the remote server, take the following steps:

1. Copy the NewPWApp directory from the local server, shown in Figure 4-3, into the Portal Enterprise application directory shown in Figure 4-2.

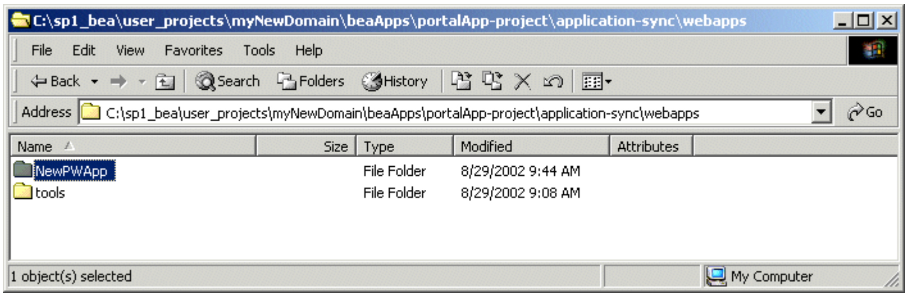
Figure 4-3 Copying the Portal Web Application



```
C:\sp1_bea\user_projects\myNewDomain\beaApps\portalApp
File  Edit  View  Favorites  Tools  Help
Back  Search  Folders  History
Address  C:\sp1_bea\user_projects\myNewDomain\beaApps\portalApp      Go

Name              Size    Type          Modified
datasync                  File Folder   8/29/2002 9:08 AM
META-INF                  File Folder   8/29/2002 9:23 AM
NewPWApp                  File Folder   8/29/2002 9:44 AM
tools                     File Folder   8/29/2002 9:08 AM
toolSupport               File Folder   8/29/2002 9:09 AM
campaign.jar      164 KB  Executabl...  8/29/2002 9:08 AM

1 object(s) selected          My Computer
```

2. Copy the `NewPWApp` directory from the `application-sync` directory on the local server, shown in Figure 4-4, into the `application-sync\webapps` directory on the remote server, shown in Figure 4-5.

Figure 4-4 Local Metadata directory


C:\sp1_bea\user_projects\myNewDomain\beaApps\portalApp-project\application-sync\webapps

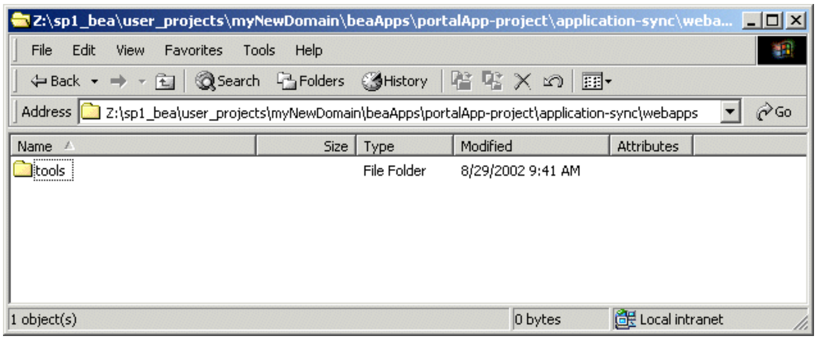
Address C:\sp1_bea\user_projects\myNewDomain\beaApps\portalApp-project\application-sync\webapps

Name	Size	Type	Modified	Attributes
NewPWApp		File Folder	8/29/2002 9:44 AM	
tools		File Folder	8/29/2002 9:08 AM	

1 object(s) selected — My Computer

File Edit View Favorites Tools Help
Back Search Folders History Go

Figure 4-5 Remote Metadata destination


Z:\sp1_bea\user_projects\myNewDomain\beaApps\portalApp-project\application-sync\weba...

Address Z:\sp1_bea\user_projects\myNewDomain\beaApps\portalApp-project\application-sync\webapps

Name	Size	Type	Modified	Attributes
tools		File Folder	8/29/2002 9:41 AM	

1 object(s) — 0 bytes — Local intranet

File Edit View Favorites Tools Help
Back Search Folders History Go

3. Proceed with the rest of the steps in this procedure.

Note: In order for a Portal Web application to include all the services available to the WebLogic platform, the following elements must be in place:

- The taglib JARs, shown in Listing 4-1 and Listing 4-2, must appear in the following directory:

  `beaApps\portalApp\<yourPortalWebApp>\WEB-INF\lib`

- The enterprise application JARs, shown in Listing 4-3, must appear in the following directory:

  `beaApps\portalApp\<yourPortalWebApp>\`

Listing 4-1 Taglib JARs Required to Support baseportal instance

```
weblogic700\common\templates\webapps\portal\baseportal\j2ee\WEB-INF\lib:
              util_taglib.jar
              webflow_servlet.jar
              ent_taglib.jar
              i18n_taglib.jar
              webflow_taglib.jar
              um_taglib.jar
              lic_taglib.jar
              es_taglib.jar
              ren_taglib.jar
              portlet_taglib.jar
              res_taglib.jar
              p13n_servlet.jar
              weblogic-tags.jar
              portal_taglib.jar
              portal_servlet.jar
              visitor_taglib.jar
              pz_taglib.jar
              ph_taglib.jar
              ps_taglib.jar
              cm_taglib.jar
```

Listing 4-2 Taglib JARs Required to Support All Portal Services

```
/weblogic700/portal/lib/commerce/web/cat_taglib.jar
/weblogic700/portal/lib/commerce/web/eb_taglib.jar
```

```
/weblogic700/portal/lib/commerce/web/productTracking_taglib.jar
/weblogic700/portal/lib/p13n/web/ad_taglib.jar
/weblogic700/portal/lib/p13n/web/cm_taglib.jar
/weblogic700/portal/lib/p13n/web/ph_taglib.jar
/weblogic700/portal/lib/p13n/web/ps_taglib.jar
/weblogic700/portal/lib/p13n/web/pz_taglib.jar
/weblogic700/portal/lib/p13n/web/tracking_taglib.jar
```

Listing 4-3 Enterprise JARs Required by Portal

```
campaign.jar
catalogws.jar
commerce_campaign_bridge_util.jar
commerce_util.jar
customer.jar
document.jar
ebusiness.jar
ejbadvisor.jar
events.jar
ldapprofile.jar
mail.jar
p13n_util.jar
payment.jar
pipeline.jar
placeholder.jar
portal.jar
portal_util.jar
```

```
property.jar

rules.jar

tax.jar

usermgmt.jar
```

Step 2: Synchronize Metadata

Open the project for your application in the E-Business Control Center and perform the synchronization step. For detailed instructions on this step, consult the chapter called “Synchronizing the New Portal Data to your Application.

Note: If you are deploying the new portal Web application to a remove server, remember to set up a new session in the E-Business Control Center, and designate the application name as `portalApp`, the enterprise application, instead of `NewPWApp`, the Web application name.

Step 3: Deploy in the WebLogic Server Console

The process of deploying a new Portal Web application consist of three steps:

- Create a new Web Module
- Configure the Web New Application
- Verify Deployment

Create a new Web Module

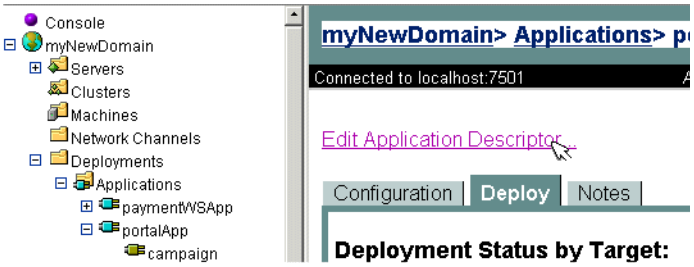
1. Navigate to the WebLogic Server Console:

   `http://<yourserver>:<port>/console/`

2. In the left pane, click on **`myNewDomain > Deployments > Applications`** > **`portalApp`**

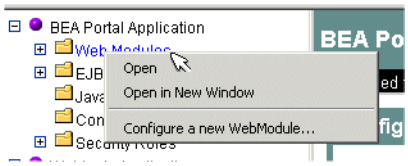
3. In the right pane, click on **Edit Application Descriptor**, as shown in Figure 4-6

Figure 4-6 Edit Application Descriptor



4. When the new browser window appears, right-click on Web Modules in the left pane, as shown in Figure 4-7.

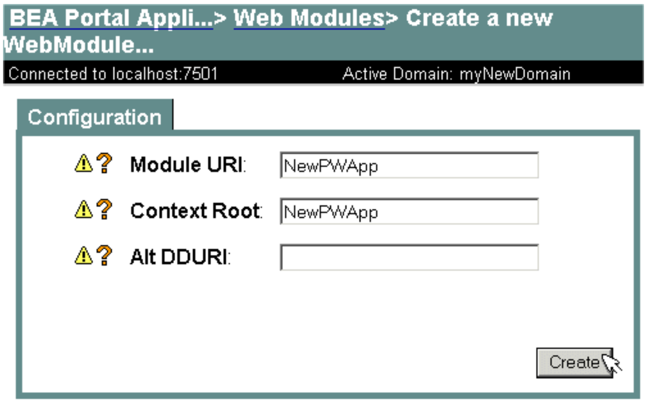
Figure 4-7 Right-click on Web Modules



5. When the Create a New Web Module screen appears, enter the URI (the native file system path is relative to the Enterprise directory, such as `beaApps\portalApp`), and the Context Root (the URL for the new portal Web application).

6. Click **Create**, then **Apply**, at the right corner of this screen, as shown in Figure 4-8.

Figure 4-8 Enter Module URI and Context Root


BEA Portal Appli...> Web Modules> Create a new WebModule...

Connected to localhost:7501 Active Domain: myNewDomain

Configuration

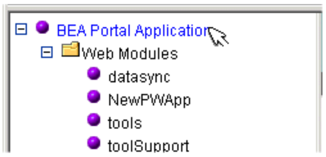
Module URI: NewPWApp

Context Root: NewPWApp

Alt DDURI:

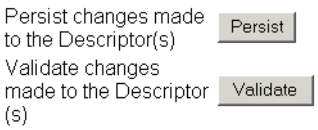
7. At the top of the left pane, click on BEA Portal Application, as shown in Figure 4-9.

Figure 4-9 Select the BEA Portal Application



8. Click **Persist** in the right pane of the same window, as shown in Figure 4-10.

Figure 4-10 Click Persist


Persist changes made to the Descriptor(s)
Persist
Validate changes made to the Descriptor (s)
Validate

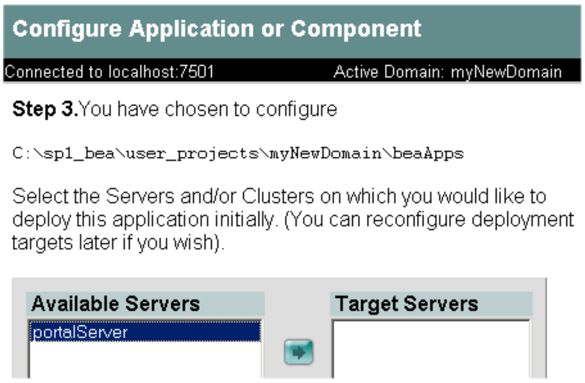
9. The message "Persistence was successful!" should appear in the right pane. Close the browser window.

Configure the Web New Application

1. Refresh the main console window, then navigate to `yourdomain > Deployments > Web Applications` and click on **Configure a new Web Application** in the right panel.
2. When the first **Locate Application or Component to configure** page appears, click on **SELECT** to the left of the **beaApps** application.
3. When the **Configure Application or Component** page appears, select the server from the list on the right, as shown in Figure 4-11.

Note: Even if you are configuring an application on a remote server, **Step 3** in Figure 4-11 will list the server using a drive letter local to the server itself, and not to the machine your web browser is running on. If this example showed a browser connected to a remote host, the fully resolved URL of that host, (instead of `localhost:7501`,) as shown in Figure 4-11, would appear in the black border next to the words "**Connected to**".

Figure 4-11 Select Server



Configure Application or Component

Connected to localhost:7501 Active Domain: myNewDomain

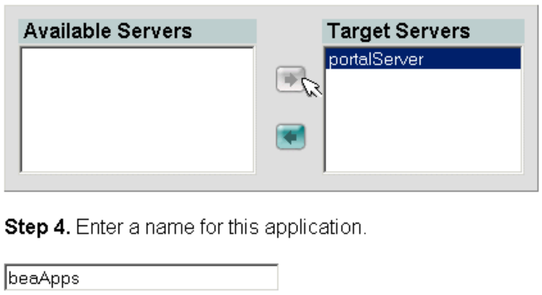
Step 3. You have chosen to configure

`C:\sp1_bea\user_projects\myNewDomain\beaApps`

Select the Servers and/or Clusters on which you would like to deploy this application initially. (You can reconfigure deployment targets later if you wish).

4. Move your server to the list of Target Servers on the right by clicking the top arrow between the lists, as shown in Step 3 of Figure 4-12.

Figure 4-12 Moving Server to Target Servers List

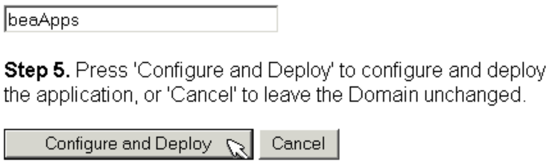


Step 4. Enter a name for this application.

beaApps

5. Confirm or edit the application name, shown in Step 4 of Figure 4-12.
6. Click on Configure and Deploy, as shown in Step 5 of Figure 4-13.

Figure 4-13 Configure and Deploy Application

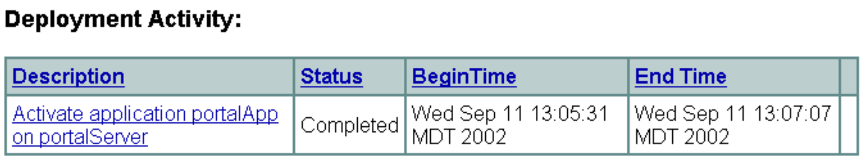


beaApps

Step 5. Press 'Configure and Deploy' to configure and deploy the application, or 'Cancel' to leave the Domain unchanged.

7. When the **`myNewDomain> Applications> portalApp`** page appears.
8. After the deployment process runs, a status message appears at the bottom of the window, indicating the deployment was successful, as shown in Figure 4-14.

Figure 4-14 Successful Deployment Message



Deployment Activity:

Description	Status	BeginTime	End Time
Activate application portalApp on portalServer	Completed	Wed Sep 11 13:05:31 MDT 2002	Wed Sep 11 13:07:07 MDT 2002

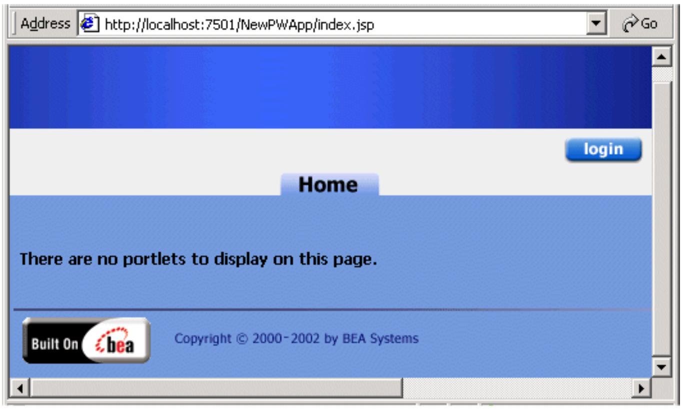
9. When the **Deployment Status by Target** page appears, verify that the status of **True** appears in the Deployed column for the new application. If it is **False**, click on **Deploy** and wait for the process to complete successfully.

Verify Deployment

Finally, verify the portal has been successfully deployed by navigating to `http:\\<hostname>:<port>\NewPWApp`, as shown in Figure 4-15.

Note: In this example, no portlets have been made visible using the WebLogic Portal Administration Tools, therefore the portal displays none in Figure 4-15.

Figure 4-15 Viewing the Deployed Portal Web Application



There are no portlets to display on this page.

Copyright © 2000-2002 by BEA Systems

Best Practice Guidelines for Deploying Your Portal

We recommend that you use a multiple environment model for developing, testing, and publishing your portal application. BEA Weblogic Platform provides a number of tools and scripts for simple deployment. These tools and best practices are discussed in detail in “WebLogic Server Deployment” at http://edocs.bea.com/wls/docs70/programming/deploying.html.

We recommend deploying your portal in the following three-stage model:

Stage 1: Deploy to a Server on Your Own Machine

Use this deployment stage for development and unit testing. We strongly recommended that you keep all development limited to this stage and only deploy to the next stage when you are finished unit testing.

Note: If you do decide to do development and unit testing in a shared environment, each developer should work within their domain to avoid overwriting database information for other developers.

Stage 2: Deploy From a Local Computer to a Staging Server

Once development and unit testing is completed, deploy to a staging server for exception testing. Any faulty code found should be resolved on a local machine and then redeployed back to a staging server.

Cluster testing, if applicable, should be performed on a staging server.

Stage 3: Deploy From the Testing Environment to a Live Production Server

Once all development and exception testing is completed, you may deploy the application to a live production server. You may also want to retest your cluster deployment at this point.

Any additional development for the application should be developed and unit tested as described in Stage 1, tested as described in Stage 2, then redeployed to the production server.

Part II Extending Portals

Once you have a functioning portal, you can increase its value by adding features and functionality. The information in this section shows you how to extend portals, turning them into interactive, content-rich, personalizable single points of access to enterprise data and applications.

This section includes information on the following subjects:

CHAPTER

5 Building Custom Templates

WebLogic Portal includes a number of wizards and templates to assist in creating domains, applications, portals and portlets quickly and efficiently. This section outlines instructions for creating your own custom templates. Domain and portal templates are consumed by the domain wizard or the portal wizard, and group portal templates are invoked using the WebLogic Portal Administration Tools.

This section includes information on the following subjects:

- Introducing Templates
- Creating a Domain Template
- Creating a Portal Template

Introducing Templates

Three templates are built into the BEA WebLogic Platform: Domain templates, portal templates and group portal templates. Domain and portal templates are built as wizard components, whereas group portals are used by the WebLogic Portal Administration Tools.

Three Types of Templates

Before learning how to develop templates for WebLogic Platform, consider the role each template type plays in creating a portal:

The Domain Wizard Template

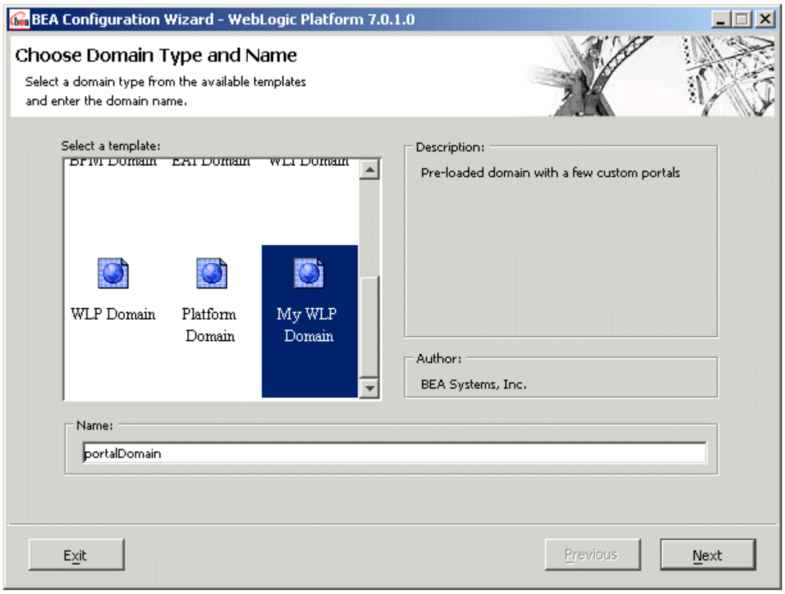
Out-of-the-box domain templates are stored in the form of JAR files in following directory:

```
<BEA_HOME>weblogic700\common\templates\domains
```

The Domain Configuration Wizard is launched from the Start -> Programs -> WebLogic700 menu, and does not require a server to be running. Gathering user input from a series of screens, the Wizard performs some directory and file copying, and then some string substitutions within the configuration files and startup scripts within the domain being created. Thus the user's choices are applied to the template and instantiated in the new domain directory.

Figure 5-1 shows the Domain Configuration Wizard explosing several domain templates.

Figure 5-1 Selecting a Custom Domain


BEA Configuration Wizard - WebLogic Platform 7.0.1.0

Choose Domain Type and Name

Select a domain type from the available templates and enter the domain name.

Select a template:

Description: Pre-loaded domain with a few custom portals

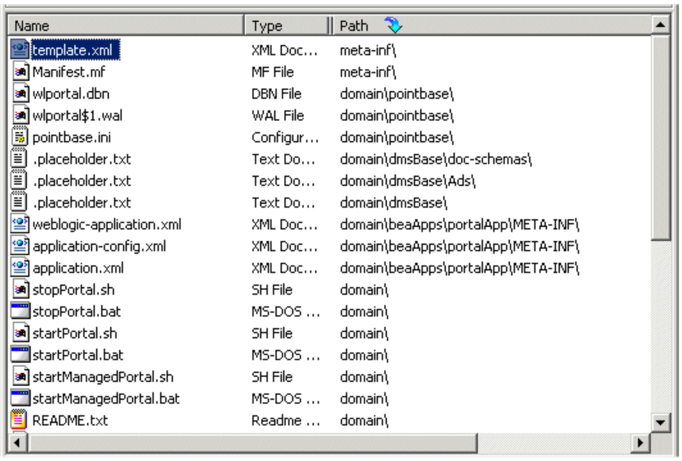
Author: BEA Systems, Inc.

Name: portalDomain

The Domain Configuration Wizard recognizes templates based on the contents of the template.xml file. Information on editing that file appears later in this procedure.

Figure 5-2 shows some of the contents of the stock WebLogic Portal domain JAR exploded to show the files within.

Figure 5-2 WebLogic Portal Domain Template Files



Name	Type	Path
template.xml	XML Doc...	meta-inf\
Manifest.mf	MF File	meta-inf\
wlportal.dbn	DBN File	domain\pointbase\
wlportal$1.wal	WAL File	domain\pointbase\
pointbase.ini	Configur...	domain\pointbase\
.placeholder.txt	Text Do...	domain\dmsBase\doc-schemas\
.placeholder.txt	Text Do...	domain\dmsBase\Ads\
.placeholder.txt	Text Do...	domain\dmsBase\
weblogic-application.xml	XML Doc...	domain\beaApps\portalApp\META-INF\
application-config.xml	XML Doc...	domain\beaApps\portalApp\META-INF\
application.xml	XML Doc...	domain\beaApps\portalApp\META-INF\
stopPortal.sh	SH File	domain\
stopPortal.bat	MS-DOS ...	domain\
startPortal.sh	SH File	domain\
startPortal.bat	MS-DOS ...	domain\
startManagedPortal.sh	SH File	domain\
startManagedPortal.bat	MS-DOS ...	domain\
README.txt	Readme ...	domain\

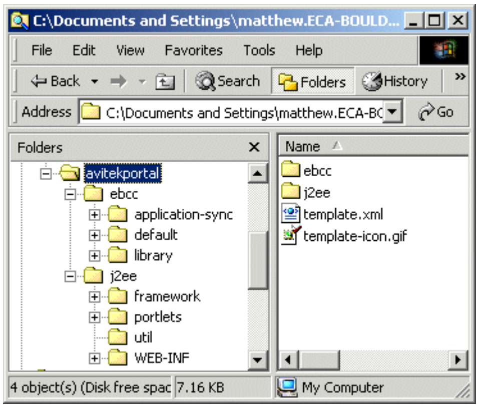
The Domain Configuration Wizard uses the `template.xml`, `config.xml` and `application.xml` files to copy resources, perform string substitutions for configuration files and scripts within the new domain.

The Portal Wizard Template

One out-of-the-box portal template is included with the installation of WebLogic Portal: Figure 5-3 shows the contents of a custom Portal Template, which would be stored in the following directory:

```
<BEA_HOME>\weblogic700\common\templates\webapps\portal
```

Figure 5-3 Avitek Portal Template file structure


C:\Documents and Settings\matthew.ECA-BOULD...
File Edit View Favorites Tools Help
Back Search Folders History
Address C:\Documents and Settings\matthew.ECA-BC Go
Folders
Name

```
avitekportal
  ebcc
    application-sync
    default
    library
  j2ee
    framework
    portlets
    util
    WEB-INF
```

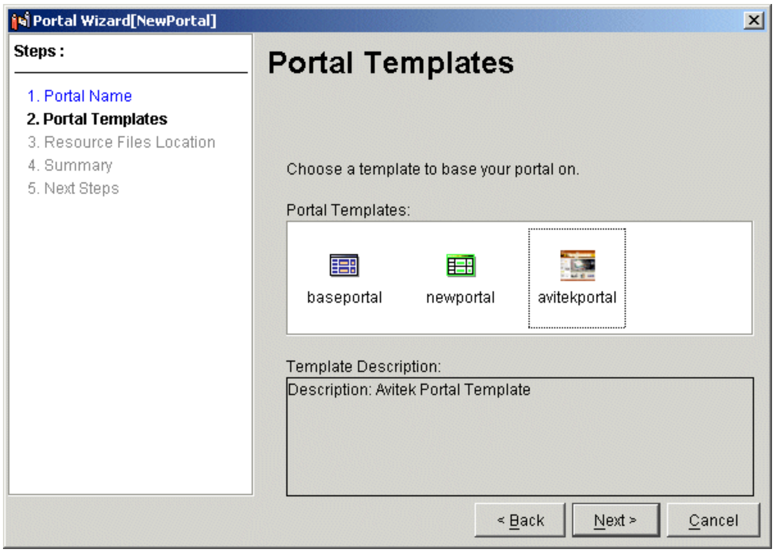
```
ebcc
j2ee
template.xml
template-icon.gif
```

4 object(s) (Disk free spac 7.16 KB My Computer

Like the Domain Configuration Wizard, the Portal Wizard consumes a file called `template.xml`, as well as a `template-icon.gif`. The `template.xml` file includes directives and variables for applying user input from the Portal Wizard to the J2EE and metadata resources contained in this folder.

Through the lens of the Portal Wizard, portal templates appear by name and icon, as shown in Figure 5-4.

Figure 5-4 Selecting Portal Templates



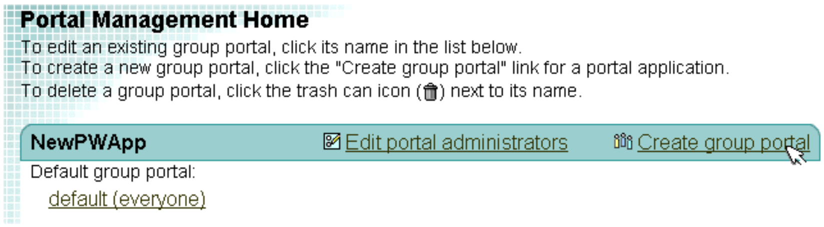
After the Portal Template is used to instantiate a portal, the WebLogic Portal Administration Tools are used to set attributes on pages, portlets, and other aspects of the portal.

The Group Portal Template

Unlike the Domain and Portal templates, the Group Portal template does not exist as a packaged entity. Rather, the WebLogic Portal Administration Tools provide a mechanism whereby every group portal is created using a template. Out-of-the-box, the only choice is the default group portal, to which no customization has been applied. Figure 5-5 shows the Create Group Portal link from the Portal Management Home page.

Note: Because Group Portal Templates are essentially reused Group Portals, this document does not address their creation. For detailed instructions on creating Group Portals, consult the tutorial called Creating a Group Portal.

Figure 5-5 Create Group Portal



Portal Management Home

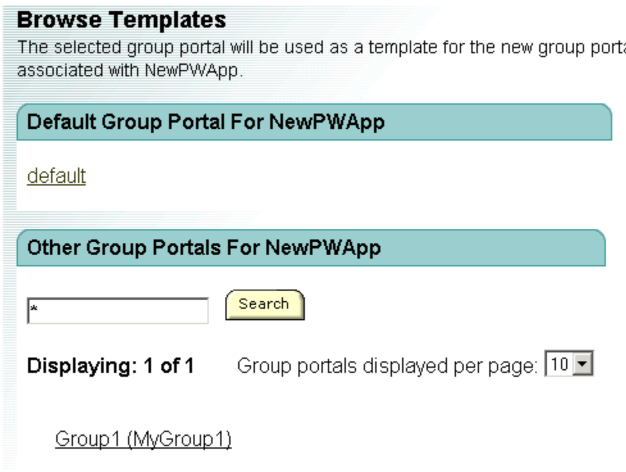
To edit an existing group portal, click its name in the list below.
To create a new group portal, click the "Create group portal" link for a portal application.
To delete a group portal, click the trash can icon (🗑) next to its name.

NewPWApp Edit portal administrators Create group portal

Default group portal:
default (everyone)

After the new group portal has been named and associated with a user group, a group portal template must be selected, as shown in Figure 5-6.

Figure 5-6 Select Group Portal Template



Browse Templates

The selected group portal will be used as a template for the new group portal associated with NewPWApp.

Default Group Portal For NewPWApp

default

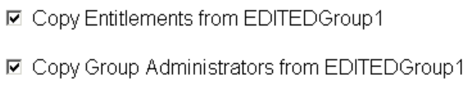
Other Group Portals For NewPWApp

Displaying: 1 of 1 Group portals displayed per page: 10

Group1 (MyGroup1)

This way, any group portal can serve as a template for new group portals. Importantly, entitlements and delegated administration can be copied from the group portal template, as shown in Figure 5-7.

Figure 5-7 Copying Entitlements and Delegated Administration from Group Portal Template



☑ Copy Entitlements from EDITEDGroup1

☑ Copy Group Administrators from EDITEDGroup1

Using Templates

Roughly three varieties of customization can be introduced into the Portal life cycle: look and feel customization can be imposed by creating stock layouts and skins, functionality can be customized by adding application elements, and subsets of these custom objects can be preserved and propagated using the group portal template.

Each type of template confers a specific advantage:

- Create a custom domain template to provide a common starting point for all the developers in an organization: this can include enterprise components such as EJBs, corporate identity components such as skins and logos, or even corporate portlets to be used across an enterprise.
- Create a custom portal template when you need to clone an existing set of portal pages, group portals, entitlements, portlets, skins and layouts.
- Create a group portal template to preserve administrative associations between user groups and portal resources such as pages, portlets, entitlements, and layouts.

Creating a Domain Template

The best way to create a custom portal domain template is to begin with a WebLogic Portal domain created using the template. Perform whatever customization is needed, everything from adding application functionality to setting entitlements on a portlet, and use the domain template as a mechanism for transmitting customization down the line to subsequent developers in your organization.

This section covers the basic steps in creating a template for the Domain Configuration Wizard:

- Step 1: Instantiate a Portal Domain
- Step 2: Customize the Portal Domain
- Step 3: Apply General Configuration
- Step 4: Package the New Domain as a Template

Step 1: Instantiate a Portal Domain

Begin by using the Domain Configuration Wizard to instantiate a WebLogic Portal domain. For detailed instructions on this step, consult the tutorial "Creating a New Portal in a New Domain.

Step 2: Customize the Portal Domain

Customize the domain using the Portal Wizard, Portlet Wizard, and by adding application functionality, look and feel components, EJBs, or other J2EE components. This section provides some detailed instructions on adding functionality to a WebLogic Portal domain. This section contains the following instruction sets, each of which can be applied separately:

- Supporting Two-Phase Deployment
- Adding All Portal Services to Your Domain
- Adding an EJB to your WebLogic Portal Domain
- Adding a Custom Layout to a Domain Template
- Adding a Custom Skin to a Domain Template

Supporting Two-Phase Deployment

If you choose to create your own portal Web application without using the template, make sure the weblogic-application.xml file shown in Listing 5-1 is saved in the following directory:

```
domain\beaApps\portalApp\META-INF\
```

This file is required in order for two-phase application deployment to function properly. It is included automatically when you create an application using the WebLogic Portal template. If you are creating an application without using the wizard, then you must manually create this file and add this entry.

Listing 5-1 weblogic-application.xml

```
<!DOCTYPE weblogic-application PUBLIC "-//BEA Systems, Inc.//DTD WebLogic
Application 7.0.0//EN"
"http://www.bea.com/servers/wls700/dtd/weblogic-application_1_0.dtd">

<weblogic-application>

  <application-param>

   <description>Required for deployment of portal

   applications</description>

   <param-name>weblogic.internal.listeners</param-name>

<param-value>com.bea.p13n.management.internal.lifecycle.J2EELifecycleListener</
param-value>

  </application-param>

</weblogic-application>
```

Adding All Portal Services to Your Domain

The template that ships with WebLogic Portal includes a subset of all the application functionality available to the portal framework, excluding content management, personalization, placeholder and ad services.

To add all this functionality to your domain, take the following steps:

1. Use the Portal Wizard to create a new portal and portal Web application in your new domain.
2. Copy the files in Listing 5-2 to the <webapp>\WEB-INF\lib directory of your new domain.
3. Insert the entry in Listing 5-3 to the `web.xml` file at `<webapp>/WEB-INF`.
4. Add the entry in Listing 5-4 to the `<webapp>/WEB-INF/weblogic.xml` file.

The portal Web application in this domain will now support all services included in the WebLogic Platform. To use this domain as a template for the Domain Configuration Wizard, follow the remaining steps in this procedure, starting with Step 3: Apply General Configuration.

Listing 5-2 JARs to add all Portal Services

```
BEA_HOME/weblogic700/portal/lib/commerce/web/cat_taglib.jar

BEA_HOME/weblogic700/portal/lib/commerce/web/eb_taglib.jar

BEA_HOME/weblogic700/portal/lib/commerce/web/productTracking_taglib.jar

BEA_HOME/weblogic700/portal/lib/p13n/web/ad_taglib.jar

BEA_HOME/weblogic700/portal/lib/p13n/web/cm_taglib.jar

BEA_HOME/weblogic700/portal/lib/p13n/web/ph_taglib.jar

BEA_HOME/weblogic700/portal/lib/p13n/web/ps_taglib.jar

BEA_HOME/weblogic700/portal/lib/p13n/web/pz_taglib.jar

BEA_HOME/weblogic700/portal/lib/p13n/web/tracking_taglib.jar
```

Listing 5-3 Entries to web.xml to add all Portal Services

```
<!-- Add this just before the </web-app> entry at the tail end of the file. -->

<!-- Filter to fire click through events -->

<filter>

    <filter-name>ClickThroughEventFilter</filter-name>


<filter-class>com.bea.p13n.tracking.clickthrough.ClickThroughEventFilter</filte
r-class>

</filter>

<filter-mapping>

    <filter-name>ClickThroughEventFilter</filter-name>

    <url-pattern>/application/*</url-pattern>

</filter-mapping>

<!-- The ShowDoc Servlet -->

<servlet>
```

```
    <servlet-name>ShowDocServlet</servlet-name>

    <servlet-class>com.bea.p13n.content.servlets.ShowDocServlet</servlet-class>

    <!-- Make showdoc always use the local ejb-ref DocumentManager -->

    <init-param>

        <param-name>contentHome</param-name>

        <param-value>java:comp/env/ejb/DocumentManager</param-value>

    </init-param>

</servlet>

<!-- The AdClickThru Servlet -->

<servlet>

    <servlet-name>adClickThru</servlet-name>

    <servlet-class>com.bea.p13n.ad.servlets.AdClickThruServlet</servlet-class>

</servlet>

<!-- The ClickThrough Servlet -->

<servlet>

    <servlet-name>clickThroughServlet</servlet-name>


<servlet-class>com.bea.p13n.tracking.clickthrough.ClickThroughServlet</servlet-
class>

</servlet>

<servlet-mapping>

    <servlet-name>ShowDocServlet</servlet-name>

    <url-pattern>/ShowDoc/*</url-pattern>

</servlet-mapping>

<servlet-mapping>

    <servlet-name>adClickThru</servlet-name>

    <url-pattern>/adClickThru/*</url-pattern>

</servlet-mapping>
```

```
<servlet-mapping>
    <servlet-name>adClickThru</servlet-name>
    <url-pattern>/AdClickThru/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>clickThroughServlet</servlet-name>
    <url-pattern>/clickThroughServlet/*</url-pattern>
</servlet-mapping>
<taglib>
    <taglib-uri>cat.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/cat_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>eb.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/eb_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>productTracking.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/productTracking_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>ad.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/ad_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>cm.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/cm_taglib.jar</taglib-location>
```

```
</taglib>
<taglib>
    <taglib-uri>ph.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/ph_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>ps.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/ps_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>pz.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/pz_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>tracking.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/tracking_taglib.jar</taglib-location>
</taglib>
<!-- This is used by the various <cm:> tags -->
<ejb-ref>
    <description>
The ContentManager EJB for this webapp
    </description>
    <ejb-ref-name>ejb/ContentManager</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.content.document.DocumentManagerHome</home>
    <remote>com.bea.p13n.content.document.DocumentManager</remote>
</ejb-ref>
```

```
<!-- This is used by the ShowDocServlet -->
<ejb-ref>
    <description>
The DocumentManager for this webapp
    </description>
    <ejb-ref-name>ejb/DocumentManager</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.content.document.DocumentManagerHome</home>
    <remote>com.bea.p13n.content.document.DocumentManager</remote>
</ejb-ref>
<!-- This is used by the Placeholder tag -->
<ejb-ref>
    <description>
The PlaceholderService Session EJB for the placeholder tag.
    </description>
    <ejb-ref-name>ejb/PlaceholderService</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.placeholder.PlaceholderServiceHome</home>
    <remote>com.bea.p13n.placeholder.PlaceholderService</remote>
</ejb-ref>
<!-- This is used by the AdClickThruServlet and the adTarget tag-->
<ejb-ref>
    <description>
The AdService for this webapp
    </description>
    <ejb-ref-name>ejb/AdService</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.ad.AdServiceHome</home>
    <remote>com.bea.p13n.ad.AdService</remote>
</ejb-ref>
<!-- This is used by the AdClickThruServlet -->
<ejb-ref>
    <description>
The AdBucketService for this webapp
    </description>
    <ejb-ref-name>ejb/AdBucketService</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.ad.AdBucketServiceHome</home>
    <remote>com.bea.p13n.ad.AdBucketService</remote>
</ejb-ref>
<!-- This is used by the various <pz:> tags -->
<ejb-ref>
    <description>
The EjbAdvisor for this webapp
    </description>
    <ejb-ref-name>ejb/EjbAdvisor</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.advisor.EjbAdvisorHome</home>
    <remote>com.bea.p13n.advisor.EjbAdvisor</remote>
</ejb-ref>
```

Listing 5-4 Entries to weblogic.xml to add all Portal Services

```
<-- Add this after the <reference-descriptor> entry near the top of the file. -->
```

```
<ejb-reference-description>
    <ejb-ref-name>ejb/ContentManager</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.DocumentManager</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/DocumentManager</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.DocumentManager</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/PlaceholderService</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.PlaceholderService</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/AdService</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.AdService</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/AdBucketService</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.AdBucketService</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/EjbAdvisor</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.EjbAdvisor</jndi-name>
</ejb-reference-description>
```

Adding an EJB to your WebLogic Portal Domain

For instance, to add an Enterprise Java Bean to your domain, take the following steps:

1. Create an EJB. For instructions on creating EJBs for the WebLogic Platform, consult Programming WebLogic Enterprise JavaBeans.
2. Place the jar file in the application directory:

   ```
   <BEA_HOME>/yourdomain/BEAapps/portalApp/META-INF
   ```

3. Insert the following reference to this jar in the `META-INF/application.xml` file:

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

   where `myEJB` is the name of your new EJB.
4. In the application node of the `domain/config.xml`, add an entry such as this:

   ```
   <EJBComponent
        Name="myEJB"
        Targets="@TARGETS"
        URI="myEJB.jar"
   />
   ```

Your EJB will be deployed with all the other application functionality at startup.

Step 3: Apply General Configuration

Aside from adding application functionality at the enterprise or Web application level, you may want to propagate more general customization by inserting portals, pages, portlets, and even skins and layouts into your customized WebLogic Portal domain template.

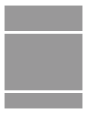
For detailed instructions on creating portals and portlets, consult the WebLogic Development Guide. For detailed instructions on creating group portals, consult the section of the WebLogic Portal Administration Guide called Creating a Portal and Group Portal.

For detailed instructions on creating custom layouts and skins, consult the section of the WebLogic Portal Development Guide called Creating a Look and Feel. This section explains how to add look and feel components to a domain such that it can be packaged within a custom WebLogic Portal domain.

Adding a Custom Layout to a Domain Template

This example shows a custom layout called "stack", a one-column vertical scheme. This layout consists of a JSP file, shown in Listing 5-5, and a thumbnail image, shown in Figure 5-8.

Figure 5-8 `thumbnail.gif` for stack layout



Listing 5-5 text of stack layout

```
<%@ taglib uri='ren.tld' prefix='layout' %>

<layout:placePortletsinPlaceholder
placeholders="top,middle,bottom" />

<center>

    <table BORDER COLS="1" WIDTH="250" >

        <tr>

            <td>

                <layout:render section='top'/>

            </td>

        </tr>


        <tr>

            <td>

                <layout:render section='middle'/>
            </td>

        </tr>


        <tr>

            <td>

                <layout:render section='bottom'/>

            </td>

        </tr>

    </table>

</center>
```

Having created a custom layout, make it available to the new domain template using the following steps:

1. To insert this layout into the domain template, save the JSP as **`template.jsp`** and the image file as **`<yourlayoutName>.gif`** in the following directory:

   `<BEA_HOME>\weblogic700\common\templates\`**`domains`**`\shared\bea\portal\projects\portalApp-project\library\portal\layouts\<yourlayoutName>\`

2. To enable this custom layout to be visible to the E-Business Control Center, save the JSP as **`template.jsp`** and the image file as **`thumbnail.gif`** in the following directory:

   `<BEA_HOME>\weblogic700\common\templates\`**`webapps`**`\portal\newportal\j2ee\framework\layouts\<yourlayoutName>\`

Warning: Filenames for layout thumbnails are not uniform. If they are not correct, the preview image may be missing in the E-Business Control Center or in the WebLogic Portal Administration Tools.

Adding a Custom Skin to a Domain Template

Having created a custom skin, use the following steps to make the custom skin available to the new domain template, take the following steps:

1. Take a screenshot of the skin on your portal, reduce it to a 1 inch width, and save it as <`yourlayoutName`>.gif.
2. Place this thumbnail in the following directory:

   ```
   <BEA_HOME>\weblogic700\common\templates\domains\shared\bea\
   portal\projects\portalApp-project\library\portal\skins\<you
   rskinName>\
   ```

3. Place the J2EE resources for the skin in the following directory:

   ```
   <BEA_HOME>\weblogic700\common\templates\webapps\portal\newp
   ortal\j2ee\framework\skins\<yourskinName>\
   ```

After these elements are added to the domain, what remains is to package the new domain in such a way that the Domain Configuration Wizard can use it to instantiate your custom domain.

Step 4: Package the New Domain as a Template

After confirming that all the customization in your domain has been added successfully, you can replicate this new functionality in the form of a domain template. This template can be used to give corporate developers a good baseline to work from, reducing unnecessary duplication of efforts at several points in the development, deployment and maintenance processes.

Entries have been made in several configuration files to support these additions. Assuming the J2EE resources are in place and archived correctly in the template JAR file, it is crucial that the metadata about these resources be entered correctly so the wizard can include them in the resulting domain. For this reason, pay special attention to the points at which the `template.xml` file makes reference to the other configuration files and J2EE resources.

Note: Any new domain template you create should include support for all portal services. For this reason, it is strongly recommended that you follow the steps outline in Step 2: Customize the Portal Domain before making your domain into a template.

To create a template from your customized domain, take the following steps:

- Open the template.xml File
- Edit the config.xml file

- Edit the Application.xml file
- Check Shell Scripts for String Substitution
- Create the Archive

Open the template.xml File

Obtain a copy of the `template.xml` file by making a copy of the following JAR file:

`<BEA_HOME>\weblogic700\common\templates\domains\portal.jar`

Unzip this archive and open the `template.xml` file in a text browser. Use this file as a reference when editing the `config.xml` file. After making any necessary edits, archive this file (in JAR form) so that it extracts to the `\META-INF` directory.

Edit the config.xml file

Open the `config.xml` file from your customized domain, and perform the substitutions shown in Listing 5-6. Replacing the populated attributes with the variables used by the template.xml allows the Domain Configuration Wizard to automatically populate user input into the fields of your `config.xml` file.

Listing 5-6 Domain node substitutions for config.xml file

```
<Server

    Name="portalServer"    replace with "@SERVER_NAME"

   ListenPort="7501"     replace with "@BEA_WEBLOGIC_SERVER_PORT@"

    NativeIOEnabled="true"

    JavaCompiler="@JAVA_HOME/bin/javac"

    ServerVersion="7.0.1.0"

    StagingMode="nostage"

    TransactionLogFilePrefix="logs/"

>

<Log FileName="logs/weblogic.log"
```

```
    Name="portalServer"/   replace with "@SERVER_NAME"
>
<SSL Enabled="true"
   ListenPort="7502"   replace with "@BEA_WEBLOGIC_SERVER_SSLPORT@"
   Name="portalServer"        replace with "@SERVER_NAME"
   ServerCertificateChainFileName="ca.pem"
   ServerCertificateFileName="democert.pem"
   ServerKeyFileName="demokey.pem"/
>
<ServerStart Name="portalServer"/>  replace with "@SERVER_NAME"
 <WebServer
  DefaultWebApp="DefaultWebApp"
  LogFileName="access.log"
  LoggingEnabled="true"
  Name="portalServer"              replace with "@SERVER_NAME"
/>
```

Note: In the `template.xml`, the @TARGETS value is used to allow the same domain to be deployed in a cluster, an admin server or a stand-alone server without modification to the `config.xml` file.

```
 <change-pair name="TARGETS">
         <before string="@TARGETS" />
         <after string="$TARGET_NAMES$" />
      </change-pair>
```

The Domain Wizard runs against your template, replacing variable entries as shown in Listing 5-7:

Listing 5-7 Variable replacements in the config.xml file

```
Original node in portal Template config.xml

<EJBComponent

    Name="campaign"

    Targets="@TARGETS"

    URI="campaign.jar"

/>

Variables populated in new config.xml file

<EJBComponent

    Name="campaign"

    Targets="portalServer"

    URI="campaign.jar"

/>
```

Edit the Application.xml file

As with the `template.xml` and `config.xml` files, extracting an example file from an existing template is a good way to start. Compare the archive file against your customized `application.xml` file. For each new Web application you want to automatically deploy at server start, an entry will need to be added in the Application node of `application.xml`. Listing 5-8 shows a Web module called `NewPWApp` inserted into the Application node.

Listing 5-8 Adding a module listing to application.xml

```
<application>

 <module>

   <web>

     <web-uri>NewPWApp</web-uri>
```

```
            <context-root>NewPWApp</context-root>

          </web>

        </module>

      ...

      </application>
```

Check Shell Scripts for String Substitution

If your application requires special startup classes or environment settings, make sure these are added to the shell scripts within your template. Listing 5-9 shows the contents of `startPortal.bat` included with the out-of-the-box portal domain template.

If your domain requires any customized startup commands, make sure any added literal references will not make your scripts difficult to maintain.

Listing 5-9 startPortal.bat from portal.jar

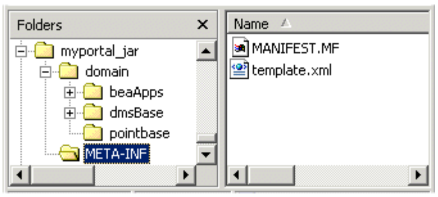
```
@ECHO OFF

SETLOCAL

REM ###########################################################################

REM (c) 2002 BEA SYSTEMS INC. All rights reserved

REM

REM BEA WebLogic Portal Server startup script.

REM This script can also install/uninstall a Portal Window Service. Use the

REM -installService or -uninstallService command-line arguments.

REM ###########################################################################

REM ###########################################################################

REM The WLP installation directory

REM ###########################################################################

SET WLP_HOME=@BEA_PORTAL_HOME_BACK_SLASH@

REM ###########################################################################
REM Set the WebLogic server name
REM ###########################################################################
SET SERVER_NAME=@MANAGED_SERVER_REGISTERED_NAME_IN_ADMIN
IF "%SERVER_NAME%"=="" SET SERVER_NAME=@SERVER_NAME
REM ###########################################################################
REM Set the WebLogic Admin Server URL, if this is a managed node.
REM Otherwise, leave this blank
REM ###########################################################################
set ADMIN_URL=@ADMIN_SERVER_URL
REM ###########################################################################
REM Set the database type
REM Valid values are: POINTBASE, ORACLE_THIN, MSSQL, SYBASE_JCONNECT, DB2_TYPE2
REM Set set-environment.bat for more details
REM ###########################################################################
SET DATABASE=@DATABASE@
REM Try to get it from the db_settings.properties file
IF not exist .\db_settings.properties goto _setenv
SET DB_SETTINGS=.\db_settings.properties
FOR /F "eol=# tokens=1,2 delims==" %%i in (%DB_SETTINGS%) do  (
    if %%i == database SET DATABASE=%%j
)
:_setenv
REM ###########################################################################
REM Set the environment
REM See set-environment.bat for more details on the available parameters
REM ###########################################################################
CALL "%WLP_HOME%\bin\win32\set-environment.bat"
```

```
REM ##############################################################################

REM Set any additional CLASSPATH information

REM ##############################################################################

SET
CLASSPATH=%CLASSPATH%;%P13N_DIR%\lib\commerce_system.jar;%P13N_DIR%\lib\campaig
n_system.jar

REM ##############################################################################

REM Start WebLogic with the above parameters.

REM See startWebLogic.cmd for more details on the available parameters.

REM ##############################################################################

set MEM_ARGS=-Xms128m -Xmx128m -XX:MaxPermSize=128m

set JAVA_OPTIONS=-Dcommerce.properties="%WLP_HOME%\weblogiccommerce.properties"

if "%1" == "-installService" goto _installService

if "%1" == "-uninstallService" goto _uninstallService

:_startWebLogic

call "%P13N_DIR%\bin\win32\startWebLogic.cmd"

goto _the_end

:_installService

call "%P13N_DIR%\bin\win32\installWebLogicService.cmd"

goto _the_end

:_uninstallService

call "%P13N_DIR%\bin\win32\uninstallWebLogicService.cmd"

goto _the_end

:_the_end

ENDLOCAL
```

Create the Archive

At this stage, all look and feel customization and functionality has been added to the portal domain, and the directory structure looks like that in Figure 5-9.

Figure 5-9 Expanded View of Custom Portal Domain

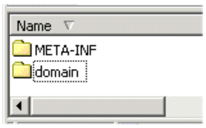


Take the following steps to create the archive:

1. Make sure the `config.xml` file and the shell scripts are in the domain directory, and that the `application.xml`, `application-config.xml` and the `weblogic-application.xml` files are inside the META-INF directories inside their respective enterprise applications.

2. Make sure that no files are at the top level of the archive folder, as shown in Figure 5-10.

Figure 5-10 Domain Template Before Being Archived



3. Enter the following command from a command line at the archive folder:

```
jar -cfM ..\myportal.jar domain META-INF
```

4. To make this domain template available to the Domain Configuration Wizard, place the JAR file in the following directory:

```
<BEA_HOME>/weblogic700/common/templates/domains/
```

Creating a Portal Template

Like the process of creating a domain template, creating a portal template is largely a matter of creating and customizing a new instance of a portal, and then packaging it such that the Portal Wizard recognizes where to put the assets.

Instantiate a New Portal

For detailed instructions on creating a new portal using the Portal Wizard, consult the section Create the New Portal.

Customize the New Portal

A large body of information is available for those customizing the behavior and appearance of a portal. For an overview and links to specific instructions, consult the WebLogic Portal Development Guide.

Apply Basic Configuration

You can now apply configuration to the new portal, including adding group portals, pages, portlets, entitlements and delegated administration settings. For detailed instructions on configuring a portal, consult the Portal Administration Guide.

Package the New Portal as a Template

Unlike a template for the Domain Configuration Wizard, a portal template does not need to be delivered in the form of a compressed archive file. However, packaging the new portal is largely a matter of editing the `template.xml` file, one of which is shown in Listing 5-12. Take the following steps to package your customized portal as a template for the Portal Wizard:

Step 1: Make Staging Directory

Create a directory called `/myportal` to serve as the staging folder for the new portal template. Create two directories inside: `/j2ee` and `/ebcc`.

Step 2: Locate Source Directories

Locate the two directories associated with the source portal: the J2EE resources, as shown in Figure 5-11, and the metadata directories, as shown in Figure 5-12.

- Inside the domain, find the Web application directory named after your portal Web application. In this example, the Web application is called NewPWApp.
- The corresponding metadata directory is called portalApp-project (named after the Enterprise application.)

Figure 5-11 J2EE Resource Directories in NewPWApp

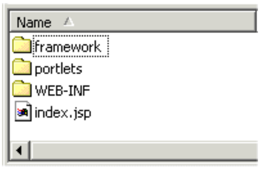
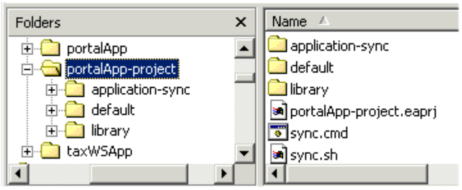

Name
framework
portlets
WEB-INF
index.jsp

Figure 5-12 Metadata directories for NewPWApp


Folders
portalApp
portalApp-project
application-sync
default
library
taxWSApp

Name
application-sync
default
library
portalApp-project.eaprj
sync.cmd
sync.sh

Step 3: Move Portal Resources

Move the resources into the template staging directory:

- Copy the contents of the `NewPWApp` directory into the `myportal/j2ee` directory.
- Copy the contents of the application-sync directory into the `myportal/ebcc` directory.

Note: Only copy metadata for resources you have added to the portal during customization. Do not copy the stock resource metadata files from the stockportal template.

Step 4: Edit template.xml

Edit the template.xml file according to the following rules:

- Listing 5-10 shows properties passed in by the Portal Wizard.
- Listing 5-11 shows properties you must set according to your portal.
- Make any other changes to the `template.xml` file required by your customized Portal Web application. It is an ANT build script. Listing 5-12 shows the full text of a template.xml for the Portal Wizard.

Listing 5-10 Properties provided by the Portal Wizard

```
<property name="template.common.lib.root.dir" value="Path to
directory containing required jar files" />

<property name="template.ebcc.root.dir" value="Path to directory
containing application data directory" />

<property name="template.j2ee.webapp.root.dir" value="Path to web
application root directory" />

<property name="template.webapp.name" value="Name of the web
application"/>

<property name="template.portal.name" value="Portal web
application name"/>

<property name="template.portal.description" value="Description of
the portal application">
```

Listing 5-11 Properties unique to your portal template

```
<property name="template.name" value="baseportal" />

<property name="template.description" value="Description: Base
Portal Template" />
```

```
<property name="template.hyperlink.text" value="After your new
portal is deployed, follow these instructions..." />

<property name="template.hyperlink.url"
value="http://edocs.bea.com/wlp/docs70/dev/newdom.htm#1003370" />
```

Step 5: Create a Thumbnail

Create an icon to signify the look of your portal template in the Portal Wizard. Name this file `template-icon.gif`. (Without this optional file, the Portal Wizard will display a stock icon.)

Step 6: Create Archive File

(Optional) Compress the contents of the portal template: From the `myportal` directory, execute the following command:

```
jar -cfM ../myportal.jar *.*
```

Step 7: Make the Archive Available

Place the resulting archive file in the following directory:

```
<BEA_HOME>\weblogic700\common\templates\webapps\portal
```

Listing 5-12 Portal template.xml

```
<?xml version="1.0"?>

<project name="Base Portal Template" default="main" basedir=".">

  <!--

    The jar or directory must contain this template.xml file AND can

    contain a template-icon.gif.  If a template-icon.gif is not

    present, a default will be provided.

  -->
  <!-- The caller should pass the following properties.

     <property name="template.common.lib.root.dir" value="Path to directory
containing required jar files" />

     <property name="template.ebcc.root.dir" value="Path to directory containing
application data directory" />

    <property name="template.j2ee.webapp.root.dir" value="Path to web application
root directory" />

     <property name="template.webapp.name" value="Name of the web application"/>

    <property name="template.portal.name" value="Portal web application name"/>

    <property name="template.portal.description" value="Description of the portal
application">

    <property name="template.hotdeploy.path" value="Path to directory containing
the portal for hot deploy"/>

     <property name="template.hotdeploy.user" value="User name for logging into
the server for hot deploy"/>

    <property name="template.hotdeploy.password" value="Password name for logging
into the server for hot deploy"/>


<property name="template.hotdeploy.adminurl" value="Server location for hot
deploy"/>

  -->


   <!-- Template Properties -->

   <property name="template.name" value="baseportal" />

   <property name="template.version" value="1.0" />

   <property name="template.type" value="portal-webapp" />

   <property name="template.description" value="Description: Base Portal
Template" />

   <property name="template.hyperlink.text" value="After your new portal is
deployed, follow these instructions..." />

   <property name="template.hyperlink.url"
value="http://edocs.bea.com/wlp/docs70/dev/newdom.htm#1003370" />
```

```
<!-- This can either be a .war or a directory for hot deployment -->

   <property name="template.hotdeploy.path"
value="${template.j2ee.webapp.root.dir}/${template.webapp.name}/"/>


   <target name="main" >

      <echo message="template.common.lib.root.dir (
${template.common.lib.root.dir} )"/>

      <echo message="template.ebcc.root.dir ( ${template.ebcc.root.dir} )"/>

     <echo message="template.j2ee.eapp.root.dir ( ${template.j2ee.eapp.root.dir}
)"/>

      <echo message="template.j2ee.webapp.root.dir (
${template.j2ee.webapp.root.dir} )"/>

      <echo message="template.appsync.dir ( ${template.appsync.dir} )"/>

      <echo message="template.webapp.name ( ${template.webapp.name} )"/>

      <echo message="template.portal.name ( ${template.portal.name} )"/>

     <echo message="template.hotdeploy.path ( ${template.hotdeploy.path} )"/>


      <!-- everything but baseportal and tools webapps -->


      <copy todir="${template.ebcc.root.dir}/"

            overwrite="no"

            preservelastmodified="yes"

            includeEmptyDirs="yes"

            filtering="no" >


         <fileset dir="ebcc/" >

            <include name="**" />

            <exclude name="application-sync/webapps/baseportal/" />
```

```
              <exclude name="application-sync/webapps/tools/" />

         </fileset>

      </copy>


      <!-- now copy baseportal webapp (exception baseportal.portal) and rename
the directory to whatever webapp the user chooses -->

      <copy
todir="${template.ebcc.root.dir}/application-sync/webapps/${template.webapp.nam
e}/"

            overwrite="no"

            preservelastmodified="yes"

            includeEmptyDirs="yes"

            filtering="no" >


         <fileset dir="ebcc/application-sync/webapps/baseportal/" >

              <include name="**" />

              <exclude name="baseportal.portal" />

         </fileset>

      </copy>


      <filter token="template.portal.description"
value="${template.portal.description}" />


      <!-- now copy the baseportal.portal file and rename it to whatever portal
name the user chooses -->

      <copy
tofile="${template.ebcc.root.dir}/application-sync/webapps/${template.webapp.na
me}/${template.portal.name}.portal"

            file="ebcc/application-sync/webapps/baseportal/baseportal.portal"

            overwrite="no"
            preservelastmodified="yes"
            includeEmptyDirs="yes"
            filtering="no" >
      </copy>

      <!-- copy all J2EE resources -->
      <copy todir="${template.j2ee.webapp.root.dir}/${template.webapp.name}/"
            overwrite="no"
            preservelastmodified="yes"
            includeEmptyDirs="yes"
            filtering="no" >

       <fileset dir="j2ee/" >
              <include name="**" />
              <exclude name="WEB-INF/weblogic.xml.stock"/>
              <exclude name="WEB-INF/web.xml.stock"/>
        </fileset>
      </copy>
     <filter token="template.portal.name" value="${template.portal.name}" />
      <filter token="template.webapp.name" value="${template.webapp.name}" />

      <copy
tofile="${template.j2ee.webapp.root.dir}/${template.webapp.name}/WEB-INF/weblog
ic.xml"
            overwrite="yes"
            preservelastmodified="yes"
            includeEmptyDirs="yes"
            filtering="no"
            file="j2ee/WEB-INF/weblogic.xml.stock">
      </copy>


      <copy
tofile="${template.j2ee.webapp.root.dir}/${template.webapp.name}/WEB-INF/web.xm
l"
            overwrite="yes"
            preservelastmodified="yes"
            includeEmptyDirs="yes"
            filtering="no"
            file="j2ee/WEB-INF/web.xml.stock">
      </copy>
   </target>
</project>
```

CHAPTER

6 Implementing User Profiles

In WebLogic Portal, users are represented by *user profiles*. User profiles provide flexibility in representing, storing, and accessing user attributes. In addition to supporting basic user profiles, WebLogic Portal provides a *Unified User Profile* (UUP) that can be used to create a virtual enterprise profile.

This section includes information on the following subjects:

- Creating a Unified User Profile
- Creating a Property Set Definition
- Enabling Visitor Self-Registration

Creating a Unified User Profile

A Unified User Profile provides the capability to leverage user data from external sources such as LDAP servers, legacy systems and databases. This allows for the use of a single profile to access user data from many different sources.

To create a UUP to retrieve user data from external sources, complete the following tasks:

1. Create an EntityPropertyManager EJB to Represent External Data
2. Deploy a ProfileManager That Can Use the New EntityPropertyManager

Create an EntityPropertyManager EJB to Represent External Data

To incorporate data from an external source, you must first create a stateless session bean that implements the methods of the `com.bea.p13n.property.EntityPropertyManager` remote interface. `EntityPropertyManager` is the remote interface for a session bean that handles the persistence of property data and the creation and deletion of profile records.

In addition, the stateless session bean should include a home interface and an implementation class. For example:

```
MyEntityPropertyManager
    extends com.bea.p13n.property.EntityPropertyManager

MyEntityPropertyManagerHome
    extends javax.ejb.EJBHome
```

Your implementation class can extend the `EntityPropertyManagerImpl` class. However the only requirement is that your implementation class is a valid implementation of the `MyEntityPropertyManager` remote interface. For example:

```
MyEntityPropertyManagerImpl extends
com.bea.p13n.property.internal.EntityPropertyManagerImpl
```

or

```
MyEntityPropertyManagerImpl extends
javax.ejb.SessionBean
```

Recommended EJB Guidelines

We recommend the following guidelines for your new EJB:

- Your custom `EntityPropertyManager` is not a default `EntityPropertyManager`. A default `EntityPropertyManager` is used to get/set/remove properties in the Portal schema. Your custom `EntityPropertyManager` does not have to support the following methods. It can throw `java.lang.UsupportedOperationException` instead:

  ```
  getDynamicProperties
  ```

`getEntityNames`

`getHomeName`

`getPropertyLocator`

`getUniqueId`

- If you want to be able to use the Portal framework and tools to create and remove users in your external data store then you must support the `createUniqueId()` and `removeEntity()` methods. However, your custom `EntityPropertyManager` is not the default `EntityPropertyManager` so your `createUniqueId()` method does not have to return a unique number. It must create the user entity in your external data store and then it can return any number, such as -1.
- The following recommendations apply to the `EntityPropertyManager()` methods that you must support:
  - `getProperty()` – Use caching. You should support the `getProperties` method to retrieve all properties for a user at once, caching them at the same time. Your `getProperty` method should use `getProperties`.
  - `setProperty()` – Use caching.
  - `removeProperties()`, `removeProperty()` – After these methods are called, a call to getProperty should return null for the property. Remove properties from the cache too.
- Your implementations of the `getProperty()`, `setProperty()`, `removeProperty()`, and `removeProperties()` methods must include any logic necessary to connect to the external system.
- If you want to cache property data, the methods must be able to cache profile data appropriately for that system. (See the `com.bea.p13n.cache` package at `../javadoc/index.html`.)
- If the external system contains read-only data, any methods that modify profile data must throw a `java.lang.UnsupportedOperationException`. Additionally, if the external data source contains users that are created and deleted by something other than your WebLogic Portal `createUniqueId` and `removeEntity` methods can simply throw an `UnsupportedOperationException`.
- To avoid class loader dependency issues, make sure that your EJB resides in its own package.

- For ease of maintenance, place the compiled classes of your custom `EntityPropertyManager` bean in your own JAR file (instead of modifying an existing WebLogic Portal JAR file).

Before you deploy your JAR file, follow the steps in the next section.

Deploy a ProfileManager That Can Use the New EntityPropertyManager

A "user type" is a mapping of a `ProfileType` name to a particular `ProfileManager`. This mapping is done in the `UserManager` EJB deployment descriptor.

To access the data in your new `EntityPropertyManager` EJB, you must do **one** of the following:

- In most cases you will be able to use the default deployment of `ProfileManager`, the `UserProfileManager`. You will modify the `UserProfileManager`'s deployment descriptor to map a property set and/or properties to your custom `EntityPropertyManager`. If you support the `createUniqueId()` and `removeEntity()` methods in your custom `EntityPropertyManager`, you can use WebLogic Portal Administration Tools to create a user of type "User" with a profile that can get/set properties using your custom `EntityPropertyManager`. For more information, refer to "Modifying the Existing ProfileManager Deployment Configuration" on page 6-5.

- In some cases you may want to deploy a newly configured `ProfileManager` that will be used instead of the `UserProfileManager`. This new `ProfileManager` is mapped to a `ProfileType` in the deployment descriptor for the `UserManager`. If you support the `createUniqueId()` and `removeEntity()` methods in your custom `EntityPropertyManager`, you can use the WebLogic Portal Administration Tools (or API) to create a user of type "MyUser" (or anything else you name it) that can get/set properties using the customized deployment of the `ProfileManager` that is, in turn, configured to use your custom `EntityPropertyManager`. For more information, refer to "Configuring and Deploying a New ProfileManager" on page 6-10.

ProfileManager is a stateless session bean that manages access to the profile values that the EntityPropertyManager EJB retrieves. It relies on a set of mapping statements in its deployment descriptor to find data. For example, the ProfileManager receives a request for the value of the DateOfBirth property, which is located in the PersonalData property set. ProfileManager uses the mapping statements in its deployment descriptor to determine which EntityPropertyManager EJB contains the data.

Modifying the Existing ProfileManager Deployment Configuration

If you use the existing UserProfileManager deployment to manage your user profiles, perform the following steps to modify the deployment configuration.

Under most circumstances, this is the method you should use to deploy your UUP. An example of this method is the deployment of the custom EntityPropertyManager for LDAP property retrieval, the LdapPropertyManager. The classes for the LdapPropertyManager are packaged in ldapprofile.jar. The deployment descriptor for the UserProfileManager EJB is configured to map the “ldap” property set to the LdapPropertyManager. The UserProfileManager is deployed in usermgmt.jar.

1. Back up the usermgmt.jar file in your enterprise application root directory.
2. From usermgmt.jar, extract META-INF/ejb-jar.xml and open it for editing.
3. In ejb-jar.xml, find the <env-entry> element, as shown in Listing 6-1:

Listing 6-1 <env-entry> Element

```
<!-- map all properties in property set ldap to ldap server -->

<env-entry>
 <env-entry-name>PropertyMapping/ldap</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>LdapPropertyManager</env-entry-value>
</env-entry>
```

and add an <env-entry> element after this to map a property set to your custom EntityPropertyManager, a shown in Listing 6-2:

Listing 6-2 Adding Another `<env-entry>` Element to Map a Property

```
<!-- map all properties in UUPExample property set to
MyEntityPropertyManager -->

<env-entry>
  <env-entry-name>PropertyMapping/UUPExample</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
```

4. In `ejb-jar.xml`, find the `<ejb-ref>` element shown in Listing 6-3

Listing 6-3 `<ejb-ref>` Element

```
<!-- an ldap property manager -->
<ejb-ref>
  <ejb-ref-name>ejb/LdapPropertyManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.bea.p13n.property.EntityPropertyManagerHome</home>
  <remote>com.bea.p13n.property.EntityPropertyManager</remote>
</ejb-ref>
```

and add a `<ejb-ref>` element after this to map a reference to an EJB that matches the name from the previous step with `ejb/` prepended as shown in Listing 6-4:

Listing 6-4 `<ejb-ref>` Element Mapping a Reference to an EJB

```
<!-- an example property manager -->
<ejb-ref>
  <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>examples.usermgmt.MyEntityPropertyManagerHome</home>
  <remote>examples.usermgmt.MyEntityPropertyManager</remote>
</ejb-ref>
```

The home and remote class names match the classes from your EJB JAR file for your custom `EntityPropertyManager`.

5. If your `EntityPropertyManager` implementation handles creating and removing profile records, you must also add Creator and Remover entries. For example:

Listing 6-5 <env-entry> Element that Adds Creator and Remover Entries

```
<env-entry>
  <env-entry-name>Creator/Creator1</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>

<env-entry>
  <env-entry-name>Remover/Remover1</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
```

This instructs the `UserProfileManager` to call your custom `EntityPropertyManager` when creating or deleting user profile records. The names "Creator1" and "Remover1" are arbitrary. All Creators and Removers will be iterated through when the `UserProfileManager` creates or removes a user profile. The value for the Creator and Remover matches the `ejb-ref-name` for your custom `EntityPropertyManager` without the `ejb/` prefix.

6. From `usermgmt.jar`, extract `META-INF/weblogic-ejb-jar.xml` and open it for editing.

7. In `weblogic-ejb-jar.xml`, find the elements described in Listing 6-6:

Listing 6-6 `weblogic-ejb-jar.xml` Elements

```
<weblogic-enterprise-bean>
  <ejb-name>UserProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
      <jndi-name>${APPNAME}.BEA_personalization.
      EntityPropertyManager</jndi-name>
    </ejb-reference-description>
```

and add an `ejb-reference-description` to map the `ejb-ref` for your custom `EntityPropertyManager` to the JNDI name. This JNDI name must match the name you assigned in `weblogic-ejb-jar.xml` in the JAR file for your customer `EntityPropertyManager`. It should look like the example in Listing 6-7:

Listing 6-7 Showing the JNDI Name

```
<weblogic-enterprise-bean>
  <ejb-name>UserProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
      <jndi-name>${APPNAME}.BEA_personalization.
      EntityPropertyManager</jndi-name>
    </ejb-reference-description>
    <ejb-reference-description>
      <ejb-ref-name>ejb/MyEntityPropertyManager
      </ejb-ref-name>
      <jndi-name>${APPNAME}.BEA_personalization.
      MyEntityPropertyManager</jndi-name>
    </ejb-reference-description>
```

Note the `${APPNAME}` string substitution variable. The WebLogic EJB container automatically substitutes the enterprise application name to scope the JNDI name to the application.

8. Update `usermgmt.jar` for your new deployment descriptors. You can use the `jar uf` command to update the modified `META-INF/` deployment descriptors.

9. Edit `META-INF/application.xml` for your enterprise application to add an entry for your custom `EntityPropertyManager` EJB module as shown in Listing 6-8:

Listing 6-8 Adding an Entry for a Custom EntityPropertyManager EJB Module

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

10. If you are using an application-wide cache, you can manage it from the Administration Console if you add a `<Cache>` tag for your cache to the `META-INF/application-config.xml` deployment descriptor for your enterprise application like this:

Listing 6-9 Adding a `<Cache>` Tag to `META-INF/application-config.xml`

```
<Cache
  Name="UUPExampleCache"
  TimeToLive="60000"
/>
```

11. Verify the modified `usermgmt.jar` and your custom `EntityPropertyManager` EJB JAR archive are in the root directory of your enterprise application and start WebLogic Server.
12. Use the WebLogic Server Administration Console to verify your EJB module is deployed to the enterprise application and then use the console to add your server as a target for the EJB module. You need to select a target to have your domain's `config.xml` file updated to deploy your EJB module to the server.
13. Use the E-Business Control Center to create a User Profile (property set) that matches the name of the property set that you mapped to your custom `EntityPropertyManager` in `ejb-jar.xml` for the `UserProfileManager` (in `usermgmt.jar`). You could also map specific property names in a property set to your custom `EntityPropertyManager`.

    Note: Be sure to synchronize the new data to your server after the property set is created.

Your new Unified User Profile type is ready to use. You can use the WebLogic Portal Administration Tools to create a user of type “User,” and it will use your UUP implementation when the “UUPExample” property set is being modified. When you call `createUser("bob", "password")` or `createUser("bob", "password", null)` on the `UserManager`, several things will happen:

- A user named “bob” is created in the security realm.
- A WebLogic Portal Server profile record is created for “bob” in the WebLogic Portal RDBMS repository.
- If you set up the Creator mapping, the `UserManager` will call the default `ProfileManager` deployment (`UserProfileManager`) which will call your custom `EntityPropertyManager` to create a record for Bob in your data source.
- Retrieving Bob's profile will use the default `ProfileManager` deployment (`UserProfileManager`), and when you request a property belonging to the “UUPExample” property set, the request will be routed to your custom `EntityPropertyManager` implementation.

Configuring and Deploying a New ProfileManager

If you are going to deploy a newly configured `ProfileManager` instead of using the default `ProfileManager` (`UserProfileManager`) to manage your user profiles, perform the following steps to modify the deployment configuration. In most cases, you will not have to use this method of deployment. Use this method only if you need to support multiple types of users that require different `ProfileManager` deployments—deployments that allow a property set to be mapped to different custom `EntityPropertyManager`s based on `ProfileType`.

An example of this method is the deployment of the custom `CustomerProfileManager` in `customer.jar`. The `CustomerProfileManager` is configured to use the custom `EntityPropertyManager` (`CustomerPropertyManager`) for properties in the “CustomerProperties” property set. The `UserManager` EJB in `usermgmt.jar` is configured to map the “WLCS_Customer” ProfileType to the custom deployment of the `ProfileManager`, `CustomerProfileManager`.

To configure and deploy a new ProfileManager, use this procedure.

1. Back up the `usermgmt.jar` file in your enterprise application root directory.

2. From `usermgmt.jar`, extract `META-INF/ejb-jar.xml`, and open it for editing.
3. In `ejb-jar.xml`, copy the entire `<session>` tag for the `UserProfileManager`, and configure it to use your custom implementation class for your new deployment of `ProfileManager`.

   In addition, you could extend the `UserProfileManager home` and `remote` interfaces with your own interfaces if you want to repackage them to correspond to your packaging (for example., `examples.usermgmt.MyProfileManagerHome`, `examples.usermgmt.MyProfileManager`).

   However, it is sufficient to replace the bean implementation class:

   You must create an `<env-entry>` element to map a property set to your custom `EntityPropertyManager`. You must also create a `<ejb-ref>` element to map a reference to an EJB that matches the name from the `PropertyMapping` with `ejb/` prepended. The `home` and `remote` class names for your custom `EntityPropertyManager` match the classes from your EJB JAR file for your custom `EntityPropertyManager`.

   Also, if your `EntityPropertyManager` implementation handles creating and removing profile records, you must also add Creator and Remover entries. This instructs your new `ProfileManager` to call your custom `EntityPropertyManager` when creating or deleting user profile records.

   Note: The name suffixes for the Creator and Remover, “Creator1” and “Remover1”, are arbitrary. All Creators and Removers will be iterated through when your `ProfileManager` creates or removes a user profile. The value for the Creator and Remover matches the `<ejb-ref-name>` for your custom `EntityPropertyManager` without the `ejb/` prefix.
4. In `ejb-jar.xml`, you must add an `<ejb-ref>` to the `UserManager` EJB section to map your `ProfileType` to your new deployment of the ProfileManager, as shown in Listing 6-10:

Listing 6-10 Adding an `<ejb-ref>` to the UserManager EJB Section

```
<ejb-ref>
  <ejb-ref-name>ejb/ProfileType/UUPExampleUser</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.bea.p13n.usermgmt.profile.ProfileManagerHome</home>
  <remote>com.bea.p13n.usermgmt.profile.ProfileManager</remote>
</ejb-ref>
```

The `<ejb-ref-name>` must start with `ejb/ProfileType/` and must end with the name that you want to use as the profile type as an argument in the `createUser()` method of `UserManager`.

5. From `usermgmt.jar`, extract `META-INF/weblogic-ejb-jar.xml` and open it for editing.

6. In `weblogic-ejb-jar.xml`, copy the `weblogic-enterprise-bean` tag, shown in Listing 6-11, for the `UserProfileManager` and configure it for your new `ProfileManager` deployment:

Listing 6-11 <weblogic-enterprise-bean> Tag for the UserProfileManager

```
<weblogic-enterprise-bean>
  <ejb-name>MyProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
        <jndi-name>${APPNAME}.BEA_personalization.
        EntityPropertyManager</jndi-name>
      </ejb-reference-description>
      <ejb-reference-description>
        <ejb-ref-name>ejb/PropertySetManager</ejb-ref-name>
        <jndi-name>${APPNAME}.BEA_personalization.
        PropertySetManager</jndi-name>
      </ejb-reference-description>
      <ejb-reference-description>
        <ejb-ref-name>ejb/MyEntityPropertyManager
        </ejb-ref-name>
        <jndi-name>${APPNAME}.BEA_personalization.
        MyEnitityPropertyManager</jndi-name>
      </ejb-reference-description>
  </reference-descriptor>
  <jndi-name>${APPNAME}.BEA_personalization.
  MyProfileManager</jndi-name>
</weblogic-enterprise-bean>
```

You must create an `<ejb-reference-description>` to map the `<ejb-ref>` for your custom `EntityPropertyManager` to the JNDI name. This JNDI name must match the name you assigned in `weblogic-ejb-jar.xml` in the JAR file for your custom `EntityPropertyManager`.

Note the `${APPNAME}` string substitution variable. The WebLogic Server EJB container automatically substitutes the enterprise application name to scope the JNDI name to the application.

7. In `weblogic-ejb-jar.xml`, copy the `<transaction-isolation>` tag for the `UserProfileManager`, shown in Listing 6-12, and configure it for your new `ProfileManager` deployment:

Listing 6-12 `<transaction-isolation>` Tag for the UserProfileManager

```
<transaction-isolation>
  <isolation-level>TRANSACTION_READ_COMMITTED
  </isolation-level>
  <method>
    <ejb-name>MyProfileManager</ejb-name>
    <method-name>*</method-name>
  </method>
</transaction-isolation>
```

8. Create a temporary `usermgmt.jar` for your new deployment descriptors and your new `ProfileManager` bean implementation class. This temporary EJB JAR archive should not have any container classes in it. Run `ejbc` to generate new container classes.

9. Edit `META-INF/application.xml` for your enterprise application to add an entry for your custom `EntityPropertyManager` EJB module, as shown in Listing 6-13:

Listing 6-13 Adding an Entry to a Custom EntityPropertyManager EJB Module

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

10. If you are using an application-wide cache, you can manage it from the WebLogic Server Administration Console if you add a `<Cache>` tag for your cache to the `META-INF/application-config.xml` deployment descriptor for your enterprise application as shown in Listing 6-14:

Listing 6-14 Adding a `<Cache>` Tag to a `META-INF/application-config.xml`

```
<Cache
  Name="UUPExampleCache" TimeToLive="60000"
/>
```

11. Verify the modified `usermgmt.jar` and your custom `EntityPropertyManager` EJB JAR archive are in the root directory of your enterprise application and start your server.
12. Use the WebLogic Server Administration Console to verify your EJB module is deployed to the enterprise application, then use the WebLogic Server Administration Console to add your server as a target for the EJB module. You must select a target to have your domain's `config.xml` file updated to deploy your EJB module to the server.
13. Use the E-Business Control Center to create a User Profile (property set) that matches the name of the property set that you mapped to your custom `EntityPropertyManager` in `ejb-jar.xml` for the `UserProfileManager` (in `usermgmt.jar`). You could also map specific property names in a property set to your custom `EntityPropertyManager`.

    Note: Be sure to synchronize the new data to your server after the property set is created.

14. Your new Unified User Profile type is ready to use. You can use the WebLogic Portal Administration Tools to create a user of type "UUPExampleUser," and it will use your UUP implementation when the "UUPExample" property set is being modified. That is because you mapped the `ProfileType` using an `<ejb-ref>` in your `UserManager` deployment descriptor, `ejb/ProfileType/UUPExampleUser`.

    Note: Tell your administrators that when they create a user in the WebLogic Portal Administration Tools, they must select the new user type.

Now, when you call `createUser("bob", "password", "UUPExampleUser")` on the `UserManager`, several things will happen:

- A user named “bob” is created in the security realm.
- A WebLogic Portal Server profile record is created for “bob” in the WebLogic Portal RDBMS repository.
- If you set up the Creator mapping, the `UserManager` will call your new `ProfileManager` deployment, which will call your custom `EntityPropertyManager` to create a record for Bob in your data source.
- Retrieving Bob's profile will use your new `ProfileManager` deployment, and when you request a property belonging to the “UUPExample” property set, the request will be routed to your custom `EntityPropertyManager` implementation.

Retrieving User Profile Data from LDAP

The LdapRealm security realm and the LdapPropertyManager unified user profile (UUP) for retrieving user properties from LDAP are independent of each other. They do not share configuration information and there is no requirement to use either one in conjunction with the other. A security realm has nothing to do with a user profile. A security realm provides user/password data, user/group associations, and group/group associations. A user profile provides user and group properties. A password is not a property.

In order to successfully retrieve the user profile from the LDAP server, ensure that you've done the following:

1. If you have already deployed the application on a WebLogic Portal instance, stop the server.
2. Deploy the `ldapprofile.jar` component within your application.

   The LdapPropertyManager EJB in `ldapprofile.jar` has been enhanced as of 7.0 SP2 to allow for the inspection of the LDAP schema to determine multi-valued versus single-value LDAP attributes, to allow for multiple `userDN`/`groupDN`, and to allow for `SUBTREE_SCOPE` searches for users and groups in the LDAP server. Following are more detailed explanations:

   Before this enhancement, an attribute that is defined as multi-valued in the LDAP server's schema, but had only one value, was stored in memory as a single value property by the LdapPropertyManager. This could cause problems in the rules engine when this situation was encountered:

- A multi-valued LDAP property had a single value.
- A rule was created in Portal that used the property as a multi-valued property.
- The rules engine expects the property to be a `java.util.Collection` because it is multi-valued, but it was not, because the LdapPropertyManager saw it as single valued and stored it that way.

This change allows a developer to configure the ejb-jar.xml deployment descriptor for the LdapPropertyManager EJB to specify that the LDAP schema be used to determine if a property is single value or multi-value.

This is done by editing `ejb-jar.xml` to setting the following env-entries:

```
<!-- Flag to specify if LDAP attributes will be determined to be
single value or multi-value via the schema obtained from the
attribute.  If false, then the attribute is stored as
multi-valued (a Collection) only if it has more than one value.
Leave false unless you intend to use multi-valued LDAP attributes
that may have only one value.  Using true adds overhead to check
the LDAP schema.  Also, if you use true beware that most LDAP
attributes are multi-value.  For example, iPlanet Directory
Server 5.x uses multi-value for givenName, which you may not
expect unless you are familiar with LDAP schemas.  -->

<env-entry>
    <env-entry-name>config/detectSingleValueFromSchema
    </env-entry-name>
    <env-entry-type>java.lang.Boolean</env-entry-type>
    <env-entry-value>true</env-entry-value>
</env-entry>

<!-- Value of the name of the attribute in the LDAP schema that
is used to determine single value or multi-value (RFC2252 uses
SINGLE-VALUE) This attribute in the schema should be true for
single value and false or absent from the schema otherwise. The
value only matters if config/detectSingleValueFromSchema is
true. -->

<env-entry>
    <env-entry-name>config/singleValueSchemaAttribute
    </env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>SINGLE-VALUE</env-entry-value>
</env-entry>
```

It is not recommended that `true` be used for `config/detectSingleValueFromSchema` unless you are going to write rules that use multi-valued LDAP attributes that have a single value. Using `config/detectSingleValueFromSchema = true` adds the overhead of checking the LDAP schema for each attribute instead of the default behavior (`config/detectSingleValueFromSchema = false`), which only stores an attribute as multi-valued (in a Collection) if it has more than one value.

This patch also implements changes that allow you to use `SUBTREE_SCOPE` searches for users and groups. It also allows multiple base `userDN` and `groupDN` to be specified. The multiple base DN can be used with `SUBTREE_SCOPE` searches enabled or disabled.

A `SUBTREE_SCOPE` search begins at a base `userDN` (or `groupDN`) and works down the branches of that base DN until the first user (or group) is found that matches the username (or group name).

To enable `SUBTREE_SCOPE` searches you must set the Boolean `config/objectPropertySubtreeScope` env-entry in the `ejb-jar.xml` for `ldapprofile.jar` to true and then you must set the `config/userDN` and `config/groupDN` env-entry values to be equal to the base DNs from which you want your `SUBTREE_SCOPE` searches to begin.

For example, if you have users in `ou=PeopleA,ou=People,dc=mycompany,dc=com` and in `ou=PeopleB,ou=People,dc=mycompant,dc=com` then you could set `config/userDN` to `ou=People,dc=mycompant,dc=com` and properties for these users would be retrieved from your LDAP server because the user search would start at the "People" ou and work its way down the branches (`ou="PeopleA" and ou="PeopleB"`).

You should not create duplicate users in branches below your base userDN (or duplicate groups below your base groupDN) in your LDAP server. For example, your LDAP server will allow you to create a user with the `uid="userA"` under both your `PeopleA` and your `PeopleB` branches. The LdapPropertyManager in `ldapprofile.jar` will return property values for the first `userA` that it finds.

It is recommended that you do not enable this change (by setting `config/objectPropertySubtreeScope` to `true`) unless you need the flexibility offered by `SUBTREE_SCOPE` searches.

An alternative to `SUBTREE_SCOPE` searches (with or without multiple base DNs) would be to configure multiple base DNs and leave `config/objectPropertySubtreeScope` set to `false`. Each base DN would have to be the DN that contains the users (or groups) because searches would not go any lower than the base DN branches. The search would cycle from one base DN to the next until the first matching user (or group) is found.

The new `ejb-jar.xml` deployment descriptor is fully commented to explain how to set multiple DNs, multiple `usernameAttributes` (or `groupnameAttributes`), and how to set the objectPropertySubtreeScope flag.

3. Start the server and deploy the application.
4. Start the WebLogic Server Administration Console for the active domain.

Creating a Property Set Definition

Property sets are the schemas for personalization attributes. They are a convenient way to give a name to a group of properties for a specific purpose. For example, in the sampleportal-project, the User Profile “Avitek” has a property set that defines properties for an e-commerce customer, such as First Name, Last Name, Home Phone, E-mail, and Customer Type. Use the E-Business Control Center to create property sets and define the properties that make up these property sets.

This section describes how to register a customer user profile:

Registering Custom User Profiles

The property set editor works the same way for all property sets. In this exercise, the E-Business Control Center will be used to create and modify User Profile properties. These examples can be used to register a custom user profile. You can follow the same procedures to create and modify property sets for Events, HTTP Requests, HTTP Sessions, and the Catalog Structure.

You can set a default profile type for each web application by setting a context parameter in `web.xml` for DEFAULT_USER_PROFILE_TYPE. For example:

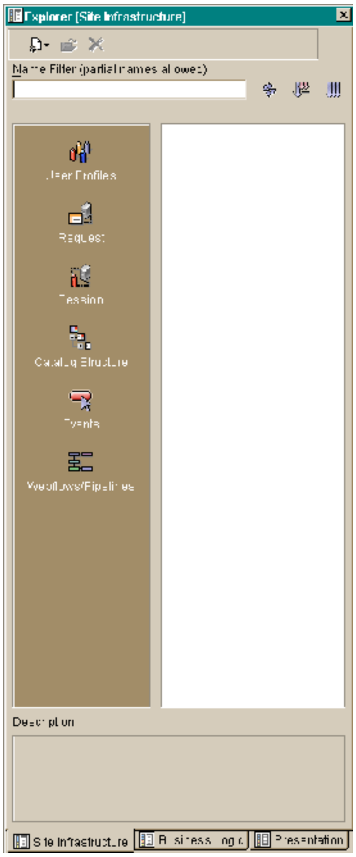
```
<context-param>
<param-name>DEFAULT_USER_PROFILE_TYPE</param-name>
<param-value>WLCS_Customer</param-value>
</context-param>
```

To register a custom user profile, complete the following steps:

1. Start the E-Business Control Center and ensure that it is connected to a server. For information on starting the E-Business Control Center and connecting it to a server, refer to "System Administration" in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/sysadmin.htm.

   The Explorer window opens as shown in Figure 6-1.

Figure 6-1 E-Business Control Center Window

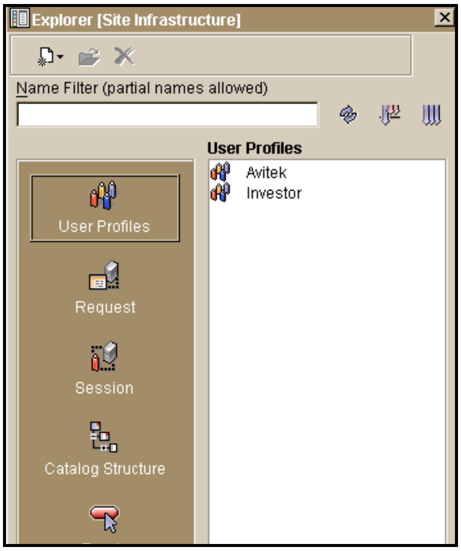


2. Open the appropriate project file. For the example in this procedure, you would open samples →portal →samplePortalDomain →beaApps →sampleportal-project.

3. Open the Event Editor as follows:

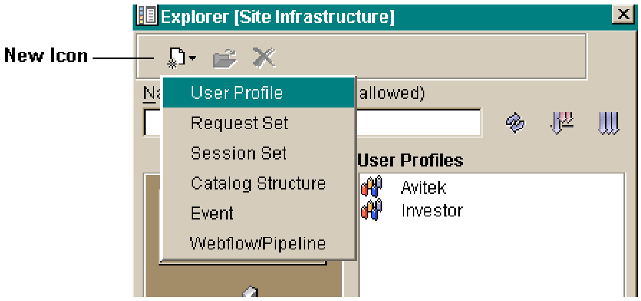
a. In the Explorer window, select the **User Profiles** icon. A list of User Profiles appears in the User Profiles field, as shown in Figure 6-2

Figure 6-2 E-Business Control Center Explorer with User Profiles Icon Selected



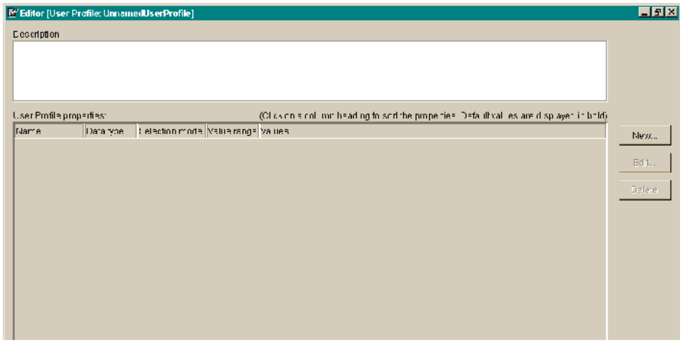
b. Click the **New** icon to open the New menu and then select **User Profile**, as shown in Listing 6-3.

Figure 6-3 New Menu (Opened by Clicking New Icon)



The User Profile Editor window appears, as shown in Figure 6-4.

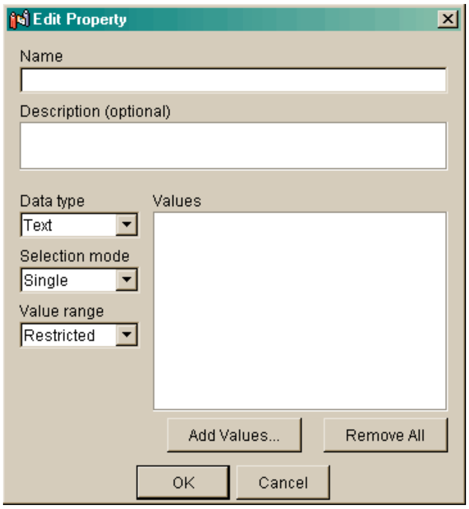
Figure 6-4 User profile Editor Window


Editor [User Profile: UnnamedUserProfile]
Description
User Profile properties:
(Click on a column heading to sort the properties. Default values are displayed in bold)
Name
Data type
Selection mode
Value range
Values
New...
Edit...
Delete

4. Click **New**.

   The Edit Property window appears (Figure 6-5).

Figure 6-5 Event Property Window



Data type: Text
Selection mode: Single
Value range: Restricted

5. In the Edit Property window, complete these steps:

   a. In the **Name** field, enter a unique name for the property no longer than 100 characters (required).

      Warning: Do not enter `LDAP` in the **Name** field.

b. In the **Description** field, enter a description of the property no longer than 254 characters (optional).

c. In the **Data type** list, select the data type. If you select Boolean as the data type, for example, the **Selection mode** and **Value range** are no longer available. The default for Boolean is Single, Restricted.

d. In the **Selection mode** list, select either **Single** or **Multiple**. The value you select here determines the number of property values you can set: one (**Single**) or multiple (**Multiple**).

e. In the Value range list, select whether the value is **Restricted** or **Unrestricted**.

f. Click **Add Values**.

One of two types of Enter Property Values windows appears. The type of Enter Property Values window that appears depends on the values selected. This is because, depending on the data type, different steps are required for entering values and setting default values. The following property categories are available:

- Properties with Boolean or a Single Value and Single Default.
- Properties with Multiple Values and Single, Multiple, or All Defaults
- Properties with Date and Time Values

Properties with Boolean or a Single Value and Single Default

To enter the default value for Boolean property or a property with a single value and a single default (unrestricted), complete the following steps:

1. In the applicable Enter Property Value window (Figure 6-6 or Figure 6-7), perform one of the following:
   - For a Boolean property, select either **True** or **False**.
   - For a Single Value, Single Default property, enter a value and click **Add**.

Figure 6-6 Enter Property Values Window—Boolean Values Required

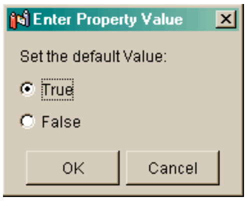
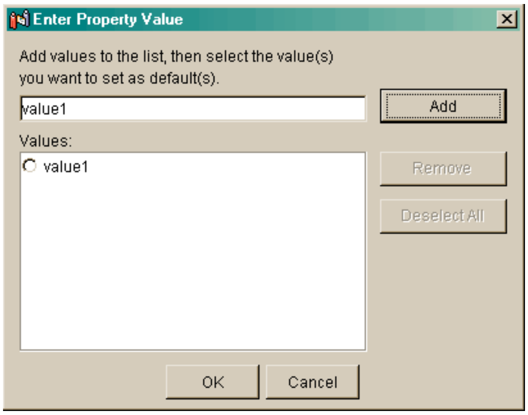


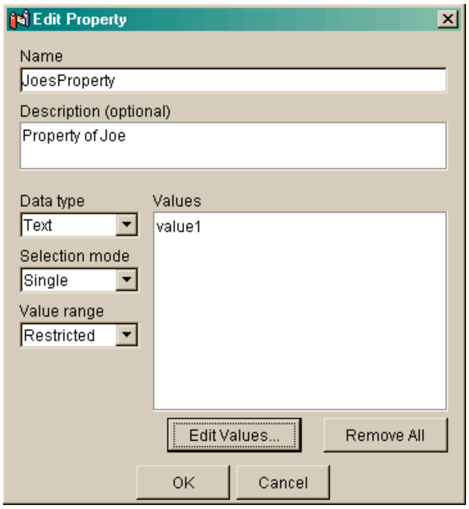
Figure 6-7 Enter Property Values Window—Single Value, Single Default Required


Enter Property Value
Add values to the list, then select the value(s) you want to set as default(s).
value1
Add
Values:
value1
Remove
Deselect All
OK
Cancel

2. Click **OK**.

   The Edit Property Value window closes, revealing the Edit Property window with the selected value(s) appearing in the Value list; for example as shown in Listing 6-8.

Figure 6-8 Edit Property Window with Text Value



Name: JoesProperty

Description (optional): Property of Joe

Data type: Text

Selection mode: Single

Value range: Restricted

Values: value1

3. Click **OK**.

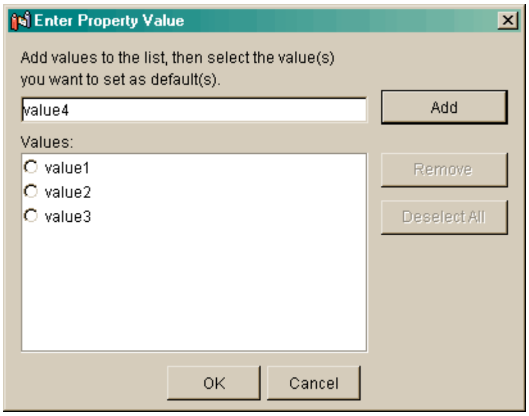
Properties with Multiple Values and Single, Multiple, or All Defaults

To enter multiple property values and set one or more defaults (unrestricted), complete the following steps:

1. In the applicable Enter Property Values window, enter a value, and then click **Add**.

   The new values will appear in the Values list box, as shown in Figure 6-9, Figure 6-10, and Figure 6-11.

Figure 6-9 Enter Property Values—Multiple Values, Single Default


Enter Property Value

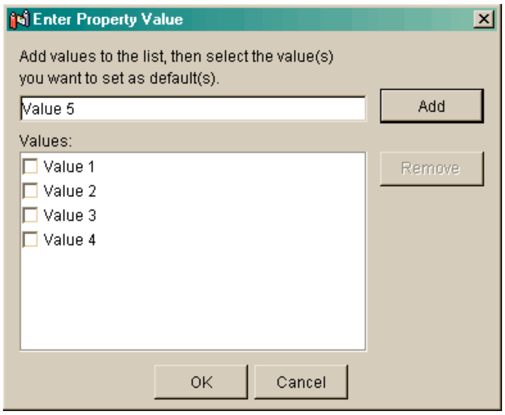
Add values to the list, then select the value(s) you want to set as default(s).

value4

Values:

value1
value2
value3

Figure 6-10 Enter Property Values—Multiple Values, Multiple Restricted Defaults


Enter Property Value

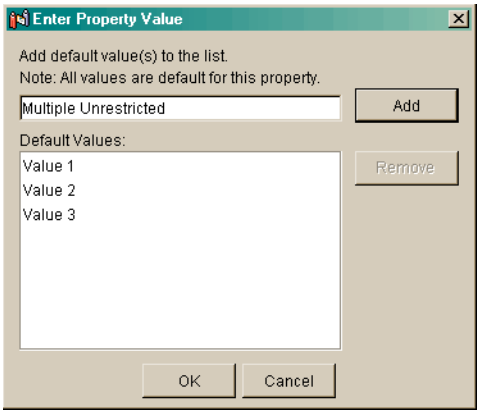
Add values to the list, then select the value(s) you want to set as default(s).

Value 5

Values:

Value 1
Value 2
Value 3
Value 4

Figure 6-11 Enter Property Values—Multiple Values, Multiple Unrestricted Defaults



2. Repeat step 1. until you have entered all necessary values.

3. To select one or more default values, complete one of the following:

   - If you do not want to select a default, go to step 5.
   - For multiple values with a single default, select the value (radio button) that you want to set as the default, and then click **OK**.

   Note: To remove the default value for a property with multiple values and a single default, click **Deselect All**.

   - For multiple values with multiple restricted defaults, select the value (check boxes) that you want to set as defaults, and then click **OK**.

   Note: For multiple values without restrictions (that is, the Value range is Unrestricted), you do not need to select any defaults.

4. In the Edit Property window, click **OK**.

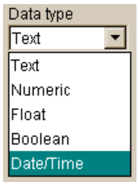
Properties with Date and Time Values

Properties with date and time values can use all Selection mode and Value range settings.

To enter date and time values and set one or more defaults, complete the following steps:

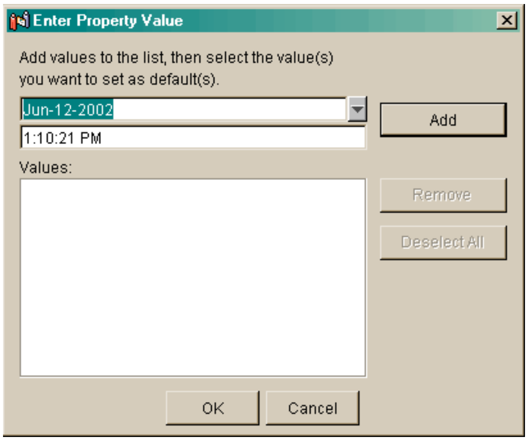
1. In the Edit Properties window, select **Date/Time** from the **Data type** drop-down list (shown in Figure 6-12) and select **Add Values**.

Figure 6-12 Date type Menu with Date/Time Selected



The Enter Property Value window for date and time values appears (Figure 6-13).

Figure 6-13 Date/Time Enter Property Value Window



Add values to the list, then select the value(s) you want to set as default(s).

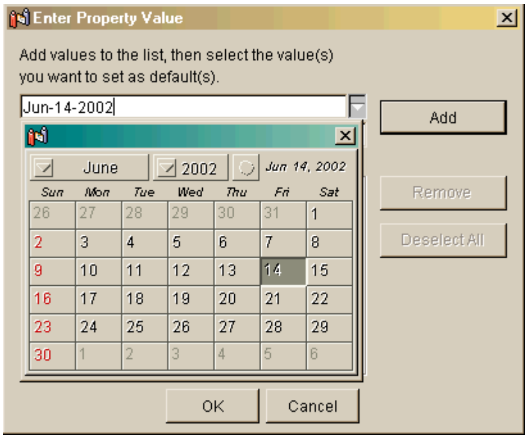
Jun-12-2002

1:10:21 PM

2. Click the drop-down arrow in the **Date** list.

A calendar appears, as shown in Figure 6-14.

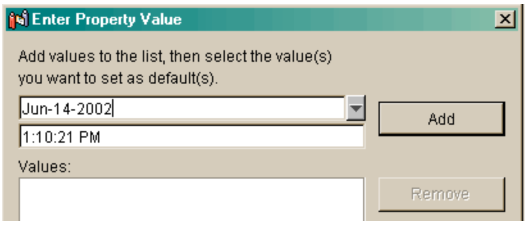
Figure 6-14 Enter Property Value Window with Calendar Displayed



3. Select a date from the calendar; for example June 14.

   The calendar disappears and the selected date appears in the date edit box, as shown in Figure 6-15

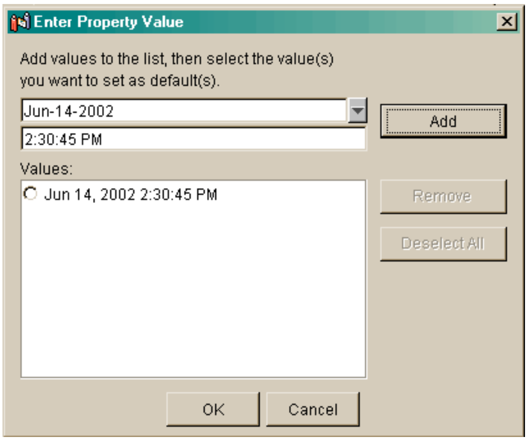
Figure 6-15 Selected Date Appears in Date Edit Box


Enter Property Value
Add values to the list, then select the value(s) you want to set as default(s).
Jun-14-2002
1:10:21 PM
Values:
Add
Remove

4. In the **Time** field, enter a time.
5. Click **Add**.

   The new time and date appear in the Values list, as shown in Figure 6-16.

Figure 6-16 Date and Time Appear in Values List



6. To add more dates and times, repeat step 1. through step 5. until you have entered all the necessary values.
7. To select one or more default values, complete one of the following:
   - If the event has a single date and time with a single default (restricted), click **OK**.
   - If the event has multiple dates and times with a single default (restricted), select the value (radio button) that you want to set as the default, and then click **OK**.
   - If the event has multiple dates and times with multiple defaults (unrestricted), select the values (check boxes) that you want to set as the default, and then click **OK**.
8. In the Edit Event Property window, click **OK**.

Updating a Registered Custom Event

Whenever you make changes to a custom event's code, you should update that event's registration. Updating the registration lets the E-Business Control Center know about the changes in the custom event and aids campaign developers using the E-Business Control Center to modify any scenario actions that refer to the event.

To update a custom event, complete the following steps.

1. Start the E-Business Control Center and ensure that it is connected to WebLogic Server.

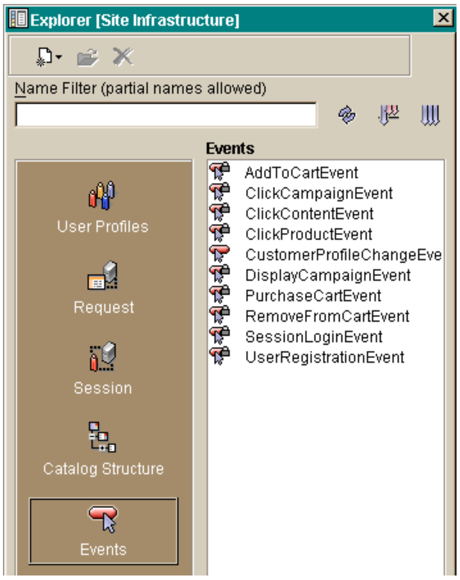
   The Explorer window opens.

2. Ensure that the correct project file is open and select the **Site Infrastructure** tab.

3. In the Explorer window, select the Events icon. A list of Events appears in the **Events** field as shown in Figure 6-17.

   Note: You cannot edit standard Events.

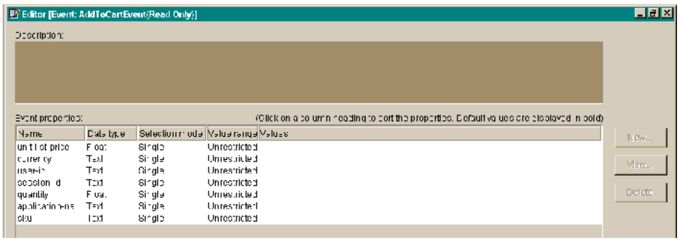
Figure 6-17 Explorer Window


Explorer [Site Infrastructure]
Name Filter (partial names allowed)
Events
User Profiles
Request
Session
Catalog Structure
Events

AddToCartEvent
ClickCampaignEvent
ClickContentEvent
ClickProductEvent
CustomerProfileChangeEve
DisplayCampaignEvent
PurchaseCartEvent
RemoveFromCartEvent
SessionLoginEvent
UserRegistrationEvent

4. Double-click the custom event that you want to edit. The Event Editor window opens as shown in Figure 6-18. The Event properties field displays a list of existing properties.

Figure 6-18 Event Editor Window



Description:

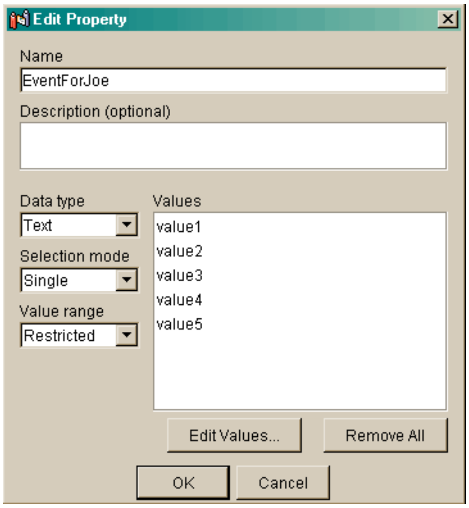
Event properties: (Click on a column heading to sort the properties. Default values are displayed in bold)

Name	Data type	Selection mode	Value range	Values
unit list price	Float	Single	Unrestricted	
currency	Text	Single	Unrestricted	
user-id	Text	Single	Unrestricted	
session-id	Text	Single	Unrestricted	
quantity	Float	Single	Unrestricted	
application-name	Text	Single	Unrestricted	
sku	Text	Single	Unrestricted	

5. In the **Event** properties field, double-clcik the property that you want to edit.

   The Edit Property window opens as shown in Figure 6-19.

Figure 6-19 Edit Property Window



Name: EventForJoe

Description (optional):

Data type: Text

Selection mode: Single

Value range: Restricted

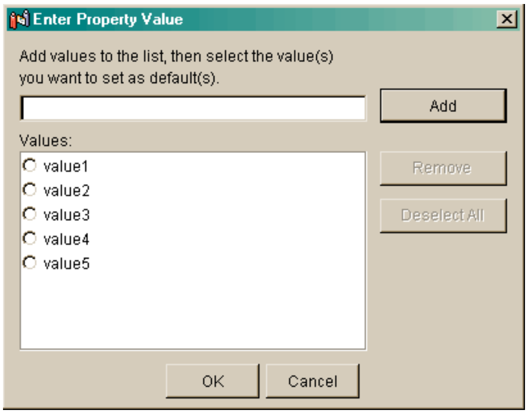
Values:
value1
value2
value3
value4
value5

6. To change the **Data type**, **Selection mode**, or **Value range**, select a setting from the appropriate list box.

   Note: If you change the property setting **Data type**, **Selection mode**, or **Value range**, the associated values will be erased.

7. To add or change values, click **Edit Values**. The Enter Property Value window opens as shown in Figure 6-20.

Figure 6-20 Enter Property Value Window



a. To remove a value, select the value, and then click **Remove**.

b. To add a value, enter the value, and then click **Add**.

c. To change a value, select the value, remove it, and then add the new value.

d. If required, select the default value or values.

e. To remove the default value for a property with multiple values and a single default, click **Deselect All**.

f. Click **OK**. The Enter Property Value window closes.

8. After you have finished updating the properties or values for the event, click **OK** in the Edit Event Property window.

Enabling Visitor Self-Registration

Visitors to Websites often need to register before they can proceed with using the site's features; for example, an online bookstore might require a visitor to register with them before they can actually buy books or other merchandise. Registration is valuable because it makes using a Website more convenient for the visitor because it stores pertinent information about them—called a customer profile—that is ncessary for each transaction, relieving the visitor of the need to re-enter this information whenever a transaction is made. It is convenient for your enterprise because it stores visitor data, which allows you to maintain information about people likely to use your service.

WebLogic Portal provides a set of JSP Webflow templates that create a customer profile as the visitor self-registers. You can use these components as is or tailor them for your specific needs. This section describes those JSPs and Webflow components and dicusses how they are used.

Implementing Customer Profile JSPs

WebLogic Portal provides a Login and Registration service comprised of five JSP templates you can use to enable visitor self-registration. You can use these templates as they are or you can modify them to meet your specific needs. This section describes those templates and shows you how to implement them.

This section discusses the following templates:

- login.jsp
- badlogin.jsp
- newuser.jsp
- newuserforward.jsp
- usercreationforward.jsp

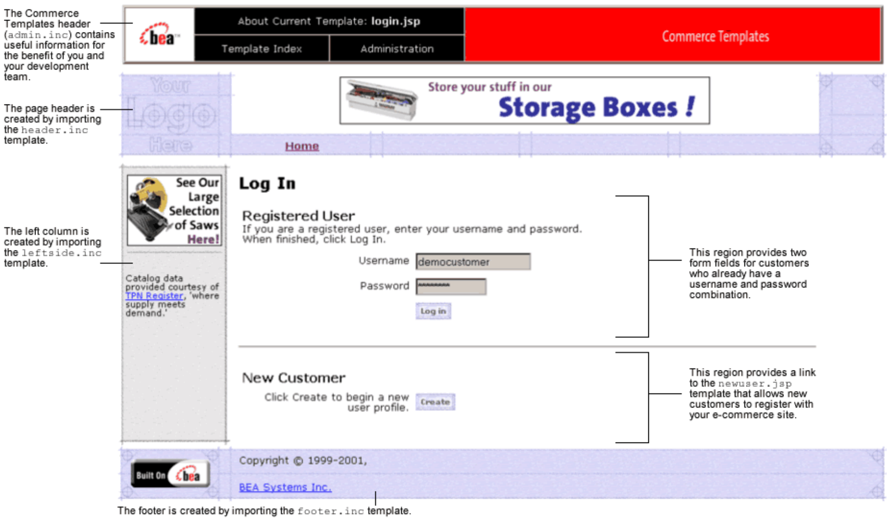
login.jsp

The `login.jsp` template provides customers who have not yet registered with your site an entry point into a page that allows them to register (create their initial customer profile) for subsequent use on the site. Since this page is the entry point to the checkout process, it also establishes mechanisms (such as sessions) that will allow customers to continue their shopping experience.

Description

Figure 6-21 shows an example of a Web page formatted with the `login.jsp` template.

Figure 6-21 login.jsp Formatted Web Page



The Commerce Templates header (admin.inc) contains useful information for the benefit of you and your development team.

The page header is created by importing the header.inc template.

The left column is created by importing the leftside.inc template.

Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Log In

Registered User

If you are a registered user, enter your username and password. When finished, click Log In.

This region provides two form fields for customers who already have a username and password combination.

New Customer

Click Create to begin a new user profile.

This region provides a link to the newuser.jsp template that allows new customers to register with your e-commerce site.

Copyright © 1999-2001,

BEA Systems Inc.

The footer is created by importing the footer.inc template.

How login.jsp Works

If an unregistered customer clicks **Create** in the portlet, the next page loaded allows the customer to create a profile and a username/password combination (`newuser.jsp`). After the customer has registered, the customer is automatically logged in and forwarded to the `newusercreation.jsp` template, which allows customers to continue shopping, view their shopping carts, or check out. If the auto-login is unsuccessful, the `login.jsp` template is loaded for the customer to enter their username and password. If the customer's login attempt is unsuccessful, the `badlogin.jsp` is loaded.

Notes: The option to proceed to checkout is only provided on the `newusercreation.jsp` template if there are items in the customer's shopping cart.

Events

The `login.jsp` template presents a customer with two buttons, only one of which is considered an event. The event triggers a particular response in the default Webflow that allows customers to continue. The other button is a standard HTML Submit button that posts the page back to the WebLogic Server for authentication. Table 6-1 provides information about the event and the business logic it invokes.

Table 6-1 login.jsp Events

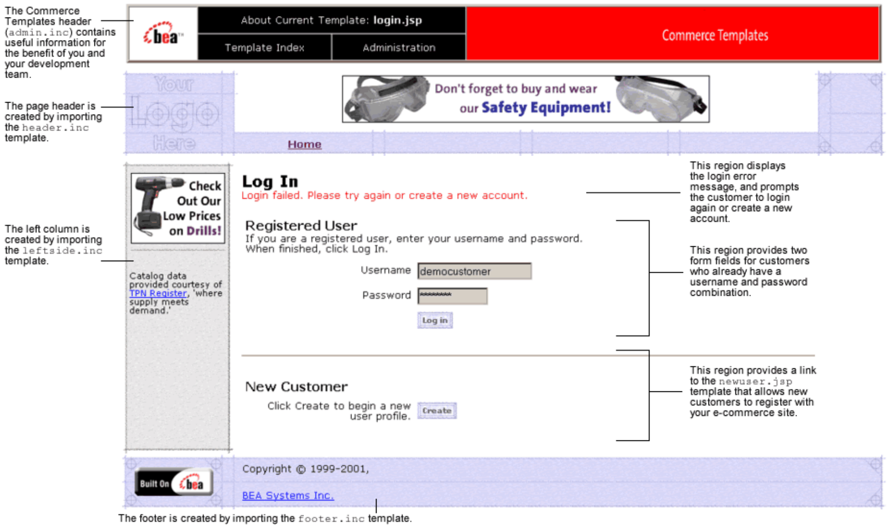
Event	Webflow Response(s)
`button.createUser`	No business logic required. Loads `newuser.jsp`.

The Login button is not an event that would trigger a Webflow response. Rather, when a customer clicks the button, control is turned over to the WebLogic Server (specifically, the RDBMS realm of the WebLogic Portal). The WebLogic Server remembers the HTTP request, determines whether the customer's username and password combination is correct, and then reinvokes the Webflow using the request.

badlogin.jsp

The `badlogin.jsp` template (shown in Figure 6-22) informs a customer that they have entered an invalid username/password combination, and allows the customer to try logging into a site again by providing a valid username/password combination. Except for the error message, it behaves exactly as the `login.jsp` template previously described.

Figure 6-22 badlogin.jsp Formatted Web Page


The Commerce Templates header (admin.inc) contains useful information for the benefit of you and your development team.
About Current Template: login.jsp
Template Index
Administration
Commerce Templates
The page header is created by importing the header.inc template.
Your Logo Here
Don't forget to buy and wear our Safety Equipment!
Home
Check Out Our Low Prices on Drills!
The left column is created by importing the leftside.inc template.
Catalog data provided courtesy of TPN Register, 'where supply meets demand.'
Log In
Login failed. Please try again or create a new account.
This region displays the login error message, and prompts the customer to login again or create a new account.
Registered User
If you are a registered user, enter your username and password. When finished, click Log In.
Username democustomer
Password
Log in
This region provides two form fields for customers who already have a username and password combination.
New Customer
Click Create to begin a new user profile.
Create
This region provides a link to the newuser.jsp template that allows new customers to register with your e-commerce site.
Built On bea
Copyright © 1999-2001,
BEA Systems Inc.
The footer is created by importing the footer.inc template.

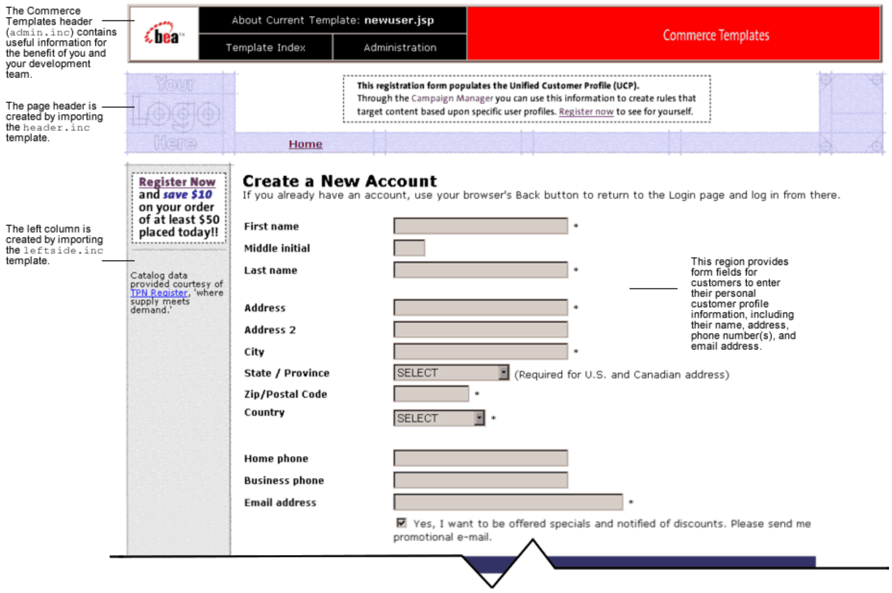
newuser.jsp

The `newuser.jsp` template allows a new customer to register with your e-commerce site by creating their customer profile, which includes personal information, shipping address information, payment information (optional), and account information.

Description

xxx through xxx show an example of how a Web page formatted with newuser.jsp might appear in a browser.

Figure 6-23 Web Page Formatted with `newuser.jsp` — Personal Information



The Commerce Templates header (`admin.inc`) contains useful information for the benefit of you and your development team.

The page header is created by importing the `header.inc` template.

The left column is created by importing the `leftside.inc` template.

This region provides form fields for customers to enter their personal customer profile information, including their name, address, phone number(s), and email address.

This registration form populates the Unified Customer Profile (UCP). Through the Campaign Manager you can use this information to create rules that target content based upon specific user profiles. Register now to see for yourself.

Register Now and *save $10* on your order of at least $50 placed today!!

Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Create a New Account

If you already have an account, use your browser's Back button to return to the Login page and log in from there.

First name		*
Middle initial		
Last name		*
Address		*
Address 2		
City		*
State / Province	SELECT	(Required for U.S. and Canadian address)
Zip/Postal Code		*
Country	SELECT	*
Home phone		
Business phone		
Email address		*

[✓] Yes, I want to be offered specials and notified of discounts. Please send me promotional e-mail.

How newuser.jsp Works

The page prior to `newuser.jsp` is the customer login page (`login.jsp`). If no errors are found after a customer enters their initial profile information, customers are auto-logged in and forwarded to a welcome page where they can select from the various links to continue shopping or check out (`newusercreation.jsp`). If errors are found, the `newuser.jsp` is reloaded with an appropriate message next to the invalid form fields.

This template is part of the `sampleapp_user` namespace in the Webflow.

JSP Templates Included by newuser.jsp

`newuser.jsp` includes three additional JSP templates when it is implemented. These JSPs provides a standardized format for both the form presentation and error handling in all JSP templates that prompt customers for shipping address, credit card information, and demographic information. These templates are described in the following paragraphs.

newaddresstemplate.inc This template provides a standardized format for both the form field presentation and error handling included in all JSP templates that prompt customers for a shipping address, except `addaddress.jsp`. The form fields are organized in a table, and upon form submission, the input processors associated with the `newaddresstemplate.inc` template will validate the form to ensure that all required fields contain values. If errors are detected, the `newaddresstemplate.inc` template will be redisplayed, with an error message at the top and the invalid field labels shown in a red (as opposed to the original black) font. Previously entered correct information will still be displayed in the form.

The behavior described above is invoked on the `newaddresstemplate.inc` template by using the `getValidatedValue` JSP tag, as shown in Listing 6-15.

Listing 6-15 Use of the `getValidatedValue` JSP Tag on newaddresstemplate.inc

```
<!-- begin table with customer's shipping address information -->

<table width="90%" border="0">
   <tr>
      <td width="26%"><webflow:getValidatedValue
         fieldName="<%=HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1%>"
         fieldValue="customerShippingAddress1" fieldStatus="status"
         validColor="black" invalidColor="red" unspecifiedColor="black"
         fieldColor="fontColor" />
         <div class="tabletext"><font color=<%= fontColor %>><b>Address </b>
            </font>
         </div>
      </td>
      <td width="74%"> <input type="text"
         name="<%=HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1%>"
         value="<%=customerShippingAddress1%>" size="30" maxlength="30">*
      </td>
   </tr>
   .
   .
   .
</table>
```

newcctemplate.inc This template provides a standardized format for both the form presentation and error handling in all JSP templates that prompt customers for credit card/payment information. The form fields are organized in a table, and upon form submission, the input processors associated with the `newcctemplate.inc` template will validate the form to ensure that all required fields contain values. If errors are detected, the `newcctemplate.inc` template will be redisplayed, with an error message at the top and the invalid field labels shown in a red (as opposed to the original black) font. Previously entered correct information will still be displayed in the form.

The behavior described above is invoked on the `newcctemplate.inc` template by using the `getValidatedValue` JSP tag, as shown in Listing 6-16.

Listing 6-16 Use of the `getValidatedValue` JSP Tag on newcctemplate.inc

```
<table>
.
.
.
   <td width="27%"><webflow:getValidatedValue
      fieldName="<%=HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER%>"
      fieldValue="customerCreditCardHolder" fieldStatus="status"
      validColor="black" invalidColor="red"
      unspecifiedColor="black"
      fieldColor="fontColor" />
      <div class="tabletext">
         <font color=<%= fontColor %>><b>Name on card</b>
         </font>
      </div>
   </td>
   <td width="73%"> <input type="text"
      name="<%=HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER%>"
      value="<%=customerCreditCardHolder%>" size="30" maxlength="50">*
   </td>
.
.
.
</table>
```

newdemographictemplate.inc This template provides a standardized format for both the form presentation and error handling in all JSP templates that prompt customers for demographic information. The radio buttons are organized in a table, and upon form submission, the input processors associated with the `newdemographictemplate.inc` template will validate the form to ensure that all required fields contain values. If errors are detected, the `newdemographictemplate.inc` template will be redisplayed, with an error message at the top of the including page and the invalid field labels shown in a red (as opposed to the original black) font. Previously entered correct information will still be displayed in the form.

The behavior described above is invoked on the `newdemographictemplate.inc` template by using the `getValidatedValue` JSP tag, as shown in Listing 6-17.

Listing 6-17 Use of the `getValidatedValue` JSP Tag on newdemographictemplate.inc

```
<webflow:getValidatedValue fieldName="<%=HttpRequestConstants.CUSTOMER_GENDER%>"
   fieldDefaultValue="<%=(String)currentPropertyValue%>"
   fieldValue="genderValue" fieldStatus="status" validColor="black"
   invalidColor="red" unspecifiedColor="black" fieldColor="fontColor" />

   <td width="26%"><div class="tabletext"><b><font color=<%= fontColor %>>
      Gender*</font></b></div>
   </td>
   <td width="74%">

   <%// get the property values for Gender
   propertyBean.setPropertyName(GENDER);
   property = propertyBean.getPropertyObject();
   if(property == null || property.getRestrictedValues() == null)
   arr = new Object[0];
   else arr = property.getRestrictedValues().toArray();%>

   <ps:getRestrictedPropertyValues propertySetName="Demographics"
      propertySetType="USER" propertyName="<%= GENDER %>" id="arr"
      result="foobar" />

   <table width="100%" border="0" cellpadding="0"
      cellspacing="0"><es:forEachInArray id="valueObject" array="<%= arr %>"
         type="String">
      <tr>
         <td width="4%"><input type="radio" name="
         <%= HttpRequestConstants.CUSTOMER_GENDER %>" value="<%= valueObject %>"
         <% if ( valueObject.equals(genderValue) ) { %>CHECKED<% } %>></td>
         <td><%= valueObject %></td>
      </tr>
      </es:forEachInArray>
   </table>
```

Events

The `newuser.jsp` template presents a customer with two buttons, each of which is considered an event. These events trigger a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or pipeline component is invoked first. Table 6-2 describes the business logic these events invoke.

Table 6-2 `newuser.jsp` Events

Event	Webflow Response(s)
`button.cancel`	`GetCategoryIP` `GetTopCategories` Pipeline
`button.save`	`CustomerProfileIP` `CustomerProfile` Pipeline

Table 6-3 briefly describes each of the Pipeline components described Table 6-2.

Table 6-3 `newuser.jsp` Associated Pipelines

Pipeline	Description
`CustomerProfile`	Contains `EncryptedCreditCardPC` and `RegisterUserPC`, and is transactional.

newuser.jsp Form Fields

The primary purpose of the `newuser.jsp` template is to allow customers to enter their profile information using various HTML form fields. It is also used to pass needed information to the Webflow.

The form fields used in the `newuser.jsp` template, most of which are imported from other templates, and a description for each of these form fields are listed in Table 6-4.

Note: If a form field is imported from another template, it is indicated in the description. Form fields without import information are in the `newuser.jsp` template.

Table 6-4 newuser.jsp Form Fields

Parameter Name	Type	Description
“`event`”	Hidden	Indicates which event has been triggered. It is used by the Webflow to determine what happens next.
“`origin`”	Hidden	The name of the current page (`newuser.jsp`), used by the Webflow.
“`namespace`”	Hidden	The namespace for the JSP; `sampleapp_user` in this JSP.
`HttpRequestConstants.CUSTOMER_FIRST_NAME`	Textbox	The customer’s first name.
`HttpRequestConstants.CUSTOMER_MIDDLE_NAME`	Textbox	The customer’s middle initial.
`HttpRequestConstants.CUSTOMER_LAST_NAME`	Textbox	The customer’s last name.
`HttpRequestConstants.CUSTOMER_ADDRESS1`	Textbox	The first line in the customer’s street address.
`HttpRequestConstants.CUSTOMER_ADDRESS2`	Textbox	The second line in the customer’s street address.
`HttpRequestConstants.CUSTOMER_CITY`	Textbox	The city in the customer’s address.
`HttpRequestConstants.CUSTOMER_STATE`	Listbox	The state in the customer’s address. Imported from `states.inc`.
`HttpRequestConstants.CUSTOMER_ZIPCODE`	Textbox	The zip code in the customer’s address.
`HttpRequestConstants.CUSTOMER_COUNTRY`	Listbox	The country in the customer’s address. Imported from `countries.inc`.
`HttpRequestConstants.CUSTOMER_HOME_PHONE`	Textbox	The customer’s home phone number.
`HttpRequestConstants.CUSTOMER_BUSINESS_PHONE`	Textbox	The customer’s business phone number.

Table 6-4 newuser.jsp Form Fields (Continued)

Parameter Name	Type	Description
`HttpRequestConstants.CUSTOMER_EMAIL`	Textbox	The customer's e-mail address.
`HttpRequestConstants.CUSTOMER_EMAIL_OPT_IN`	Checkbox	Indicates that the customer wants to receive promotional items via e-mail.
`HttpRequestConstants.SAME_AS_ABOVE`	Checkbox	Indicates that the customer's shipping address is the same as the contact address. Imported from `newaddresstemplate.inc`.
`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1`	Textbox	The first line in the customer's shipping address. Imported from `newaddresstemplate.inc`.
`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS2`	Textbox	The second line in the customer's shipping address. Imported from `newaddresstemplate.inc`.
`HttpRequestConstants.CUSTOMER_SHIPPING_CITY`	Textbox	The city in the customer's shipping address. Imported from `newaddresstemplate.inc`.
`HttpRequestConstants.CUSTOMER_SHIPPING_STATE`	Listbox	The state in the customer's shipping address. Imported from `newaddresstemplate.inc`.
`HttpRequestConstants.CUSTOMER_SHIPPING_ZIPCODE`	Textbox	The zip/postal code in the customer's shipping address. Imported from `newaddresstemplate.inc`.
`HttpRequestConstants.CUSTOMER_SHIPPING_COUNTRY`	Listbox	The country in the customer's shipping address. Imported from `newaddresstemplate.inc`.
`HttpRequestConstants.CUSTOMER_GENDER`	Radio buttons	Identifies the customer as male or female. Imported from `newdemographictemplate.inc`.
`HttpRequestConstants.CUSTOMER_DATE_OF_BIRTH`	Textboxes	The customer's date of birth. Imported from `newdemographictemplate.inc`.
`HttpRequestConstants.CUSTOMER_OCCUPATION`	Radio buttons	The customer's job description. Imported from `newdemographictemplate.inc`.

Table 6-4 newuser.jsp Form Fields (Continued)

Parameter Name	Type	Description
`HttpRequestConstants.` `CUSTOMER_EMPLOYMENT_STATUS`	Radio buttons	Identifies if the customer has a job at the time of registration. Imported from `newdemographictemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_MARITAL_STATUS`	Radio buttons	Identifies the customer's marital status. Imported from `newdemographictemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_EDUCATION_LEVEL`	Radio buttons	Identifies how much formal education the customer has completed. Imported from `newdemographictemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_INCOME_RANGE`	Radio buttons	Identifies the customer's yearly income. Imported from `newdemographictemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_QUALITY`	Radio buttons	Ranks customer from beginner to expert in using your product. Imported from `newdemographictemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_TYPE`	Listbox	The type of the customer's credit card. Imported from `newcctemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_HOLDER`	Textbox	The name on the credit card. Imported from `newcctemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_NUMBER`	Textbox	The number of the customer's credit card. Imported from `newcctemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_MONTH`	Listbox	The month of the customer's credit card expiration date. Imported from `newcctemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_YEAR`	Listbox	The year of the customer's credit card expiration date. Imported from `newcctemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_ADDRESS1`	Textbox	The first line in the customer's billing address. Imported from `newcctemplate.inc`.
`HttpRequestConstants.` `CUSTOMER_CREDITCARD_` `ADDRESS2`	Textbox	The second line in the customer's billing address. Imported from `newcctemplate.inc`.

Table 6-4 newuser.jsp Form Fields (Continued)

Parameter Name	Type	Description
`HttpRequestConstants.CUSTOMER_CREDITCARD_CITY`	Textbox	The city in the customer's billing address. Imported from `newcctemplate.inc`.
`HttpRequestConstants.CUSTOMER_CREDITCARD_STATE`	Listbox	The state in the customer's billing address. Imported from `newcctemplate.inc`.
`HttpRequestConstants.CUSTOMER_CREDITCARD_ZIPCODE`	Textbox	The zip/postal code in the customer's billing address. Imported from `newcctemplate.inc`.
`HttpRequestConstants.CUSTOMER_CREDITCARD_COUNTRY`	Listbox	The country in the customer's billing address. Imported from `newcctemplate.inc`.
`HttpRequestConstants.USER_NAME`	Textbox	An identity chosen by the customer for login.
`HttpRequestConstants.PASSWORD`	Password	A password chosen by the customer for login.
`HttpRequestConstants.CONFIRM_PASSWORD`	Password	Confirmation of the password chosen by the customer for login.

Note: Parameters that are literals in the JSP code are shown in quotes, while non-literals will require scriptlet syntax (such as `<%= HttpRequestConstants.USER_NAME %>`) for use in the JSP.

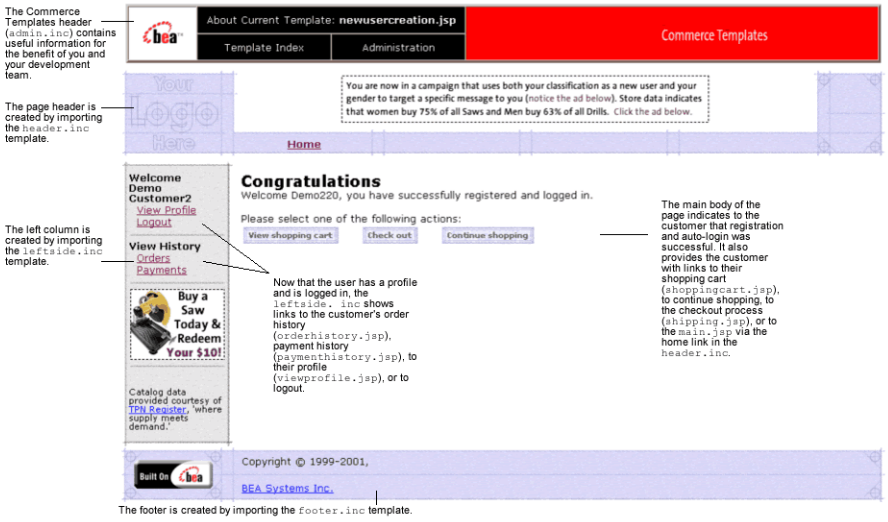
newusercreation.jsp

The `newusercreation.jsp` template informs a customer who has just created a new user profile that they have been logged in and that registration was successful. It also provides the customer with the opportunity to return to their shopping experience through several navigation options.

Description

Figure 6-24 shows an example of a Web page formatted with `newusercreation.jsp`.

Figure 6-24 Web Page Formatted with `newusercreation.jsp`


The Commerce Templates header (`admin.inc`) contains useful information for the benefit of you and your development team.

The page header is created by importing the `header.inc` template.

The left column is created by importing the `leftside.inc` template.

Now that the user has a profile and is logged in, the `leftside. inc` shows links to the customer's order history (`orderhistory.jsp`), payment history (`paymenthistory.jsp`), to their profile (`viewprofile.jsp`), or to logout.

The main body of the page indicates to the customer that registration and auto-login was successful. It also provides the customer with links to their shopping cart (`shoppingcart.jsp`), to continue shopping, to the checkout process (`shipping.jsp`), or to the `main.jsp` via the home link in the `header.inc`.

The footer is created by importing the `footer.inc` template.

About Current Template: **newusercreation.jsp**

You are now in a campaign that uses both your classification as a new user and your gender to target a specific message to you (notice the ad below). Store data indicates that women buy 75% of all Saws and Men buy 63% of all Drills. Click the ad below.

Congratulations

Welcome Demo220, you have successfully registered and logged in.

Please select one of the following actions:

Welcome Demo Customer2

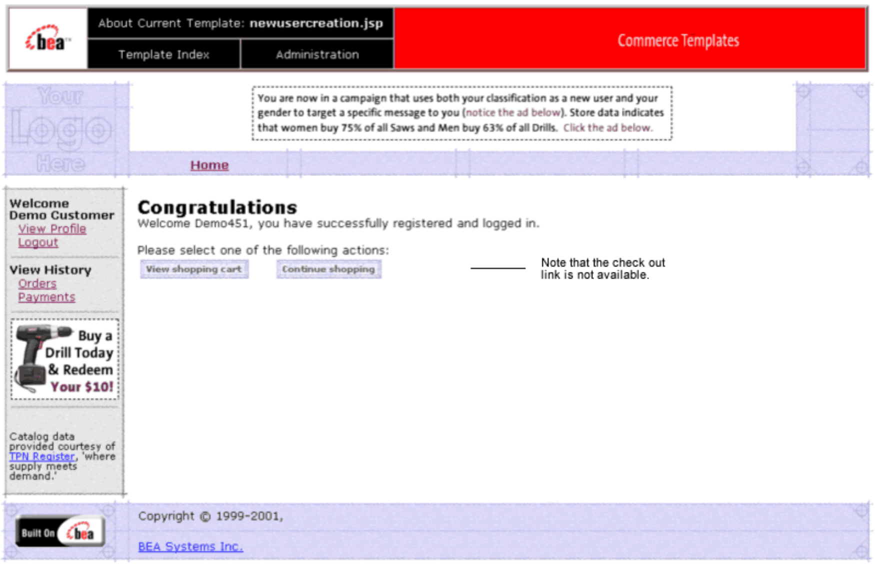
Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Copyright © 1999-2001,

BEA Systems Inc.

The option to proceed to checkout is only provided on the `newusercreation.jsp` template if there are items in the customer's shopping cart. Otherwise, the `newusercreation.jsp` template will leave out this option as shown in Figure 6-25.

Figure 6-25 Web Page Formatted with `newusercreation.jsp` When Shopping Cart is Empty



About Current Template: **newusercreation.jsp**

Template Index | Administration

Commerce Templates

You are now in a campaign that uses both your classification as a new user and your gender to target a specific message to you (notice the ad below). Store data indicates that women buy 75% of all Saws and Men buy 63% of all Drills. Click the ad below.

Home

Welcome Demo Customer
View Profile
Logout

View History
Orders
Payments

Buy a Drill Today & Redeem Your $10!

Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Congratulations

Welcome Demo451, you have successfully registered and logged in.

Please select one of the following actions:

View shopping cart | Continue shopping

Note that the check out link is not available.

Copyright © 1999-2001,

BEA Systems Inc.

How newusercreation.jsp Works

Customers arrive at the `newusercreation.jsp` template when they have successfully created a new user profile and the auto-login—using Java Authentication and Authorization Service (JAAS)—has completed. If the customer creates a new profile, but the auto-login does not complete successfully, the customer is routed back to the `login.jsp` template and will not see the `newusercreation.jsp` template. After manual login, the customer is routed to the `main.jsp` template.

Note: If a customer had created a profile on a previous visit and logged in using the `login.jsp` template, the customer would simply be taken to the protected page the customer was trying to access.

From the `newusercreation.jsp` template, the customer can return to their shopping cart (`shoppingcart.jsp`), continue shopping, continue to the checkout process (`shipping.jsp`), view their order history (`orderhistory.jsp`), view their profile (`viewprofile.jsp`), view their payment history (`paymenthistory.jsp`), logout, or return to the main catalog page (`main.jsp`).

Note: The option to proceed to checkout is only provided on the `newusercreation.jsp` template if there are items in the customer's shopping cart.

This template is part of the `sampleapp_user` namespace in the Webflow.

Events

Every time a customer clicks a button to view more detail about an order, it is considered an event. Each event triggers a particular response in the default Webflow that allows them to continue. While this response can be to load another JSP, it is usually the case that an input processor and/or Pipeline is invoked first. Table 6-5 provides information about these events and the business logic they invoke.

Table 6-5 `newusercreation.jsp` Events

Event	Webflow Response(s)
`link.shoppingcart`	`InitShoppingCartIP`
`button.checkout`	`InitShippingMethodListIP`
`link.home`	`GetTopCategoriesIP` `GetTopCategories` Pipeline

newuserforward.jsp

The `newuserforward.jsp` template directs unregistered users to the `newuser.jsp` when that user clicks an ad placeholder that contains a static URI. This is necessary because dynamic URIs are not supported in placeholders. The `newuserforward.jsp` template then forwards the user to `newuser.jsp`. Additionally, the `newuserforward.jsp` bridges the transition from a non-secure to a secure connection (`.http` to `.https`).

Description

This template does not render a Web page or any other visible output. Its code is shown in Listing 6-18.

Listing 6-18 `newuserforward.jsp` Code

```
<% String s = com.bea.p13n.appflow.webflow.WebflowJSPHelper.
   createWebflowURL(pageContext, "sampleapp_main", "login.jsp",
   "button.createUser", true); %>

<% response.sendRedirect(s) ; %>
```

Table 6-6 shows the key template components.

Table 6-6 Template Components

Type of Component	Components
Included templates	None
Tag libraries	None
Imported Java packages	None

How newuserforward.jsp Works

The page prior to `newuserforward.jsp` can be any page that an anonymous user can access. However, this template is only needed if an unregistered user clicks the ad placeholder that prompts them to register. The static URI in the placeholder accesses the `newuserforward.jsp` which then forwards the user to the `newuser.jsp` template.

This template is part of the `sampleapp_main` namespace in the Webflow.

Events

The `newuserforward.jsp` template has one event, which triggers a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 6-7 provides information about this event and the business logic it invokes.

Table 6-7 newuserforward.jsp Events

Event	Webflow Response(s)
`button.createUser`	`newuser.jsp`

usercreationforward.jsp

The `usercreationforward.jsp` template forwards new users to the `newusercreation.jsp` template after the registration and auto-login process using JAAS is completed by the Webflow. Once the user is created, the request must be flushed; the `usercreationforward.jsp` template allows that to happen.

Description

This template does not render a Web page or any other visible output. Its code is shown in Listing 6-19.

Listing 6-19 `usercreationforward.jsp` Code

```
<% String s = WebflowJSPHelper.createWebflowURL(pageContext,
   "sampleapp_user", "usercreationforward.jsp",
   "forward.usercreation", true); %>

<% response.sendRedirect(s) ; %>
```

The `usercreationforward.jsp` template uses Java classes in the `com.bea.p13n.appflow.webflow.WebflowJSPHelper` package and must include this import statement:

```
<%@ page import="com.bea.p13n.appflow.webflow.
   WebflowJSPHelper*" %>
```

How usercreationforward.jsp Works

The page prior to `usercreationforward.jsp` is the `newuser.jsp` template. When new users save their profiles, they are auto-logged in using JAAS and if the login is successful, because the old request must be flushed, the `usercreationforward.jsp` is needed to redirect the user to the `newusercreation.jsp` template.

This template is part of the `sampleapp_user` namespace in the Webflow.

Events

The `usercreationforward.jsp` template has one event. This event triggers a particular response in the default Webflow that allows customers to continue. While this response can be to load another JSP, it is usually the case that an input processor or Pipeline is invoked first. Table 6-8 provides information about this event and the business logic it invokes.

Table 6-8 usercreationforward.jsp Events

Event	Webflow Response(s)
`forward.usercreation`	`newusercreation.jsp`

Webflow Components Used in Visitor Self-Registration

The templates described in "Implementing Customer Profile JSPs" on page 6-33 use Webflow components called input processors and pipelines to execute much of the necessary business logic to enable visitor self-registration. This section describes the key Webflow components implemented.

This section includes information on the following subjects:

- Input Processors
- Pipeline Components

Note: See "Setting Up Portal Navigation" on page 9-1 for information about using, creating, or modifying a Webflow and using input processors and pipeline components.

Input Processors

The following input processors represent Java classes invoked to carry out more complex visitor regiatration tasks when invoked by the Webflow mechanism. These processors are:

- CustomerProfileIP
- LoginCustomerIP

For more information on input processors, see "Types of Nodes" on page 9-3.

CustomerProfileIP

`CustomerProfileIP` (all input processor names end in the letters "IP") processes the input of `newuser.jsp` and allows the customer to store their profile. It also creates and places a `CustomerValue` object into the Pipeline Processor session.

Class Invoked	`examples.wlcs.sampleapp.customer.webflow.CustomerProfileIP`

Required `HTTPServletRequest` Parameters (Personal Information)	`HttpRequestConstants.CUSTOMER_FIRST_NAME`
	`HttpRequestConstants.CUSTOMER_MIDDLE_NAME`
	`HttpRequestConstants.CUSTOMER_LAST_NAME`
	`HttpRequestConstants.CUSTOMER_ADDRESS1`
	`HttpRequestConstants.CUSTOMER_ADDRESS2`
	`HttpRequestConstants.CUSTOMER_CITY`
	`HttpRequestConstants.CUSTOMER_STATE`
	`HttpRequestConstants.CUSTOMER_ZIPCODE`
	`HttpRequestConstants.CUSTOMER_COUNTRY`
	`HttpRequestConstants.CUSTOMER_HOME_PHONE`
	`HttpRequestConstants.CUSTOMER_BUSINESS_PHONE`
	`HttpRequestConstants.CUSTOMER_EMAIL`
	`HttpRequestConstants.CUSTOMER_EMAIL_OPT_IN`
	(**code location**: `newuser.jsp` template.)

Required **`HTTPServletRequest`** **Parameters (Demographic Information)**	`HttpRequestConstants.CUSTOMER_INCOME_RANGE`
	`HttpRequestConstants.CUSTOMER_EDUCATION_LEVEL`
	`HttpRequestConstants.CUSTOMER_DATE_OF_BIRTH`
	`HttpRequestConstants.CUSTOMER_GENDER`
	`HttpRequestConstants.CUSTOMER_OCCUPATION`
	`HttpRequestConstants.CUSTOMER_MARITAL_STATUS`
	`HttpRequestConstants.CUSTOMER_EMPLOYMENT_STATUS`
	`HttpRequestConstants.CUSTOMER_QUALITY`
	(**code location**: `newdemographictemplate.inc` template.)

Required **`HTTPServletRequest`** **Parameters (Shipping Information)**	`HttpRequestConstants.SAME_AS_ABOVE` (**code location**: `newuser.jsp` template.)

	`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1`
	`HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS2`
	`HttpRequestConstants.CUSTOMER_SHIPPING_CITY`
	`HttpRequestConstants.CUSTOMER_SHIPPING_STATE`
	`HttpRequestConstants.CUSTOMER_SHIPPING_ZIPCODE`
	`HttpRequestConstants.CUSTOMER_SHIPPING_COUNTRY`
	`HttpRequestConstants.DEFAULT_SHIPPING_ADDRESS`
	(**code location**: `newaddresstemplate.inc` template.)

`HTTPServletRequest` **Parameters (Payment Information)**	`HttpRequestConstants.CUSTOMER_CREDITCARD_TYPE`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_NUMBER`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_MONTH`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_YEAR`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS1`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS2`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_CITY`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_STATE`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_ZIPCODE`
	`HttpRequestConstants.CUSTOMER_CREDITCARD_COUNTRY`
	(**code location**: `newcctemplate.inc` template.)

Required `HTTPServletRequest` Parameters (Account Information)	`HttpRequestConstants.USER_NAME`
	`HttpRequestConstants.PASSWORD`
	`HttpRequestConstants.CONFIRM_PASSWORD`
	(**code location**: `newuser.jsp` template.)

Required Pipeline Session properties	None
Updated Pipeline Session properties	PipelineSession`Constants.CUSTOMER` PipelineSession`Constants.PASSWORD` `PipelineSessionConstants.CREDITCARD_KEY` (only if customer provides a credit card update).
Removed Pipeline Session properties	None
Validation	Checks that the required fields contain values and checks that the credit card number is not less than 16 digits (15 digits for AMEX type). Also checks that the password and confirm password fields contain matching values.
Exceptions	`InvalidInputException`, thrown when required fields are empty or credit card number is less than 16 digits (15 digits for AMEX type).

LoginCustomerIP

`LoginCustomerIP` processes the input of `login.jsp` and allows the customer to access the secure pages of the site. It also creates and places a `CustomerValue` object into the Pipeline Processor session.

Class Invoked	`examples.wlcs.sampleapp.customer.webflow.`
	`   LoginCustomerIP`

Required `HTTPServletRequest` Parameters	None

Required Pipeline Session properties	`PipelineSessionConstants.CUSTOMER`
	`PipelineSessionConstants.PASSWORD`
	`PipelineSessionConstants.CREDITCARD_KEY` (only if the customer provides a credit card update).

Updated Pipeline Session properties	None
Removed Pipeline Session properties	`PipelineSessionConstants.PASSWORD`
Validation	Verifies that the username and password are correct.
Exceptions	`InvalidInputException`, thrown if either the username or password is invalid. `ProcessingException`, thrown if the username is invalid or cannot get authentication.

Pipeline Components

This section provides a brief description of each pipeline component associated with the Customer Login and Registration Services JSP template(s). These Pipelines are processor nodes a Webflow invokes to initiate the execution of specific tasks related to visitor registration.

Note: Some pipeline components extend other, base pipeline components. For more information on the base classes, see the *Javadoc*.

For more information on pipeline components, see "Types of Nodes" on page 9-3.

This section contains information on these pipeline components:

- RegisterUserPC
- EncryptCreditCardPC

RegisterUserPC

`RegisterUserPC` (all pipeline component names end in the letters "PC") retrieves the `CustomerValue` object and password from the Pipeline Processor session and creates a `CUSTOMER` attribute.

Class Name	examples.wlcs.sampleapp.customer.pipeline. RegisterUserPC
Contained in	CustomerProfile Pipeline
Required Pipeline Session Properties	PipelineSessionConstants.CUSTOMER PipelineSessionConstants.PASSWORD
Updated Pipeline Session Properties	None
Removed Pipeline Session Properties	PipelineSessionConstants.PASSWORD
Type	Java class
JNDI Name	None
Exceptions	PipelineException, thrown when the pipeline component cannot create the user.

EncryptCreditCardPC

EncryptCreditCardPC uses the CREDITCARD_KEY object to retrieve a customer credit card, encrypts the credit card number, and then adds the modified credit card back to the PipelineSession CustomerValue attribute.

Class Name	examples.wlcs.sampleapp.customer.pipeline. EncryptCreditCardPC
Description	
Contained in	CustomerProfile Pipeline
Required Pipeline Session Properties	PipelineSessionConstants.CREDITCARD_KEY
Updated Pipeline Session Properties	PipelineSessionConstants.CUSTOMER

Removed Pipeline Session Properties	PipelineSession`Constants.CREDITCARD_KEY`
Type	Java class
JNDI Name	None
Exceptions	`PipelineException`, thrown when the pipeline component cannot find the user in the Pipeline Processor session or the `creditcard_key` is invalid or the encryption did not complete successfully.

CHAPTER

7 Adding Security to a Portal

A Web server authenticates users and determines which resources within the server users can create, access, or modify. To do this, the Web server uses a security realm. When a user attempts to access a particular resource, the server tries to authenticate and authorize that user by checking the access control list (ACL) and permissions that are assigned to the user within the realm. You can set up multiple security realms, but each instance of a Web server can use only one realm. The server uses the same security realm for your Web site developers and for your visitors.

This section contains information on the following subjects:

Implementing Portal Security

If you choose to use the basic implementation of the RDBMS security realm supplied by BEA, it will be available when you install WebLogic Portal. No further configuration is necessary.

Note: The WebLogic Portal RDBMS Realm is a different implementation than the WebLogic Server RDBMS Realm. The WebLogic Server RDBMS Realm is for testing and is not suitable for use in a production environment. WebLogic Portal's RDBMS realm is shipped in the `BEA_HOME/weblogic700/portal/lib/p13n_system.jar` file and the implementing class is in `com.bea.p13n.security.realm.RDBMSRealmas` configured with the `config.xml` file entry: `RealmClassName="com.bea.p13n.security.realm.RDBMSRealm`.

Integrating with an LDAP Security Realm

If you don't want to use the basic RDBMS security realm, one popular alternative is to use a lightweight directory access protocol (LDAP) server as your security realm. This section describes how to integrate an LDAP server with WebLogic Portal. This section includes the following topics:

- Supported LDAP Servers
- Integrating an LDAP Security Realm

Supported LDAP Servers

WebLogic Portal supports these LDAP servers:

- Netscape Directory Server
- Microsoft Site Server

- Novell Directory Services
- Open LDAP Directory Services

You can find templates for each of these services in "Supported Server Templates" on page 7-7.

Integrating an LDAP Security Realm

This section shows how WebLogic Portal integrates with a third-party LDAP security realm; security realms are the method used by WebLogic Portal to authenticate users. While WebLogic Portal provides a default user store based on a RDBMS, this can be replaced with an LDAP realm, which uses an LDAP server for security information.

Configuring the LDAP Server for Integration

Configuring the LDAP security realm involves defining attributes that enable the LDAP Security realm in WebLogic Server to communicate with the LDAP server and the attributes that describe how users and groups are stored in the LDAP directory. The LDAP tree and schema is different for every LDAP server.

In "Supported Server Templates" on page 7-7, you can find templates for the supported LDAP servers. These templates specify default configuration information used to represent users and groups in each of the supported LDAP servers. You choose the template that corresponds to the LDAP server you want to use and then fill in the attributes described in Table 7-1.

Note: In LDAP V1, you can configure these attributes from the LDAP Realm Create screen, on the tab specified in Table 7-1.

Table 7-1 LDAP Realm Configurable Attributes

Attribute	Description
`User DN`	A list of attributes and their values that, when combined with the attributes in the User Name Attribute attribute, uniquely identifies an LDAP User. Configure this attribute on the Users tab in the LDAP Realm Create screen.

Table 7-1 LDAP Realm Configurable Attributes

Attribute	Description
`Group DN`	List of attributes and values that, combined with the Group Name Attribute attribute, uniquely identifies a Group in the LDAP directory. Configure this attribute on the Groups tab in the LDAP Realm Create screen.
`Principal`	DN of the LDAP User that WebLogic Server uses to connect to the LDAP server. This user must be able to list LDAP Users and Groups. Configure this attribute on LDAP Realm tab in the LDAP Realm Create screen.
`Credential`	Password that authenticates the LDAP User defined in the Principal attribute. Configure this attribute on LDAP Realm tab in the LDAP Realm Create screen.

For instructions for configuring the LDAP security realm, please refer to “Configuring the LDAP Security Realm” in the *WebLogic Server Administration Guide* at http://e-docs.bea.com/wls/docs61/adminguide/cnfgsec.html#1052314.

In addition to defining attributes that enable communication with the LDAP server, you will have to define certain groups in your LDAP server to correspond to the security role mappings that have been set for portal delegated administration. To set these groups up you should:

1. Read “Administering Users and Groups” at http://e-docs.bea.com/wlp/docs70/admin/usrgrp.htm, especially the section “Creating Administrative Users.”

2. To set up WebLogic Server System Administrators it is recommended that you use the provided `fileRealm.properties` file and use the WebLogic administration console to administer the `weblogic` and `system` users. They are members of the `Administrators` group, which is a subgroup of the special WebLogic Server groups `Operators`, `Deployers`, and `Monitors`.

3. Set up one or more WebLogic Portal System Administrators.

   a. Create the group `SystemAdministrator` in your LDAP server.

   b. Add the desired users to this group in your LDAP server.

4. Set up the groups required for delegated administration of portals. All users that will be designated at Portal Administrators or Group Portal Administrators must be added to the proper groups before using the WebLogic Portal Administration Tools to specify them as Portal Administrators or Group Portal Administrators.

   a. Create the groups `AdminEligible` and `DelegatedAdministrator` in your LDAP server.

   b. Add the desired users to both of these groups.

   c. These users are now candidates for designation as Portal Administrators or Group Portal Administrators. You would have to use the WebLogic Portal Administration Tools to persist the data required to enable them as administrators.

   d. When removing all delegated administration capabilities for a user it is recommended that they first be removed from the `AdminEligible` and `DelegatedAdministrator` group in your LDAP server. However, you may remove them from the groups after revoking their privileges in the WebLogic Portal Administration Tools if you prefer.

5. Create groups in your LDAP server that will be associated with your group portals.

   It is not necessary to add your Portal Administrators and Group Portal Administrators to these groups, but it would be recommended to do this so that your administrators may log into the group portals to verify their work.

Configuring LDAP-based Security Realms for WebLogic Server and Portal 7.0

Perform the following steps for configuring LDAP-based security realms for WebLogic Server 7.0 and WebLogic Portal 7.0.

WebLogic Server Setup

1. Define the LDAP security realm – See the “Compatibility Security” section of the online help for the WebLogic Server Administration Console.

2. Configure the caching realm – See the “Compatibility Security” section of the online help for the WebLogic Server Administration Console.

3. Configure the default caching realm.

   a. Open the WebLogic Server Administration Console.
   b. Click the **Compatibility Security** node in the left pane.
   c. Click the **FileRealm** tab.
   d. Select the Caching Realm you just created and click **Apply**.
4. Define a `system` user (or another user as the WebLogic administrator), and define `guest/guest` realm in your LDAP directory. See the "Compatibility Security" section of the online help for the WebLogic Server Administration Console for more information.
5. Define an `Administrators` group and add your system user to it (or another user as the WebLogic administrator) for your realm in your LDAP directory. See the "Compatibility Security" section of the online help for the WebLogic Server Administration Console for more information.
6. Test WebLogic Server.
   a. Shut down the server.
   b. Re-start the server.
   c. Log in to the WebLogic Administration Console as the WebLogic administrator you created.

WebLogic Portal Setup

1. Define the following groups and users for your realm in your LDAP directory:

Group	**User**
Administrators	system
SystemAdministrator	administrator
AdminEligible	
DelegatedAdministrator	

2. Add a new Group Portal.
   a. Create the group in LDAP.

b. Create a new user to administer the Portal.

c. Add the user to the `AdminEligible` group, `DelegatedAdministrator` group, and the new group you just created for the Portal.

d. Use the following URL to log in to WebLogic Portal Administration Tools as a Portal System Administrator: `http://localhost:7501/portalAppTools/index.jsp`.

e. Create a new Group Portal. Specify the new group and new administrator.

3. Add a new Delegated Administrator.

a. Create a new user to administer the Portal.

b. Add the user to the `AdminEligible` group, `DelegatedAdministrator` group, and the group for the Portal.

c. Use the following URL to log in to WebLogic Portal Administration Tools as the Group Portal Administrator: `http://localhost:7501/portalAppTools/index.jsp`.

d. Create a new Group Portal Administrator. Specify the new group and new administrator.

Supported Server Templates

Listing 7-1 through Listing 7-4 are the templates you can use to configure supported LDAP servers. You can copy these templates from here directly into the `config.xml` file for your application.

Warning: Each line in the following code examples must appear on a single line. The examples shown below have been formatted to fit the margins of this document and some lines have been broken to facilitate that formatting. If you paste this text into the `config.xml` file, be sure to concatenate the lines that are broken so that they appear on a single line in your code.

Listing 7-1 Default Netscape Customer Security Realm Template

```
<CustomRealm
   Name="defaultLDAPRealmForNetscapeDirectoryServer"
   RealmClassName="weblogic.security.ldaprealmv2.LDAPRealm"
   Password="*secret*"
   ConfigurationData="server.host=ldapserver.example.com;server.principal=uid=
      admin,
   ou=Administrators, ou=TopologyManagement, o=NetscapeRoot;user.dn=ou=people,
   o=beasys.com;user.filter=(&amp;(uid=%u)(objectclass=person));group.dn=
      ou=groups,
   o=beasys.com;group.filter=(&amp;(cn=%g)(objectclass=groupofuniquenames));
      membership.filter=(&amp;(uniquemember=%M)(objectclass=
      groupofuniquenames));"
   Notes="This is provided as an example. Before enabling this Realm, you must
   edit the configuration parameters as appropriate for your environment."
/>
```

Listing 7-2 Default Microsoft Customer Security Realm Template

```
<CustomRealm
   Name="defaultLDAPRealmForMicrosoftSiteServer"
   RealmClassName="weblogic.security.ldaprealmv2.LDAPRealm"
   Password="*secret*"
   ConfigurationData="server.host=ldapserver.example.com;server.principal=cn=
      Administrator,
   ou=Members, o=ExampleMembershipDir;user.dn=ou=Members,
   o=ExampleMembershipDir;user.filter=(&amp;(cn=%u)(objectclass=member));
      group.dn=ou=Groups,
   o=ExampleMembershipDir;group.filter=(&amp;(cn=%g)(objectclass=mgroup));
      membership.scope.depth=1;microsoft.membership.scope=sub;membership.
      filter=(|(&amp;(memberobject=%M)(objectclass=memberof))
      (&amp;(groupobject=%M)(objectclass=groupmemberof)));"
   Notes="This is provided as an example. Before enabling this Realm,
      you must edit the configuration parameters as appropriate for your
      environment."
/>
```

Listing 7-3 Default Novell Customer Security Realm Template

```
<CustomRealm
   Name="defaultLDAPRealmForNovellDirectoryServices"
   RealmClassName="weblogic.security.ldaprealmv2.LDAPRealm"
   Password="*secret*"
   ConfigurationData="server.host=ldapserver.example.com;server.principal=cn=
      admin,
   o=example.com;user.dn=ou=people,
   o=example.com;user.filter=(&amp;(cn=%u)(objectclass=person));group.dn=ou=
      groups,
   o=example.com;group.filter=(&amp;(cn=%g)(objectclass=groupofuniquenames));
      membership.filter=(&amp;(member=%M)(objectclass=groupofuniquenames));"
   Notes="This is provided as an example. Before enabling this Realm, you must
      edit the configuration parameters as appropriate for your environment."
/>
```

Listing 7-4 Default OpenLDAP Security Realm Template

```
<CustomRealm
   Name="defaultLDAPRealmForOpenLDAPDirectoryServices"
   RealmClassName="weblogic.security.ldaprealmv2.LDAPRealm"
   Password="*secret*"
   ConfigurationData="server.host=ldapserver.example.com;server.principal=cn=
      Manager,
   dc=example, dc=com;user.dn=ou=people, dc=example,
   dc=com;user.filter=(&amp;(uid=%u)(objectclass=person));group.dn=ou=groups,
   dc=example,
   c=com;group.filter=(&amp;(cn=%g)(objectclass=groupofuniquenames));membership.
      filter=(&amp;(uniquemember=%M)(objectclass=groupofuniquenames));"
   Notes="This is provided as an example. Before enabling this Realm, you must
      edit the configuration parameters as appropriate for your environment."
/>
```

Using Wildcards for User Lookup in an LDAP Realm

This change has been implemented to allow true wildcard searches of the LDAP server when using the `weblogic.security.ldaprealmv2.LDAPRealm`.

The original implementation used a `getUsers()` method that did a search for `uid=*` and returned all users. The enumeration of all users was iterated through to find matches to the search string input in the WebLogic Portal Administration Tools.

The new implementation does a true wildcard search by using a new `getUsers(String searchString, int maxResults)` method that is implemented by extending the original LDAPRealm with a new `com.bea.p13n.security.realm.PortalLDAPRealm`.

The original WebLogic Portal Administration Tools used the `CachingRealm.getUsers()` method to conduct a search that returned all users in the LDAPRealm. They used a search filter that looked like this:

```
(&(uid=*)(objectclass=person))
```

This search result (all users in the realm) was iterated through and users with names that matched a wildcard search expression in the request were put into a list and were displayed as links on the page.

The new `PortalLDAPRealm.getUsers(String searchString, int maxResults)` method allows a true wildcard search of the LDAP server, using a search filter that looks like this:

```
(&(uid=a*)(objectclass=person))
```

Your search String is used in the `(uid=...)` expression in the filter. To use this patch, the <CustomRealm> is configured to use a RealmClassName of `com.bea.p13n.security.realm.PortalLDAPRealm` instead of `weblogic.security.ldaprealmv2.LDAPRealm`.

Warning: A wildcard LDAP search can be a slow operation for a large LDAP server. This is not a characteristic of WLS or Portal. This is a characteristic of LDAP and the libraries used to search it. LDAP servers populated with millions of users can be very slow for wildcard searches. To verify expected response times for wildcard searches with your system you could use an ldapsearch utility to measure a typical wildcard search, like `"a*"`. For example, for the iPlanet Directory Server you could type something like this on one line:

```
ldapsearch -b "ou=People, dc=beasys, dc=com"
-D "uid=admin, ou=Administrators, ou=TopologyManagement,
o=NetscapeRoot"
-h myserver.mydomain.com -p 389 -s sub -w password -z100
"(&(uid=a*)(objectclass=person))"
```

A fast alternative is to use a specific user id as the search string. With a user population of millions of users, an LDAP search for `uid=a*` will typically be timed out by your LDAP server while a search for the specific `uid=administrator` will be very fast. The new PortalLDAPRealm, with its `getUsers(String searchString, int maxResults)` method allows you to either search with a wildcard (slow) or with an exact username (fast).

Recommendations for Using this Patch with a Large LDAP Server

- You may need to set `GroupMembershipCacheTTL=0` to disable the group membership cache. This will speed up calls to `Group.isMember()` because this causes an LDAP search to determine group membership for the specific user without populating the group membership cache for all the other members of the group.

  Use the WLS console: **CompatibilitySecurity > CachingRealm > yourCachingRealm > Configuration > Groups > GroupMembershipCacheTTL** (set it to 0).

  The disabling of this cache is allowed by this patch because the changes to the `weblogic.security.ldaprealmvw.LDAPRealm` include the changes made in in CR090409 to allow disabling of the GroupMembershipCache.

- Configure your LDAP server and your `user.dn/group.dn` so that your `user.dn` and `group.dn` contain only groups and users that are required by your Portal application. This will greatly speed up LDAP operations if you have a large number of users/groups that are used by your company that are never needed in your Portal application. Remember that the LDAPRealm defaults to subtree scope searches (this can be changed in `config.xml`).

- Wildcard searches are permitted now that this patch has been made but they should not be used if your LDAP server is too big to handle them. Test your system by using the ldapsearch command line utility or the LDAPSearch utility for the Netscape SDK 4.1 (WebLogic Server 7.0 ships with the Netscape SDK 4.1 and it is what is used by the LDAPRealm). You can prevent your administrators from using wildcard searches by modifying the WebLogic Portal Administration Tools JSP by deleting the wildcard buttons and checking for `"*"` in the search string that is input by the administrator.

- Do not use the WebLogic Portal Administration Tools to browse groups with very large LDAP repositories. The Group tools search for all members of all groups and display a tree of the group hierarchy. For a large LDAP server this is a slow operation. If you must set properties for a group and your LDAP server is too big for the Group tools to work, then it is recommended that you programatically set the property for the group using the JSP tags or API.

Adding User Profile Information to LDAP Users

The LdapRealm security realm and the LdapPropertyManager unified user profile (UUP) for retrieving user properties from LDAP are independent of each other. They do not share configuration information and there is no requirement to use either one in conjunction with the other. A security realm has nothing to do with a user profile. A security realm provides user/password data, user/group associations, and group/group associations. A user profile provides user and group properties. A password is not a property.

For information on setting up UUP and retrieving user profile properties for LDAP users, see Chapter 6, “Implementing User Profiles.”

Switching to a WebLogic 7.0 Security Framework Security Realm

WebLogic Portal 7.0 Service Pack 4 adds support for the WebLogic Security Framework introduced in WebLogic Server 7.0. This enables the replacement of WebLogic Server 6.x security realms with WebLogic Server 7.0 realms. Prior to 7.0 SP4 WebLogic Portal 7.0 users were restricted to using the Compatibility Realm.

Even if you are using a new WebLogic Server 7.0 realm, it is possible to continue using users and groups from a Compatibility RDBMS Realm by using the RDBMSAuthenticator. Alternatively, you can use the LDAP data store embedded with WebLogic Server, or a commercial LDAP provider such as Iplanet, or a custom authentication provider that you have developed.

Setup depends on which authentication provider you use:

- Upgrading a Portal from Compatibility Security to WebLogic Server 7.0 Security With RDBMS – This is the recommended option if you already have users and groups in an existing RDBMS security realm, such as the one that shipped with WebLogic 7.0 SP2 and earlier.
- Upgrading a Portal from Compatibility Security to WebLogic Server 7.0 Security with Embedded LDAP – This is the recommended option if you have other applications that use the embedded WebLogic Server LDAP data store and

you would like to share user and group information with a WebLogic Portal application.

- Upgrading a Portal from Compatibility Security to WebLogic Server 7.0 Security with a Commercial LDAP Provider – This is the recommended option if you would like to capitalize on a third-party LDAP provider that you already use.

As of WebLogic Portal 7.0 Service Pack 4, you can also use multiple authentication providers. However, there are some limitations. See "Multiple Authentication Providers Support in WebLogic Portal 7.0 SP4" on page 7-22.

In WebLogic 7.0 the security Mbeans are in binary form in the `<domain>/userConfig` directory. During these exercises you can check your security Mbeans by using the WebLogicMBeanDumper:

```
java weblogic.management.commo.WebLogicMBeanDumper -
includeDefaults -name Security:* mbean_security.out
```

Upgrading a Portal from Compatibility Security to WebLogic Server 7.0 Security With RDBMS

This is the recommended option if you already have users and groups in an existing RDBMS security realm, such as the one that shipped with WebLogic 7.0 SP2 and earlier. By retaining user and group information in your RDBMS data store, you can avoid migrating it to an LDAP data store. The RDBMS security realm provides a high-performance, robust solution.

These instructions explain how to use a WebLogic Server 7.0 realm with users and groups from a WebLogic Server 6.x RDBMS Compatibility realm. Users and groups will be stored in a previously used Compatibility Realm.

1. Copy `rdbmsAtnProvider.jar` from the `weblogic700/portal/lib` directory to the `weblogic700/server/lib/mbeantypes` directory. You must use a version of this JAR that was built for the version of WebLogic Server that you are using. For example, you cannot use a 7.0 SP2 version of this JAR with 7.0 SP4. If you use an incompatible `rdbmsAtnProvider.jar` then you may see a ClassCastException in the WebLogic Server administration console when you try to create the authentication provider.

2. In the WebLogic Server administration console, navigate to the Authentication Providers page under **<your domain> > Security > Realms > myrealm > Providers > Authentication Providers**.
3. Click **Configure a new RDBMSAuthenticator**.
4. Name the authentication provider and click **Create**.
5. Make sure the “Control Flag” is set to REQUIRED, and click **Apply**.
6. Click **Details**, make appropriate changes for your database, and click **Apply**.
7. Navigate to **<your domain> > Security > Realms > myrealm > Users** and make sure existing users are listed for the RDBMSAuthenticator.
8. Navigate to **<your domain> > Security > Realms > myrealm > Groups** and make sure existing groups are listed for the RDBMSAuthenticator.
9. Before you switch from the Compatibility realm to the RDBMSAuthenticator you should set up users and groups for WebLogic Server administration in your RDBMS schema. Use the WebLogic Server Administration Console to do this, *not* the WebLogic Portal Administration Tools. These users and groups are defined in your domain’s `fileRealm.properties` file, but that file will not be used by the RDBMSAuthenticator when you switch to the new security framework.

   Add the following users and groups, making the users members of the appropriate group. Create users and groups for the RDBMSAuthenticator, not the Default Authenticator.

Group	**User**
Administrators	<your WebLogic system user, e.g., system>
Monitors	
Operators	
Deployers	

10. Verify that your WebLogic Server `system` user is a member of the `Administrators` group.

11. Navigate back to the Authentication Providers page under **<your domain> > Security > Realms > myrealm > Providers > Authentication Providers** and delete the `DefaultAuthenticator`.
12. In the WebLogic Server administration console, select your domain. Then, on the **Security > General** tab, change the Default Realm from `CompatibilityRealm` to `myrealm`. Apply the change.
13. Double-check for the existence of the proper groups and users for the RDBMSAuthenticator in **<your domain> > Security > Realms > myrealm**.
14. Restart the server.

    Notice the server will start with myrealm from now on. If the server fails to start due to an authenticator misconfiguration, you can switch back to the Compatibility Realm by removing the `userConfig` subdirectory under your domain directory, then restarting the server. You must then restart the configuration procedure.

Core Groups required for WebLogic Portal

When using the RDBMS Authenticator, there are no additional steps required in order to use WebLogic Portal applications. The core groups required for the proper operation of WebLogic Portal are pre-populated in the PointBase database that ships with WebLogic Portal, or are created during the steps for Switching to Other Databases in the *Administration Guide*.

Running the WLP Samples

When using the RDBMS Authenticator, there are no additional steps required in order to use the WebLogic Portal sample applications. The users and groups required for the sample applications are pre-populated in the PointBase database that ships with WebLogic Portal, or are created during the steps for Switching to Other Databases in the *Administration Guide*.

Upgrading a Portal from Compatibility Security to WebLogic Server 7.0 Security with Embedded LDAP

This is the recommended option if you have other applications that use the embedded WebLogic Server LDAP data store and you would like to share user and group information with a WebLogic Portal application.

These instructions explain how to upgrade from the WebLogic Server 7.0 default Compatibility RDBMS Realm to the Default Authenticator. Users and groups will be stored in the LDAP data store that is embedded in WebLogic Server.

It is essential to create the proper users and groups. If this is not done, the server will fail to start and a `java.lang.SecurityException: Authentication denied` message will appear in the console. If you do encounter this error, remove the entire `userConfig` directory, which is located under your domain directory, then restart the configuration procedure. Users and groups created in the embedded LDAP server will not be lost if you do not delete the staging directory. The staging directory is the directory that is named after your server and is used for 2-phase deployment in the internal LDAP server.

1. Create the core users and groups in your LDAP server.

   There are certain users and groups that are required for the proper operation of WebLogic Portal and WebLogic Server. These need to be set up in the embedded LDAP data store. To do this, go to the WebLogic Server administration console and expand the **<domain> > Security > Realms > myrealm** section. To create users, click the **Users** item and select **Configure a new User**. Likewise for groups, click the **Groups** item and select **Configure a new Group**.

   a. For WebLogic Portal, add the following users and groups, making the users members of the appropriate group:

Group	User
SystemAdministrator	administrator
AdminEligible	
DelegatedAdministrator	

b. Place users you would like to be able to manage portals in the `AdminEligible` and `DelegatedAdministrator` groups. The `administrator` user will be able to administer portals as a member of the `SystemAdministrator` group.

c. For WebLogic Server, add the following users and groups, making the users members of the appropriate group:

Group	**User**
Administrators	<your WebLogic system user, e.g., system>
Monitors	
Operators	
Deployers	

Note that `Monitors`, `Operators`, and `Deployers` are not required to contain the `Administrators` group like in `fileRealm.properties`. That is because the role mappings in the new security framework automatically make an `Admin` user able to monitor, operate, and deploy.

2. If you would like to use the WebLogic Portal sample applications, create the sample users and groups.

a. For the wlcs sample, add the following users and group, making the users members of the appropriate group:

Group	**Users**
wlcs_customer	bobsmith, suecarpenter, dangreen, democustomer

b. For sampleportal, add a top-level group called “Avitek”, and add sub-groups as follows:

Group	**Sub-Groups**
Avitek	CustomerService, FinancialAdvisor, Investor, Approver

c. In addition, add the following users and groups, making the users members of the appropriate groups. Note that some groups were created in the previous step. Notice also that some users belong to multiple groups.

Group	**Users**
<no group assigned>	acme, demo
Group1	visitor1,visitor2,visitor3,visitor4,visitor5
Group2	visitor6,visitor7,visitor8,visitor9,visitor10
Approver	visitor1
CustomerService	visitor5
FinancialAdvisor	visitor2, visitor3
Investor	visitor4, visitor6, visitor7
SystemAdministrator	demosa1, demosa2, demosa3
AdminEligible	admin1, admin2, admin3, admin4, demopa1, demopa2, demoga1, demoga2, demoga3
DelegatedAdministrator	demopa1, demopa2, demoga1, demoga2, demoga3

3. In the WebLogic Server administration console, click your domain. Then, on the **Security -> General** tab, change the Default Realm from `CompatibilityRealm` to `myrealm`. Apply the change.

4. Restart the server.

Notice the server will start with myrealm from now on. See the <Notice> in `weblogic.log` for <Security initializing using realm myrealm>. If the server fails to start due to an authenticator misconfiguration, you can switch back to the Compatibility Realm by removing the `userConfig` subdirectory under your domain directory, then restarting the server. You must then restart the configuration procedure.

Users and groups created in the embedded LDAP server will not be lost if you do not delete the staging directory. The staging directory is the directory that is named after your server and is used for 2-phase deployment and the internal LDAP server.

Upgrading a Portal from Compatibility Security to WebLogic Server 7.0 Security with a Commercial LDAP Provider

This is the recommended option if you would like to capitalize on a third-party LDAP provider that you already use.

These instructions explain how to upgrade from the 7.0 default Compatibility RDBMS Realm to an Authenticator backed by a commercial LDAP server, such as the IPlanet Authenticator. Users and groups will be stored in a commercial LDAP server.

It is essential to create the proper users and groups. If this is not done, the server will fail to start and a `java.lang.SecurityException: Authentication denied` message will appear in the WebLogic Server administration console. If you do encounter this error, remove the entire `userConfig` directory, which is located under your domain directory, then restart the configuration procedure.

1. In the WebLogic Server administration console, navigate to the Authentication Providers page under **<your domain> > Security > Realms > myrealm > Providers > Authentication Providers**.
2. Click the link to configure a new authenticator for your particular LDAP server. For example, click **Configure a new IPlanet Authenticator**.
3. Make sure the “Control Flag” is set to REQUIRED, and click **Create**.
4. Configure your Authenticator according to the instructions for Configuring an LDAP Authentication Provider in the WebLogic Server *Managing WebLogic Security* guide.
5. Create the core users and groups in your LDAP server.

   There are certain users and groups that are required for the proper operation of WebLogic Portal and WebLogic Server. These need to be set up in your LDAP data store.

a. For WebLogic Portal, add the following users and groups, making the users members of the appropriate group:

Group	**User**
SystemAdministrator	administrator
AdminEligible	
DelegatedAdministrator	

b. Place users you would like to be able to manage portals in the `AdminEligible` and `DelegatedAdministrator` groups. The `administrator` user will be able to administer portals as a member of the `SystemAdministrator` group.

c. For WebLogic Server, add the following users and groups, making the users members of the appropriate group:

Group	**User**
Administrators	<your WebLogic system user, e.g., system>
Monitors	
Operators	
Deployers	

6. If you would like to use the WebLogic Portal sample applications, create the sample users and groups.

a. For the wlcs sample, add the following users and group, making the users members of the appropriate group:

Group	**Users**
wlcs_customer	bobsmith, suecarpenter, dangreen, democustomer

b. For sampleportal, add a top-level group called "Avitek", and add sub-groups as follows:

Group	**Sub-Groups**
Avitek	CustomerService, FinancialAdvisor, Investor, Approver

In addition, add the following users and groups, making the users members of the appropriate groups. Note that some groups were created in the previous step. Notice also that some users belong to multiple groups.

Group	**Users**
<no group assigned>	acme, demo
Group1	visitor1,visitor2,visitor3,visitor4,visitor5
Group2	visitor6,visitor7,visitor8,visitor9,visitor10
Approver	visitor1
CustomerService	visitor5
FinancialAdvisor	visitor2, visitor3
Investor	visitor4, visitor6, visitor7
SystemAdministrator	demosa1, demosa2, demosa3
AdminEligible	admin1, admin2, admin3, admin4, demopa1, demopa2, demoga1, demoga2, demoga3
DelegatedAdministrator	demopa1, demopa2, demoga1, demoga2, demoga3

7. Navigate to the Authentication Providers page under **<your domain> > Security > Realms > myrealm > Providers > Authentication Providers** and delete the `DefaultAuthenticator`.

8. In the WebLogic Server administration console, click on your domain. Then, on the **Security > General** tab, change the Default Realm from `CompatibilityRealm` to `myrealm`. Apply the change.

9. Restart the server.

Notice the server will start with myrealm from now on. See the <Notice> in `weblogic.log` for <Security initializing using realm myrealm>. If the server fails to start due to an authenticator misconfiguration, you can switch back to the Compatibility Realm by removing the `userConfig` subdirectory under your domain directory, then restarting the server. You must then restart the configuration procedure.

Multiple Authentication Providers Support in WebLogic Portal 7.0 SP4

The WebLogic Portal user and group management framework communicates with only one authentication provider for basic user and group operations. Therefore, it is required that a system property be set to specify which authentication provider to use when the WebLogic Server is configured with multiple Authentication Providers.

How WebLogic Portal 7.0 uses the WebLogic Server Security Framework

WebLogic Portal 7.0 relies completely on WebLogic Server for login authentication. For authorization, WebLogic Portal has its own user and group management framework for some user/group management operations. WebLogic Portal's Delegated Administration framework uses the user and group management framework and entitlements framework for authorization. The Delegated Administration framework is not based on JAAS authorization so it does not use an authorization provider.

Limited Support of Multiple Authentication Providers in WebLogic Portal 7.0 SP4

The WebLogic Portal Administration Tools can use only one authentication provider. By default, if you have configured multiple providers, the WebLogic Portal Administration Tools use the authentication provider that is "most capable" in terms of offered functionality. However, you can force the WebLogic Portal Administration Tools to use a specific authentication provider by specifying the following system property:

```
com.bea.p13n.usermgmt.AuthenticationProviderName=<provider_displa
y_name>
```

The system property is specified as a `java -D` switch on the command line for starting the server.

Users from non-specified providers can log in to the portal and personalize their portal just like a user from an external custom security realm could do with the old WebLogic Server 6.x style of security.

Resetting of user passwords using the WebLogic Portal UserManager EJB (which is used by the `um` [user management] JSP tag library) only works for users who are available from the specified provider, because the Portal UserManager is only aware of a single authentication provider.

Group membership used by a portal should be consistent across providers. For example, if `user1` is in `Group1`, and `Group1` is associated with the `Group1` group portal, then `user1` should belong to `Group1` in all providers, otherwise the user may not be able to access the proper group portal if he is authenticated with a provider that does not have this group membership set up.

What Is Not Supported for Multiple Authentication Providers in WebLogic Portal 7.0 SP4

The use of multiple authentication providers with WebLogic Portal 7.0 SP4 has a larger impact on the WebLogic Portal Administration Tools than on the portal application itself. The following limitations exist:

- The WebLogic Portal Administration Tools can see only users/groups from the specified provider.
- Portal users belonging to the `SystemAdministrator` group (known as SAs) lose their super power for overall portal management because they cannot manage users/groups for the non-specified providers.
- Use of the realm configuration cleanup tool in the WebLogic Portal Administration Tools (to clean up unused user/group profiles) will delete the profiles of users/groups from the non-specified providers (though not the users/groups themselves). The "profile" consists of user properties that have been persisted for a user/group in the portal schema. A "profile" is not the existence of the user in the security realm (username/password and group membership).
- Setting a user password will fail if the user is from a non-specified provider.

Other Supported Security Realms

In addition to LDAP, WebLogic Server supports these security realms:

- Windows NT Security Realm

  This security realm uses Windows NT account information to authenticate users. Users and groups defined through Windows NT can be used by your Web application. You can use the WebLogic Server Administration Console to view this realm, but you must use the facilities provided by Windows NT to define users and groups.

- UNIX Security Realm

  A UNIX security realm executes a native program, `wlauth`, to authenticate users and groups using UNIX login IDs and passwords. On some UNIX platforms, `wlauth` uses a Pluggable Authentication Module (PAM) that allows you to configure authentication services in a UNIX platform without altering applications that use those services. On UNIX platforms for which PAM is not available, `wlauth` uses the standard login mechanism, including shadow passwords when they are supported. You can use the Administration Console to view this realm, but you must use the facilities provided by the UNIX platform to define users and groups.

- File Realm

  When you start the server, the File realm creates user, group, and ACL objects from properties defined through the WebLogic Server Administration Console and stores them in the `fileRealm.properties` file.

  Note: The File realm is designed for use with 10,000 or fewer users. If you have more than 10,000 users, use an alternate security realm.

Enabling Secure Sockets Layer Security

The Webflow and Pipeline mechanisms that direct the presentation and business logic associated with WebLogic Portal's Commerce JSP templates make use of the Secure Sockets Layer (SSL) and declarative transport mechanisms. Links that invoke protected JSP files, as well as certain Input Processors and Pipelines, need to be accessed via the HTTPS protocol. There are a number of these links already defined in the Commerce (`wlcs`) Web application's `web.xml` deployment descriptor. Secured JSP templates that rely on SSL also require a setting in the `web.xml` file that indicates the transport guarantee. This guarantee can be `CONFIDENTIAL` or `INTEGRAL`.

- A `CONFIDENTIAL` setting prevents other entities from observing the contents of the transmission.
- An `INTEGRAL` setting prevents the data from being changed while transmitting between the client and server.

See "Setting Up Portal Navigation" on page 9-1 for information on Webflows and Pipelines.

Note: For SSL connections to work, you must have a valid SSL certificate from a certificate authority set up on your server.

config.xml Requirements for SSL

To enable SSL for your Web application, you need to ensure that the domain's `config.xml` file has SSL enabled, as shown in Listing 7-5.

Listing 7-5 Enabling SSL in the `config.xml` File

```
<server>
.
.
.
   <SSL Enabled="true" ListenPort="7502" Name="portalServer"
      ServerCertificateChainFileName="ca.pem"
      ServerCertificateFileName="democert.pem"
   ServerKeyFileName="demokey.pem"/>

</server>
```

The SSL attribute should also identify the necessary certificate filenames, the server key filename, and the server name.

`config.xml` is stored in `<BEA_HOME>/user_projects/<YOUR_DOMAIN>`.

where `<YOUR_DOMAIN>` is the domain folder created when you ran the Configuration Wizard.

web.xml Requirements for SSL

You must also ensure that the secure listening ports in the Web application's deployment descriptor (`web.xml`) match that set in `config.xml`, as shown in Listing 7-6.

Listing 7-6 Identifying Listen Ports

```
<context-param>
  <param-name>HTTPS_PORT</param-name>
  <param-value>7502</param-value>
</context-param>
```

Enabling HTTPS_URL_PATTERNS

Enabled HTTPS_URL_PATTERNS for portal pages (the CreatePageChangeURLTag tag) as decribed above. The entries have the form /groupPortalDisplayName/pageName. If the user is not authenticated, the group portal name will be "DEFAULT_GROUP_PORTAL" so if you wish to specify that the page change URL for a page called "home" uses HTTPS when no one is logged in specify /DEFAULT_GROUP_PORTAL/home in the HTTPS_URL_PATTERNS section of web.xml

Example:

```
<context-param>

   <param-name>HTTPS_URL_PATTERNS</param-name>

   <param-value>

      /framework/security/login.jsp,

      /framework/security/new_user.jsp,

      /security/NewUser.inputprocessor,

      /security/LoginIP.inputprocessor,

      /groupPortal1/page1,

      /groupPortal2/page1

   </param-value>

</context-param>
```

Also added a FORCE_HTTPS_FOR_AUTH_USERS option to web.xml. This will cause all CreateWebflowURLTag derived tags to generate https URLs if the user has been authenticated and the tag is not specifically coded to use http. The web.xml entry should be as follows, with value true to enable the feature and false to turn it off.

```
<context-param>

   <param-name>FORCE_HTTPS_FOR_AUTH_USERS</param-name>

   <param-value>true</param-value>

</context-param>
```

If the FORCE_HTTPS_FOR_AUTH_USERS is enabled but the user is not logged in the HTTPS_URL_PATTERNS will be checked. If FORCE_HTTPS_FOR_AUTH_USERS is enabled and the user is logged in HTTPS_URL_PATTERNS are ignored and all URLs will be https unless the tag has been specifically coded for http.

See "Enabling HTTPS_URL_PATTERNS" on page 7-27 for more information.

Enabling Single Sign-On

With single sign-on enabled, a visitor needs only to sign-on once to access multiple web applications, provided those applications are running on the same server or cluster. Enabling single sign-on requires these steps:

1. Ensure that the user has the same cookies for each Web application.
2. Ensure that the same user properties are used for each Web application.

Setting the Cookie Name

Set the cookie name for each application to which the visitor will have single sign-on access. To do this, edit the `weblogic.xml` file located in `<BEA_HOME>weblogic700\samples\portal<PORTAL_DOMAIN>\beaApps\<PORTAL_APP>\<PORTAL>\WEB-INF\` *for each application* to which the visitor will have single sign-on access.

1. Open the specific `weblogic.xml` file and locate the `<session-param>` element that identifies the `CookieName` parameter, as shown in Listing 7-7.

Listing 7-7

```
<session-param>
   <param-name>CookieName</param-name>
   <param-value>JSESSIONID_SAMPLEPORTAL</param-value>
</session-param>
```

2. Change the `<param-value>` value to the appropriate cookie name.
3. Repeat steps 1 and 2 for each Web application.

Setting the User Properties

Each Web application to which a visitor has single sign-on access must use the same user property sets for the specific visitor. You will need to ensure this by editing these property sets and making them match. For details on editing user property sets, see "Creating a Property Set Definition" on page 6-18.

CHAPTER

8 Portal Content Management

A key component of any portal is its content. WebLogic Portal provides content by using a Content Manager. A Content Manager provides content and document management capabilities for use in personalization services to target users with dynamic web content. It works with files or with content managed by third-party vendor tools. While developing portal resources, you will have to configure the Content Manager and use various content-related tags so that the user has access to the most relevant content available.

This section includes information on the following subjects:

- Adding Content by Using the Bulk Loader
- Using Content-Selector Tags and Associated JSP Tags
- Integrating External Content Management Systems
- Constructing Content Queries

Adding Content by Using the Bulk Loader

The easiest way to add content to a portal is to use the bulk loader scripts provided by BEA. To implement this strategy, use this procedure:

1. Publish files to a directory on the file system at a specified interval. Ideally, you should publish content to subdirectories under `\dmsBase` directory in the appropriate domain folder. For example, place ads in:

`<BEA_HOME>\weblogic700\samples\portal\<DOMAIN_NAME>\dmsBase\ads`

2. When you are done publishing the content to the file on your directory system, run the BulkLoader by running one of the following scripts:

   - If you are loading text or image files, run `loaddocs.bat` (`loaddocs.sh` for UNIX users).
   - If you are loading ads for placeholders or campaigns, run `loadads.bat` (`loadads.sh` for UNIX users).

   The load scripts can be found at this location:

   `<BEA_HOME>\weblogic700\samples\portal\<DOMAIN_NAME>\`

   You can run these scripts either by typing them at the command line (or **Start | Run** in Windows NT or 2000) or by going to the Windows Explorer, locating the script you want to run, and double-clicking it in the file list. If you run the BulkLoader from a command line, you can also use any of the switches listed in Table 8-1.

Table 8-1 Bulk Loader Switch Settings

Switch Setting	Description
`-verbose`	Emits verbose messages.
`+verbose`	Runs quietly [default].
`-recurse`	Recurses into directories [default].
`+recurse`	Does not recurse into directories.
`-delete`	Removes document from database.
`+delete`	Inserts documents into database [default].
`-metaparse`	Parses HTML files for `<meta>` tags [default].
`+metaparse`	Does not parse HTML files for `<meta>` tags.
`-cleanup`	If specified, this only performs a table cleanup using the `-d` argument as the document base. (All files will need to be under that directory.)
`+cleanup`	Turns off table cleanup (do a document load) [default].
`-hidden`	Specifies to ignore hidden files and directories [default].

Table 8-1 Bulk Loader Switch Settings

Switch Setting	Description
`+hidden`	Specifies to include hidden files and directories.
`-inheritProps`	Specifies to have metadata properties be inherited when recursing [default].
`+inheritProps`	Specifies to have metadata properties not be inherited when recursing.
`-truncate`	Attempts to truncate data values if they are too large for the database (controlled via `loader.properties`).
`+truncate`	Does not attempt to truncate data values [default].
`-ignoreErrors`	Ignores any errors while loading a document (errors will still be reported).
`+ignoreErrors`	Stops processing on any error [default].
`-htmlPat <pattern>`	Specifies a pattern for determining which files are HTML files when determining whether to do the `<meta>` tag parse. This can be specified multiple times. If none are specified, `*.htm` and `*.html` are used.
`-properties <name>`	Specifies the location of the `loaddocs.properties` file that should contain the `connectionPool` definition. This file may contain `jdbc.column.<columnName>=<propname>` entries similar to the -`columnMap` argument.
`-conPool <name>`	Specifies the `connectionPool` name from the properties file from which the BulkLoader should get the connection information.
`-schema <name>`	Specifies the path to the schema file the BulkLoader will generate (defaults to `document-schema.xml`).
`+schema`	If specified, then no schema file will be created.
`-schemaName <name>`	Specifies the name of the schema generated by the BulkLoader. Defaults to “LoadedData”.

Table 8-1 Bulk Loader Switch Settings

Switch Setting	Description
`-encoding <name>`	Specifies the file encoding to use. Defaults to your system's default encoding. (See your JDK documentation for the valid encoding names.)
`-commitAfter <num>`	Commits the JDBC transaction after this many documents are loaded. Defaults to: only at the end of the full load.
`-match <pattern>`	Specifies a file pattern the BulkLoader should include. This can be specified multiple times. If none are specified, all files and directories are included.
`-ignore <pattern>`	Specifies a file pattern the BulkLoader should not include. This can be specified multiple times.
`-d <dir>`	Specifies the docBase that non-absolute paths will be relative to. If not specified, "." (current directory) is used.
`-mdext <ext>`	Specifies the filename extension for metadata property files. The value should starts with a `"."` (defaults to `.md.properties).`
`-filter <filter class>`	Specifies the class name of a `LoaderFilter` to run files through. This can be specified multiple times to add to the list of Loader Filters.
`+filters`	Clears the current list of Loader Filters. (This will clear the default filters as well.)
`--`	Everything after this is considered a file or directory.
`-columnMap <file.properties>`	Specifies a properties file containing the `jdbc.column.<columnName>=<propname,...>` list of additional columns to the DOCUMENT table (see `-column`). This cannot be used to override behavior for standard columns.
`-column <columnName>=<propName,...>`	Specifies an additional column to the DOCUMENT table and the property names that map onto the column. This cannot be used to override behavior for standard columns.
`+columns`	Clears any configured additional columns.

3. Make the sure that the DocumentConnectionPool in the application's `META-INF/application-config.xml` or through the WebLogic Server console is configured to point to the correct directory.

4. In order for the docPool to reload the XML schema files, it must be restarted.

For this strategy to succeed, the CMS must be able to publish document metadata in one of the three ways the BulkLoader supports:

- In a corresponding `md.properties` file; this file should be a standard Java-style properties file containing the user-defined metadata for the document. For example, for image`.gif` the file would be `image.gif.md.properties`.
- In the document file; for HTML files, the metadata can be placed in standard `<META name="…" content="…">` tags in the document file itself.
- In some other form for which a LoaderFilter can be written to gather the metadata.

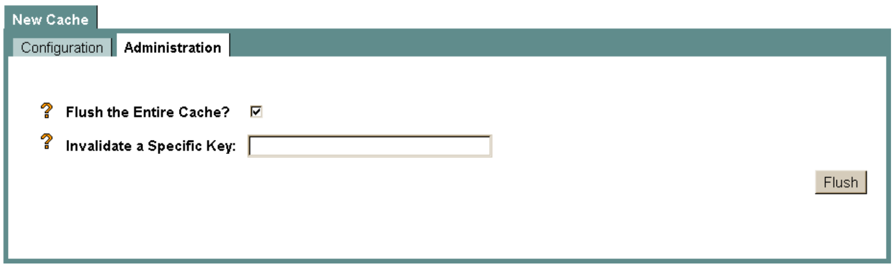
BulkLoader Performance Tips

The following suggestions will improve BulkLoader performance.

Clean Up the Cache

If you've made a lot of small updates to a page, before running the BulkLoader, you should clean up the cache. On the Cache Manager Administration tab on the Administration Console, select **Flush Entire Cache** and click **Flush**, as shown in Figure 8-1.

Figure 8-1 Flushing the Cache



Flush the Entire Cache? ☑

Invalidate a Specific Key:

Flushing is particularly helpful for these caches:

- `DocumentMetaDataCache`
- `AdBucketServicesCache`
- `DocumentContentCache`
- `AdServicesCache`

Restarting the ConnectionPool

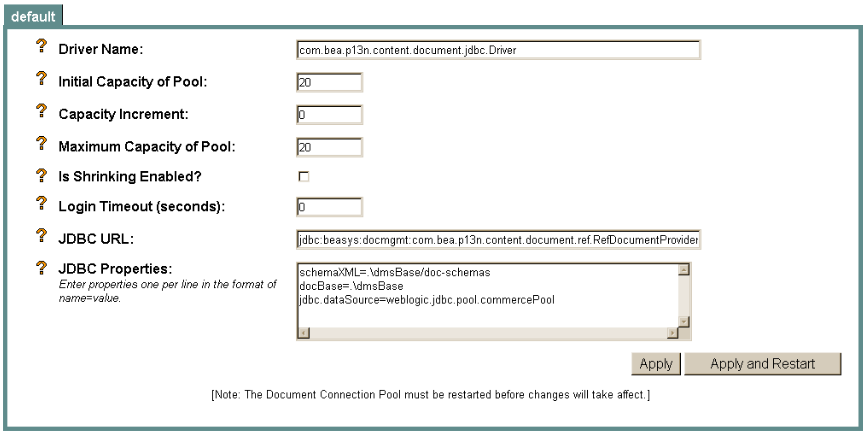
If you used the E-Business Control Center to add new metadata properties, you need to restart the connectionPool. To do so:

1. From left pane of the Administration Console, open the following node:

   `<yourDomain> →Deployments →Applications →<yourPortal> → Service Configurations →Documents →DocumentConnectionPool Services →Default`

   The Document Connection Pool Services Default tab appears in the right pane of the Console, as shown in Figure 8-2.

Figure 8-2 Restarting the ConnectionPool



default

Driver Name:	com.bea.p13n.content.document.jdbc.Driver
Initial Capacity of Pool:	20
Capacity Increment:	0
Maximum Capacity of Pool:	20
Is Shrinking Enabled?	☐
Login Timeout (seconds):	0
JDBC URL:	jdbc:beasys:docmgmt:com.bea.p13n.content.document.ref.RefDocumentProvider

JDBC Properties: *Enter properties one per line in the format of name=value.*	schemaXML=.\dmsBase/doc-schemas
	docBase=.\dmsBase
	jdbc.dataSource=weblogic.jdbc.pool.commercePool

[Note: The Document Connection Pool must be restarted before changes will take affect.]

2. Click **Apply and Restart**.

Configuring the Content Manager

If you are using a third-party Content Management System to populate a portal, you will need to configure it to work with WebLogic Portal. This section explains the configuration procedures you will need to perform.

The Content Manager is a run-time subsystem that provides access to content through tags and EJBs. When developing JSPs, the Content Management tags allow you to receive an enumeration of content objects by querying the content database directly by using a search expression syntax. The Content Manager component works alongside the other components to deliver personalized content, but does not have a GUI-based tool for edit-time customization.

The following section describes the tasks required to configure a Content Manager. It includes information on the following subjects:

- Configuring the DocumentManager EJB Deployment Descriptor
- Configuring the PropertySetManager EJB Deployment Descriptor for Content Management
- Configuring DocumentManager MBeans
- Setting Up Document Connection Pools
- Editing a DocumentConnectionPool MBean in the WebLogic Console
- Configuring the Web Application

Configuring the DocumentManager EJB Deployment Descriptor

The `DocumentManager` EJB deployment descriptor handles the EJB portion of the Content Management component configuration and must be configured to recognize the correct environmental settings. To configure the `DocumentManager` EJB, ensure that the following environment settings are in its deployment descriptor:

- `DocumentManagerMBeanName`—specifies the name of the `DocumentManager` MBean to use to configure this `DocumentManager`.
  - In the enterprise application's `application-config.xml` file, a `<DocumentManager>` entry must exist with a `Name` attribute equal to the value specified in the deployment descriptor. If this is not specified in the deployment descriptor, it defaults to "default".
- `DocumentConnectionPoolName`—specifies the name of the `DocumentConnectionPool` MBean that this `DocumentManager` should use.
  - In the application's `application-config.xml` file, a `<DocumentConnectionPool>` entry must exist with a `Name` attribute equal to the value specified in the deployment descriptor.
  - If the `DocumentManager's DocumentConnectionPoolName` attribute is configured to use MBeans, then the value in the deployment descriptor is ignored.
- `jdbc/docPool`—specifies the J2EE resource reference to the `javax.sql.DataSource` that this `DocumentManager` should use to access a document connection pool.

  If `jdbc/docPool` is specified in the deployment descriptor, then:
  - `DocumentConnectionPoolName` is ignored, and
  - Any `DocumentConnectionPool` MBeans in the application's `application-config.xml` file are also ignored.
- `PropertyCase`—specifies how the `DocumentManager` modifies the incoming property name.
  - If the `PropertyCase` attribute of the `DocumentManager` MBean being used is set, the value in the deployment descriptor is ignored.
  - If this is *lower*, all property names are converted to lowercase.
  - If this is *upper*, all property names are converted to uppercase.
  - If this is anything else or not specified, property names are not modified.

  Note: Use *lower* or *upper* depending upon the document connection pool implementation being used. For the document reference implementation, do not specify the `PropertyCase`.

Configuring the PropertySetManager EJB Deployment Descriptor for Content Management

The DocumentManager needs to be integrated into the PropertySetManager EJB deployment descriptor so that content property sets are exposed to the system. To configure the `PropertySetManager` EJB Deployment Descriptor for content management, add the following environment settings:

- `repository/CONTENT`—specifies the fully qualified class name of `com.bea.p13n.property.PropertySetRepository` implementation. Use `com.bea.p13n.content.PropertySetRepositoryImpl` to integrate with the Content Management component.
- `ejb/ContentManagers/[type]`— the `com.bea.p13n.content.PropertySetRepositoryImpl` looks for all environment entries starting with `ejb/ContentManagers`. It expects these to be J2EE EJB references to `ContentManagers` (or subclasses).

  To integrate a `ContentManager` or `DocumentManager` with the `PropertySetManager`, add an EJB reference here. For example, `ejb/ContentManagers/Document` is mapped to the standard `DocumentManager`.

Alternately, you can set the `JNDIName` attribute of the `DocumentManager` MBean to the JNDI Home name of the `DocumentManager`. The `${APPNAME}` construct can be used in the value; it will be replaced by the current J2EE application name. The `com.bea.p13n.content.PropertySetRepositoryImpl` will automatically pick up those `DocumentManagers` and the J2EE EJB reference is not required.

Configuring DocumentManager MBeans

The `DocumentManager` implementation uses `DocumentManager` MBeans to maintain its configuration. A deployed `DocumentManager` finds which `DocumentManager` MBean to use from the `DocumentManagerMBeanName` EJB deployment descriptor setting. You will need to configure the `DocumentManager` MBeans in the application so that their values correspond to the `Name` attributes in the `DocumentManagerMBeanName` EJB deployment descriptor.

To configure a `DocumentManager` MBean, you can modify the application's `META-INF/application-config.xml` file to add or change the following XML, as shown in Listing 8-1.

Listing 8-1 Modifying the `<DocumentManager>` Element in an Application's `META-INF/application-config.xml` File

```
<DocumentManager
  Name="default"
  DocumentConnectionPoolName="default"
  PropertyCase="none"
  MetadataCaching="true"
  MetadataCacheName="documentMetadataCache"
  UserIdInCacheKey="false"
  ContentCaching="true"
  ContentCacheName="documentContentCache"
  MaxCachedContentSize="32768"
>
</DocumentManager>
```

Do not try to change these attributes from within the application-config.xml file. Instead, use the WebLogic Server Administration Console, as described in "Using the WebLogic Server Administration Console to Modify DocumentManager MBeans" on page 8-10.

Using the WebLogic Server Administration Console to Modify DocumentManager MBeans

To modify DocumentManager MBeans by using the WebLogic Server Administration Console, use this procedure.

1. Start WebLogic Server and open a Web browser.

2. Open the WebLogic Server Administration Console by typing the following URL in the Address field:

   `http://<hostname>:<port>/console`

   For example, if you are launching the console on the server itself, the default URL is:

```
http://localhost:7501/console
```

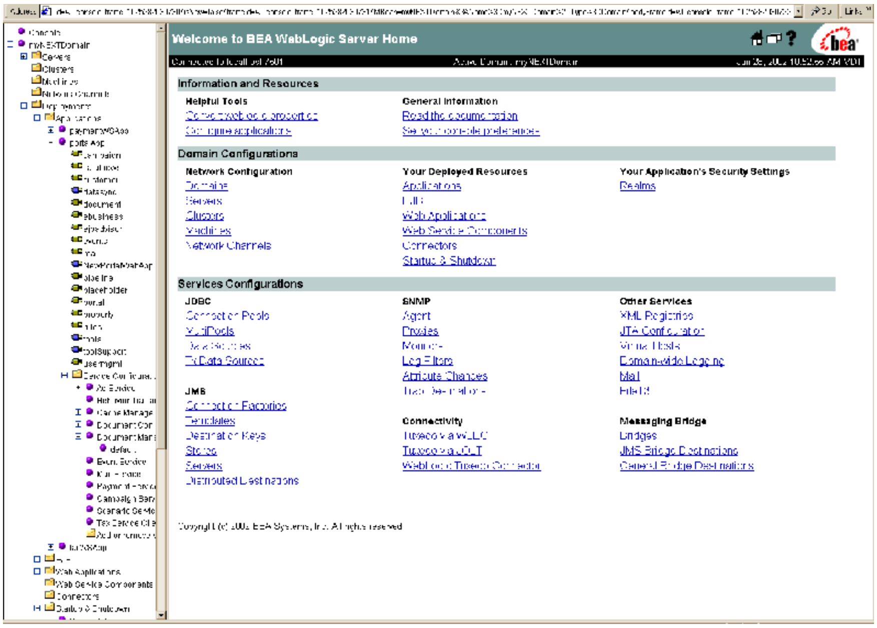
3. Press **Enter**.

   The WebLogic Server sign-in screen appears.

4. Sign in by entering your username and password and clicking **Sign in**. The default username/password pair is `system/weblogic`.

5. The WebLogic Server Administration Console appears, as show in Figure 8-3.

Figure 8-3 WebLogic Server Administration Console



Welcome to BEA WebLogic Server Home

Connected to localhost:7501 — Active Domain: myNEXTDomain — Jan 28, 2002 10:52:55 AM MDT

Information and Resources

Helpful Tools	General Information
Convert weblogic.properties	Read the documentation
Configure applications	Set your console preferences

Domain Configurations

Network Configuration	Your Deployed Resources	Your Application's Security Settings
Domains	Applications	Realms
Servers	EJB	
Clusters	Web Applications	
Machines	Web Service Components	
Network Channels	Connectors	
	Startup & Shutdown	

Services Configurations

JDBC	SNMP	Other Services
Connection Pools	Agent	XML Registries
MultiPools	Proxies	JTA Configuration
Data Sources	Monitors	Virtual Hosts
Tx Data Sources	Log Filters	Domain-wide Logging
	Attribute Changes	Mail
	Trap Destinations	FileT3

JMS	Connectivity	Messaging Bridge
Connection Factories	Tuxedo via WLEC	Bridges
Templates	Tuxedo via JOLT	JMS Bridge Destinations
Destination Keys	WebLogic Tuxedo Connector	General Bridge Destinations
Stores		
Servers		
Distributed Destinations		

Copyright (c) 2002 BEA Systems, Inc. All rights reserved

6. In the left pane, drill down to the DocumentManager MBean for your applicaion. Select these nodes:

   `MyDomain` →`Deployments` →`Applications` →`MyApplication` →`Service Configuration` →`Document Manager` →`MyMBean`

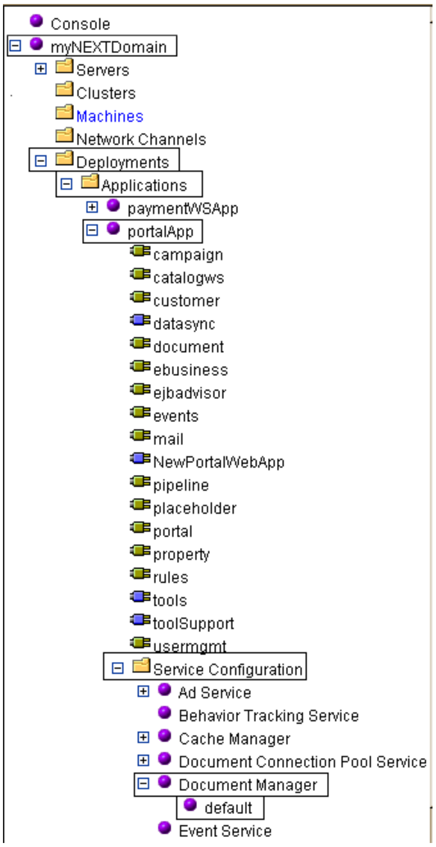
   Where:

   - MyDomain is the domain inwhich the application resides.
   - MyApplication is the Web application that uses the MBean.

- MyMBean is the actual MBean you want to configure.

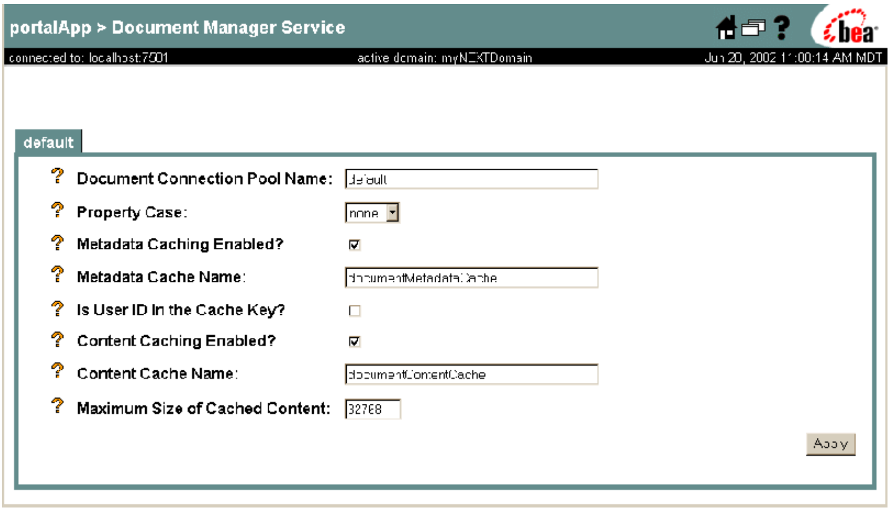
Figure 8-4 shows an example of the nodes you would select to drill down to the Default MBean for the DocumentManager for an application called portalApp.

Figure 8-4 Drilling Down to an MBean



When you select the MBean node, the DocumentManager Services screen for that MBean appears in the right pane, as shown in Figure 8-5. Note the MBean name on the tab.

Figure 8-5 DocumentManager Service Screen


portalApp > Document Manager Service
connected to: localhost:7501
active domain: myNEXTDomain
Jun 20, 2002 11:00:14 AM MDT
default
Apply

Document Connection Pool Name: default
Property Case: none
Metadata Caching Enabled? ✓
Metadata Cache Name: documentMetadataCache
Is User ID In the Cache Key? ☐
Content Caching Enabled? ✓
Content Cache Name: documentContentCache
Maximum Size of Cached Content: 32768

7. Change the attributes as necessary and click **Apply**. These attributes are listed in Table 8-2.

Table 8-2 `DocumentManager` **MBean Attributes**

Attribute	Screen Label/Control Type	Description
`DocumentConnection PoolName`	**Document Connection Pool Name** Text Box	Specifies the name of the `DocumentConnectionPool` MBean the `DocumentManager` should use.

Table 8-2 `DocumentManager` MBean Attributes

Attribute	Screen Label/Control Type	Description
`PropertyCase`	**Property Case** Drop-down List	Specifies how the `DocumentManager` modifies the incoming property name. ■ If this is *lower*, all property names are converted to lowercase. ■ If this is *upper*, all property names are converted to uppercase. ■ If this is anything else or not specified, property names are not modified. Use *lower* or *upper* depending upon the document connection pool implementation being used. For the document reference implementation, do not specify the `PropertyCase`.
`MetadataCaching`	**Metadata Caching Enabled?** Checkbox	Specifies whether the `DocumentManager` should cache Document metadata from searches. Use `true` to have the DocumentManager cache search results in the `com.bea.p13n.cache.Cache` specified by `MetadataCacheName`; otherwise, use `false`. This defaults to `true`.
`MetadataCacheName`	**MetaData Cache Name** Text Box	Specifies the name of the `com.bea.p13n.cache.Cache` to use if `MetadataCaching` is set to `true`. This defaults to `documentMetadataCache`.
`UserIdInCacheKey`	**Is Userid in the Cache Key?** Checkbox	Specifies whether the user's identifier should be used as part of the key when caching document metadata or content. This defaults to `true`. If using WebLogic Portal reference implementation document management system, set this to `false`.
`ContentCaching`	**Content Caching Enabled?** Checkbox	Specifies whether the `DocumentManager` should cache document content (that is, the bytes of the document). Use `true` to have the `DocumentManager` cache document content bytes; otherwise use `false`. This value defaults to `true`.

Table 8-2 `DocumentManager` MBean Attributes

Attribute	Screen Label/Control Type	Description
`ContentCacheName`	**Content Cache Name** Text Box	Specifies the name of the `com.bea.p13n.cache.Cache` to use if `ContentCaching` is set to `true`. This defaults to `documentContentCache`.
`MaxCachedContentSize`	**Maximum Size of Cached Content** Text Box	Specifies the maximum size of a document's content bytes that the `DocumentManager` will cache, if ContentCaching is `true`. This defaults to 32768 (32K) bytes.
`JNDIName`	n/a	Specifies the JNDI home name of the `DocumentManager` EJB that is connected to this MBean. The `${APPNAME}` construct can be used in the value; it will be replaced by the current J2EE application name. This is used by the `com.bea.p13n.content.PropertySetRepositoryImpl` to tie document property set information into the `PropertySetManager`.

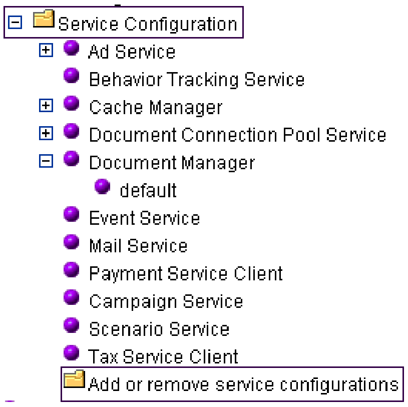
Disabling an MBean

You can enable or disable an MBean from the WebLogic Server Administration Console. To do so, use this procedure.

Note: This procedure assumes that you are connected to WebLogic Server and have the Administartion console open, as you did in "Using the WebLogic Server Administration Console to Modify DocumentManager MBeans."

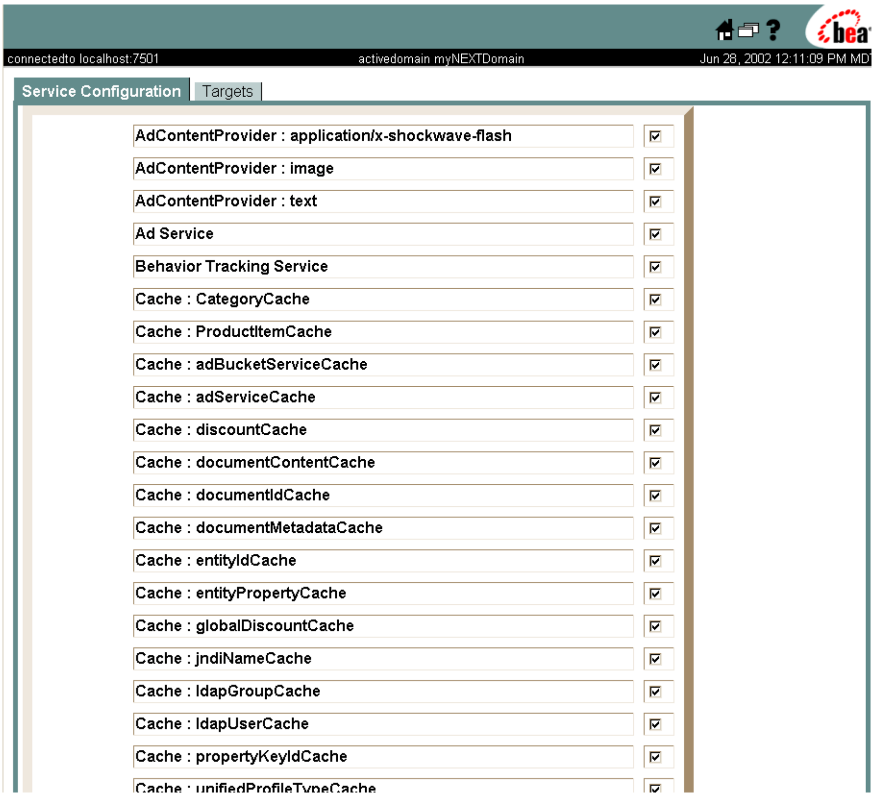
1. In the left pane, drill down to the **Add or remove service configurations** node, which is under the Service Configurations node, as shown in Figure 8-6.

Figure 8-6 Selecting the Add or remove service configurations Node



The Service Configuration screen appears in the right pane, as shown in Figure 8-7.

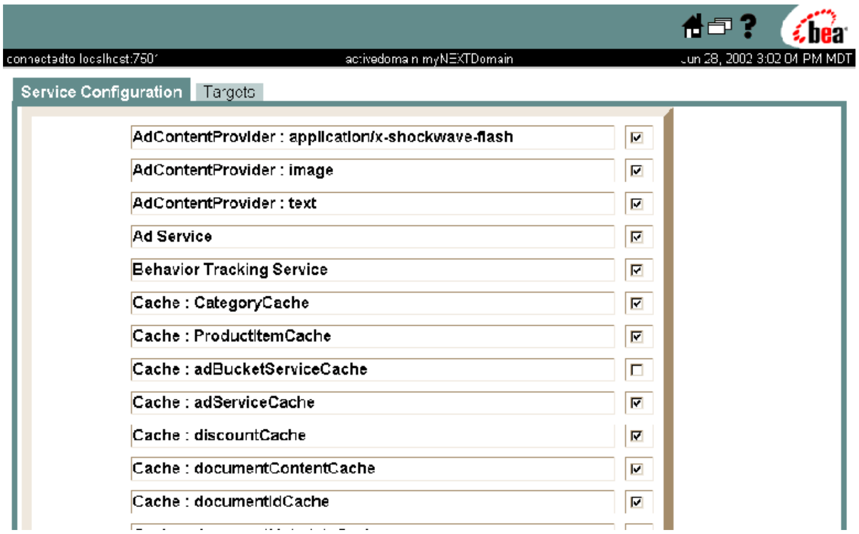
Figure 8-7 Service Configuration Screen


connectedto localhost:7501
activedomain myNEXTDomain
Jun 28, 2002 12:11:09 PM MDT
Service Configuration
Targets

AdContentProvider : application/x-shockwave-flash	✓
AdContentProvider : image	✓
AdContentProvider : text	✓
Ad Service	✓
Behavior Tracking Service	✓
Cache : CategoryCache	✓
Cache : ProductItemCache	✓
Cache : adBucketServiceCache	✓
Cache : adServiceCache	✓
Cache : discountCache	✓
Cache : documentContentCache	✓
Cache : documentIdCache	✓
Cache : documentMetadataCache	✓
Cache : entityIdCache	✓
Cache : entityPropertyCache	✓
Cache : globalDiscountCache	✓
Cache : jndiNameCache	✓
Cache : ldapGroupCache	✓
Cache : ldapUserCache	✓
Cache : propertyKeyIdCache	✓
Cache : unifiedProfileTypeCache	✓

2. Locate the MBean you want to remove from the list and click the associated checkbox to deselect it, as shown in Figure 8-8.

Figure 8-8 Service Configuration Screen with MBean Deselected


connectedto localhost:7501
activedomain myNEXTDomain
Jun 28, 2002 3:02:04 PM MDT
Service Configuration
Targets

AdContentProvider : application/x-shockwave-flash	✓
AdContentProvider : image	✓
AdContentProvider : text	✓
Ad Service	✓
Behavior Tracking Service	✓
Cache : CategoryCache	✓
Cache : ProductItemCache	✓
Cache : adBucketServiceCache	☐
Cache : adServiceCache	✓
Cache : discountCache	✓
Cache : documentContentCache	✓
Cache : documentIdCache	✓

3. Deselect each MBean you want to disable. When you are done, click **Submit**.

   You receive the message "You must redeploy your application [APP_NAME] for these changes to take place."

4. Redeploy the application or restart the server.

Restoring a Disabled MBean

To restore the MBean disabled in the preceding procedure, click its acssociated checkbox to select it. Again, once you've restored all necessary MBeans, click **Submit**.

Setting Up Document Connection Pools

The `DocumentManager` implementation uses connection pools to a specialized JDBC driver to handle searches on content. A deployed `DocumentManager` finds the document connection pool to use via either the `DocumentConnectionPoolName` attribute of its `DocumentManager` MBean or the `DocumentConnectionPoolName` EJB deployment descriptor setting. That value must correspond to a `DocumentConnectionPool` MBean.

To configure a `DocumentConnectionPool` MBean, modify the application's `META-INF/application-config.xml` file by adding or changing the XML shown in Listing 8-2:

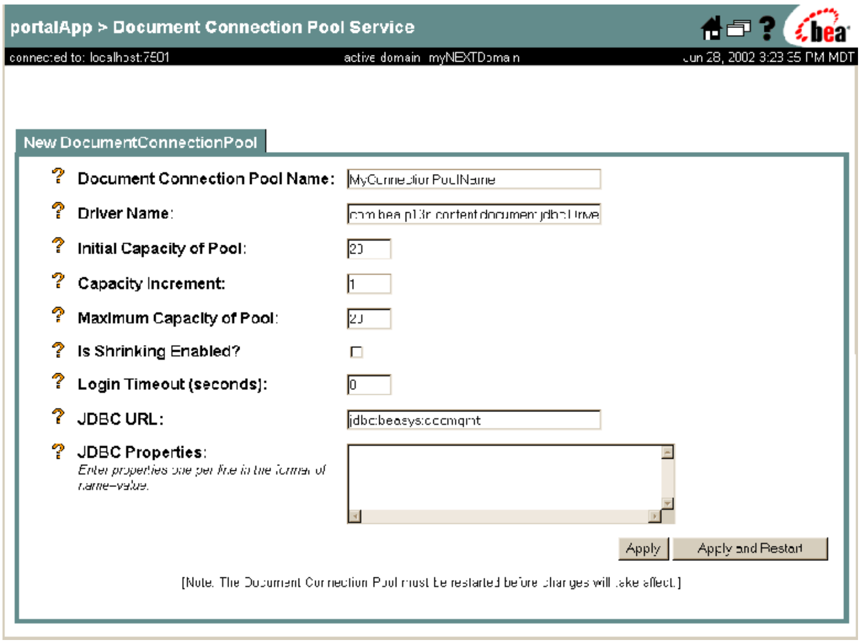
Listing 8-2 Modifying an Application's `META-INF/application-config.xml` File to Configure a `DocumentConnectionPool` MBean

```
<DocumentConnectionPool

Name="default"DriverName="com.bea.p13n.content.document.
jdbc.Driver"URL="jdbc:beasys:docmgmt:com.bea.p13n.content
   document.ref.RefDocumentProvider"
Properties="jdbc.dataSource=weblogic.jdbc.pool.commercePool;
   schemaXML=D:/bea/user_projects/myNEWdomain/dmsBase/
   doc-schemas;docBase=D:/bea/user_projects/myNEWdomain/dmsBase"
InitialCapacity="20"
MaxCapacity="20"
CapacityIncrement="0"

/>
```

Editing a DocumentConnectionPool MBean in the WebLogic Console

As with the DocumentManager MBeans, you must modify the DocumentConnectionPool MBean by using the WebLogic Server Administration Console. See the procedure in "Using the WebLogic Server Administration Console to Modify DocumentManager MBeans" on page 8-10 for instructions. Note that, for the DocumentConnectionPool MBeans, you need to select the Document Connection Pool Service node and make the changes on the Document Connection Pool Service screen, shown in Figure 8-9.

Figure 8-9 Document Connection Service Pool Screen



portalApp > Document Connection Pool Service

connected to: localhost:7501 active domain myNEXTDomain Jun 28, 2002 3:23:35 PM MDT

New DocumentConnectionPool

Document Connection Pool Name:	MyConnectionPoolName
Driver Name:	com.bea.p13n.content.document.jdbc.Driver
Initial Capacity of Pool:	20
Capacity Increment:	1
Maximum Capacity of Pool:	20
Is Shrinking Enabled?	☐
Login Timeout (seconds):	0
JDBC URL:	jdbc:beasys:docmgmt
JDBC Properties: *Enter properties one per line in the format of name=value.*	

[Note. The Document Connection Pool must be restarted before changes will take effect.]

The attributes of the `DocumentConnectionPool` MBean that you can change are listed in Table 8-3.

Table 8-3 DocumentConnectionPool MBean Attributes

Attribute	Screen Label/ Control Type	Description
`DriverName`	**Driver Name** Text Box	Specifies the JDBC driver class name to use. This should be set to `com.bea.p13n.content.document.jdbc.Driver`

Table 8-3 DocumentConnectionPool MBean Attributes

Attribute	Screen Label/ Control Type	Description

`URL`	**JDBC URL**
	Text Box
		Specifies the JDBC URL to use.
		■ For WebLogic Portal's reference implementation document management system, this should be set to:
		`jdbc:beasys:docmgmt:com.bea.p13n.content.document.ref.RefDocumentProvider.`
		■ For a different Document Provider, use:
		`jdbc:beasys:docmgmt:<classname>`
		where `<classname>` is the fully qualified class name of the implementation of `com.bea.p13n.content.document.spi.DocumentProvider.`

`Properties`	**JDBC Properties** Text Box	The semi-colon separated list of `name=value` pairs which will be passed to the `DocumentProvider` specified in the `URL`. Table 8-4 lists the properties that the reference implementation understands.
`InitialCapacity`	**Initial Capacity of Pool** Text Box	Specifies the initial number of connections to create when the document connection pool is started.
`MaxCapcity`	**Maximum Capacity of Pool** Text Box	Specifies the maximum number of connections this pool will ever create and maintain.
`CapacityIncrement`	**Capacity Increment** Text Box	Specifies the number of connections the pool will create whenever it needs to create an available connection.

Table 8-3 DocumentConnectionPool MBean Attributes

Attribute	Screen Label/ Control Type	Description
`LoginTimeout`	n/a	Specifies the amount of time to wait for a connection: after this time expires, an exception is thrown. Use 0 or less to have the pool not timeout, which is the default.
`ClassPath`	n/a	Specifies the semicolon -separated list of additional directories and JARs the connection pool should use when attempting to load the `Driver` and the `DocumentProvider` classes. All paths are assumed to be relative to the application directory.

Table 8-4 Describes the valid reference implementation properties you can set for the DocumentConnectionPool MBean.

Table 8-4 Reference Implementation Properties

`Property`	Description
`jdbc.dataSourceSp`	Specifies the JNDI name of the javax.sql.DataSource to use to get database connections. This datasource should be connected to the database that contains the DOCUMENT and DOCUMENT_METADATA tables.
`jdbc.url`	Specifies the JDBC URL to connect to. If jdbc.dataSource is specified, this is ignored.
`jdbc.driver`	Specifies the JDBC driver class to load. If jdbc.dataSource is specified, this is ignored.
`jdbc.isPooled`	If true, or if `jdbc.url` starts with `jdbc:weblogic:pool` or `jdbc:weblogic:jts`, or if `jdbc.dataSource` is specified, then assumes the connection is pooled and won't cache it. If anything else, assumes the connection is not pooled and will maintain one connection.

Table 8-4 Reference Implementation Properties

Property	**Description**
`jdbc.supportsLikeEscapeClause`	Specifies whether the underlying database supports the SQL LIKE ESCAPE clause. If this is not specified, the connection will be queried.
`jdbc.docBase`	Specifies under which base directory the documents are stored. Assumes all paths coming from the database are relative to this directory.
`jdbc.schemaXML`	Specifies the path to the directory containing XML files following the doc-schemas DTD which contain the property set information. The system will recurse through the directory, loading all files ending in .xml.
`jdbc.isolationLevel`	Configures the transaction isolation level to set on the database connections. This can be one of the following: READ_COMMITTED READ_UNCOMMITTED SERIALIZABLE REPEATABLE_READ NONE If not specified, it defaults to SERIALIZABLE. For further details, see the Javadoc API documentation for `java.sql.Connection`.
`jdbc.column.<colName>`	Specifies an additional column to the DOCUMENT table. The value is the comma-separated list of property names that map onto that column. This can be specified multiple times. This should be used in conjunction with the `-columnMap` and/or `-column` arguments to the BulkLoader. If the same property is mapped to more than one column, the result is indeterminate.

You can edit a DocumentConnectionPool MBean to change attribute and property values as needed by using the WebLogic Server Administration Console, as shown in Figure 8-9.

Configuring the Web Application

You need to configure the Web application to have access to the J2EE resources (such as EJBs, servlets, and JSP tag libraries) required to access the content management services. This means you will need to configure EJB references to `ejb/ContentManager` and `ejb/DocumentManager`. Additionally, you need to have the `com.bea.p13n.content.servlets.ShowDocServlet` mapped into your Web Application. BEA suggests that you to map it to the `/ShowDoc/* URL` in your Web Application, as shown in Listing 8-3.

Listing 8-3 Mapping the ShowDocServlet

```
<servlet>
  <servlet-name>ShowDocServlet</servlet-name>
  <servlet-class> com.bea.p13n.content.servlets.ShowDocServlet
  </servlet-class>

 <!-- Make showdoc always use the local ejb-ref DocumentManager -->

  <init-param>
    <param-name>contentHome</param-name>
    <param-value>java:comp/env/ejb/DocumentManager</param-value>
  </init-param>

</servlet>

...

<servlet-mapping>
  <servlet-name>ShowDocServlet</servlet-name>
  <url-pattern>/ShowDoc/*</url-pattern>
</servlet-mapping>
```

This will allow the `ShowDoc/` URI under your Web Application’s context root (for example, `/wlcs/ShowDoc`) to be sent to the `ShowDocServlet`. The `contentHome` `<init-param>` will cause that `ShowDocServlet` to always use the `ejb/DocumentManager` EJB reference; you can take this out to allow `ShowDocServlet` to obey any `contentHome` request parameters.

To access the Content Management tag libraries, you will need to:

- Copy the `cm_taglib.jar` file in the Web Application's `WEB-INF/lib` directory. (It can be copied from `WL_PORTAL_HOME/lib/p13n/web`.)
- Make sure that `cm.tld` is mapped to `/WEB-INF/lib/cm_taglib.jar` in a `<taglib>` entry in your Web Application's `WEB-INF/web.xml` file.

Using Content-Selector Tags and Associated JSP Tags

A content selector is one of several mechanisms that WebLogic Portal provides for retrieving documents from a content management system. You can use content selector JSP tags and a set of other JSP tags to retrieve and display the content targeted by the content selector.

This section describes how to use content-selector tags and their associated JSP tags to manage content. It includes information on the following topics:

- Using the <pz:contentSelector> Tag
- Associated Tags That Support Content Selectors
- Using Content Selector Tags and Associated Tags

For information on how WebLogic Portal's content-related JSP tags map to WebLogic Portal's content management service provider interface (SPI), see the "Personalization JSP Tags" section of the JavaServer Page Guide at http://edocs.bea.com/wlp/docs70/jsp/p13njsp.htm.

Using the <pz:contentSelector> Tag

The `<pz:contentSelector>` selector tag allows you to do the following:

- Identify the Content Selector Definition
- Identify the JNDI Home for the Content Management System
- Define the Array That Contains Query Results

- Create and Configure the Cache to Improve Performance

Identify the Content Selector Definition

The content selector definition created in the E-Business Control Center determines the conditions that activate a content selector and the query that the active content selector runs.

To refer to this definition, use the `rule` attribute:

```
<pz:contentSelector rule= { definition-name | scriptlet } >
```

You can use a scriptlet to determine the value of the `rule` attribute based on additional criteria. For example, you use a content selector in a heading JSP (`heading.inc`), which is included in other JSPs. You can create different content selectors for each page that includes `heading.inc`.

A scriptlet is used in `heading.inc` to provide a value based on the page that currently displays the included JSP file. Listing 8-4 shows an example.

Listing 8-4 Using a Scriptlet in `heading.inc`

```
 String banner = (String)pageContext.getAttribute("bannerPh");
   banner = (banner == null) ? "cs_top_generic" : banner;

%>

<!-- ------------------------------------------------------------
-->

<table width="100%" border="0" cellspacing="0" cellpadding="0"
   height="108">

  <tr><td rowspan="2" width="147" height="108">
  <pz:contentSelector rule="<%= banner %>" ... />

</td>
```

Identify the JNDI Home for the Content Management System

The content selector tag must use the `contentHome` attribute to specify the JNDI home of the content management system. If you use the reference content management system or a third-party integration, you can use a scriptlet to refer to the default content home. Because the scriptlet uses the `ContentHelper` class, you must first use the following tag to import the class into the JSP:

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
```

Then, when you use the content selector tag, specify the `contentHome` as described in Listing 8-5.

Listing 8-5 Specifying `contentHome` in a Content-Selector Tag

```
<pz:contentSelector
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
... />
```

If you create your own content management system, you must specify the JNDI home for your system instead of using the ContentHelper scriptlet. In addition, if your content management system provides a JNDI home, you can specify that one instead of using the ContentHelper scriptlet.

Define the Array That Contains Query Results

You can use the attributes described in Table 8-5 to configure the array that contains the results of the content-selector query.

Table 8-5 Attributes that Define the Array that Contains Query Results

Attribute	Description
`id`	Specifies a name for the array. This attribute is required. For example, `<pz:contentSelector id="docs" .../>` places documents in an array named `docs`.

Table 8-5 Attributes that Define the Array that Contains Query Results

Attribute	Description
`max`	Limits the number of documents the content selector places in its array. For example, `<pz:contentSelector max="10" .../>` causes the content selector to stop retrieving documents when the array contains 10 documents. This attribute is optional and defaults to `-1`, which means no maximum.
`sortBy`	Uses one or more document attributes to sort the documents in the array. The syntax for `sortBy` follows the SQL *order by clause* syntax. This attribute is optional. If you do not specify this attribute, the content selector returns the query results in the order that the content management system returns them. For example, `<pz:contentSelector sortBy="creationDate" .../>` places the documents that were created first at the beginning of the array. The tag `<pz:contentSelector sortBy="creationDate ASC, title DESC" .../>` places older documents at the beginning of the array. If any documents were created on the same day, it sorts those documents counter-alphabetically by title.

Create and Configure the Cache to Improve Performance

To extend accessibility of retrieved content and to improve performance, you can optionally use content selector attributes to create and configure a cache that contains the array contents. Without the cache, you can access the content selector array only from the current JSP page, and only for the customer request that created it. In addition, each time a customer requests a JSP that contains the content selector tag, the content selector must run the query, potentially slowing the overall performance of WebLogic Portal.

To cache the contents of the array, use the attributes listed in Table 8-6.

Table 8-6 Attributes that Cache the Contents of an Array

Attribute	Description

`useCache`	Determines whether the content selector places the array in a cache. To activate the cache, set this attribute to `true`. For example, `<pz:contentSelector cache="true" ...>`.
	To deactivate the cache, set the attribute to `false` or do not include it. For example, the following statements are equivalent:
	`<pz:contentSelector cache="false" .../>`
	`<pz:contentSelector ...>`

`cacheId`	Assigns a name to the cache. If you do not specify this attribute, the cache uses the name of the array (which you must specify with the `id` attribute). If you want to access the cache from a JSP or user session other than the one that created the array, you must specify a `cacheId`.
`cacheTimeout`	Specifies the number of milliseconds that WebLogic Portal maintains the cache. The content selector does not re-run the query until the number of seconds expires. For example, you create the following tag: `<pz:contentSelector cache="true" cacheTimeout=”300000” .../>` A customer requests the page that contains this content selector tag. The user leaves the page but, 2 minutes (120000 milliseconds) later, requests it again. The content selector evaluates its conditions, but because only 120000 milliseconds have expired since the content selector created the cache, it does not re-run the query. Instead, it displays the documents in the cache.

Table 8-6 Attributes that Cache the Contents of an Array

Attribute	Description
`cacheScope`	Determines from where the cache can be accessed. You can provide the following values for this attribute: ■ `application`. Any JSP page in the Web application that any customer requests can access the cache. ■ `session` (the default). Any JSP in the Web application that the current customer requests can access the cache. ■ `page`. Only the current JSP that any customer requests can access the cache. ■ `request`. Only the current user request can access the cache. If a customer re-requests the page, the content selector re-runs the query and recreates the cache.

Associated Tags That Support Content Selectors

The JSP tags listed in Table 8-7 support content selector functions.

Table 8-7 JSP Tags that Support Content-Selector Functions

Tag	Description
`<um:getProfile>`	Retrieves the profile of the customer who is currently viewing the page. A content selector uses the customer profile to evaluate any conditions that involve customer properties. For example, if you create a content selector that runs a query for all customers in the Gold Customer customer segment, the content selector must access the customer profile to determine if it matches the customer segment. Even if a content selector does not currently use the customer profile for its conditions, we recommend that you include the `<um:getProfile>` tag; its affect on performance is minimal and with the tag, customer-profile conditions can be added to the content selector without requiring a developer to modify JSPs. The tag must be located closer to the beginning of the JSP than the content selector tag.

Table 8-7 JSP Tags that Support Content-Selector Functions

Tag	Description
`<es:forEachInArray>`	Iterates through the array that contains the results of a content selector query. With this tag, you can use the following to work with the documents in the array: ■ The `System.out.println` method to print each item in the array. ■ The `<cm:getProperty>` tag to retrieve one or more attributes of the documents in the array. You can use the attributes to construct the HTML that a browser requires to display the documents. For example, you use the `<cm:getProperty>` tag to determine the value of a `MIME-type` attribute. If the MIME-type of a document in the array is an image, you print the HTML `<img>` tag with the appropriate attributes. ■ You can also use attributes of the `<pz:contentSelector>` tag, such as `sortBy`, to work with the attributes of documents in the array. ■ The `<cm:printProperty>` to print one or more attributes of the documents in the array. For example, you can use this tag to print a list of document titles that the content selector retrieves.

Using Content Selector Tags and Associated Tags

The combination of content selector definitions, tag attributes, and associated JSP tags creates a powerful set of tools for matching documents to customers in specific contexts. The following tasks are the most common uses of content selectors and associated tags:

- Retrieving and Displaying Text-Type Documents
- Retrieving and Displaying Image-Type Documents
- Retrieving and Displaying a List of Documents
- Accessing a Content Selector Cache on a Different JSP

For information on how WebLogic Portal's content-related JSP tags map to WebLogic Portal's content management service provider interface (SPI), see the "Personalization JSP Tags" section of the JavaServer Page Guide at http://edocs.bea.com/wlp/docs70/jsp/p13njsp.htm.

Retrieving and Displaying Text-Type Documents

To retrieve and display text-type documents, use this procedure:

Note: This procedure assumes that the content selector query created in the E-Business Control Center includes a filter to retrieve only text documents.

1. Open a JSP in a text editor.
2. Near the beginning of the JSP, add the lines shown in Listing 8-6 to import classes and tag libraries if they are not already in the JSP:

Listing 8-6 Code to Import Classes and Tag Libraries

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
```

3. Add the following tag to get the customer profile, if the tag is not already in the JSP:

   ```
   <um:getProfile>
   ```

   If the JSP already uses this tag for some other purpose, it probably includes other attributes. Make sure that the tag is closer to the beginning of the JSP than the `<pz:contentSelector>` tag, which you use in the next step.

4. Add the tags shown in Listing 8-7, where `SpringSailing` is the name of the content selector that was created in the E-Business Control Center:

Listing 8-7 Content Selector Tag Example 1

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="textDocs"/>
<es:forEachInArray array="<%=textDocs%>" id="aTextDoc"
type="com.bea.p13n.content.Content">
<p><cm:printDoc id="aTextDoc"/></p>

</es:forEachInArray>
```

Note: To verify the content type before you display it, you can surround the `<% "<P>" + aTextDoc + "</P>" %>` scriptlet with another scriptlet. Listing 8-8 shows an example:

Listing 8-8 Verifying the Content Type

```
<% if (aTextDoc.getMimeType().contains("text") != -1)
{
  %>
    <p><cm:printDoc id="aTextDoc"/></p>
<%
}
%>
```

5. Save the JSP. If you deploy the Web application as a `WAR` file, re-jar the Web application and deploy it.

   WebLogic Portal deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Portal waits for the page-check interval to expire before deploying any changes.

Retrieving and Displaying Image-Type Documents

To retrieve and display image-type documents, use this procedure:

1. Open a JSP in a text editor.

2. Near the beginning of the JSP, add the lines shown in Listing 8-9 to import classes and tag libraries if they are not already in the JSP:

Listing 8-9 Code to Import Classes and Tag Libraries if They are not Already in the JSP

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
```

3. Add the following tag to get the customer profile, if the tag is not already in the JSP:

   `<um:getProfile>`

   If the JSP already uses this tag for some other purpose, it probably includes other attributes. Make sure that the tag is closer to the beginning of the JSP than the `<pz:contentSelector>` tag, which you create in the next step.

4. Add the tags shown in Listing 8-10, where `SpringSailing` is the name of the content selector that was created in the E-Business Control Center:

Listing 8-10 Content Selector Tag Example 2

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="ImageDocs"/>

<es:forEachInArray array="<%=ImageDocs%>" id="anImageDoc"
type="com.bea.p13n.content.Content">

  <img src="ShowDoc/<cm:printProperty

  id="anImageDoc" name="identifier" encode="url"/>"

</es:forEachInArray>
```

Note: The above tags assume that the content selector query that was created in E-Business Control Center includes a filter to retrieve only image documents. To verify the content type before you display it, you can surround the `<img>` tag with a scriptlet. Listing 8-11 shows an example:

Listing 8-11 Surrounding an `<img>` Tag with a Scriptlet

```
<% if (anImageDoc .getMimeType().contains("image"))
{
%>
  <img src="ShowDoc/<cm:printProperty
  id="anImageDoc" name="identifier" encode="url"/>">
}
%>
```

5. Save the JSP. If you deploy the Web application as a `.war` file, re-jar the Web application and deploy it.

   WebLogic Portal deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Portal waits for the page-check interval to expire before deploying any changes..

Retrieving and Displaying a List of Documents

To retrieve and display a list of documents, use this procedure:

1. Open a JSP in a text editor.
2. Near the beginning of the JSP, add the lines shown in Listing 8-12 to import classes and tag libraries if they are not already in the JSP:

Listing 8-12 Code to Import Classes and Tag Libraries if They are not Already in the JSP

```
<%@ page import="com.bea.p13n.content.ContentHelper"%> <%@
taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
```

3. Add the following tag to get the customer profile, if the tag is not already in the JSP:

   `<um:getProfile>`

   If the JSP already uses this tag for some other purpose, it probably includes other attributes. Make sure that the tag is closer to the beginning of the JSP than the `<pz:contentSelector>` tag, which you create in the next step.

4. Add the tags shown inListing 8-13 , where `SpringSailing` is the name of the content selector that was created in the E-Business Control Center:

Listing 8-13 Content Selector Tags Example 3

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="docs"/>
<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">

   <li>The document title is: <cm:printProperty id="aDoc"
   name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

5. Save the JSP. If you deploy the Web application as a `.war` file, re-jar the Web application and deploy it.

   WebLogic Portal deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Portal waits for the page-check interval to expire before deploying any changes.

Accessing a Content Selector Cache on a Different JSP

To access a content selector cache on a Different JSP, use this procedure:

1. In a text editor, open the JSP page that contains the content selector tag. For example, you want to cache the results of the following tag:
   `<pz:contentSelector rule="SpringSailing" id="docs".../>`

2. Add attributes to the content selector tag as shown in Listing 8-14:

Listing 8-14 Content Selector Tag Attributes

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="docs"
useCache="true" cacheId="SpringSailingDocs"
cacheTimeout="120000"
cacheScope="application" />
```

These attributes create a cache that WebLogic Portal maintains for 2 minutes (120000 milliseconds) and that can be accessed using the name `SpringSailingDocs` by any user from any page in the Web application. For more information about possible values for cacheScope, see "Create and Configure the Cache to Improve Performance" on page 8-28.

3. Save and deploy the JSP.
4. In a text editor, open the JSP from which you want to access the cache.
5. Use a content-selector tag that is identical to the tag you created in step 2. For example, on the current JSP, add the tag shown in Listing 8-15:

Listing 8-15 Adding an Identical Tag

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="docs"
useCache="true" cacheId="SpringSailingDocs" cacheTimeout="120000"
cacheScope="application" />
```

6. Save and deploy the JSP.

Integrating External Content Management Systems

For customers who have larger amounts of content and want more control over the publishing and tagging of content, BEA partners with third-party vendors to add flexibility to WebLogic Portal. Third-party content management systems provide robust, content-creation management solutions while the Content Manager personalizes and serves the content to the end user.

Integration Strategies

BEA recommends three strategies for integrating a third-party content management system with the WebLogic Portal:

- Have the CMS publish the documents onto the file system and use the reference implementation's BulkLoader to load them into the database. This is the same process described in "Adding Content by Using the Bulk Loader" on page 8-1, except that you must ensure that the third-party CMS loads data into the appropriate location at a regularly scheduled interval. See "Adding Content by Using the Bulk Loader" on page 8-1 for complete information.
- Write an implementation of the `DocumentProvider` interface. See "Adding Content by Implementing a DocumentProvider Interface" on page 8-40.
- Have the CMS publish into the reference implementation document repository. See "Publishing to Reference Implementation" on page 8-48.

Adding Content by Implementing a DocumentProvider Interface

The DocumentProvider object is the entry point into an SPI implementation. It consists of methods that access the underlying content management system. When developing a DocumentProvider, you do not need to be concerned about transactional state or thread safety. Since a DocumentProvider does not need to perform write actions, a transaction is not required to access a DocumentProvider.

Implementing a DocumentProvider interface involves writing implementations of Java interfaces contained in the `com.beasys.p13n.content.document.spi` package. These interfaces are:

- `DocumentProvider`
- `DocumentIterator`
- `DocumentMetadataDef`
- `DocumentDef`
- `DocumentSchemaDef`

The following sections describe how these interfaces are implemented to integrate a CMS with WebLogic Portal.

For more information on these interfaces, see the Javadoc for `com.beasys.p13n.content.document.spi`.

The following steps present a high-level description of how to implement a DocumentProvider interface.

Step 1. Ensure that the CMS Meets the Minimum Use Requirements

To successfully integrate a Content Management System into WebLogic Portal, the CMS must support the capabilities listed in Table 8-8.

Table 8-8 CMS Minimum Use Requirements

Requirement	Description
Unique document Ids	There must be a single key that identifies a document as unique from all the other documents in the system, and it must be possible to represent the key as a String. For example, some content management systems assign a document an object id, while others (including the reference implementation) use the relative path.
Document Metadata	There must be a way to retrieve all the metadata about a document. The metadata must be comprised of the standard WebLogic Portal types (Boolean, Integer, Float, DateTime, String, Multi-valued), or be able to be converted to those types. This metadata must include, at a minimum, the size of the document in bytes and the MIME 1.0 mime type of the document.
Document content retrieval	There must be some way to retrieve the raw bytes of the document.
Searching	The CMS must allow some mechanism to search for documents based upon a query against the document metadata. (Full-text searches are not required in WebLogic Portal.)
Schemas	The CMS must provide a mechanism to expose the document metadata schemas. Schemas describe which of the metadata attributes and their types will be associated with which types of documents. The rules editor then uses this schema information to let you create the content selector rules that make personalization possible. (Regular document searching methods do not use the schema information, so non-personalized document retrieval is possible without schemas.)
Java access	The CMS must support some mechanism by which Java code can access the documents. These may include, but are not limited to, Java class libraries, a native shared library which JNI can access, socket based access (such as HTTP, DCOM, client-server), DBMS level access, file based access, or eLink capable access. (Java access is optional when publishing into the reference implementation supplied by WebLogic Portal.)

If all of these requirements cannot be met in some fashion, it will not be possible to fully integrate the CMS with WebLogic Portal. Additionally, since WebLogic Portal does not provide document creation and editing functionality, the CMS must have some way for users to create and edit documents.

Step 2. Write the SPI Implementation

Next, you need to code the SPI by implementing the interfaces described in Table 8-9 and the additional default and helper classes listed in "`DefaultDocumentProvider`" on page 8-43. These interfaces provide two-way communication between your Web application and the CMS by taking BEA objects and converting them to objects recognizable by the CMS.

Table 8-9 DocumentProvider Interfaces

Interface	Description
`DocumentIterator`	The DocumentIterator interface extends the java.util.Iterator interface by adding a `close()` method. The `close()` method is invoked when the DocumentIterator is no longer used. It clears any resources tied to the DocumentIterator. The methods invoked by this interface should not return null. If an exception is necessary, the methods should throw a DocumentException. If the results set is empty, than an empty DocumentIterator should be returned.
`DocumentMetadataDef`	The DocumentMetadataDef interface represents the metadata attributes of a document. This interface contains methods for retrieving both explicit and implicit (CMS defined) metadata. The `getProperty(String name)` method for retrieving implicit metadata should not respond to the explicit attributes names: identifier, size, version, author, creationDate, lockedBy, modifiedDate, modifiedBy, description, comments, and mimeType. To retrieve the explicit properties, the infrastructure calls the corresponding individual methods.

Table 8-9 DocumentProvider Interfaces

Interface	Description
`DocumentDef`	The DocumentDef interface represents the raw bytes of a document's content. Although this interface is required to implement two methods, primarily the `openStream()` method will be invoked. It is highly recommended that the InputStream returned from `openStream()` supports the `skip()` and `available()` methods in an efficient manner. The `skip()` method will be used when returning chunks of the content bytes to WebLogic Portal. The `available()` method will be used to determine how many more bytes are available from the stream.
`DocumentSchemaDef`	The DocumentSchemaDef represents a single available schema from the underlying CMS. It contains methods to query the attribute names, types, possible values, and descriptions.

For information on how WebLogic Portal's content-related JSP tags map to the WebLogic Portal SPI, see the "Personalization JSP Tags" section of the JavaServer Page Guide at http://edocs.bea.com/wlp/docs70/jsp/p13njsp.htm.

Additional DocumentProvider Classes to Implement

In addition to the interfaces listed in Table 3-1, you can use some of the abstract classes in the `com.bea.p13n.content.document.ref` package as base classes to implement some SPI functionality. These classes include:

- `DefaultDocumentProvider`
- `DefaultDocumentIterator`
- `DefaultDocumentMetadata`
- `DefaultDocumentSchema`
- `DefaultDocument`
- `FileDocument`
- `URLDocument`

Other classes that can help in developing a DocumentProvider implementation include:

- `com.bea.p13n.content.document.ref.DocumentComparator`
- `com.bea.p13n.content.expression.SortCriteria`
- `com.bea.p13n.content.expression.ExpressionHelper`
- `com.bea.p13n.content.expression.ExpressionAdapter`
- `com.bea.p13n.content.MimeTypeHelper`
- `com.bea.p13n.util.DefaultEntityResolver`
- `com.bea.p13n.util.jdbc.JdbcHelper`
- `com.bea.p13n.util.WildCard`

Note: Consult the WebLogic Portal Javadoc for further details about each class.

Implementing Search and Schema Methods

In general, integating a third-party CMS requires that you implement both search methods and schema methods.

Search methods return metadata about a document; that is, the data that determines the appearance of the data returned. You will need to determine how the search criteria passed into the search object corresponds to the search methods allowed by the CMS and then define this mapping in order for the search to work.

Schema methods return the “bytes”; that is, something that represents the data to be consumed by the Web application. Assuming that the necessary data exists in the polled CMS, these methods will return:

- The specified schema object
- A list of all schema names
- A map of schema names

Step 3. Place Code Into the Application

Once the SPI implementation is written, WebLogic Portal needs to be configured to use it. To do this, you can either:

- Modify the Existing DocumentConnectionPool

OR

- Configure a New DocumentConnectionPool and DocumentManager

Modify the Existing DocumentConnectionPool

The first method is to simply modify an existing DocumentConnectionPool in the application's `META-INF/application-config.xml`. The DocumentManager implementation uses connection pools to a specialized JDBC driver to handle searches. A deployed DocumentManager finds the document connection pool to use via either the DocumentConnectionPoolName attribute of its DocumentManager MBean or the DocumentConnectionPoolName EJB deployment descriptor setting.

Configure a New DocumentConnectionPool and DocumentManager

The second way to configure WebLogic Portal is to set up a new DocumentConnectionPool in the application's `META-INF/application-config.xml` file and to deploy a new DocumentManager EJB in the application.

First, create a new connection pool by following the procedure outlined in the "Using the WebLogic Server Administration Console to Modify DocumentManager MBeans" on page 8-10 Be sure to give the DocumentConnectionPool a unique name in the application (for example, "`myConnectionPool`").

Next, put together the new EJB by using the following procedure:

1. Unjar the `<application-directory>/document.jar` file to a temporary directory.
2. Add your implementation code to the directory appropriately.
3. In `META-INF/ejb-jar.xml`, create a new `<session>` entry based upon the DocumentManager entry. Be sure to change the `<ejb-name>` entry to a unique name; for example, in Listing 8-16, that is NewsletterDocumentManager:

Listing 8-16 Creating a New <session>

```
<!--  The Newsletter DocumentManager  -->
<session>
<ejb-name>NewsletterDocumentManager</ejb-name>
   <home>com.bea.p13n.content.document.DocumentManagerHome</home>
   <remote>com.bea.p13n.content.document.DocumentManager</remote>
   <ejb-class>com.bea.p13n.content.document.internal.
```

```
      SPIFastDocumentManagerImpl</ejb-class>
   <session-type>Stateless</session-type>

   <transaction-type>Container</transaction-type>

   <!--
    This controls which DocumentManager MBean this instance
    will look for in the application-config.xml.
   -->

   <env-entry>
      <env-entry-name>DocumentManagerMBeanName</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>newsletter</env-entry-value>
   </env-entry>
</session>
```

4. In the new `<session>` entry in `META-INF/ejb-jar.xml`, change the `<env-entry-value>` in the `<env-entry>` for DocumentManagerMBeanName to a unique name ("`NewsletterDocumentManager`" in this example). This value will be used later in the `application-config.xml` file.

5. In `META-INF/ejb-jar.xml`, add `<assembly-descriptor>` and `<container-transaction>` entries for your new `<session>` entry. You can copy the existing ones, being sure to change `<ejb-name>` to the value used above ("`NewsletterDocumentManager`" in this example). Listing 8-17 shows an example:

Listing 8-17 Adding `<assembly-descriptor>` and `<container-transaction>` Entries

```
<assembly-descriptor>
   <container-transaction>
      <method>
         <ejb-name>NewsletterDocumentManager</ejb-name>
         <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   .
   .
   .
</assembly-descriptor>
```

6. Edit `META-INF/weblogic-ejb-jar.xml` to create a new `<weblogic-enterprise-bean>` entry based upon the DocumentManager entry. Be sure to change the `<ejb-name>` entry to the name used above ("NewsletterDocumentManager" in this example). Listing 8-18 shows an example.

Listing 8-18 Creating a New `<weblogic-enterprise-bean>` Entry

```
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
      <ejb-name>NewsletterDocumentManager</ejb-name>
      <entity-descriptor>
         <persistence>
            <persistence-type>
               <type-identifier>WebLogic_CMP_RDBMS
               </type-identifier>
               <type-version>7.0</type-version>
               <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml
               </type-storage>
            </persistence-type>
            <persistence-use>
               <type-identifier>WebLogic_CMP_RDBMS
               </type-identifier>
               <type-version>6.0</type-version>
            </persistence-use>
         </persistence>
      </entity-descriptor>

   <jndi-name>${APPNAME}.BEA_portal_examples.
      NewsletterDocumentManage</jndi-name>

</weblogic-enterprise-bean>
```

7. In `META-INF/weblogic-ejb-jar.xml`, in the new `<weblogic-enterprise-bean>` entry, change the `<jndi-name>` to a desired value. This value will be the JNDI name used in EJB references to your new DocumentManager. For example:

```
<jndi-name>${APPNAME}.BEA_portal_examples.
   NewsletterDocumentManager</jndi-name>
```

8. Jar up the temporary directory back into `document.jar`.

Step 4. Make the .jar Accessible to the Application

Next, you need to make the `.jar` file you created accessible to the application accessing the CMS. You can do this by using one of these methods:

- Place the `.jar` into a `.jar` file already deployed in the application.
- Place the `.jar` into a `.jar` file referenced in the `META-INF/Manifest.mf` file's Class-Path entry in the .jar file that initializes the DocumentConnectionPool (for example, `document.jar`).
- Put the .jar name in a classpath attribute on the DocumentConnectionPool MBean in the application's `META-INF/application-config.xml` file

Step 5. Restart the Server

Because, by this point, you have made a number of configuration changes and changes to classpaths, we recommend that you restart the server.

Step 6. Apply the Portal

Finally, to complete the integration, you need to create the portal into which the CMS data will appear. For instructions on creating a portal or portlet, please refer to "Step 3: Add a Portlet" on page 2-23.

Publishing to Reference Implementation

This strategy involves directly publishing to the WebLogic Portal reference implementation database tables and XML schema files.

To implement this strategy:

1. Put the document entries and document metadata into the database tables.
2. Put the document metadata schema into WebLogic Portal XML schema files.
3. Put the document files on the file system.

Constructing Content Queries

This section, which provides guidelines for constructing queries to a content management system, includes information on the following topics.

- Structuring Queries
- Using Comparison Operators to Construct Queries
- Constructing Queries Using Java
- Using the Document Servlet

Structuring Queries

WebLogic Portal queries are similar in syntax to the SQL string syntax that supports basic Boolean-type comparison expressions, including nested parenthetical queries. In general, the query includes a metadata property name, a comparison operator, and a literal value. For example:

```
attribute_name comparison_operator literal_value
```

Note: For more information about the query syntax, see the *Javadoc* API documentation for `com.bea.p13n.content.expression.ExpressionHelper`.

The following constraints apply to queries constructed using this syntax:

- String literals must be enclosed in single quotes.
  - `'WebLogic Server'`
  - `'football'`
- Date literals can be created using a simple `toDate` method that takes one or two `String` arguments enclosed in single quotes. The first argument, if two are supplied, is the `SimpleDateFormat` format string; the second argument is the date string. If only one argument is supplied, it should include the date string in 'MM/dd/yyyy HH:mm:ss z' format.

   - `toDate('EE dd MMM yyyy HH:mm:ss z', 'Thr 08 Nov 2001 16:56:00 MDT')`
   - `toDate('02/23/2005 13:57:43 MST')`

- Use the `toProperty` method to compare properties whose names include spaces or other special characters. In general, use `toProperty` when the property name does not comply with the Java variable-naming convention that uses alphanumeric characters.
   - `toProperty ('My Property') = 'Content'`
- To include a scope into the property name, use either `scope.propertyName` or the `toProperty` method with two arguments.
   - `toProperty ('myScope', 'myProperty')`

   Note: The reference implementation document management system ignores property scopes.
- Use \ along with the appropriate character(s) to create an escape sequence that includes special characters in string literals.
   - `toProperty ('My Property\'s Contents') = 'Content'`
- Additionally, use Java-style Unicode escape sequences to embed non-ASCII characters in string literals.
   - Description like `'*\u65e5\u672c\u8a9e*'`

   Notes: The query syntax can contain only ASCII and extended ASCII characters (0-255).

   Use `ExpressionHelper.toStringLiteral` to convert an arbitrary string to a fully quoted and escaped string literal which can be put in a query.
- The *`now`* keyword, only used on the literal value side of the expression, refers to the current date and time.
- Boolean literals are either `true` or `false`.
- Numeric literals consist of the numbers themselves without any text decoration (like quotation marks). The system supports scientific notation in the forms; for example, *`1.24e4`* and *`1.24E-4`*.
- An exclamation mark (!) can be placed at an opening parenthesis to negate an expression.

- `!(keywords contains 'football') || (size >= 256)`

- The Boolean `and` operator is represented by the literal `&&`.
  - `author == 'james' && age < 55`
- The Boolean `or` operator is represented by the literal `||`.
  - `creationDate > now || expireDate < now`

The following examples illustrate full expressions:

Example 1:

```
((color='red' && size <=1024) || (keywords contains 'red' &&
creationDate < now))
```

Example 2:

```
creationDate > toDate ('MM/dd/yyyy HH:mm:ss', '2/22/2000 14:51:00')
&& expireDate <= now && mimetype like 'text/*'
```

Using Comparison Operators to Construct Queries

To support advanced searching, the system allows construction of nested Boolean queries incorporating comparison operators. Table 8-10 summarizes the comparison operators available for each metadata type.

Table 8-10 Comparison Operators Available for Each Metadata Type

Operator Type	Characteristics
Boolean (==, !=)	Boolean attributes support an equality check against `Boolean.TRUE` or `Boolean.FALSE`.
Numeric (==, !=, >, <, >=, <=)	Numeric attributes support the standard equality, greater than, and less than checks against a `java.lang.Number`.

Table 8-10 Comparison Operators Available for Each Metadata Type (Continued)

Operator Type	Characteristics
Text (==, !=, >, <, >=, <=, like)	Text strings support standard equality checking (case sensitive), plus lexicographical comparison (less than or greater than). In addition, strings can be compared using wildcard pattern matching (that is, the *`like`* operator), similar to the SQL LIKE operator or DOS prompt file matching. In this situation, the wildcards will be * (asterisk) to match any string of characters and ? (question mark) to match any single character. Interval matching (for example, using [ ]) is not supported. To match * or ? exactly, the quote character is \ (backslash).
Datetime (==, !=, >, <, >=, <=)	Date/time attributes support standard equality, greater than, and less than checks against a `java.sql.Timestamp`.
Multi-valued Comparison Operators (contains, containsall)	Multi-valued attributes support a *`contains`* operator that takes an object of the attribute's subtype and checks that the attribute's value contains it. Additionally, multi-valued attributes support a *`containsall`* operator, which takes another collection of objects of the attribute's subtype and checks that the attribute's value contains all of them. Single-valued operators applied to a multi-valued attribute should cause the operator to be applied over the attribute's collection of values. Any value that matches the operator and operand should return `true`. For example, if the multi-valued text attribute *`keywords`* has the values *BEA*, *Computer*, and *WebLogic*, and the operand is *BEA*, then the < operator returns `true` (*BEA* is less than *Computer*), the > operator returns `false` (*BEA* is not greater than any of the values), and the == operator returns `true` (*BEA* is equal to *BEA*).
User Defined Comparison Operators	Currently, no operators can be applied to a user-defined attribute.

Notes: The search parameters and expression objects support negation of expressions using a bit flag (!).

The reference implementation content management system has only single-value Text and Number properties. All implicit properties are single-value Text.

Constructing Queries Using Java

To construct queries using Java syntax instead of using the query language supplied with the Content Management component, see the *Javadoc* API documentation for `com.bea.p13n.content.expression.ExpressionHelper`.

The `ContentManager` session bean is the primary interface to the functionality of the Content Management component. Using a `ContentManager` instance, content is returned based on a `com.bea.p13n.content.expression.Search` object with an embedded `com.bea.p13n.expression.Expression`, which represents the expression tree.

In the expression tree, the following caveats apply for it to be valid for the ContentManager:

- Each branch node can only be of the following types. Any other branch node type is invalid.

```
com.bea.p13n.expression.operator.logical.LogicalAnd,
com.bea.p13n.expression.operator.logical.LogicalOr,
com.bea.p13n.expression.operator.logical.LogicalMulitAnd, or
com.bea.p13n.expression.operator.logical.LogicalMultiOr.
```

- Each leaf node can only be of the following types. Any other branch node type is invalid.

```
com.bea.p13n.expression.operator.comparative.Equals,
com.bea.p13n.expression.operator.comparative.GreaterOrEquals,
com.bea.p13n.expression.operator.comparative.GreaterThan,
com.bea.p13n.expression.operator.comparative.LessOrEquals,
com.bea.p13n.expression.operator.comparative.LessThan,
com.bea.p13n.expression.operator.comparative.NotEquals,
com.bea.p13n.expression.operator.string.StringLike,
com.bea.p13n.expression.operator.collection.CollectionContains, or
com.bea.p13n.expression.operator.collection.CollectionsContainsAll
```

- Any valid branch or leaf node may be contained in a `com.bea.p13n.expression.operator.logical.LogicalNot` node.
- In each leaf node, the left side will always be a `com.bea.p13n.content.expression.PropertyRef` node, which must contain `Strings` for `getPropertySet()` and `getPropertyName()`.

- The right side of these leaf nodes can be a `java.util.Collection, Long, Double, String,` or `java.sql.Timestamp`:
  `com.bea.p13n.expression.operator.comparative.Equals, com.bea.p13n.expression.operator.comparative.NotEquals, com.bea.p13n.expression.operator.comparative.GreaterOrEquals, com.bea.p13n.expression.operator.comparative.GreaterThan, com.bea.p13n.expression.operator.comparative.LessOrEquals, com.bea.p13n.expression.operator.comparative.LessThan, or com.bea.p13n.expression.operator.collection.CollectionContains`
- The right side of `com.bea.p13n.expression.operator.string.StringLike` leaf nodes can be a `String`. Anything else is invalid.
- The right side of `com.bea.p13n.expression.operator.collection.CollectionsContainsAll` leaf nodes can be a `java.util.Collection`. Anything else is invalid.

Using the Document Servlet

The Content Management component includes a servlet capable of outputting the contents of a `Document` object. This servlet is useful when streaming the contents of an image that resides in a content management system or to stream a document's contents that are stored in a content management system when an HTML link is selected. Table 8-11 shows the Request/URL parameters that the servlet supports.

Table 8-11 Request Parameters Supported by the Document Servlet

Request Parameter	Required	Description
`contentHome`	Maybe	If the *`contentHome`* initialization parameter is not specified, then this is required and will be used as the JNDI name of the DocumentHome. If the *`contentHome`* initialization parameter is specified, this is ignored.
`contentId`	No	The string identifier of the Document to retrieve. If not specified, the servlet looks in the `PATH_INFO`.

Table 8-11 Request Parameters Supported by the Document Servlet (Continued)

Request Parameter	Required	Description
`blockSize`	No	The size of the data blocks to read. The default is 8K. Use 0 or less to read the entire block of bytes in one operation.

The servlet supports only `Documents`, not other subclasses of `Content`. It sets the *`Content-Type`* to the `Document's` mimeType and, the *`Content-Length`* to the `Document's` size, and correctly sets the *`Content-Disposition`*, which should present the correct filename when the file is saved from a browser.

Example 1: Usage in a JSP

This example searches for news items that are to be shown in the evening, and displays them in a bulleted list.

```
<cm:select sortBy="creationDate ASC, title ASC"

query=" type = ‘News’ && timeOfDay = ‘Evening’ && mimeType like
‘text/*’ "id="newsList"/>

<ul>

<es:forEachInArray array="<%=newsList%>" id="newsItem"
type="com.bea.p13n.content.Content">

     <li><a href="ShowDoc/<cm:printProperty id="newsItem"
     name="identifier" encode="url"/>"><cm:printProperty
     id="newsItem" name="title" encode="html"/></a>

  </es:forEachInArray>

</ul>
```

Example 2: Usage in a JSP

This example searches for image files that match keywords that contain *bird* and displays the image in a bulleted list.

```
<cm:select max="5" sortBy="name" id="list"

query=" KeyWords like ‘*birds*’ && mimeType like ‘image/*’ "
```

```
contentHome="java:comp/env/ejb/MyDocumentManager"/>

<ul>

<es:forEachInArray array="<%=list%>" id="img"
type=”com.bea.p13n.content.Content”>

   <li><img src="/ShowDoc/<cm:printProperty id="img"
   name="identifier"
   encode="url"/>?contentHome=<es:convertSpecialChars
   string="java:comp/env/ejb/MyDocumentManager"/>">

<es:forEachInArray>

</ul>
```

CHAPTER

9 Setting Up Portal Navigation

Portal navigation is achieved through the use of Webflow, a mechanism designed to help you build Web applications and maintain a separation between presentation logic and underlying business processes. When the visitor causes an event, such as clicking **Next** on a page, Webflow determines what the visitor will see next. At appropriate times during a visitor's interaction, the Webflow may also invoke Pipelines, predefined, specialized components used to validate data or to execute back-end business processes.

Because the Webflow's centralized XML configuration files specify the order in which pages are displayed to your Web site's visitors, use of the Webflow mechanism may reduce the amount of work necessary to create and modify the flow of your Web site.

This section contains information on the following subjects:

- Building a Webflow
- Creating a Pipeline and Adding it to a Webflow
- Synchronizing the Webflow to the Application
- Creating a New Input Processor
- Extending Webflow by Creating Extension Presentation and Processor Nodes

Building a Webflow

This section shows you how to build a basic Webflow by adding the necessary nodes to the Webflow and connecting those nodes to each other with transitions.

To build a Webflow, you need to complete the following steps:

- Step 1. Create the Webflow
- Step 2. Add Nodes to the Webflow Canvas
- Step 3. Identify the Begin Node
- Step 4. Create Transitions Between Nodes

Completing these steps will provide you with a rudimentary Webflow. Subsequent information in this section will complete the Webflow-building process by showing you how to:

- Create and add a Pipeline to the Webflow (see "Creating a Pipeline and Adding it to a Webflow" on page 9-22)
- Synchronize the Webflow to the application (see "Synchronizing the Webflow to the Application" on page 9-36)

Note: While the procedures outlined here and in subsequent parts of this section imply a specific sequence for creating Webflows and Pipelines, this is done only to facilitate the document and is not required. You can follow whatever sequence is most accommodating to your development needs. For example, you might want to create Pipelines for a Webflow before actually creating the Webflow. Do note that you cannot synchronize a Webflow to an application until the Webflow exists.

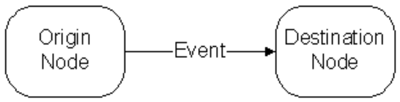
Understanding Webflow Components

Before attempting to build and implement a Webflow, you should understand some of the Webflow components you will be using in the ensuing procedures.

Nodes and Transitions

Nodes are the graphical representation of the functionality of a state in the Webflow. Depending on the node type, there are a number of predefined **events** that may occur (such as a visitor clicking a link on a Web page). When a particular event happens, the Webflow decides which subsequent node to invoke to continue the flow. This process is referred to as a **transition**, and is illustrated in Figure 9-1.

Figure 9-1 Generic Webflow Transition



Note that, as shown in Figure 9-1, nodes can be referred to as origin or destination nodes, depending on their location in a transition.

Types of Nodes

There are two main types of nodes: presentation nodes and processor nodes. Each of the presentation and processor nodes can be used as origin or destination nodes within the Webflow.

Presentation Nodes

Presentation nodes represent states in which the Webflow presents or displays something to a person interacting with the Web application. The form of the presentation can be:

- HTML
- JavaServer Page (JSP)
- Java servlets

You can also create extension (custom) presentation nodes for use in the Webflow. For more information about extension presentation nodes, see "How to Create an Extension Presentation Node" on page 9-40.

Processor Nodes

Processor nodes represent states in which the Webflow invokes more specialized components to handle such activities as form validation or the back-end business logic that drives the site's presentation. The processor nodes available for use are described in Table 9-1.

Table 9-1 Webflow Processor Node Types

Processor Node Type	Description
Input Processors	Input Processors are predefined, specialized Java classes that carry out more complex tasks when invoked by the Webflow mechanism. Input Processors are typically used to validate HTML form data, or to provide conditional branching within a Web page. For example, an Input Processor may contain code that verifies whether a date has been entered in the correct format, as opposed to embedding that code within the same JSP that displays the form fields. Input Processors contain logic that is specific to the Web application, and are therefore loaded by the Web application's container.
Pipelines	A Pipeline is also a type of processor node that may be invoked by the Webflow. Pipelines initiate the execution of specific tasks related to your business process, and can be transactional or nontransactional. For example, if a visitor attempts to move to another page on your Web site but you want to persist the visitor's information to a database first, you could use a Pipeline. Pipelines contain business logic that may apply to multiple Web applications within a larger enterprise application, and are therefore loaded by the Enterprise Java Bean (EJB) container. All Pipelines are collections of individual Pipeline Components, which can be implemented as Java objects or stateless session Enterprise JavaBeans (EJBs). Pipeline Components are the parts of a Pipeline that actually perform the tasks associated with the underlying business logic. When these tasks are complex, Pipeline Components may also make calls to external services (other business objects).
Extension Processor Nodes	Extension (custom) processor nodes for use in the Webflow. For more information about extension processor nodes, see "Extending Webflow by Creating Extension Presentation and Processor Nodes."

Wildcard Nodes

If the Webflow cannot locate a specific presentation or processor node to complete a transition, the Webflow will search for a *wildcard* presentation or processor node to use as the origin node. Therefore, wildcard presentation nodes and wildcard processor nodes implement default behavior for your Web application. Put another way, wildcard nodes allow you to abstract common functionality and to locate that functionality in a single place in your Webflow. Use Wildcard nodes only when you haven't explicitly defined destination nodes in the Webflow. You may have one wildcard presentation node and one wildcard processor node per namespace.

An example of a Wildcard node might be when you want a link called **Help** (which is present on every page) to always point to a JSP containing help information. To do so, you could use a wildcard presentation origin. Further, you might always want exceptions returned from processor nodes to transition to JSP containing detailed information about the error. You could handle both of these situation with a wildcard processor node.

Note: A slight impact on performance might occur if the Webflow must search for a wildcard node, as more processing is involved.

Types of Transitions

There are two types of transitions: event and exception.

- An event transition represents the processing logic between two nodes when an event occurs on one of them and that event is successful.
- An exception transition represents the processing logic that occurs when activity on a processor node fails and an some sort of exception must be thrown. These transitions usually direct the transaction.

For more information on the events that cause transitions, see "Types of Events" on page 9-5.

Types of Events

Each node in a Webflow responds to events, which cause transitions (that is, movement from an origin node to a destination node). However, the types of events a node responds to depends on whether the node is a presentation node or a processor node.

Presentation nodes respond to the following events:

- Links
- Buttons

In other words, when a visitor to the Web site clicks a link or a button, the Webflow responds to that event. A response might be to transition to another presentation node (such as a JSP) or to a processor node (such as an Input Processor to validate visitor-provided form data).

Processor nodes respond to the following events:

- Exceptions
- Return objects

Exceptions occur when an Input Processor or Pipeline does not execute properly, and indicates an error state. Otherwise, these processor nodes return an object that the Webflow can use to continue processing.

Note: Webflow used in portal applications may respond to more events than those described above.

Encoding Webflow URLs

New context parameters ENCODE_URLS, ESCAPE_URLS, ENCODE_STATIC_URLS, and ESCAPE_STATIC_URLS have been added for 70sp5 that control the default setting for URL escaping and encoding in webflow URLs. See WebflowJSPHelper Javadoc for more information regarding escaping and encoding of URLs.

ENCODE_URLS is the default setting indicating whether webflow URLs should be encoded. Allowable param-values are true and false. If not set the default, default is true to ensure backward compatibility.

```
<context-param>
  <param-name>ENCODE_URLS</param-name>
  <param-value>false</param-value>
</context-param>
```

ESCAPE_URLS is the default setting indicating whether webflow URLs should be escaped. Allowable param-values are NO_URL_ESCAPE, ESCAPE_URL, CALCULATE_ESCAPE. If not set the default, default is URL_ESCAPE to ensure backward compatibility.

```
<context-param>
  <param-name>ESCAPE_URLS</param-name>
  <param-value>CALCULATE_ESCAPE</param-value>
</context-param>
```

The above 2 defaults are for the following tags and also for the WebflowJSPHelper createWebflowURL methods that do not have escape or encode as a parameter:

```
<wf:createWebflowURL/>

<wf:form/>

<wf:validatedForm/>

<portal:createWebflowURL/>

<portal:form/>

<portal:validatedForm/>

<portal:createPortalPageChangeURL/>

<portlet:createWebflowURL/>

<portlet:form/>

<portlet:validatedForm/>

<portlet:createPortletEditURL/>

<portlet:createPortletUneditURL/>

<portlet:createPortletMinimizeURL/>

<portlet:createPortletUnminimizeURL/>

<portlet:createPortletMaximizeURL/>

<portlet:createPortletUnmaximizeURL/>

<portlet:createPortletFloatURL/>
```

ENCODE_STATIC_URLS it the default setting indicating whether static URLs should be encoded. Allowable param-values are true and false. If not set the default, default is true to ensure backward compatibility.

```
<context-param>

   <param-name>ENCODE_STATIC_URLS</param-name>

   <param-value>false</param-value>

</context-param>
```

ESCAPE_STATIC_URLS is the default setting indicating whether static URLs should be escaped. Allowable param-values are NO_URL_ESCAPE, ESCAPE_URL, CALCULATE_ESCAPE. If not set the default, default is URL_ESCAPE to ensure backward compatibility.

```
<context-param>

   <param-name>ESCAPE_STATIC_URLS</param-name>

   <param-value>CALCULATE_ESCAPE</param-value>

</context-param>
```

The above 2 defaults are for the <wf:createResourceURL/> tag and the WeblfowJSPHelper createStaticResource methods that do not have escape or encode as a parameter.

URLs need to be encoded if you wish to maintain session state and the browser does not accept cookies. URLs will only need to be encoded if the browser does not accept cookies.

URLs must be escaped if they will contain any characters (with some exceptions - see below) that would be encoded using java.net.URLEncoder.encode(). Note however that even when escaping is on the entire URL is not encoded rather the URL will be tokenized using the characters ':', '/', '?', '=', and '&' then the substrings between the tokens are encoded using java.net.URLEncoder.encode(). The tokenizing is necessary so that the URL will still be recognized by the Webflow engine.

Note that escaping is relatively costly and should be avoided if possible but is required if the URL might have any characters other then those ignored by java.net.URLEncoder.encode(), that is any characters other than 'a' through 'z', 'A' through 'Z, '0' through '9', '-', '_', '.', or '*' with the exception of the tokenizing characters mentioned above. If the content of the URL will not be determined until runtime and might contain "illegal" characters you should either have escaping on or use the calculate feature. Calculate should be used with care. For sites that have a small number of URLs that will need escaping using calculate rather then always escaping will result in a performance improvement. But, since calculate first checks the URL then encodes if needed, sites where most URLs need escaping will have poorer performance using calculate rather then escaping all URLs.

Webflow Tools and Buttons

You create Webflows by using the Webflow Editor, as depicted in Figure 9-4. Most activity in which you engage will require selecting either a tool button or a command button. Table 9-2 describes the tool buttons available and Table 9-3 describes the command buttons.

Table 9-2 Webflow Editor Tools

Tool	Function	Description
↖	Selection Tool	Allows you to select and move nodes, event transitions, and exception transitions. Also allows you to add elbows to transitions. This is the default tool for the Webflow Editor. **Note:** This tool will stay selected until you select another one.
⊏	Event Tool	Allows you to add an event transition between two nodes, or a self-referring event transition. **Note:** This tool will stay selected until you select another one.
⊏	Exception Tool	Allows you to add an exception transition between two nodes, or a self-referring exception transition. **Note:** This tool will stay selected until you select another one.
🗎	Presentation Node	Allows you to add a new Presentation Node to the Editor canvas.

Table 9-2 Webflow Editor Tools (Continued)

Tool	Function	Description
	Wildcard Presentation Node	Allows you to add a new Wildcard Presentation Node to the Editor canvas.
	Input Processor Node	Allows you to add a new Input Processor Node to the Editor canvas.
	Pipeline Node	Allows you to add a new Pipeline Node to the Editor canvas.
	Wildcard Processor Node	Allows you to add a new Wildcard Processor Node to the Editor canvas.
	Extension (Custom) Processor Node	Allows you to add a new Extension (Custom) Processor Node to the Editor canvas. **Note:** The Extension (Custom) Processor Node tool is disabled until the Webflow Editor detects a new node in the `webflow-extensions.wfx` file.
	Proxy Node	Allows you to add a Proxy Node to the Editor canvas. You should create a Proxy Node any time you want to refer to a node that is defined in another namespace.

Table 9-3 Webflow Command Buttons

Tool	Function	Description	Keyboard Shortcut
	Print button	Allows you to print the entire Webflow namespace or Pipeline to a printer.	Ctrl+P
	Delete button	Deletes the selected Webflow component. The Delete button is disabled until a Webflow component is selected.	Delete

Table 9-3 Webflow Command Buttons (Continued)

Tool	Function	Description	Keyboard Shortcut
	Zoomed Overview button	Allows you to view the entire Webflow namespace or Pipeline at a glance.	Ctrl+Z
	Validate the Selected Node button	Allows you to run the Webflow validation feature on the selected node. The Validate the Selected Node button is disabled until a Webflow component is selected.	Ctrl + V
	Validate All button	Allows you to run the Webflow Editor's validation feature on the entire Webflow namespace, or the Pipeline Editor's validation feature on the entire Pipeline.	Alt + V
	Set Up Configuration Error Page Name button	Allows you to specify the name and path to the configuration error page. **Note:** Only available in the Webflow Editor.	--
	Pipeline Component Editor button	Opens the Pipeline Component Editor, which allows you to manage Pipeline Components. **Note:** Only available in the Pipeline Editor.	--

Step 1. Create the Webflow

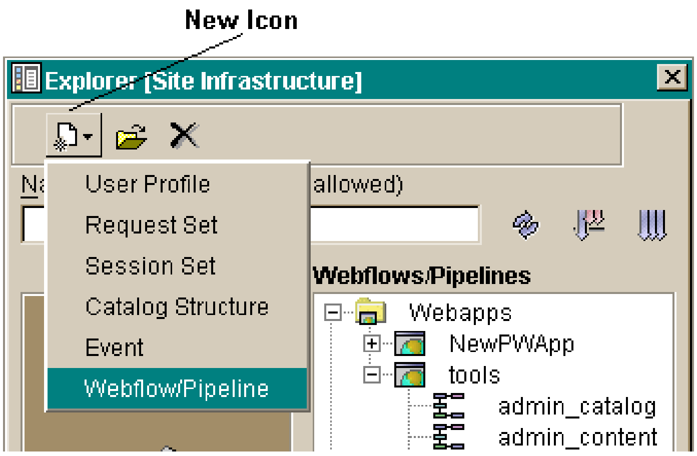
To create a Webflow, use this procedure.

Note: This procedure assumes the following:

- You have either opened an existing project or have created a new one.
- The E-Business Control Center is running.
- The **Site Infrastructure** tab is selected.

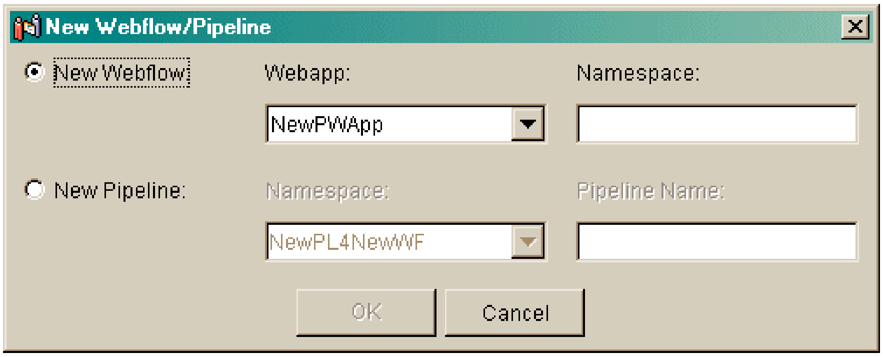
1. Click the New icon to open the drop-down menu and select **Webflow/Pipeline**, as shown in Figure 9-2.

Figure 9-2 E-Business Control Center with New Drop-Down Menu Open



The New Webflow/Pipeline dialog box appears.

Figure 9-3 New Webflow/Pipeline Dialog Box



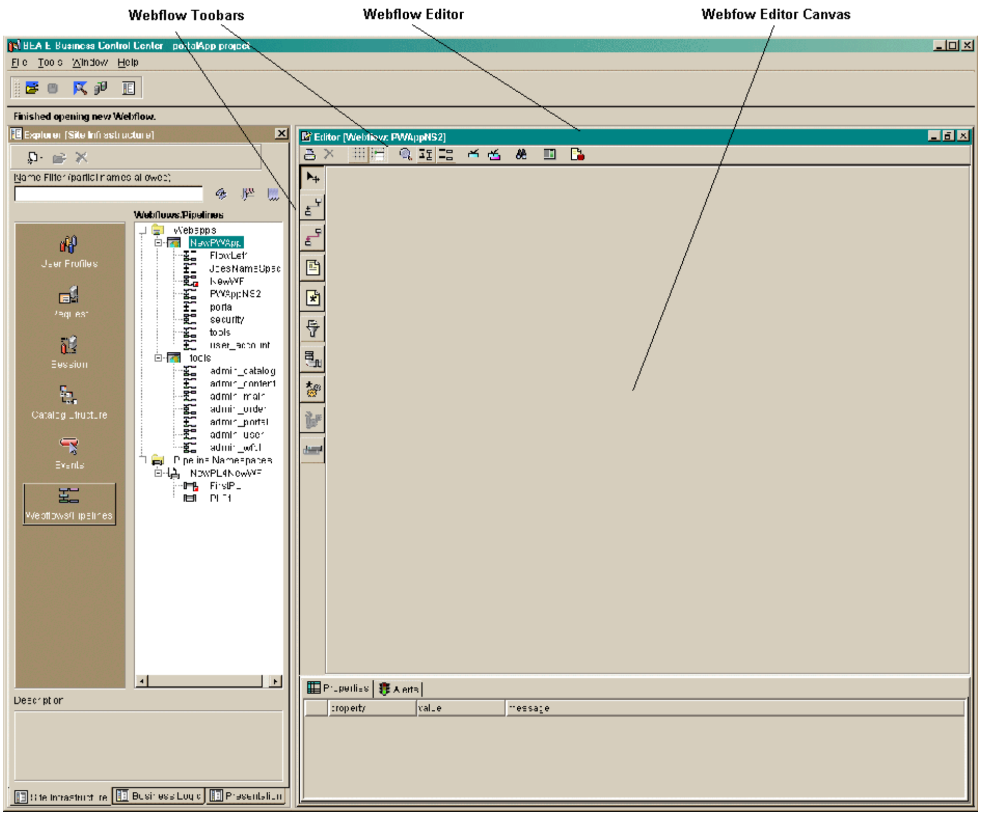
2. Ensure that **New Webflow** is selected, then open the **Webapp:** drop-down menu and select the Web application for which you are creating the webflow (this value might already be selected). Selecting a Web application is important because Webflows are scoped to a Web application.

3. Type in a Namespace name in the **Namespace** edit box. A namespace is used to scope a Webflow so that multiple Webflows can be used in a single Web application without conflicting. Note that each Web application can have multiple namespaces.

4. Click **OK**.

   The Webflow Editor appears in the right-hand pane of the E-Business Control Center. Figure 9-4 shows an example.

Figure 9-4 Webflow Editor



Step 2. Add Nodes to the Webflow Canvas

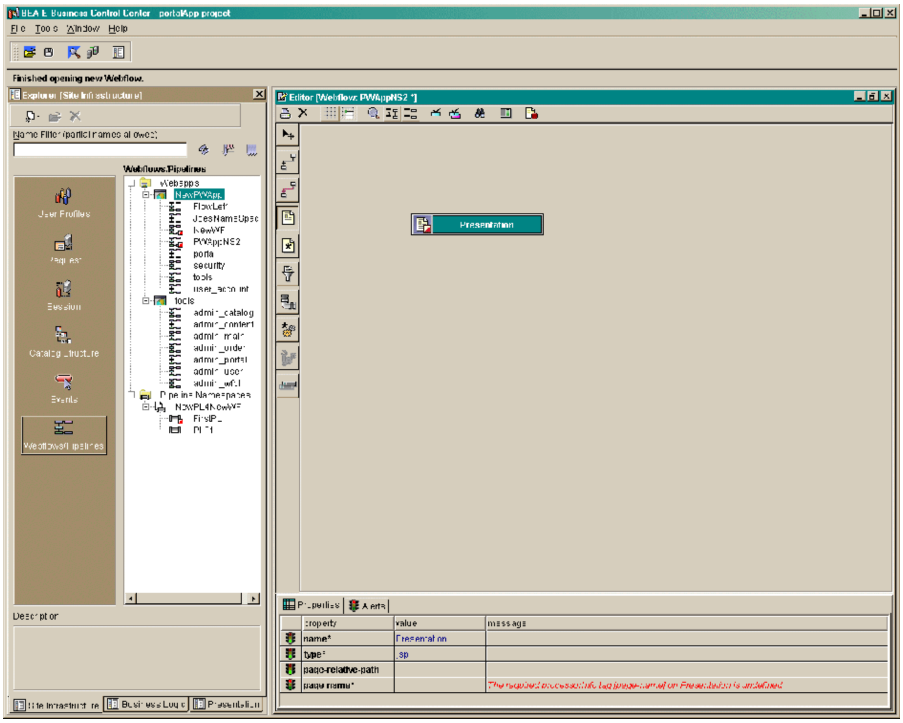
A Webflow is composed of two types of nodes: presentation nodes and processor nodes. Each of the presentation and processor nodes can be used as origin or destination nodes within the Webflow. For more information on Webflow nodes, see "Understanding Webflow Components" on page 9-2.

Add the first node to the Webflow Editor canvas by doing the following:

1. Select the appropriate tool and click the pointer (cross-hairs) anywhere on the canvas. Table 9-2 describes each of the tools in the Webflow Editor palette.

   The node appears on the canvas where you click; for example, if you want the first node to be a Presentation node (that is, a JSP, HTML file, or other presentation file), you would select the Presentation Node tool and then click over the canvas to place the node there.

Figure 9-5 Placement of a Presentation Node

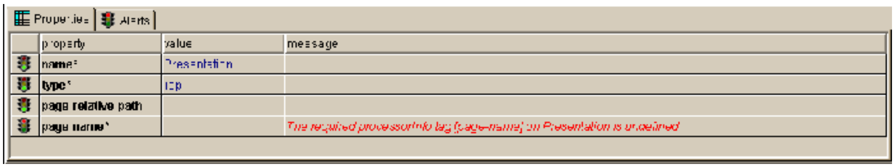


	property	value	message
	name*	Presentation	
	type*	jsp	
	page-relative-path		
	page name*		The required processor/info tag (page-name) on Presentation is undefined

Note: The Presentation Tool stays selected. This enables you to add as many Presentation nodes as necessary without reselecting the tool for each node. Also, you needn't worry about where on the canvas you place node. You can move it wherever you want by selecting the node and dragging it to the preferred location.

When you place a node on the canvas, the node's property editor, shown in Figure 9-6, opens in a pane below the canvas. You can also open the property editor for a node by clicking the node itself.

Figure 9-6 Presentation Node Property Editor



	property	value	message
	name*	Presentation	
	type*	jsp	
	page relative path		
	page name*		The required processorInfo tag [page-name] on Presentation is undefined

2. In the property editor, type the following information, as necessary:

Table 9-4 Property Editors

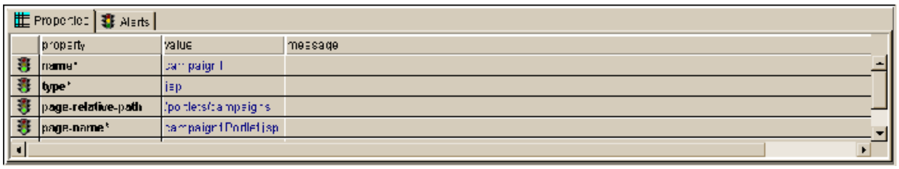
Property	Description
name	The name of the node. This value will appear in the node on the canvas. This value will auto-fill when you fill out the **page-name** field described below.
type	The type of file the node references. For a presentation node, valid types are: ■ **portal** ■ **jsp** ■ **html/htm** ■ **servlet**
page-relative-path	The relative path to the file the node references.

Table 9-4 Property Editors

Property	Description
page-name	The filename that the node references. The value entered here will be used to fill in the **name** property, although without the file extension (which is populated automatically, based upon what was selected for type); for example, if you enter `pres01` here and select **jsp** as the type, when you tab out of this field, `.jsp` will be appended to the page-name and the **name** value will change to `pres01`.

When completed, the Property Editor might look like the sample shown in Figure 9-7

Figure 9-7 Sample Completed Property Editor for a Presentation Node



	property	value	message
	name*	campaign1	
	type*	jsp	
	page-relative-path	/portlets/campaigns	
	page-name*	campaign1Portlet.jsp	

3. Continue adding nodes to the canvas until you've added all the nodes necessary for the webflow.

Step 3. Identify the Begin Node

The begin node is used as the starting point for the visitor's interaction with the application. It is designated as the initial entry point or state of the Webflow, which automatically transitions to a presentation or processor node. The begin node is generally a presentation node in the form of a JSP.

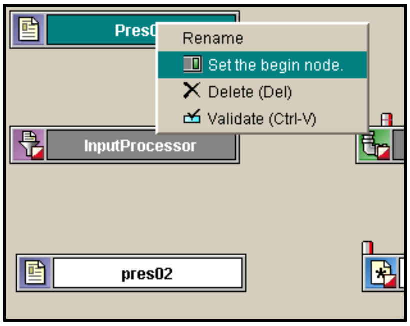
If a URL does not specify an origin, namespace, or event, the Webflow mechanism looks for a begin node in the default namespace. Although the begin node is optional, BEA recommends that you have at least one defined in your default namespace.

To identify a node as the begin node, use this procedure:

1. Right-click the node you want to assign as the Begin Node.

The node's Context menu appears, as shown in Figure 9-8.

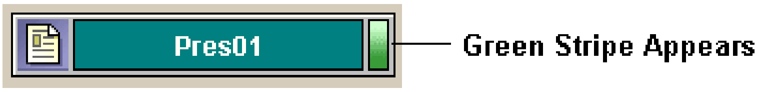
Figure 9-8 Presentation Node Context Menu



2. Select **Set the begin node**.

   The node is marked by a green stripe to the right of the node name, as shown in Figure 9-9.

Figure 9-9 Appearance of the Begin Node



Note: A webflow can have only one begin node.

Step 4. Create Transitions Between Nodes

With all the nodes placed on the canvas, connect them by creating transitions between them. Create these transitions by using the Event Tool and the Exception Tool, as shown in Table 9-2. The Exception Tool is red and the Event Tool is black.

Adding an Event Transition

To add an Event transition, do the following:

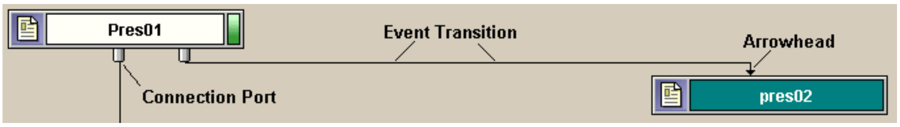
1. Click the Event Tool to activate it.
2. Place the pointer on the edge of the node from which transition will begin.

If the connection is permissible, a small orange square will appear on the edge of the node.

3. Drag the pointer to the node that marks the end of the transition.

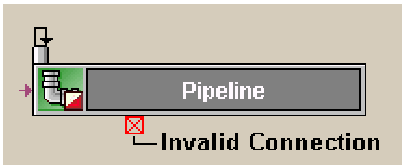
   A grey connection port will appear on the node at the transition origin site and, If the connection is permissible, an arrowhead will appear where the transition ends, as shown in Figure 9-10.

Figure 9-10 Node-to-Node Event Transition



If the connection is not allowed (for example if you are trying create an exception link on a presentation node), a red square filled with a red **X** will appear on the node's edge, as shown in Figure 9-11.

Figure 9-11 Invalid Connection Indicator



4. Continue adding Event transitions as necessary to connect all nodes designed to respond to events.

Adding an Exception Transition

Use the Exception Tool to connect any nodes that might require exception handling to the node that will do that handling. For example, an input processor might accept data that needs to be validated before it can be passed. If that data contains an error, it might need to throw an exception. The Exception Tool lets you process the exception and display any results or other information. Note that the transition link between the node appears in red.

To add an Event transition, do the following:

1. Click the Event Tool to activate it.

2. Place the pointer on the edge of the node from which transition will begin.

   If the connection is permissible, a small orange square will appear on the edge of the node.

3. Drag the pointer to the node that marks the end of the transition.

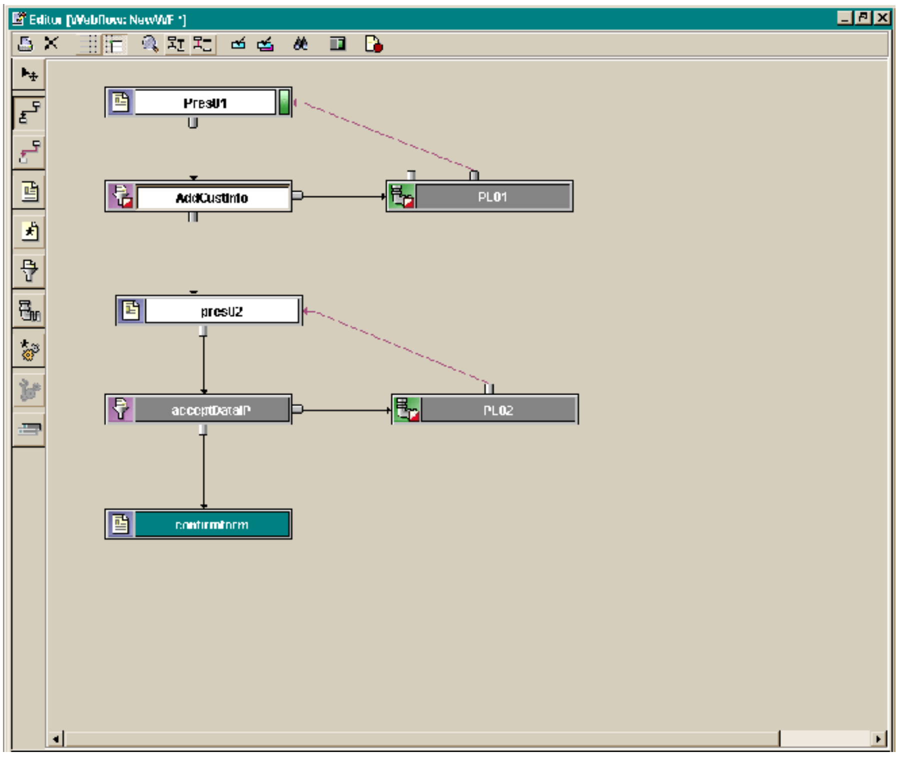
   A grey connection port will appear on the node at the transition origin site and, If the connection is permissible, an arrowhead will appear where the transition ends.

   If the connection is not allowed (for example if you are trying create an exception link on a presentation node), a red square filled with a red **X** will appear on the node's edge, as shown in Figure 9-11.

4. Continue adding Exception transitions as necessary to connect all nodes designed to respond to exceptions.

With all nodes in place and connected by the proper transitions, the webflow might appear as the one shown in Figure 9-12.

Figure 9-12 Sample Webflow Layout



Using the Transition Tools

In addition to adding a transition (as described in "Adding an Event Transition" on page 9-17 and "Adding an Exception Transition" on page 9-18), you can also move an existing transition's connection port, add elbows to transitions, and delete a transition.

Moving a Connection Port Connection ports accepting transitions are called input connection ports; connection ports where transitions originate are called output connection ports. In some cases, it may be helpful to move the node's connection port. To reposition the connection port on a node, follow these steps:

1. Select the Transition tool or the Selection tool.
2. Click and hold the mouse button on the connection port, then drag the connection port to the desired location on the node.

3. Release the mouse button to place the connection port in the new location.

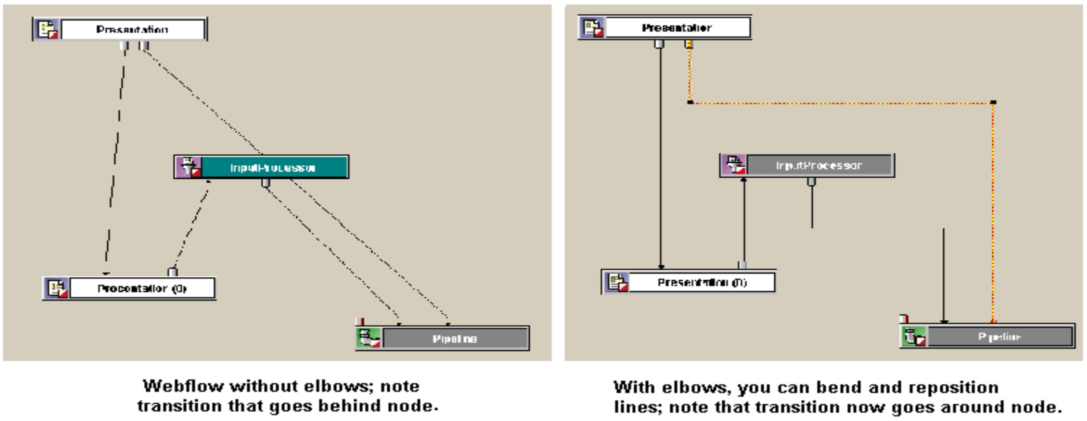
The connection port associated with a self-referring transition can only be moved along the same node edge.

Moving a Transition to Another Node You can move the end point of a transition (indicated by an arrowhead) from one node to another (assuming that the connection is allowable). Do the following:

1. Select the Transition tool or the Selection tool.
2. Select the end point (arrowhead) you want to move.
3. Holding down the left mouse button, drag the arrowhead to the node to which you want to connect it.
4. Release the mouse button.

Creating, Moving, and Deleting Elbows in Transition Lines You can also reposition transition lines on an Editor's canvas by moving, creating or deleting elbows. **Elbows** allow you to bend portions of the transition line, as shown in Figure 9-13, to enhance the appearance of the flow.

Figure 9-13 Elbows in Transitions


Presentation
InputProcessor
Presentation (0)
Pipeline
Webflow without elbows; note transition that goes behind node.
Presentation
InputProcessor
Presentation (0)
Pipeline
With elbows, you can bend and reposition lines; note that transition now goes around node.

To create a new elbow, follow these steps:

1. Single-click a transition to view the existing elbows, which appear as black squares.

2. Click and hold the mouse button anywhere on the transition line (except on an existing elbow) and drag the mouse to add the new elbow. The selected elbow appears as an orange square.
3. Release the mouse button to create the elbow in that location.

To move an elbow in an existing transition, follow these steps:

1. Single-click a transition to view any existing elbows, which appear as black squares.
2. Click and hold the mouse button on the elbow as you drag it to the desired position. The selected elbow appears as an orange square.
3. Release the mouse button to place the elbow in the new location.

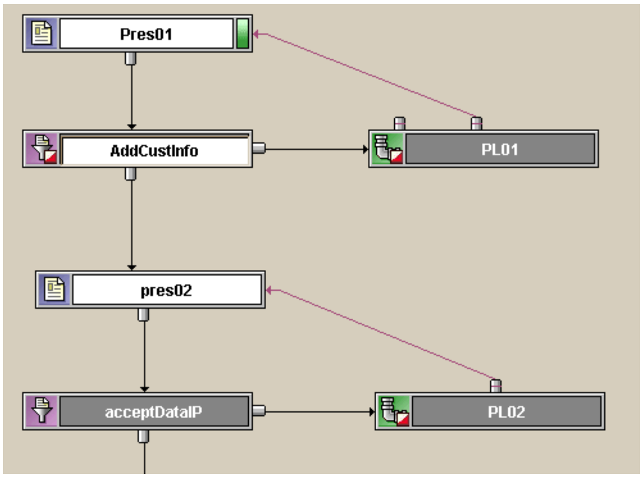
To delete an existing elbow, follow these steps:

1. Single-click a transition to view the existing elbows, which appear as black squares.
2. Click the elbow you want to delete to select it. The selected elbow appears as an orange square.
3. Click the Delete button on the Editor's toolbar, or press the Delete key.

Creating a Pipeline and Adding it to a Webflow

Figure 9-14 is a detailed look at a Webflow. In this example, you will see that two nodes are labeled PL01 and PL02. These nodes are called Pipelines.

Figure 9-14 Webflow Example: Detail Showing Pipeline Nodes



A **Pipeline** is a type of processor node that is typically used in a Webflow to execute back-end business logic. Each Pipeline is comprised of a number of Pipeline Components that perform specific tasks. BEA provides a number of Pipeline Components that are packaged with the WebLogic Portal product suite that you may want to reuse in your own Pipelines. However, you may also want to create your own Pipeline Components to execute your organization's specific business processes.

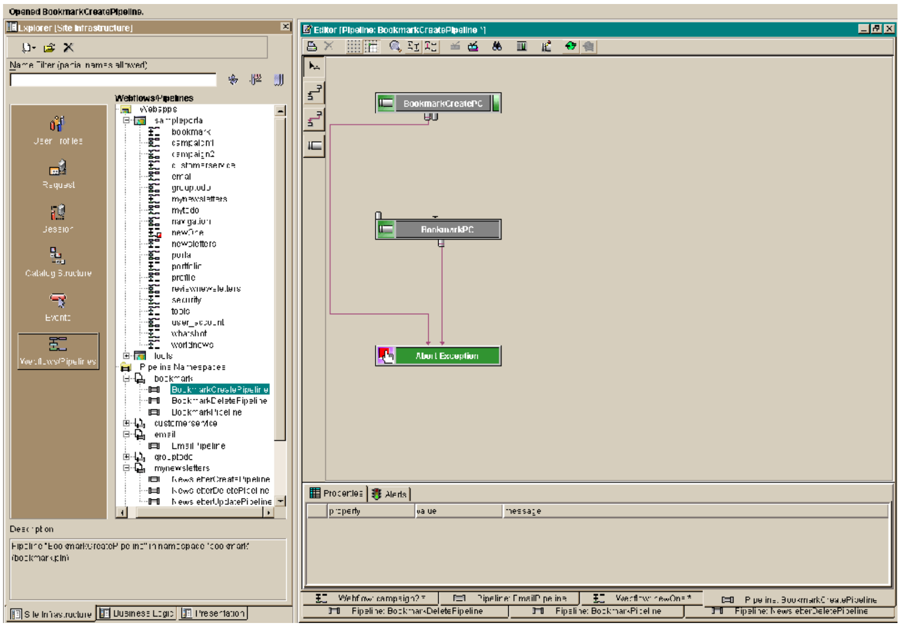
This section will show you how to create a pipeline by following these steps:

- Step 1: Create a New Pipeline Component
- Step 2: Add the New Pipeline Component to the Webflow

Understanding the Pipeline Editor

As with Webflows, you create a Pipeline and add functionality to it by using the Pipeline Editor in the E-Business Control Center, as shown in Figure 9-15 (see also Figure 9-18).

Figure 9-15 E-Business Control Center with Pipeline Editor Displayed



The Pipeline Editor has a set of display and behavior buttons and a set of command buttons along its top border and a set of tool buttons along its left border. Table 9-5 describes the tool buttons, Table 9-6 describes the display, and behavior buttons and Figure 9-7 describes the command buttons.

Table 9-5 Pipeline Editor Tools

Tool	Function	Description
↖	Selection Tool	Allows you to select and move Pipeline Components, event transitions and exception transitions. This is the default tool for the Pipeline Editor. **Note:** This tool will stay selected until you select another one.
⤴	Event Tool	Allows you to add an event transition between two nodes. **Note:** This tool will stay selected until you select another one.

Table 9-5 Pipeline Editor Tools

Tool	Function	Description
	Exception Tool	Allows you to add an exception transition between two nodes, or a self-referring exception transition. **Note:** This tool will stay selected until you select another one.
	Begin Node	Allows you to designate one of the Pipeline Components already on the Editor canvas as the Begin Node for the current Pipeline.
	Pipeline Component	Allows you to add new Pipeline Components to the Editor canvas.

Table 9-6 Pipeline Display and Behavior Buttons

Tool	Function	Description
	Show/Hide Grid button	Allows you to show or hide a grid background in the Editor canvas.
	Snap to Grid button	Allows you to control whether or not Webflow components are automatically placed to the nearest grid point on the Editor canvas when you release the mouse button.
	Link Optimization button	Allows you to control whether or not the connectors on each node will be automatically moved around the perimeter of the node as the node is moved on the Editor canvas.
	Show/Hide Exceptions button	Allows you to show or hide exception transitions in the Editor canvas.
	Make This Pipeline Transactional button	Allows you to specify whether the Pipeline should be transactional. **Note:** Only available in the Pipeline Editor.

Table 9-6 Pipeline Display and Behavior Buttons (Continued)

Tool	Function	Description
	Include Pipeline Session in Transaction button	Allows you to specify whether the Pipeline Session should be included in the transaction. Enabled only if the Pipeline Transaction button is on. **Note:** Only available in the Pipeline Editor.

The Pipeline Editor uses the same command buttons the Webflow Editor use (see Table 9-3), with the addition of the the button described in Table 9-7.

Table 9-7 Pipeline Command Buttons

Tool	Function	Description
	Pipeline Component Editor button	Opens the Pipeline Component Editor, which allows you to manage Pipeline Components. **Note:** Only available in the Pipeline Editor.

Understanding what these buttons do and how to use them will make creating Pipelines a quick and easy process.

Step 1: Create a New Pipeline Component

This section contains the steps needed to create the Pipeline Component used by the Webflow.

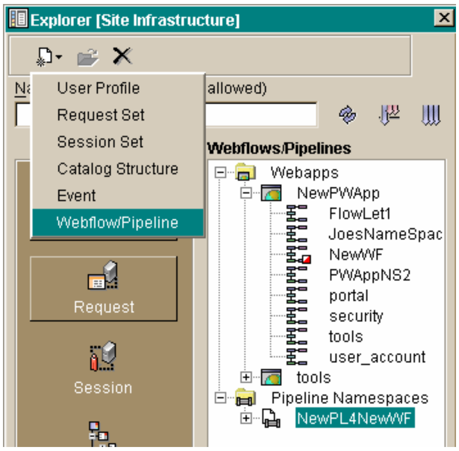
Note: This procedure assumes the following:

- You have either opened an existing project or have created a new one.
- The E-Business Control Center is running.
- The **Site Infrastructure** tab is selected.

1. In the left pane of the Explorer, click Webflow/Pipelines. A list of Webflows and Pipelines appears in the Explorer's right pane.

2. In the Webflows/Pipelines list, expand the Pipeline Namespaces folder.
3. Open the namespace that will host the Pipeline. With pipelines, a namespace is used to scope a pipeline so that multiple pipelines can be used in a single enterprise application without conflicting.
4. Click the **New** icon above the Project name to open the **New** menu, as shown in Figure 9-16.

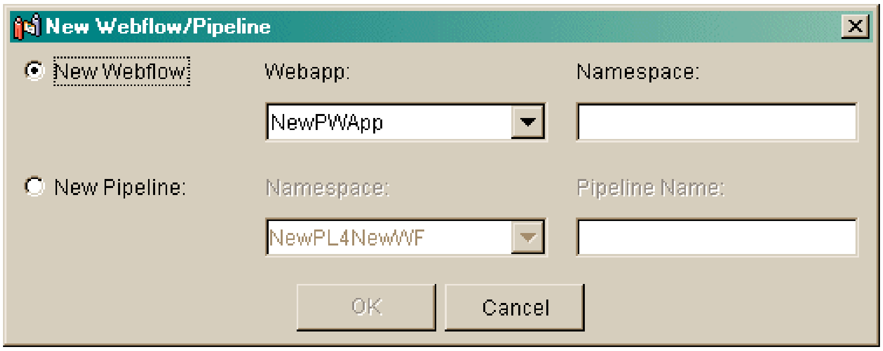
Figure 9-16 New Menu for Selected Pipeline Namespace



5. Select **Webflow/Pipeline**.

   The New Webflow/Pipeline window opens, as shown in Figure 9-17.

Figure 9-17 New Webflow/Pipeline Window



6. Create the new Pipeline by doing the following:

   a. Select the **New Pipeline** radio button.

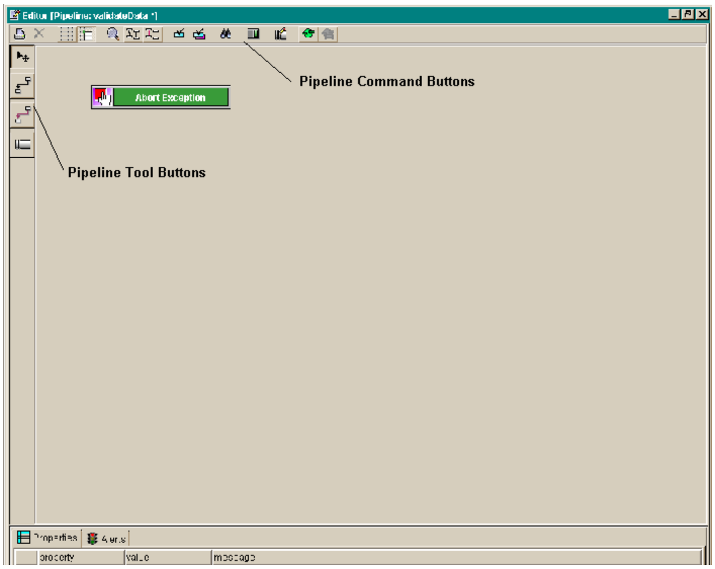
   b. In the **Namespace** list box, select the namespace that will host the Pipeline; for example, `bookmark`.

   c. In the **Pipeline Name** field, enter the name you want to call the Pipeline; for example `validate Bookmark`.

   d. Click **OK** button.

   The Pipeline Editor appears, as shown in Figure 9-18.

Figure 9-18 Pipeline Editor [Pipeline: validateData]



Note that when the Pipeline Editor opens, the Abort Exception node is automatically placed on the canvas. This node is used for exception handling and causes the Webflow to abort when an exception can't be resolved by normal processing.

7. Create the first Pipeline component by doing the following:

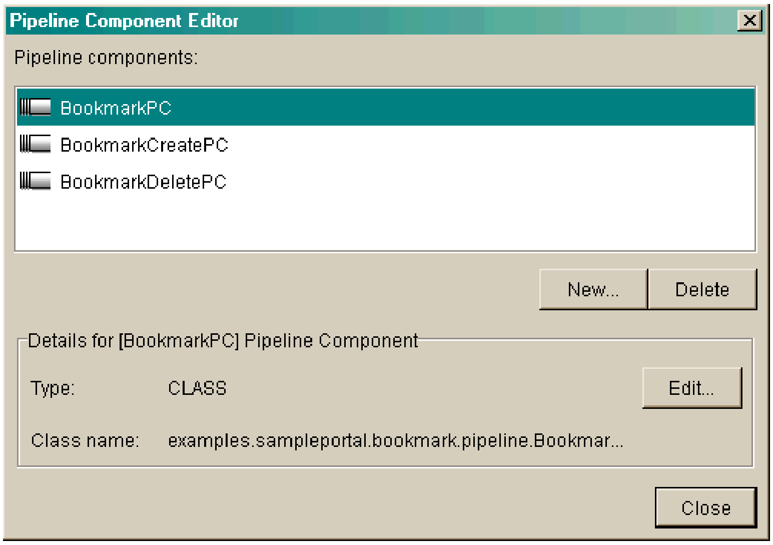
   a. Click and hold the Abort Exception Node and drag it down near the bottom of the canvas.

b. Select the Pipeline Component Node tool (see Table 9-7).

c. Place the crosshairs on the Pipeline Editor canvas somewhere above the Abort Exception Node, where you want to place the new Pipeline Component. Click to place the new Pipeline Component node on the Editor canvas.

d. Click the Pipeline Component Editor button (see Table 9-7). The Pipeline Component Editor window opens, as shown in Figure 9-19.

Figure 9-19 Pipeline Component Editor



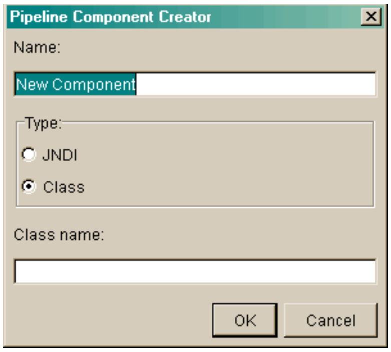
Details for [BookmarkPC] Pipeline Component

Type: CLASS

Class name: examples.sampleportal.bookmark.pipeline.Bookmar...

e. In the Pipeline Component Editor window, click **New**. The Pipeline Component Creator window opens, as shown in Figure 9-20.

Figure 9-20 Pipeline Component Creator



f. In **Name**, enter the name you want to call the Pipeline component; for example, `validateBookmark`.

g. Select **Class** as the **Type**.

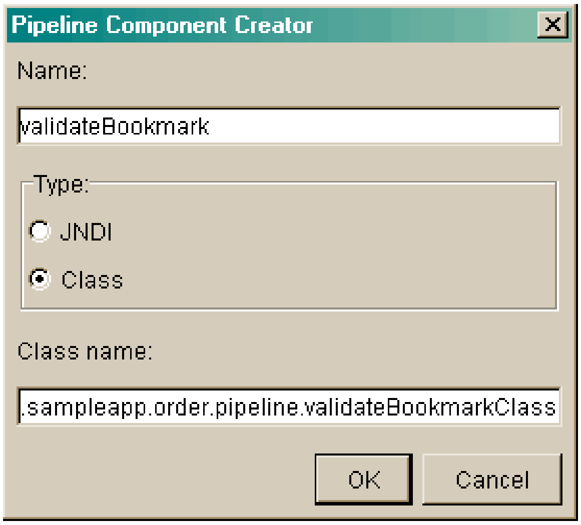
   The **Type** value is important because Pipeline components can be either a regular Java class or a session EJB. Selecting **Class** implements a regular Java class while selecting **JNDI** implements a session EJB

h. In **Class name**, enter the full package name of the class that the Pipeline component will reference; for example:

```
examples.wlcs.sampleapp.order.pipeline.ValidateBookmarkClass
```

   The completed dialog box might look like the example in Figure 9-21.

Figure 9-21 Completed Pipeline Component Creator Dialog Box



Name:

validateBookmark

Type:

JNDI [✓ Class]

Class name:

.sampleapp.order.pipeline.validateBookmarkClass

i. Click **OK** to close the Pipeline Component Creator window.

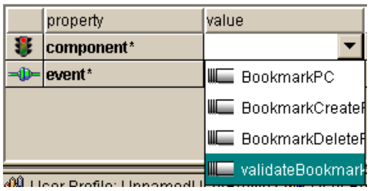
j. In the Pipeline Component Editor window, click **Close**.

The new Pipeline component appears in the **Pipeline components** list on the Pipeline Component Editor.

k. Select the new Pipeline node.

l. In the Properties Editor, located below the canvas, select the Pipeline name you entered in step f from the component*/Value pop-up list, as shown in Figure 9-22.

Figure 9-22 Properties Editor—Component Selection



property	value
component*	
event*	

The new component name appear in the **component*/value** cell.

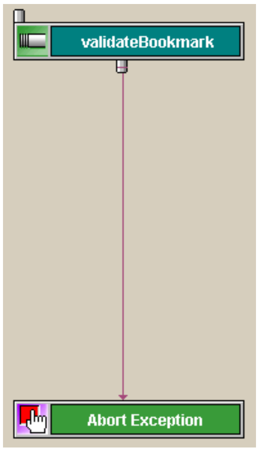
8. In this example, we want any exception to cause you to exit the Pipeline. Therefore, the node must be connected to the `Abort Exception` node. Connect the new Pipeline component to the `Abort Exception` node, as follows:

   a. Click the Exception tool.

   b. Position the transition by moving the mouse to bottom edge of the `validateBookmark` node. A solid orange square indicates an acceptable connection location, and the cursor changes to indicate a transition addition.

   c. Hold and drag the mouse to the `Abort Exception` node. Release the mouse to connect the transition to the `Abort Exception` node (Figure 9-23).

Figure 9-23 validateBookmark – Abort Exception Connection

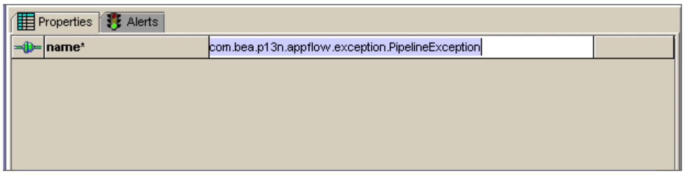


   d. Select the Exception event. The event line turns orange.

   e. In the Properties Editor, select the second column (Figure 9-24), and enter the package name to the exception class, as follows:

   ```
   com.bea.p13n.appflow.exception.PipelineException
   ```

Figure 9-24 Properties Editor—Exception Value



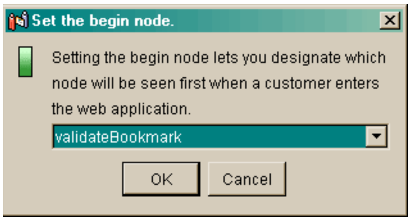
name* com.bea.p13n.appflow.exception.PipelineException

9. Make the `validateBookmark` node a Begin Node, as follows:

   a. Click the Begin Node Tool (see "Begin Node" on page 9-25).

      A message appears explaining that you are about to create a Begin node, as shown in Figure 9-25.

Figure 9-25 Begin Node Message Box



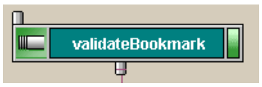
Setting the begin node lets you designate which node will be seen first when a customer enters the web application.

validateBookmark

   b. Select **OK** in the message window.

   c. validateBookmark become the Begin node, as indicated by the green bar on its right edge, as shown in Figure 9-26.

Figure 9-26 validateBookmark as Begin Node



10. Save the new Pipeline by selecting **File** → **Save** from the E-Business Control Center Main toolbar.

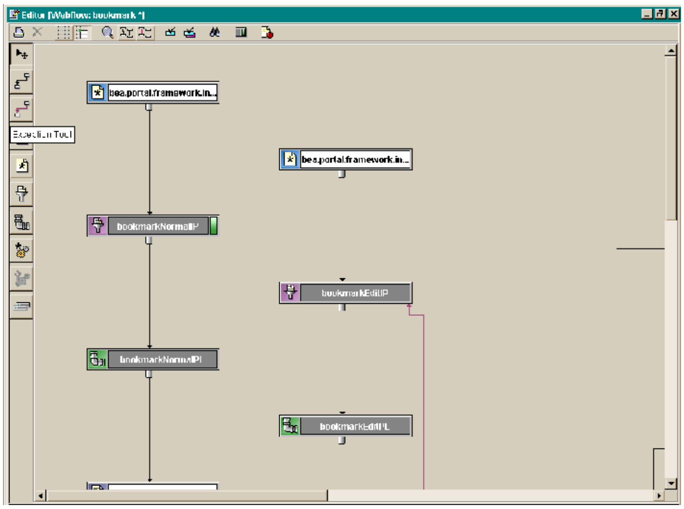
Step 2: Add the New Pipeline Component to the Webflow

Now, you need to add the new pipeline component to a Webflow. To do so, use the following procedure.

1. In the right pane of the Explorer, open the Web application to which you want to add the Pipeline. In that Web application, double-click the Webflow to which you want to add the Pipeline.

   The Webflow editor opens, as shown in Figure 9-27.

Figure 9-27 Webflow Editor showing completed Webflow


Editor [Webflow: bookmark *]
Exception Tool
bea.portal.framework.in...
bea.portal.framework.in...
bookmarkNormalIP
bookmarkEditIP
bookmarkNormalPI
bookmarkEditPL

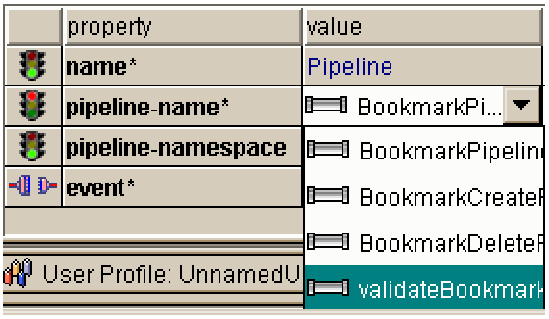
2. Select the event between the `bea.portal.framework...` and `bookmarkEditIP` nodes and press **Delete**.

3. Add the `validateBookmark` Pipeline to the Webflow, as follows:

   a. Click the Pipeline Node tool (see Table 9-5) and place the pointer between `bea.portal.framework...` and `bookmarkEditIP` nodes. Click once to add the new Pipeline Node onto the Editor canvas in that location.

   b. In the Properties Editor, below the canvas, select the second column of the **pipeline-name*** row, and then select `validateBookmark` from the drop-down list, as shown in Figure 9-28.

Figure 9-28 Properties Editor—Component Selection



property	value
name*	Pipeline
pipeline-name*	BookmarkPi...
pipeline-namespace	
event*	

BookmarkPipelin
BookmarkCreateF
BookmarkDeleteF
validateBookmark

User Profile: UnnamedU

4. Connect the `validateBookmark` Pipeline node to the other Pipeline nodes, as follows:

   a. With the Event tool (see Table 9-5) connect the `validateBookmark` node to the `bea.portal.framework...` node, click the Event you just created, and then in the Properties Editor →Properties tab, change the name of the event to "`success`" and then press **Enter**.

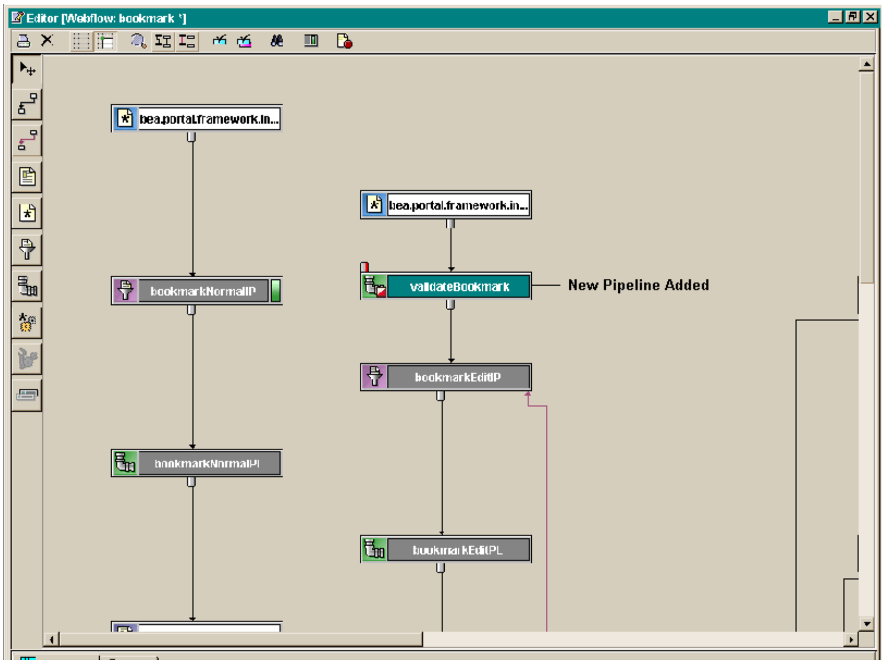
      Note: The word "success" is case sensitive. Be sure to use a lowercase "s".

   b. With the Event tool, connect the `validateBookmark` node to the `bookmarkEditIP` node.

   c. Click the Event you just created, then in the Properties Editor's Properties tab, change the name of the event to "`success`" (if necessary), and press **Enter**.

   Figure 9-29 shows the Webflow Editor canvas after this process is completed. (The nodes have been moved to make the drawing easier to see.)

Figure 9-29 Event Transitions



5. To save the changes, select **File** -> **Save** from the E-Business Control Center Main toolbar.

Synchronizing the Webflow to the Application

The Webflow that you just created must be synchronized to the Web application in order for it to work. Data synchronization loads the Webflow's and Pipeline's XML definition into the database and the master data repository, which is an in-memory data store.

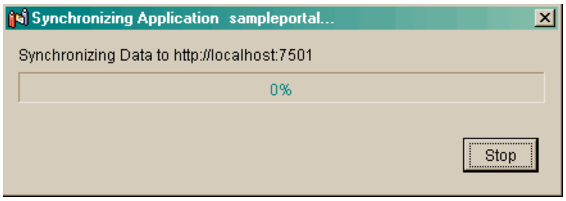
Warning: All application data is synchronized at once. If you and other developers concurrently synchronize data to a single enterprise application, it is possible to overwrite each others' work or create sets of changes that are incompatible and difficult to debug. To prevent this possibility, synchronize to separate instances of your application.

To synchronize the new pipeline component and the modified Webflow, use the following procedure:

1. Start WebLogic Server and open the E-Business Control Center.
2. Go to **Tools** → **Synchronize**.

   After a few seconds, the Synchronize progress meter appears (Figure 9-30).

Figure 9-30 Synchronize Progress Meter



Synchronizing Application sampleportal...

Synchronizing Data to http://localhost:7501

0%

   This meter will indicate to you that synchronization is in progress. When the process is complete, the progress meter will so indicate and **Stop** changes to **Close**.

3. Click **Close**.

The Webflow is synchronized to the application.

Creating a New Input Processor

As discussed earlier in this section, Input Processors are predefined, specialized Java classes that carry out more complex tasks when invoked by the Webflow mechanism. They are typically used to validate HTML form data or to provide conditional branching within a Web page. BEA has developed a number of Input Processors that are packaged with the WebLogic Portal. While you may want to reuse these processors in your own applications, you might also want to create your own for use in your applications' Webflows.

Creating an Input Processor with the InputProcessor Interface

To create a new Input Processor, you must implement the `com.bea.p13n.appflow.webflow.InputProcessor` interface by providing the details of the `process()` method, as shown in Listing 9-1.

Listing 9-1 Implementing `Process()` for the InputProcessor Interface

```
public java.lang.Object process(javax.servlet.http.HttpServletRequest req,
                                java.lang.Object requestContext)
                        throws ProcessingException
```

This interface processes the `HttpServletRequest` or the `PipelineSession` present in the `HttpSession`. The return object can be anything, but it must have a meaningful implementation of **toString()**. The webflow executor will call `toString()` on the returned object to generate the event for the processor.

Parameters	`req` - the HttpServletRequest object `requestContext` - the Object that uniquely identifies the request
Returns	An Object, which has implemented a meaningful form of `toString()`.
Throws	`ProcessingException` or one of its sub-classes

Naming an Input Processor

The name of an Input Processor should end with the suffix `IP`. For example, an Input Processor that is responsible for deleting a shipping address might be called `DeleteShippingAddressIP`. This naming convention should help you keep track of Input Processors more easily.

Executing Business Logic with Input Processors

Execution of business (application) logic should typically not be done within Input Processors. Specifically, Input Processors should not call Enterprise JavaBeans (EJBs) or attempt to access a database. All such logic should be implemented in Pipeline Components. Although it is possible to execute this logic within an Input Processor, such logic could not be transactional, and would defeat a primary purpose of the Webflow architecture. By separating business logic from the presentation logic, your Web site is inherently flexible in nature. Modifying or adding functionality can be as simple as creating and plugging in new Pipelines and/or Input Processors.

Extending the InputProcessorSupport Class

Alternately, your new Input Processor can extend the `com.bea.p13n.appflow.webflow.InputProcessorSupport` class, as shown in Listing 9-2. As its name implies, this abstract class allows you to use static helper methods that provide additional support for an Input Processor. If your new Input Processor class must extend some other class, however, you will not be able to take advantage of the `InputProcessorSupport` class.

Listing 9-2 Extending the `InputProcessorSupport` class

```
public abstract class InputProcessorSupport
                         extends java.lang.Object
                implements InputProcessor, ValidatedFormConstants
```

Note: For more information about implementing the `InputProcessorSupport` class, refer to the Javadoc for `com.bea.p13n.appflow.webflow`.

When you are using the Webflow Editor to specify the properties for an Input Processor node you placed on the canvas, simply include the class name of your newly created Input Processor in the appropriate field. There are no additional activities you need to perform to make your Input Processor work with the existing Webflow mechanism.

Extending Webflow by Creating Extension Presentation and Processor Nodes

If creating new input processors and pipeline components to add to those BEA provides does not meet your needs, you may also choose to extend the Webflow mechanism by creating classes that can be used as Extension (Custom) Presentation or Processor Nodes. Once you create the classes associated with these nodes, you will need to register the new nodes in the `webflow-extensions.wfx` file. This section shows you how to perform these tasks.

How to Create an Extension Presentation Node

To create an Extension (Custom) Presentation Node, use this procedure:

1. Create a class that implements the `com.bea.p13n.appflow.webflow.PresentationNodeHandler` interface. Be sure your class returns a URL to which the `WebflowServlet` servlet can forward.

2. Register your extension node in the `webflow-extensions.wfx` file so it can be used in the Webflow and Pipeline Editors. See "Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors" on page 9-42.

WebLogic Portal uses an Extension (Custom) Presentation Node named `portal`, which you can view as an example. Portal uses this extension node to indicate to the portal Webflow that the contents of the portlet are to remain unchanged (that is, it indicates that the last URL should be displayed). The `portal` node's implementation class is `LastContentUrlNodeHandler.java`.

How to Create an Extension Processor Node

Extension (Custom) Processors are processors that your organization (as opposed to BEA) develops for use in your applications' Webflows. Imagine you want to create an Extension (Custom) Processor that functions at the same level as an Input Processor processor or Pipeline Processor. Extension Processors may be used to perform activities not currently supported by the Webflow. However, the flow in and out of an Extension Processor is still governed by the Webflow mechanism. Extension Processors are represented as nodes in the Webflow Editor, much like the Input Processor and Pipeline Nodes are, but with a slightly different representation for easy identification.

For example, you may want to create an Extension (Custom) Processor that works with the BEA Rules Engine to support different Webflows based on some condition, such as membership in a customer segment. Another, more simple example might be a layout manager processor that automatically includes a header and footer in your JSP when given the page's body content. In fact, we have already created such a processor.

To create an Extension (Custom) Processor Node, use this procedure:

1. Create a class that implements the `com.bea.p13n.appflow.webflow.Processor` interface to define the Extension Processor. Listing 9-3 shows a typical implementation of the `Processor` interface.

Listing 9-3 Implementing the `Processor` Interface

```
public java.lang.Object process(java.lang.String webapp,
                                java.lang.String namespace,
                                javax.servlet.http.HttpServletRequest request,
                                java.lang.Object requestContext)
                        throws java.lang.Exception
```

This interface executes the processor indicated by the request. The return object can be anything, but it must have a meaningful implementation of `toString()`.

The webflow executor will call `toString()` on the returned object to generate the event for the processor.

Parameters	`webapp` - a String containing the webapp name `namespace` - a String containing the namespace name `name` - a String containing the processor name (`foo.inputprocessor`, bar.pipeline) `request` - the HttpServletRequest `requestContext` - the Object object that uniquely identifes the associated request
Returns	the results, as an Object. Can be anything, but must implement `toString()`.
Throws	`java.lang.Exception`, if an error occurs

2. Register your processor in the `webflow-extensions.wfx` file so it can be used in the Webflow and Pipeline Editors.

Note: For instructions on how to register your extension node in the `webflow-extensions.wfx` file, see "Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors" on page 9-42.

Making Your Extension Presentation and Processor Nodes Available in the Webflow and Pipeline Editors

After you have created an Extension (Custom) Presentation or Processor Node, you must make that node available to other developers on your team by registering the node in the `webflow-extensions.wfx` file.

Notes: The `webflow-extensions.wfx` file resides within the `<BEA_HOME>/user_projects/myNEWDomain/beaApps/portalApp-project/default/webflow/` folder (where `<BEA_HOME>` is where you installed WebLogic Portal; for example:

```
bea/user_projects/myNEWDomain/beaApps/portalApp-project/
   default/webflow/
```

Registering an Extension (Custom) Processor Node will cause its corresponding tool on the Webflow Editor palette to become enabled once you restart the E-Business Control Center.

Registering an Extension Presentation Node

To register an Extension Presentation Node in the `webflow-extensions.wfx` file, follow these steps:

1. Open the `webflow-extensions.xml` file, which resides in the `<BEA_HOME>/user-projects/myNEWDomain/beaApps/portalApp-project/application-synch/webapps/<APPLICATION>` folder (where `<BEA_HOME>` is youre BEA parent directory and `<APPLICATION>` is the specific Web application.
2. Add an `<end-node>` element to the `<end-node-registration>` list.
3. Assign your presentation node a name with the Name attribute, and specify the class of the underlying node implementation with the Node-handler attribute.
4. Define the input parameters that the class expects upon invocation, using `<node-processor-input>` elements. Give each parameter a Name, and if the parameter is optional, assign the Required attribute a value of `false`.

   Note: This information will be used in the Webflow and Pipeline Editors' Property Editors.
5. Save the `webflow-extensions.wfx` file, and restart the E-Business Control Center.

"Registering an Extension Presentation Node" provides an example of registering an Extension Presentation Node in the `webflow-extensions.wfx` file.

Registering an Extension Processor Node

To register an Extension Processor Node in the `webflow-extensions.wfx` file, follow these steps:

1. Open the `webflow-extensions.xml` file, which resides in the `<BEA_HOME>/user-projects/myNEWDomain/beaApps/portalApp-project/application-synch/webapps/<APPLICATION>` folder (where `<APPLICATION>` is the specific Web application.

2. Add a `<process>` element to the `<process-registration>` list.

3. Assign your processor a name with the `Name` attribute, and specify the class of the underlying processor implementation with the `Executor` attribute.

4. Define the input parameters that the class expects upon invocation, using `<node-processor-input>` elements. Give each parameter a name, and if the parameter is optional, assign the `Required` attribute a value of `false`.

   Note: This information will be used in the Webflow and Pipeline Editors' Property Editors.

5. Save the `webflow-extensions.wfx` file, and restart the E-Business Control Center.

CHAPTER

10 Creating a Look-and-Feel

WebLogic Portal provides two powerful mechanisms for managing the look and feel of your portals: skins and layouts. Skins are graphics and HTML style settings that define the visual style of a portal; for example color, fonts, and icons. Layouts are HTML-based matrixes into which portlets can be placed. The WebLogic Portal platform provides a ready-made structure for these components can be created, managed, and provided to users across applications.

This section shows you how to create skins and layouts for a portal. Some prerequisite knowledge about creating cascading style sheets (CSS), creating graphics, and defining HTML tables is required.

This section includes information on the following subjects:

- Portal Look-and-Feel Structure
- Using Skins
- Using Layouts

Portal Look-and-Feel Structure

When you see a portal in a Web browser, you see a single, unified Web page. That portal is made up of many JSPs. For example, in addition to the page content, a portal may use separate JSPs for the header, footer, and vertical and horizontal navigation bars.

Each JSP used in a portal's structure contains the coding necessary to implement part of the portal's overall look and feel. For example, each JSP can contain references to the CSS and to graphics for determining a portal's skin; and the portal's main content page contains the logic necessary to implement which layout is chosen for the portal.

When you create a portal in the E-Business Control Center with the Portal Wizard, the portal template BEA provides is called **baseportal**. One of the functions of this template is to use a predefined set of JSPs that define the look-and-feel structure to create the new portal. In this section, the procedures for creating skins and layouts are presented in the context of the **baseportal** structure, which is also used for the Avitek Financial Portal Example provided with WebLogic Portal.

Using Skins

A *skin* is a collection of files that includes a cascading style sheet (`.css` file) and a directory of images that define the look and feel of a portal. Every button, banner, portlet header, background color, and font characteristic is determined by the CSS and the graphics. Skins also include a thumbnail graphic of the skin for preview purposes.

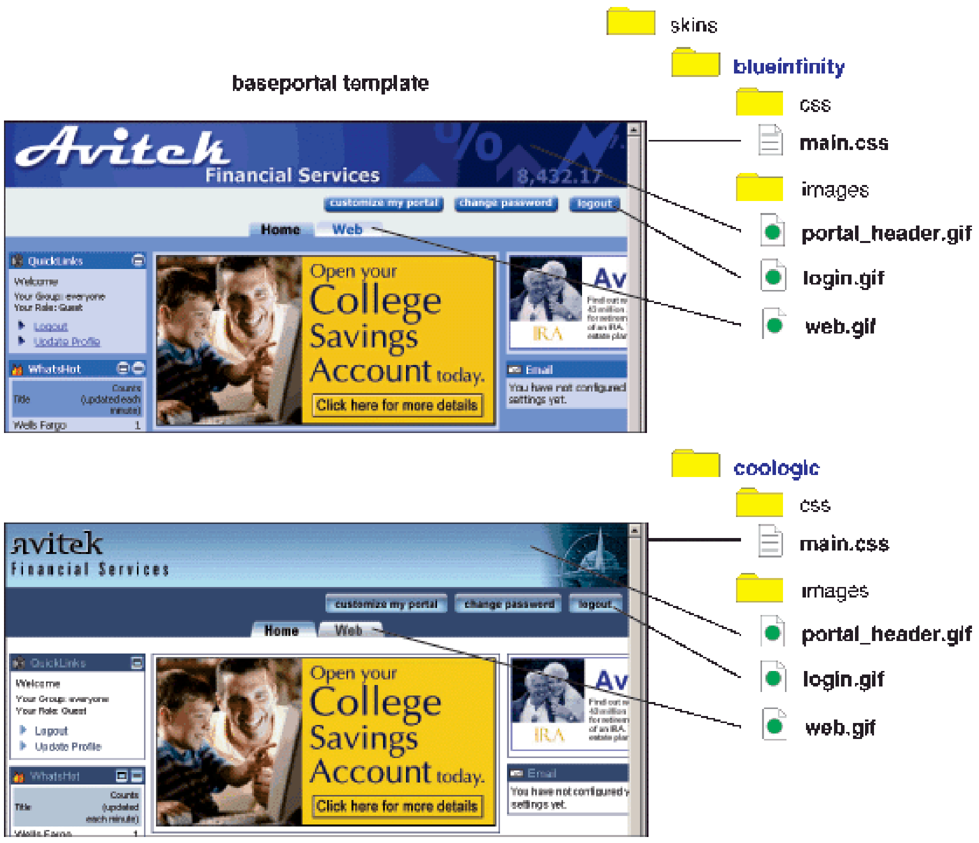
This section includes information on the following subjects:

- Creating Skins
- Storing Skins
- Making Skins Available

Creating Skins

To enable seamless switching between skins in a portal, the CSS and graphics for one skin must be named identically to the CSS and graphics in the other skins, as shown in Figure 10-1.

Figure 10-1 Different Skins Applied to a Portal Created with the baseportal Template



Note: Any content graphics that appear on the page, such as ads and graphics in portlets, are stored either in a content management system or in images subdirectories beneath portlet directories. However, portlet title bar graphics and background colors are controlled by the skin, as shown in Figure 10-1.

Creating skins is simply a matter of creating or modifying CSS and graphics files, with the same names as those used in other skins, but under a different directory. Creating skins also involves creating a thumbnail graphic for preview purposes. Use your favorite graphics tool to create and modify skin and thumbnail graphics.

When creating graphics for a skin, be aware that larger graphics slow down page loading in a browser.

Skins Provided by BEA

WebLogic Portal includes a set of predefined skins that are used when you create a portal with the Portal Wizard. These skins are located in:

```
<BEA_HOME>\weblogic700\common\templates\webapps\portal\baseportal
\
j2ee\framework\skins
```

These can be used as templates to create entirely new skins.

Storing Skins

The next section describes the directory structure in which you must store new skins. To make skins available to a portal, store the CSS and graphics in the directory structure that is referenced by the metadata in a portal. Table 10-1 shows where to store the necessary pieces that make up a skin.

The skin thumbnail graphic is the only part of the skin not stored on the server. The thumbnail is for previewing a selected skin in the E-Business Control Center.

Table 10-1 Where to Store Skins Resources

Resource	Location
Skin thumbnail graphic	`<BEA_HOME>\user_projects\`*`your_domain`*`\beaApps\portalApp-project\library\portal\skins\`*`skin_folder`*
CSS file	`<BEA_HOME>\user_projects\`*`your_domain`*`\beaApps\portalApp\`*`your_portal`*`\framework\skins\`*`skin_folder`*`\css`
Skin graphics	`<BEA_HOME>\user_projects\`*`your_domain`*`\beaApps\portalApp\`*`your_portal`*`\framework\skins\`*`skin_folder`*`\images`

Making Skins Available

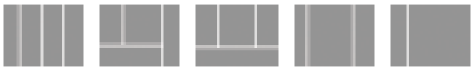
Making skins available in your portal involves using the E-Business Control Center to add the skins to a portal definition, synchronizing the updated portal definition to the server, and applying the skins to the portal with the WebLogic Portal Administration Tools.

Procedures for making skins available for administration are found in the WebLogic Portal *Administration Guide*. See "Administering Portal and Portlet Attributes" at http://edocs.bea.com/wlp/docs70/admin/frmwork.htm.

Using Layouts

A layout is a JSP. It combines simple HTML formatting with simple JSP tags that define sections of a page where portlets are placed. A layout also includes a thumbnail graphic representation of itself for preview purposes. Figure 10-2 shows thumbnail representations of layouts that are provided with WebLogic Portal.

Figure 10-2 Thumbnails of Default Layouts Provided with WebLogic Portal



This section includes information on the following subjects:

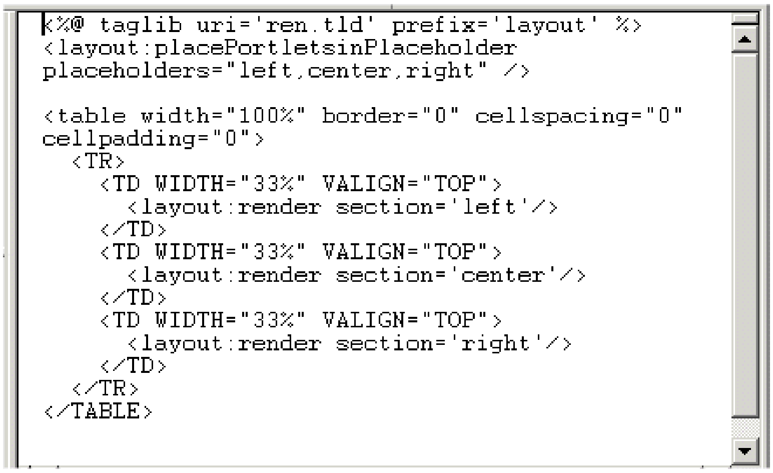
- Creating Layouts
- Storing Layouts
- Making Layouts Available

Creating Layouts

This procedure shows you how to create a new layout for an existing portal.

To view an existing layout, open the following file in a text editor: `your_portal\framework\layouts\threecolumn\template.jsp`, as shown in Figure 10-3.

Figure 10-3 Source view of the "threecolumn" layout



```
<%@ taglib uri='ren.tld' prefix='layout' %>
<layout:placePortletsinPlaceholder
placeholders="left,center,right" />

<table width="100%" border="0" cellspacing="0"
cellpadding="0">
  <TR>
    <TD WIDTH="33%" VALIGN="TOP">
      <layout:render section='left'/>
    </TD>
    <TD WIDTH="33%" VALIGN="TOP">
      <layout:render section='center'/>
    </TD>
    <TD WIDTH="33%" VALIGN="TOP">
      <layout:render section='right'/>
    </TD>
  </TR>
</TABLE>
```

To create a new layout:

1. Create a text file called `template.jsp` and store it in an appropriate layout folder. See "Storing Layouts" on page 10-8.

   Note: The layout name is determined by the name of the folder in which `template.jsp` is stored.

2. On the first line, enter:

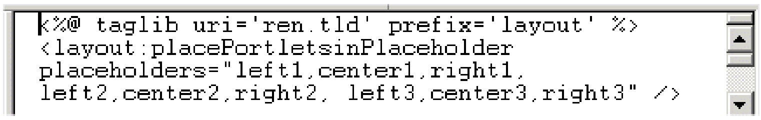
   ```
   <%@ taglib uri='ren.tld' prefix='layout' %>
   ```

3. On the second line, insert the following tag and enter the names of the sections (placeholders) in which portlets will be placed in the layout. Put the placeholder names in a list of comma-separated values in the `placeholders` attribute, as shown in the following:

```
<layout:placePortletsinPlaceholder
placeholders='<name1>,<name2>,<name3>,<name4>' />
```

where `<name*>` is a unique section name, as shown in Figure 10-4.

Figure 10-4 Adding new layout placeholder names



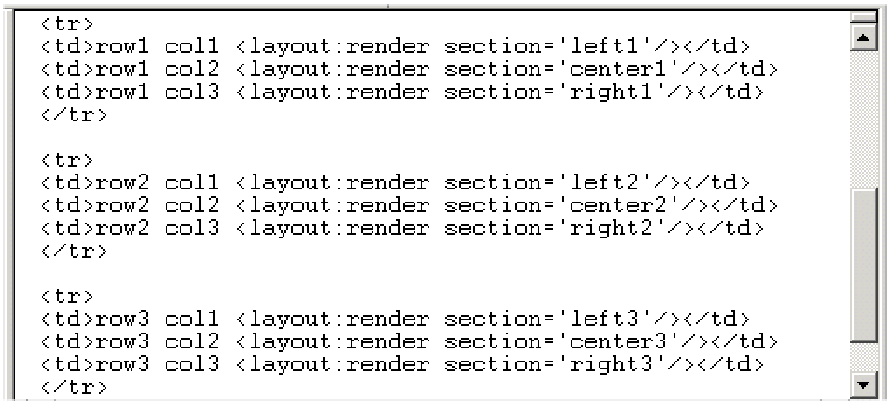
```
<%@ taglib uri='ren.tld' prefix='layout' %>
<layout:placePortletsinPlaceholder
placeholders="left1,center1,right1,
left2,center2,right2, left3,center3,right3" />
```

4. In the HTML body of the layout, insert the following tag for each placeholder you defined in the previous step. These tags determine where portlets will be placed in the layout.

   ```
   <layout:render section='<name>'/>
   ```

   where `<name>` is one of your placeholder names. You can insert these tags in any valid HTML (though frames are not supported); you are not restricted to putting them in tables. Figure 10-5 shows the tags placed in table cells.

Figure 10-5 Inserting placeholders into the HTML fragment



```
<tr>
<td>row1 col1 <layout:render section='left1'/></td>
<td>row1 col2 <layout:render section='center1'/></td>
<td>row1 col3 <layout:render section='right1'/></td>
</tr>

<tr>
<td>row2 col1 <layout:render section='left2'/></td>
<td>row2 col2 <layout:render section='center2'/></td>
<td>row2 col3 <layout:render section='right2'/></td>
</tr>

<tr>
<td>row3 col1 <layout:render section='left3'/></td>
<td>row3 col2 <layout:render section='center3'/></td>
<td>row3 col3 <layout:render section='right3'/></td>
</tr>
```

Note: The `<layout:placePortletsinPlaceholder>` and `<layout:render>` tags perform all the logic of retrieving and placing portlets in your layout and remembering which portlets are put in which placeholders.

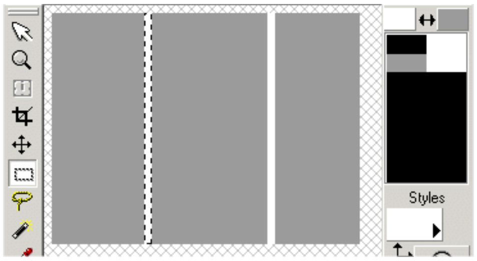
5. Save the file.

6. Create a thumbnail graphic representation of your layout. This thumbnail provides the layout preview in the E-Business Control Center and in the WebLogic Portal Administration Tools when you select the layout name in the interface. Using a graphics editor, copy and modify an existing thumbnail graphic, as shown in Figure 10-6. Save the graphic as two separate files: one called `thumbnail.gif` and the other *`layout_name`*`.gif`, where *`layout_name`* is the name of the layout folder.

   A set of predefined layouts and thumbnail graphics are located in subdirectories under the following directory:

   ```
   <BEA_HOME>\weblogic700\common\templates\webapps\portal\
   baseportal\j2ee\framework\layouts
   ```

Figure 10-6 Creating a layout thumbnail



The next section describes the directory structure in which you must store new layouts.

Storing Layouts

To make a layout available in a portal, store the new layout and the thumbnail graphic in a specific directory structure in two different locations. Also, use two different names for the thumbnail graphic, as described in the previous section. Table 10-2 shows the names and locations of the files.

Table 10-2 Where to Store Layout Resources

Resource	Location
Layout thumbnail graphic: *`layout_name`*`.gif`	`<BEA_HOME>\user_projects\`*`your_domain`*`\beaApps\portalApp-project\library\portal\layouts\`*`layout_folder`* The *`layout_name`* must be the same as the *`layout_folder`*.

Table 10-2 Where to Store Layout Resources

Resource	Location
Layout thumbnail graphic: `thumbnail.gif`	`<BEA_HOME>\user_projects\`*`your_domain`*`\beaApps\portalApp\`*`your_portal`*`\framework\layouts\`*`layout_folder`*
`template.jsp`	`<BEA_HOME>\user_projects\`*`your_domain`*`\beaApps\portalApp-project\library\portal\layouts\`*`layout_folder`* **and** `<BEA_HOME>\user_projects\`*`your_domain`*`\beaApps\portalApp\`*`your_portal`*`\framework\layouts\`*`layout_folder`* The name of the *`layout_folder`* is used as the name of the layout.

Making Layouts Available

Making layouts available in your portal involves using the E-Business Control Center to add the layout to a portal definition, synchronizing the updated portal definition to the server, and applying the layout to the portal with the WebLogic Portal Administration Tools.

Procedures for making layouts available are found in the WebLogic Portal *Administration Guide*. See “Administering Portal and Portlet Attributes” at http://edocs.bea.com/wlp/docs70/admin/frmwork.htm.

CHAPTER

11 Extending Portlets

To invoke advanced features in portlets, and make these portlets available to administrators, developers use several tools and many procedures. This section includes information on the following subjects:

- Basic Portlet Customization
  - Moving a Portlet Between Portal Web Applications
  - Moving a Portlet Between Domains
  - Creating Categories for Portlets
- Portlets and the Framework
  - Simple JSP Portlets
  - WebFlow Portlets
  - Web Service Portlets
- Portalizing an Existing Web Application
- Performance Tuning

Basic Portlet Customization

One of the most basic customizations is moving a portlet from one portal into another. Understanding this procedure will help you understand many of the other tasks in this section, since many procedures and tools are introduced here. This section covers the following basic portlet customizations:

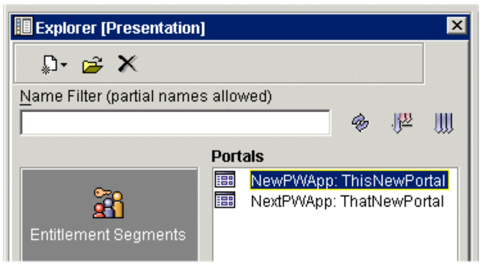
- Moving a Portlet Between Portal Web Applications

- Moving a Portlet Between Domains
- Creating Categories for Portlets

Moving a Portlet Between Portal Web Applications

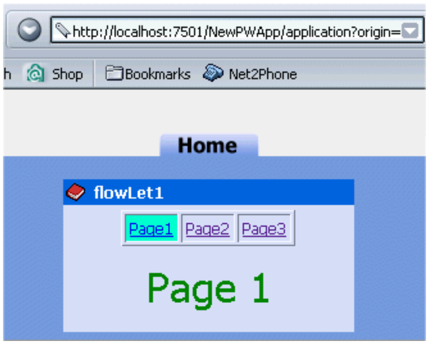
This section explains the process of moving a portlet from one portal Web application to another, and is based on the assumption that both portals have been deployed correctly. Figure 11-2 shows the two portals with their associated Web applications.

Figure 11-1 Portals and associated Web applications



This example illustrates moving a portlet called `flowlet1`, shown in Figure 11-2, from one portal Web application (NewPWApp) to another (NextPWApp).

Figure 11-2 flowLet1 portlet in the NewWPApp



Take the following steps to add the flowLet1 portlet to NextPWApp portal:

- Step 1: Copy J2EE Resources into New Web Application

- Step 2: Edit the Target Web Application Metadata
- Step 3: Synchronize the Project
- Step 4: Make the New Portlet Visible and Available

Step 1: Copy J2EE Resources into New Web Application

If you open the flowLet1 portlet inside the E-Business Control Center, you can see it is visible among the available portlets.

Because the original portlet uses a custom Webflow, this file must be moved before the target portal Web application can use this portlet. Figure 11-3 shows the flowLet1 Webflow file in the `\webapps` directory for NewPWApp being copied into the `\webapps` directory for NextPWApp.

Figure 11-3 Copying the navigation Webflow

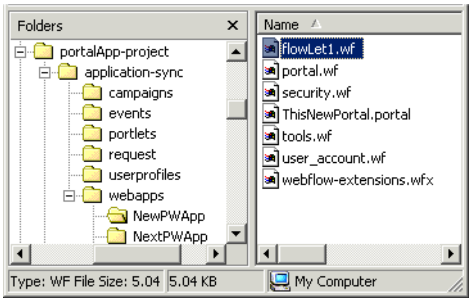
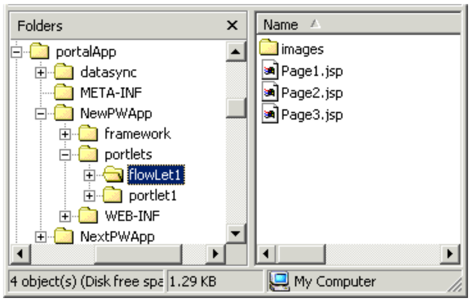


Figure 11-4 shows the portlet folder flowLet1 being copied from the NewPWApp application folder into the NextPWApp Web application folder. The JSPs and images that make up your portlet are stored inside this directory.

Figure 11-4 Copying J2EE resources



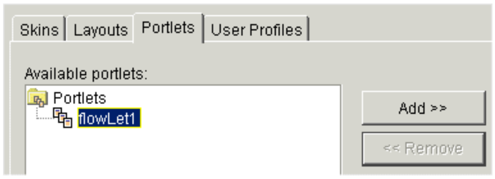
Now that the J2EE resources have been copied into the new Web application, the metadata can be edited to point to these resources.

Step 2: Edit the Target Web Application Metadata

Use the E-Business Control Center to add the flowLet1 portlet to the `ThatNewPortal` portal:

1. From the Presentation Tab in the E-Business Control Center, click on **portal** and select the portal called That New Portal.
2. When the portal editor opens, click on the General tab to the top right.
3. Click on the portlets tab halfway down the General page, select the flowLet1 portlet from the list of available portlets, and click **Add**, as shown in Figure 11-5.

Figure 11-5 Adding flowLet1 to ThatNewPortal



4. Close the General tab, click on the Pages tab, select the Home portal page and click **Edit**.
5. Select the flowLet1 portlet from the list of available portlets to the lower left of the definition screen for the Home portal page, and click **Add**. Click **OK** to close this window.

6. Save this project.

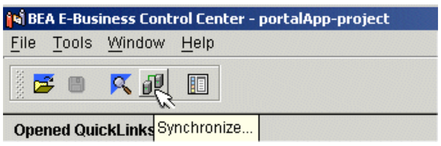
Step 3: Synchronize the Project

Use the E-Business Control Center to sync the project.

Note: For more details on the E-Business Control Center, consult "Setting up the E-Business Control Center" in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/sysadmin.htm.

1. From the E-Business Control Center toolbar, click the **Synchronize** button, as shown in Figure 11-6.

Figure 11-6 Click Synchronization button



2. When the Synchronizing Application window shows that synchronization is complete, click **Close**.

Step 4: Make the New Portlet Visible and Available

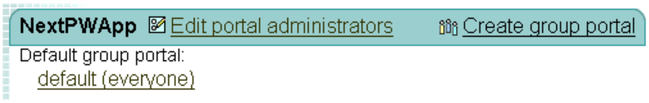
The new portlet called flowLet1 is now deployed, but must be made available using the WebLogic Portal Administration Tools.

1. In your Web browser, navigate to the following URL: http://<hostname>:<port>/portalAppTools. Login as administrator/password, and click **Portal Management**, as shown in Figure 11-7.

Figure 11-7 Go to Portal Management

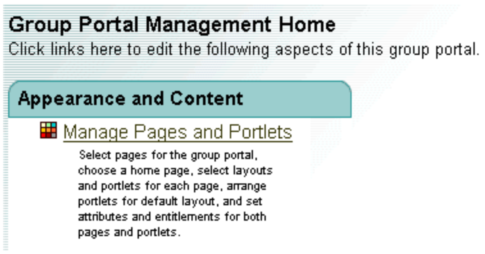
2. From the Portal Management Home page, click the Default Portal under the NextPWApp, as shown in Figure 11-8**.**

Figure 11-8 Select Default Group Portal



3. From the Group Portal Management Home page, click **Manage Pages and Portlets** as shown in Figure 11-9.

Figure 11-9 Manage Pages and Portlets



Group Portal Management Home

Click links here to edit the following aspects of this group portal.

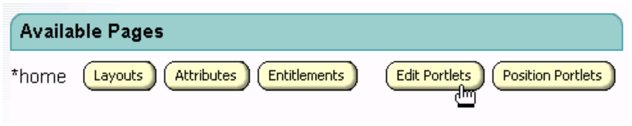
Appearance and Content

Manage Pages and Portlets

Select pages for the group portal, choose a home page, select layouts and portlets for each page, arrange portlets for default layout, and set attributes and entitlements for both pages and portlets.

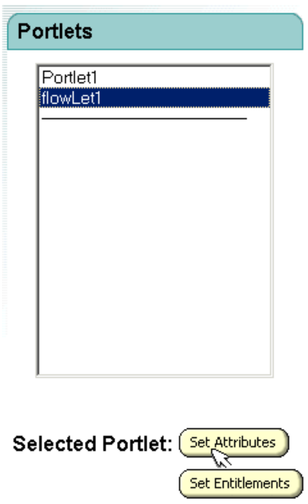
4. Next to the portal page, click **Edit Portlets**, as shown in Figure 11-10.

Figure 11-10 Click Edit Portlets



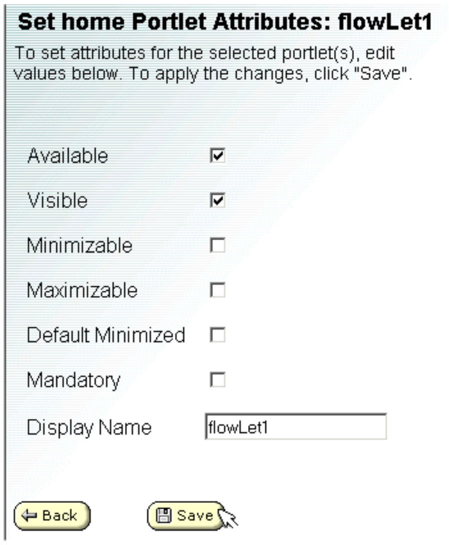
5. Select the flowLet1 portlet from the list and click Set Attributes, as shown in Figure 11-11.

Figure 11-11 Select Set Attributes for flowLet1 portlet



6. Set the Portlet's attributes to Visible and Available and click Save, as shown in Figure 11-12.

Figure 11-12 Set Portlet Attributes

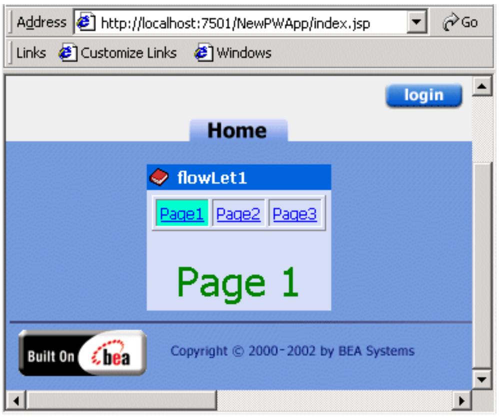

Set home Portlet Attributes: flowLet1

To set attributes for the selected portlet(s), edit values below. To apply the changes, click "Save".

Available	☑
Visible	☑
Minimizable	☐
Maximizable	☐
Default Minimized	☐
Mandatory	☐
Display Name	flowLet1

7. Verify the results by navigating to `http://<hostname>:<port>/NextPWApp/index.jsp`. The result should resemble that shown in Figure 11-13.

Figure 11-13 Viewing flowLet1 in NextPWApp



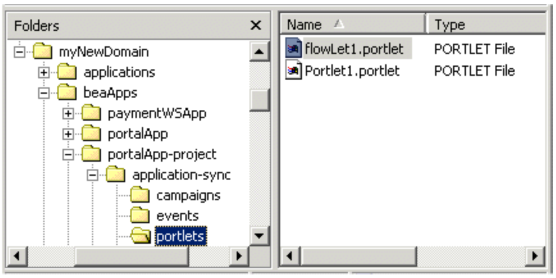
Moving a Portlet Between Domains

Assuming both domains are correctly installed and deployed instances of the latest release of WebLogic Portal, the procedure for moving a portlet from one domain to the other is almost identical to that used for Moving a Portlet Between Portal Web Applications.

1. When copying the J2EE files, also copy the .portlet file to the `portalApp-project\portlets` directory on the target server. Figure 11-14 shows where the .portlet file resides in the `portalApp-project\portlets` directory on the origin domain.

2. Complete the remainder of the steps listed in the section Moving a Portlet Between Portal Web Applications.

Figure 11-14 The .portlet file



Finding Missing Resources Through Error Messages

If you are missing necessary portlet resources in a domain, you will receive error messages either at server startup or when you try to access portal pages that use those resources. (Missing portlet resources do not prevent the server from starting.)

For example, you may receive HTTP session errors in your browser such as 404, or render-time errors that say, for example, "Functionality temporarily unavailable." Missing metadata can also cause errors.

Look for error messages in your server's command window to determine which resources are missing. For example, Listing 11-1 shows that a portlet called WebFlowPortlet is missing a file called `header.jsp`. This message was generated in the command window when the portal page containing this portlet was accessed.

Listing 11-1 Error caused by missing portlet resources

```
<Jun 12, 2002 10:38:59 AM MDT> <Error> <HTTP> <101214> <Included
resource or file "/NewWebApp/portlets/WebflowPortlet/header.jsp"
not found from requested resource
"/NewWebApp/framework/portal.jsp".>
```

The E-Business Control Center also catches some errors when you are editing portlets. For example, a warning dialog box alerts you if the portlet you are editing is missing an associated Webflow file.

Creating Categories for Portlets

The E-Business Control Center lets you group portlets into categories for easier management. This section illustrates how to create categories and manage portlets within them.

Preparing to Work With Categories

The steps in this section require that the following preparations be in place:

- Working instance of WebLogic Portal 7.0
- Fully deployed portal
- WebLogic Portal Server must be running.

Warning: Trying to create a new category before a portal has been defined in the E-Business Control Center will generate an error.

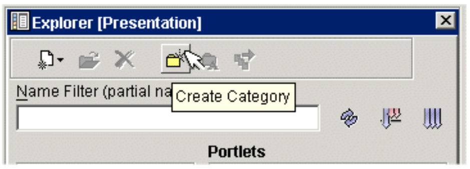
Creating Portlets and Categories

1. From the Presentation Tab in the E-Business Control Center, use the Portlet Wizard to create some generic portlets, as shown in Chapter 2, "Creating a New Portal in a New Domain." To follow along with this example, name the new portlets as follows:

- WallStreet
- MainStreet
- MidWest
- SouthEast
- LatinAmerica

2. From the Presentation Tab in the E-Business Control Center, click the **Portlets** icon and click the new folder icon, as shown in Figure 11-15.

Figure 11-15 Creating a new category

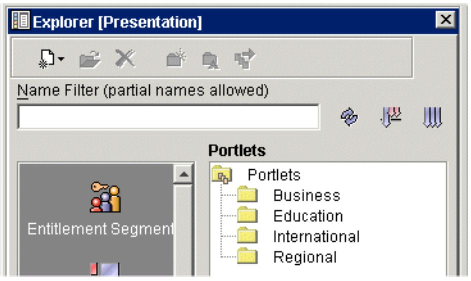


3. The New Category screen appears. Type “Business” in the name field and click **OK**. Repeat this step to create the following new categories:

- Education
- Regional
- International

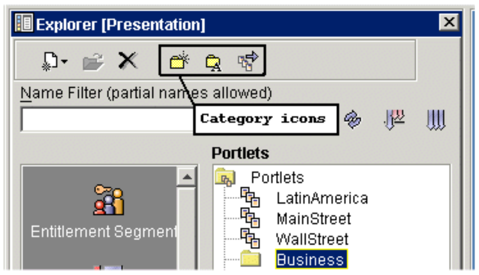
The resulting category list should resemble that shown in Figure 11-16.

Figure 11-16 New categories



Note: Selecting one of the portlets makes the three icons to the right available, as shown in Figure 11-17.

Figure 11-17 Category icons in toolbar

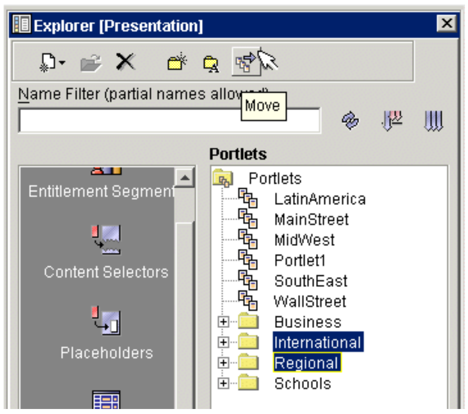


Moving Portlets and Categories

This section explains how to move portlets and categories using the E-Business Control Center.

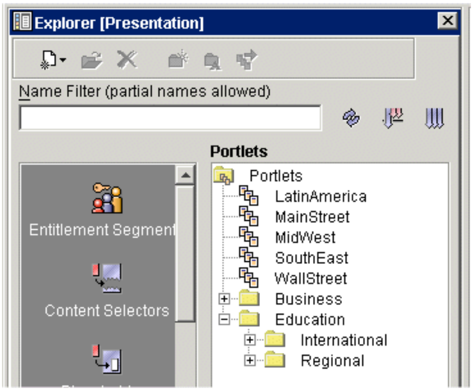
1. Select the International and Regional categories, then click the **Move** icon on the far right of the toolbar, as shown in Figure 11-18.

Figure 11-18 Click Move category



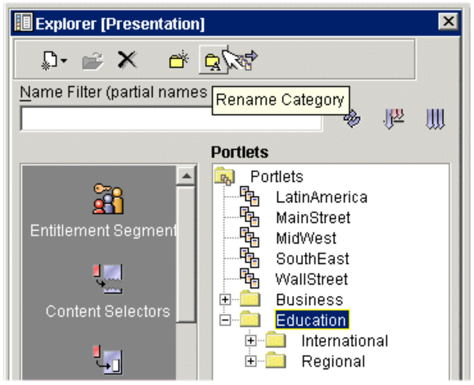
2. The Moving Category International screen appears. Select Education and click **Move**. When the Moving Category Regional window appears, do the same. The results should look similar to those in Figure 11-19.

Figure 11-19 Categories and portlets



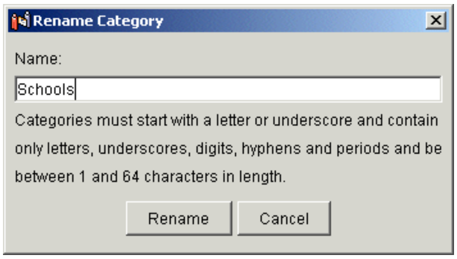
3. Select the Education category, then click the **Rename** icon second from the right in the toolbar, as shown in Figure 11-20.

Figure 11-20 Select a category and click rename



4. The Rename category screen appears. Type in “Schools” and click **Rename**, as shown in Figure 11-21.

Figure 11-21 Renaming a category



Categories must start with a letter or underscore and contain only letters, underscores, digits, hyphens and periods and be between 1 and 64 characters in length.

5. Move the MidWest and SouthEast portlets into the Schools category by selecting them, clicking the **Move** icon, shown in Figure 11-22, and selecting a destination as shown in Figure 11-23.

Figure 11-22 Moving portlets into a Category

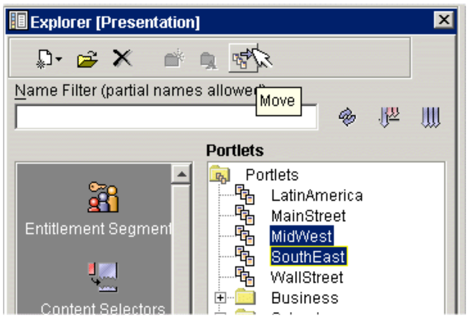
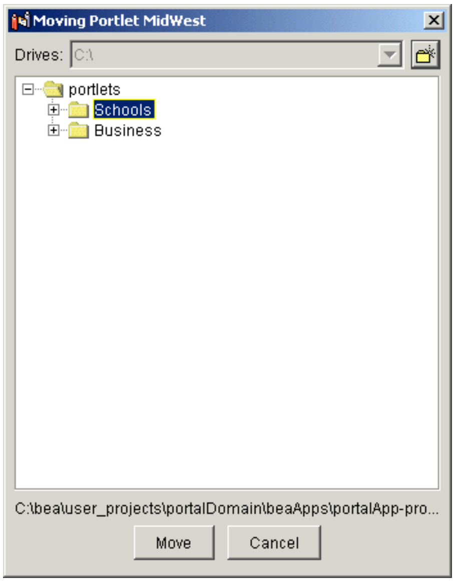
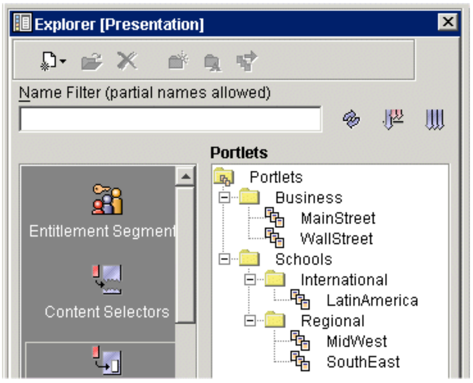


Figure 11-23 Selecting destination folder



6. Using the category icons, move the MainStreet and WallStreet portlets into the Business category, the LatinAmerica portlet into the International category, and the MidWest and SouthEast portlets into the Regional categories. The results should look like those in Figure 11-24.

Figure 11-24 Portlets and categories arranged



Adding Portlets to Existing Categories

This section explains how to place new portlets into existing categories. In this example, the following portlets are created:

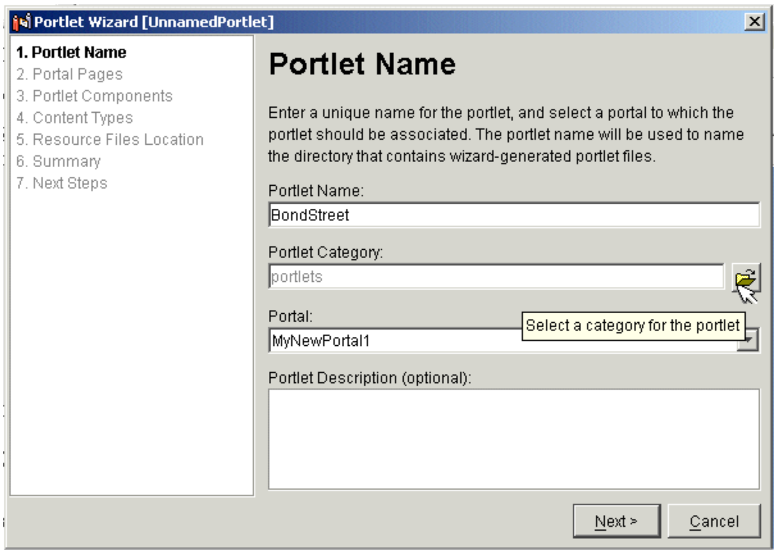
- BondStreet
- UK
- Asia

To add portlets to existing categories, take the following steps:

1. From the Presentation Tab in the E-Business Control Center, use the Portlet Wizard to create the new portlets.

Note: When naming the portlets, click **Select a Category**, as shown in Figure 11-25.

Figure 11-25 Select a category for the portlet



Portlet Wizard [UnnamedPortlet]

Portlet Name

Enter a unique name for the portlet, and select a portal to which the portlet should be associated. The portlet name will be used to name the directory that contains wizard-generated portlet files.

Portlet Name: BondStreet

Portlet Category: portlets

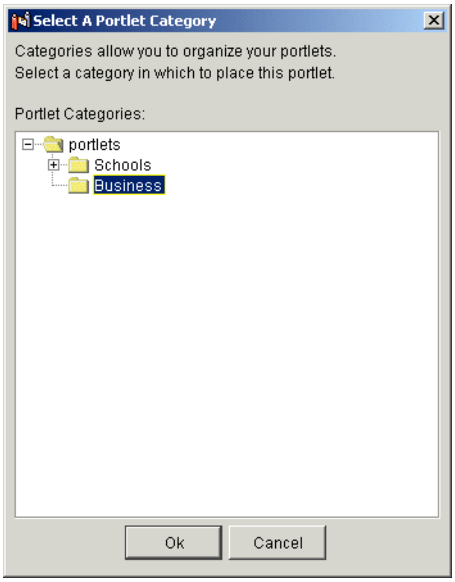
Portal: MyNewPortal1

Portlet Description (optional):

Select a category for the portlet

2. The Select a portlet category screen appears, as shown in Figure 11-26. Select Business and click **OK**.

Figure 11-26 Placing a new portlet inside an existing category



Categories allow you to organize your portlets.
Select a category in which to place this portlet.

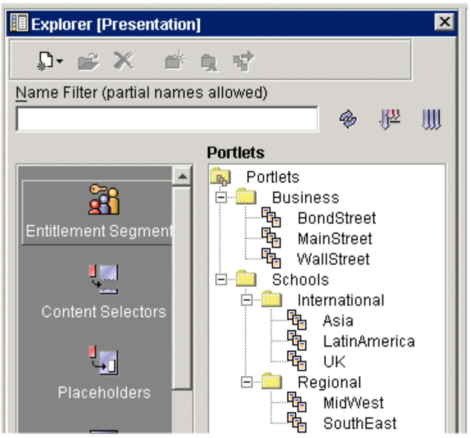
Portlet Categories:

- portlets
  - Schools
  - Business

3. Proceed with the rest of the steps in the Portlet Wizard, and repeat for the BondStreet, UK, and Asia portlets.

4. The resulting portlets should resemble those shown in Figure 11-27.

Figure 11-27 Portlets in their categories



- Portlets
  - Business
    - BondStreet
    - MainStreet
    - WallStreet
  - Schools
    - International
      - Asia
      - LatinAmerica
      - UK
    - Regional
      - MidWest
      - SouthEast

Portlets and the Framework

The BEA WebLogic platform provides extensive support for customizing portlets, including JSP tag libraries designed to expose robust functionality while requiring minimal Java scripting within the actual JSPs that constitute a portlet. Use the information in this section to create the following portlet types:

- **Simple JSP Portlets**: These portlets can contain scriptlets and can invoke personalization features, but do not use individual WebFlows.
- **WebFlow Portlets**: The portal wizard can be used to create three different Webflow portlets. Navigation and inter-portlet scenarios can also be realized by customizing a WebFlow file and associating it with an individual portlet.
- **Web Services Portlets**: Code can be added to a portlet that invokes various interactions with other programs, either locally or across the Internet.

The remainder of this section explains each portlet type, providing several examples of each and explaining some techniques for creating portlets that best make use of the BEA WebLogic platform.

Simple JSP Portlets

To illustrate some of the power, flexibility, and ease-of-use inherent in the WebLogic Portal platform, this subsection includes the following examples:

- The scriptDemo Portlet
- Calling ActiveX Components from a Portlet

The scriptDemo Portlet

The scriptDemo portlet, shown in Figure 11-30, appears to the left on the Home page of an example portal created using the Portal Wizard.

Preparing

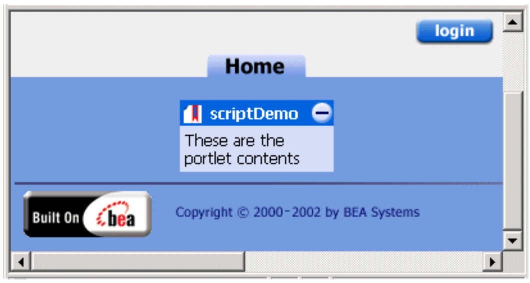
Make the following preparations to work through this example:

1. Create a new portal called "MyNewPortal1", and a new portal Web application "OldPortalWebApp", as explained in Step 2: Create the New Portal, in Chapter 2, "Creating a New Portal in a New Domain."
2. Create a new portlet, called "scriptDemo" as explained in Step 3: Add a Portlet, in Chapter 2, "Creating a New Portal in a New Domain."
3. Make the "scriptDemo" portlet visible, as explained in Step 4: Make New Portlet Visible, in Chapter 2, "Creating a New Portal in a New Domain."

Before Continuing

Ensure your portal looks similar to that shown in Figure 11-28 before a user is logged in, or that shown in Figure 11-29 with a user logged in.

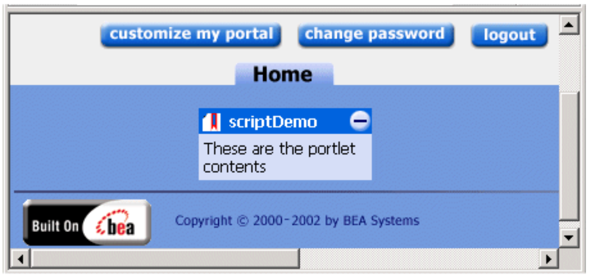
Figure 11-28 Starting point with anonymous user



These are the portlet contents

Copyright © 2000-2002 by BEA Systems

Figure 11-29 Starting Point with user logged on



These are the portlet contents

Copyright © 2000-2002 by BEA Systems

Installing the scriptDemo portlet

With the raw materials are in place, you can now transform the portlet by replacing the `content.jsp` to change its behavior. Listing 11-2 shows a very simple scriptlet that illustrates how to use generic Java scriptlets to interact with the Portal framework in a portlet.

Listing 11-2 scriptDemo portlet version 1

```
<center> Test Portlet </center>
<br><hr>
<%

A Simple Java Scriptlet
<br>
/*
Create a new String variable and set the value to an empty string.
*/
  String userName = "";
/*
```

To get the actual user name, first get the java.security.Principal object from the `Javax.servlet.http.HttpServletRequest` object using the `getUserPrincipal()` method. The request object is available to the JSP directly. For the Principal object, use the full package name.

```
*/

  java.security.Principal principal = request.getUserPrincipal();
/*
```

If the principal object is null then the user has not logged in. For this example, ignore the not-logged in case by using an if statement to only process the value if it is not null.

```
*/
  if (principal != null)
  {
/*
*/
    userName = principal.getName();
  }
%>
<%--
```

Display the value of the userName variable using a scriptlet. Note that the scriptlet is embedded in the HTML and is denoted by the <%= %> block.

```
--%>
The user name is : <%= userName %>
<br>
```

Steps

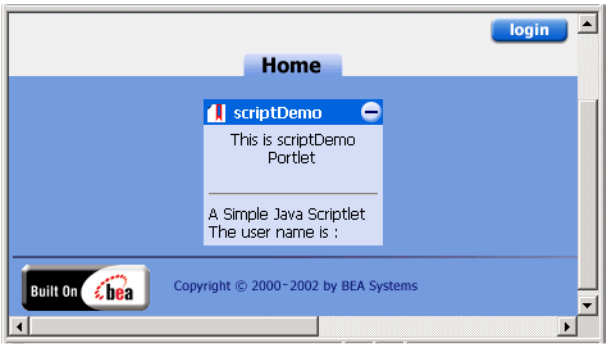
Take the following steps to place this new code inside the scriptDemo portlet:

1. In a text editor, save the code from Listing 11-2 as `content.jsp` in the following directory:

   ```
   <BEA_HOME>\portalDomain\beaApps\portalApp\
   <PortalWebApp>\portlets\scriptDemo\
   ```

2. Refresh the browser, and verify that the results resemble Figure 11-30. Notice that because the portlet content was altered, the user has been logged out. Because the portlet now appears with an anonymous user, you can see that no user name appears in the portlet.

Figure 11-30 scriptDemo portlet with anonymous user



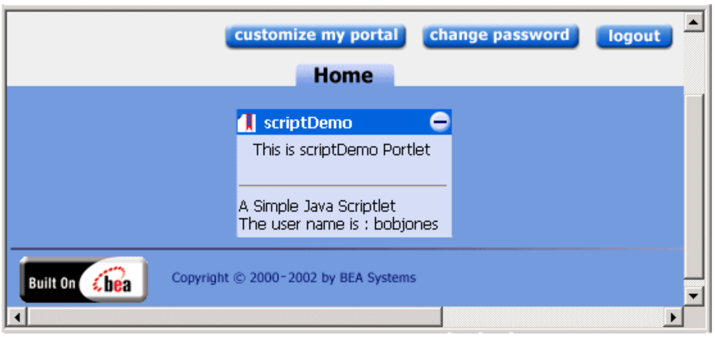
This is scriptDemo Portlet

A Simple Java Scriptlet
The user name is :

3. Click **Login** in the top right corner of the portal.
4. When the Portal Login page appears, click **Sign up now** under the New Users column to the right.
5. When the New User Registration page appears, create a new user called “bobjones”, and enter “password” in the password fields, and click **Submit**.

Figure 11-31 shows what the code in the scriptDemo portlet displays when a user is logged on. Notice that the user name is filled in.

Figure 11-31 scriptDemo portlet with a user logged on



This is scriptDemo Portlet

A Simple Java Scriptlet
The user name is : bobjones

Copyright © 2000-2002 by BEA Systems

Note: For more information on using JSP tags in your custom portlets, consult the following resources:

- JavaServer Page Reference Guide
- WebLogic Portal *Javadoc* API documentation.

Calling ActiveX Components from a Portlet

To call an ActiveX component from within your portlet, use the HTML <OBJECT> tag, as shown in Listing 11-3 and Listing 11-4.

These examples include a CODEBASE parameter so that if the local machine doesn't have the component installed in their registry, the component can be downloaded from Microsoft.

Note: In these examples, the portal is only communicating with the ActiveX components through HTML, which comes back from the portal to the browser, and tells the browser to load and run the ActiveX component.

Listing 11-3 Calling Outlook Using Active X - Calendar Example

```
<OBJECT CLASSID="clsid:0006F063-0000-0000-C000-000000000046"

CODEBASE="http://activex.microsoft.com/activex/controls/office/
outlctlx.CAB#ver=9,0,0,3203"

    id=OVCtl1 width=100% height=200>
```

```
  <param name="Folder" value="Calendar">
  <param name="Namespace" value="MAPI">
  <param name="Restriction" value="">
  <param name="DeferUpdate" value="0">
</OBJECT>
```

Listing 11-4 Calling Outlook Using Active X - Inbox Example

```
<OBJECT CLASSID="clsid:0006F063-0000-0000-C000-000000000046"
CODEBASE="http://activex.microsoft.com/activex/controls/office/ou
tlctlx.CAB#
ver=9,0,0,3203"
        id=OVCtl1 width=100% height=200>
  <param name="Folder" value="Inbox">
  <param name="Namespace" value="MAPI">
  <param name="Restriction" value="">
  <param name="DeferUpdate" value="0">
</OBJECT>
```

Note: ActiveX components only work inside Microsoft Internet Explorer with appropriate security settings which allow them to run.

WebFlow Portlets

This section begins by showing the types of webflow portlets that can be created by the portlet wizard. Next, some fundamental aspects of WebFlow are illustrated using simple scripting examples. The following topics are handled in the section:

- Three Webflow Portlets

- How a Portlet Handles a Refresh Event
- Making a Portlet Respond to a Custom Event
- Sharing State from One Portlet to Another

Three Webflow Portlets

The Portlet Wizard in the E-Business Control Center can now produce three different types of Webflow portlets:

- Navigation Bar, in which each page in the set has a navigation bar with links to every other page in the set.
- Sequential, in which pages are traversed in sequence via the Next and Previous buttons.
- Hierarchical, in which a parent page features links to each child page. Child pages link back to the parent, but not to one another.

Preparing

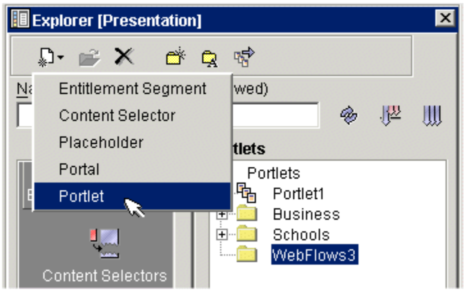
This section illustrates adding new Webflow portlets to the application `NewPWApp`, as shown throughout this chapter. To work through these examples, make sure the portal server is running, and that portal-project is open in the E-Business Control Center.

Creating a Navigation Bar Webflow Portlet

This procedure shows how to use the portlet wizard to create a Navigation Bar Webflow portlet.

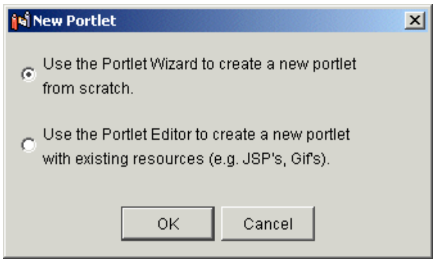
1. From the Presentation Tab in the E-Business Control Center, select New Portlet, as shown in Figure 11-32.

Figure 11-32 Starting the Portlet Wizard



2. Elect to use the portlet wizard, as shown in Figure 11-33, and click **OK**.

Figure 11-33 Elect to use the Portlet Wizard

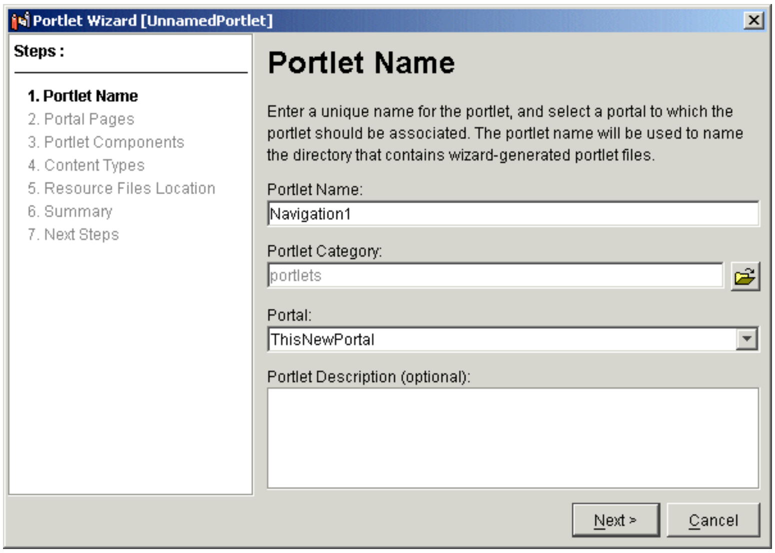

New Portlet

[✓ Use the Portlet Wizard to create a new portlet from scratch.]

Use the Portlet Editor to create a new portlet with existing resources (e.g. JSP's, Gif's).

3. Name the portlet Navigation, as shown in Figure 11-34, then click **Next**.

Figure 11-34 Naming the Navigation Portlet



Portlet Name

Enter a unique name for the portlet, and select a portal to which the portlet should be associated. The portlet name will be used to name the directory that contains wizard-generated portlet files.

Portlet Name: Navigation1

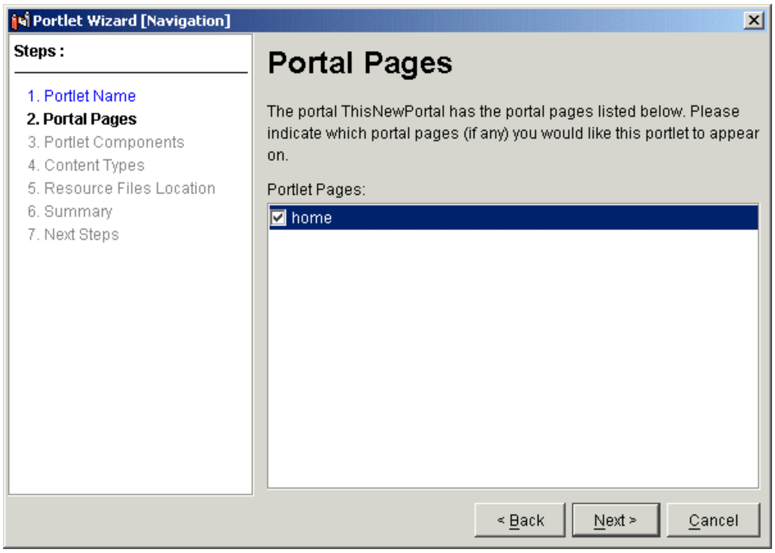
Portlet Category: portlets

Portal: ThisNewPortal

Portlet Description (optional):

4. Associate the new portlet with the Home page, as shown in Figure 11-35, and click **Next**.

Figure 11-35 Associate the portlet with the Home page



Portal Pages

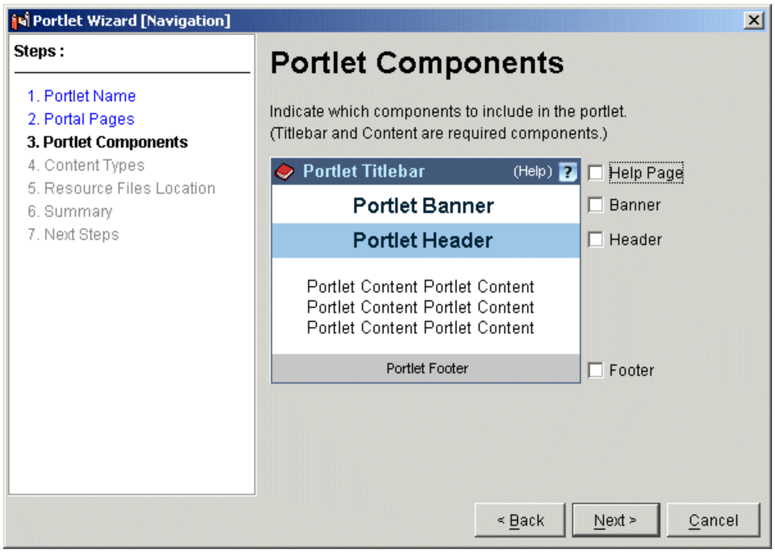
The portal ThisNewPortal has the portal pages listed below. Please indicate which portal pages (if any) you would like this portlet to appear on.

Portlet Pages:

[✓] home

5. Select portlet components, as shown in Figure 11-36, and click **Next**.

Figure 11-36 Select components



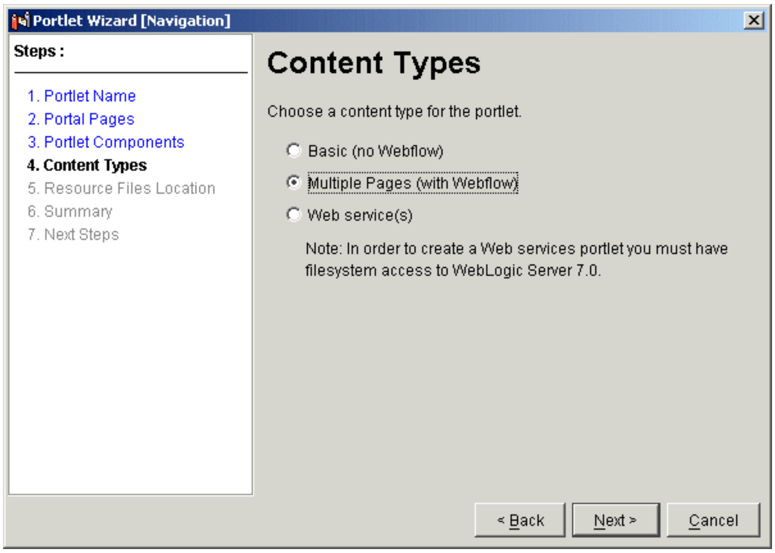
Portlet Components

Indicate which components to include in the portlet. (Titlebar and Content are required components.)

[ ] Help Page
[ ] Banner
[ ] Header
[ ] Footer

6. Select webflow content type, as shown in Figure 11-37, and click **Next**.

Figure 11-37 Select Webflow content type



Content Types

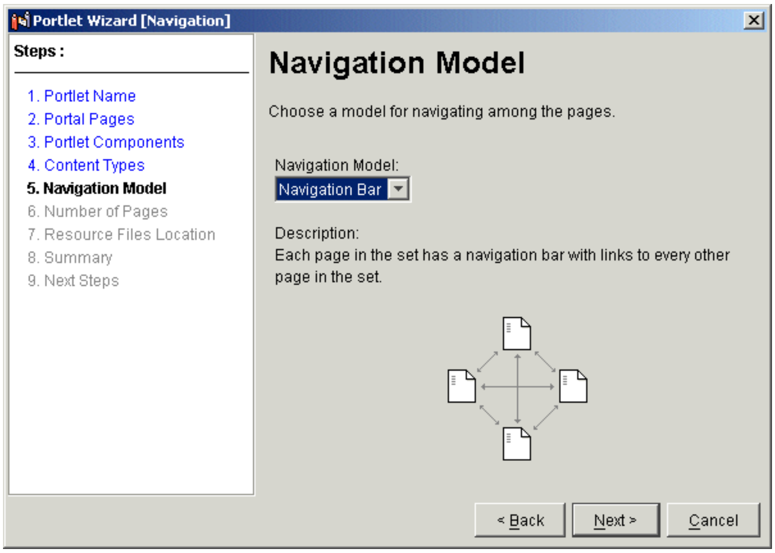
Choose a content type for the portlet.

- ○ Basic (no Webflow)
- ● Multiple Pages (with Webflow)
- ○ Web service(s)

Note: In order to create a Web services portlet you must have filesystem access to WebLogic Server 7.0.

7. Select Navigation Bar model, as shown in Figure 11-38, and click **Next**.

Figure 11-38 Select Navigation Bar Model



Navigation Model

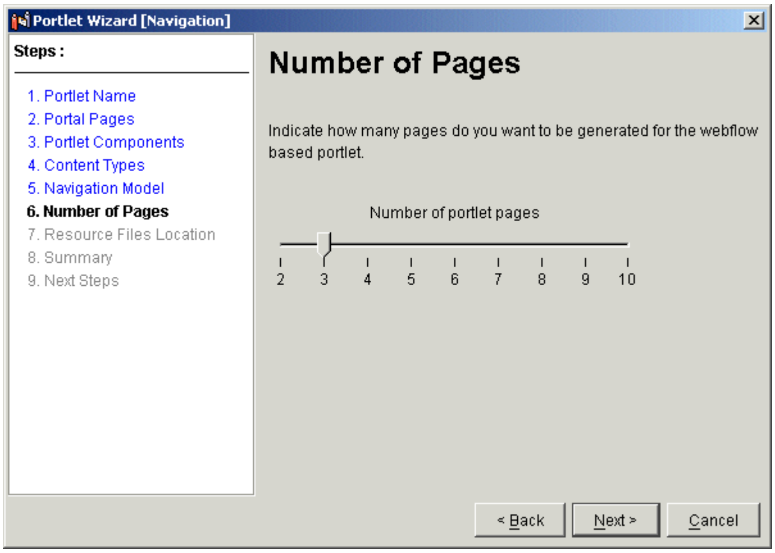
Choose a model for navigating among the pages.

Navigation Model: Navigation Bar

Description:
Each page in the set has a navigation bar with links to every other page in the set.

8. Select number of pages, as shown in Figure 11-39, and click **Next**.

Figure 11-39 Select number of pages



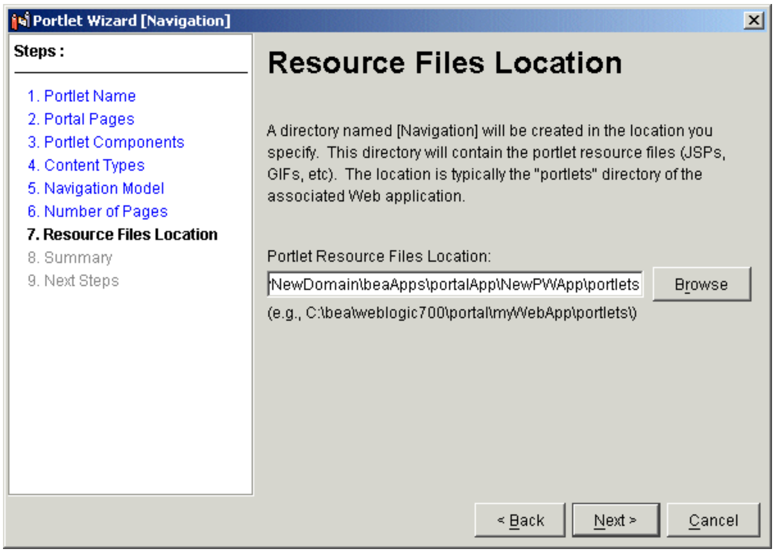
Number of Pages

Indicate how many pages do you want to be generated for the webflow based portlet.

Number of portlet pages: 3

9. Confirm resource files location, as shown in Figure 11-40, and click **Next**.

Figure 11-40 Confirm Resource Files Location



Resource Files Location

A directory named [Navigation] will be created in the location you specify. This directory will contain the portlet resource files (JSPs, GIFs, etc). The location is typically the "portlets" directory of the associated Web application.

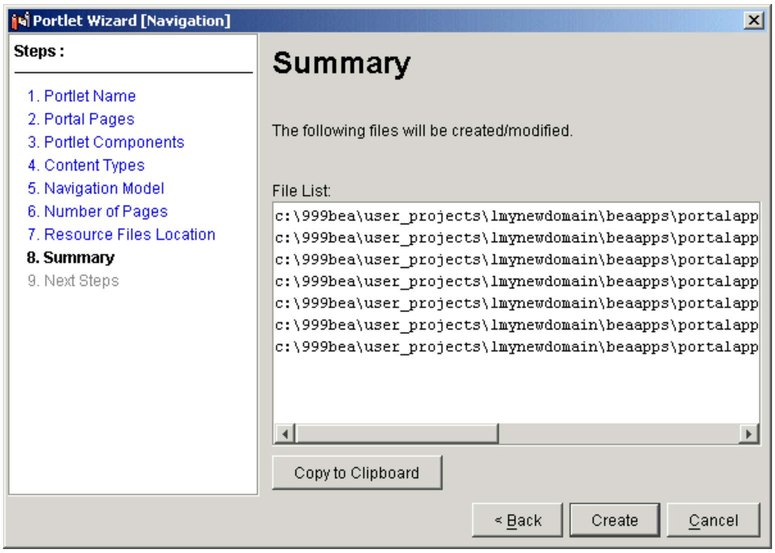
Portlet Resource Files Location:

`NewDomain\beaApps\portalApp\NewPWApp\portlets`

(e.g., C:\bea\weblogic700\portal\myWebApp\portlets\)

10. Confirm summary of files to be created, as shown in Figure 11-41, and click **Create**.

Figure 11-41 Confirm Summary



Summary

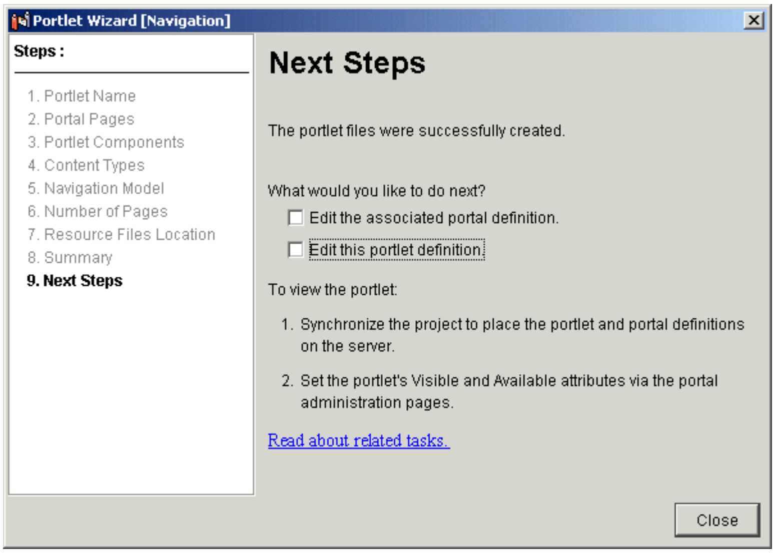
The following files will be created/modified.

File List:

```
c:\999bea\user_projects\lmynewdomain\beaapps\portalapp
c:\999bea\user_projects\lmynewdomain\beaapps\portalapp
c:\999bea\user_projects\lmynewdomain\beaapps\portalapp
c:\999bea\user_projects\lmynewdomain\beaapps\portalapp
c:\999bea\user_projects\lmynewdomain\beaapps\portalapp
c:\999bea\user_projects\lmynewdomain\beaapps\portalapp
c:\999bea\user_projects\lmynewdomain\beaapps\portalapp
```

11. Confirm next steps, as shown in Figure 11-42, and click **Close**.

Figure 11-42 Confirm Next Steps



Next Steps

The portlet files were successfully created.

What would you like to do next?

- [ ] Edit the associated portal definition.
- [ ] Edit this portlet definition.

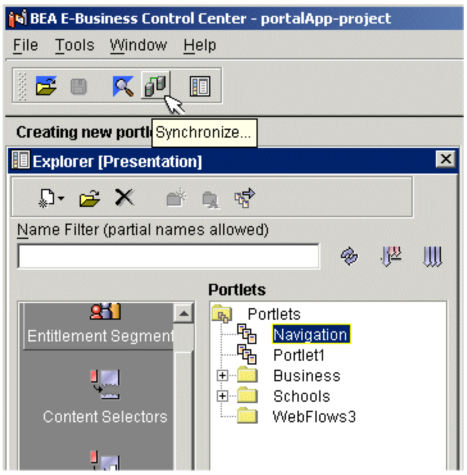
To view the portlet:

1. Synchronize the project to place the portlet and portal definitions on the server.
2. Set the portlet's Visible and Available attributes via the portal administration pages.

Read about related tasks.

12. Synchronize the project, as shown in Figure 11-43.

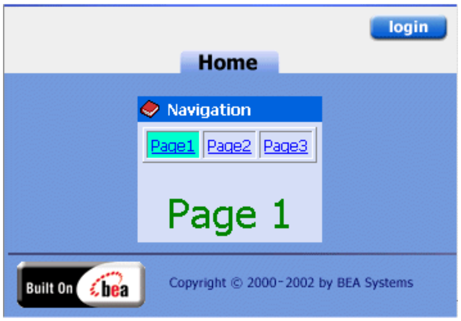
Figure 11-43 Synchronize project



13. Use the WebLogic Portal Administration Tools to make the portlet visible and available, as shown in the section called "Step 4: Make the New Portlet Visible and Available" on page 5.

14. View the portlet, as shown in Figure 11-44.

Figure 11-44 Viewing the Navigation Bar Portlet

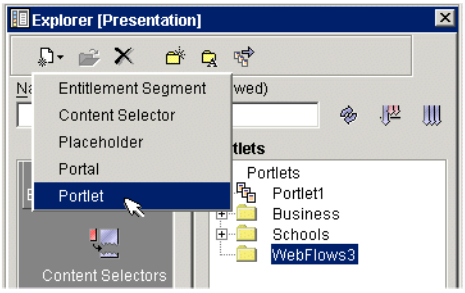


Creating a Sequential Webflow Portlet

This procedure shows how to use the portlet wizard to create a Sequential Webflow portlet.

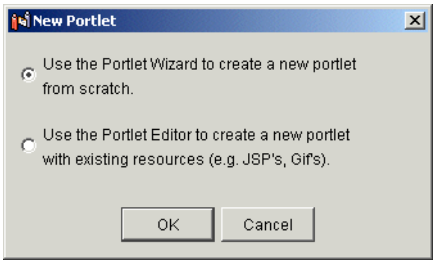
1. From the Presentation Tab in the E-Business Control Center, select New Portlet, as shown in Figure 11-45.

Figure 11-45 Starting the Portlet Wizard



2. Elect to use the portlet wizard, as shown in Figure 11-46, and click **OK**.

Figure 11-46 Elect to use the Portlet Wizard

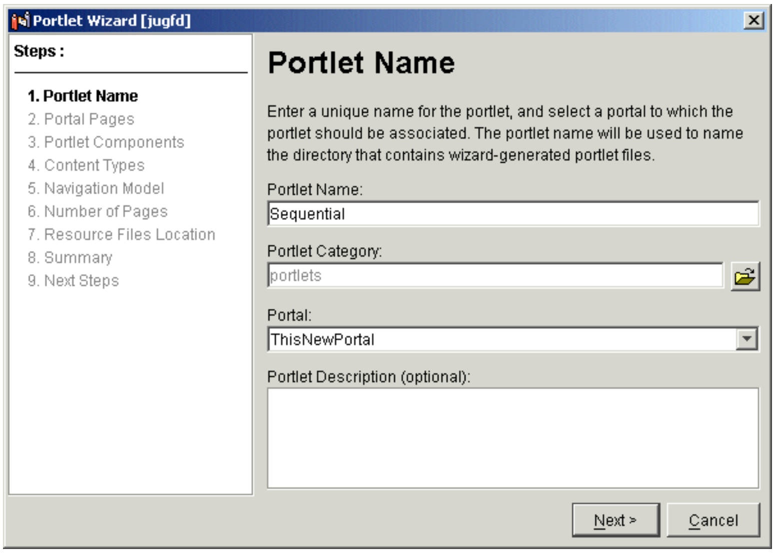


[✓ Use the Portlet Wizard to create a new portlet from scratch.]

Use the Portlet Editor to create a new portlet with existing resources (e.g. JSP's, Gif's).

3. Name the portlet Sequential, as shown in Figure 11-47, then click **Next**.

Figure 11-47 Naming the Sequential Webflow Portlet



Portlet Name

Enter a unique name for the portlet, and select a portal to which the portlet should be associated. The portlet name will be used to name the directory that contains wizard-generated portlet files.

Portlet Name: Sequential

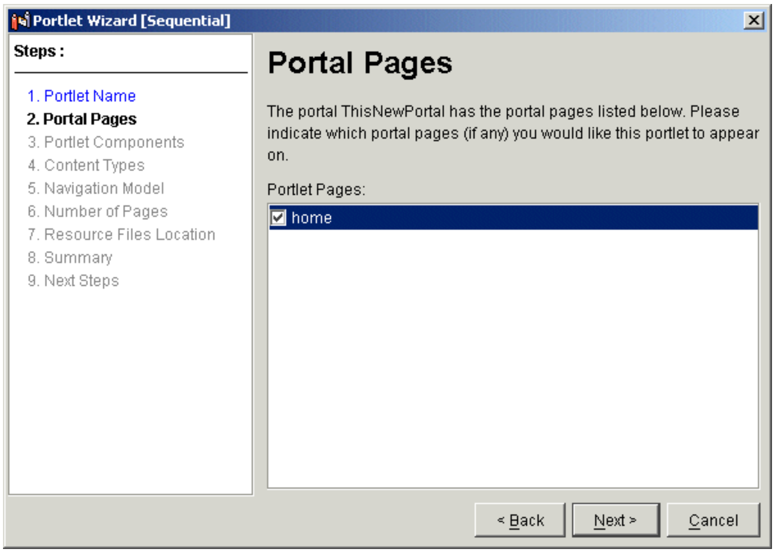
Portlet Category: portlets

Portal: ThisNewPortal

Portlet Description (optional):

4. Associate the portlet with the Home page, as shown in Figure 11-48, and click **Next**.

Figure 11-48 Associate the portlet with the Home page



Portal Pages

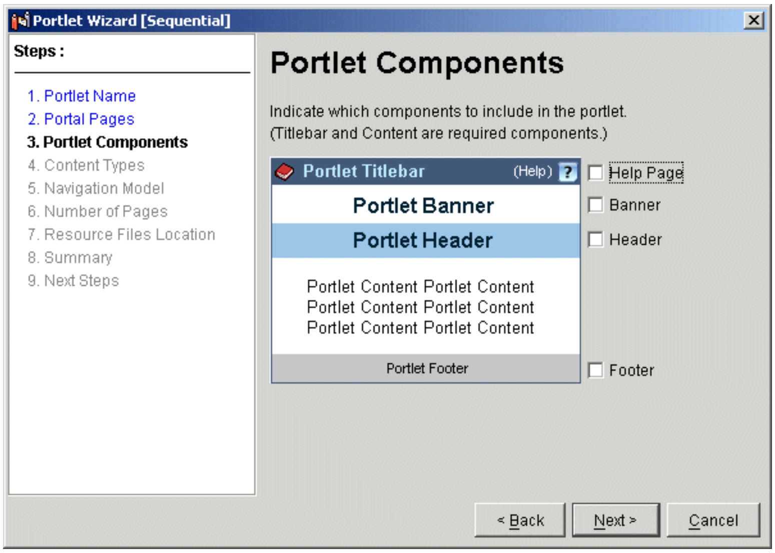
The portal ThisNewPortal has the portal pages listed below. Please indicate which portal pages (if any) you would like this portlet to appear on.

Portlet Pages:

[✓] home

5. Select portlet components, as shown in Figure 11-49, and click **Next**.

Figure 11-49 Select Portlet Components



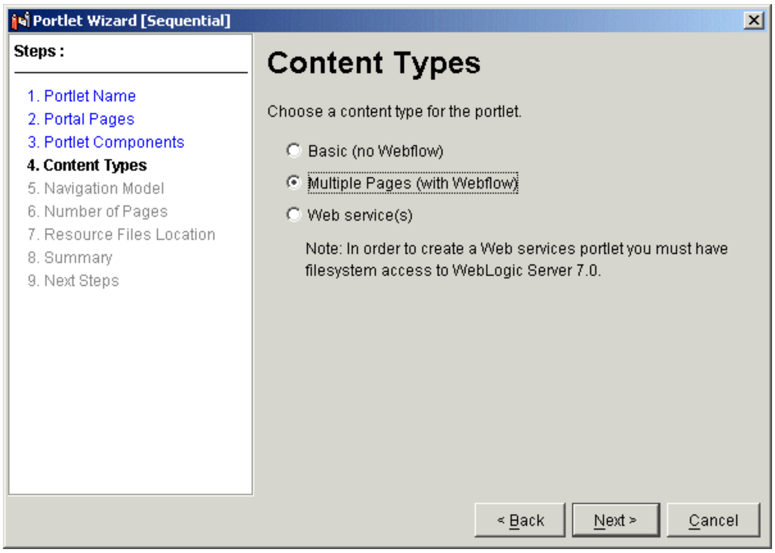
Portlet Components

Indicate which components to include in the portlet.
(Titlebar and Content are required components.)

☐ Help Page
☐ Banner
☐ Header
☐ Footer

6. Select webflow content type, as shown in Figure 11-50, and click **Next**.

Figure 11-50 Select Webflow content type



Content Types

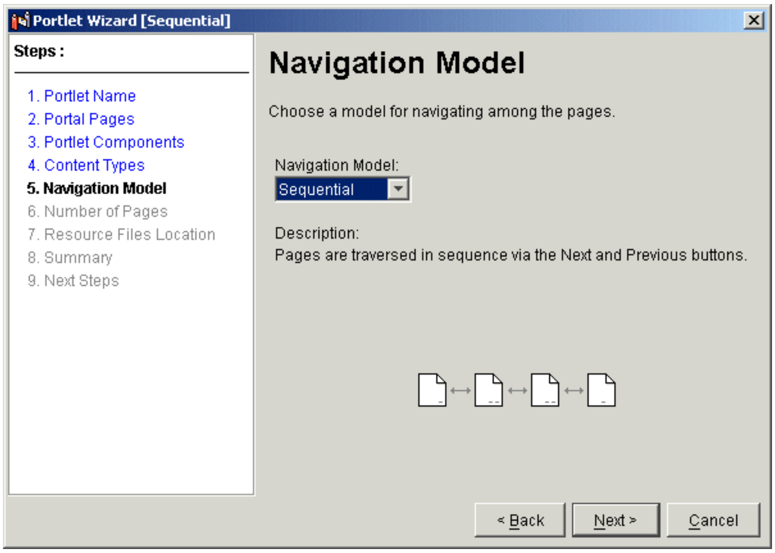
Choose a content type for the portlet.

( ) Basic (no Webflow)
[✓ Multiple Pages (with Webflow)]
( ) Web service(s)

Note: In order to create a Web services portlet you must have filesystem access to WebLogic Server 7.0.

7. Select Sequential Navigation model, as shown in Figure 11-51, and click **Next**.

Figure 11-51 Select Sequential model



Navigation Model

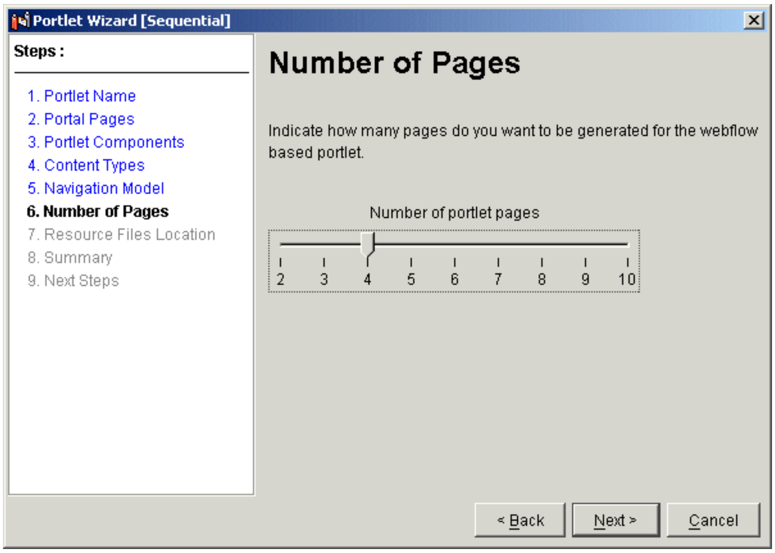
Choose a model for navigating among the pages.

Navigation Model: Sequential

Description:
Pages are traversed in sequence via the Next and Previous buttons.

8. Select number of pages, as shown in Figure 11-52, and click **Next**.

Figure 11-52 Select number of pages



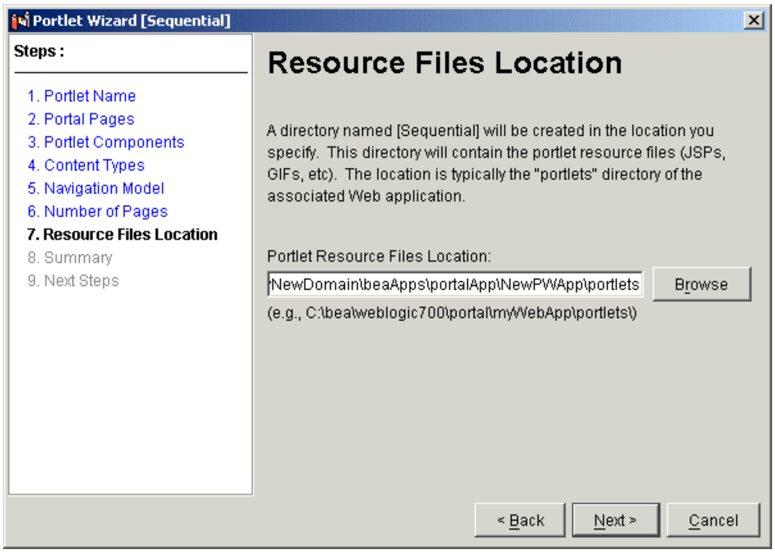
Number of Pages

Indicate how many pages do you want to be generated for the webflow based portlet.

Number of portlet pages: 4

9. Confirm resource files Location, as shown in Figure 11-53, and click **Next**.

Figure 11-53 Confirm Resource Files Location



Resource Files Location

A directory named [Sequential] will be created in the location you specify. This directory will contain the portlet resource files (JSPs, GIFs, etc). The location is typically the "portlets" directory of the associated Web application.

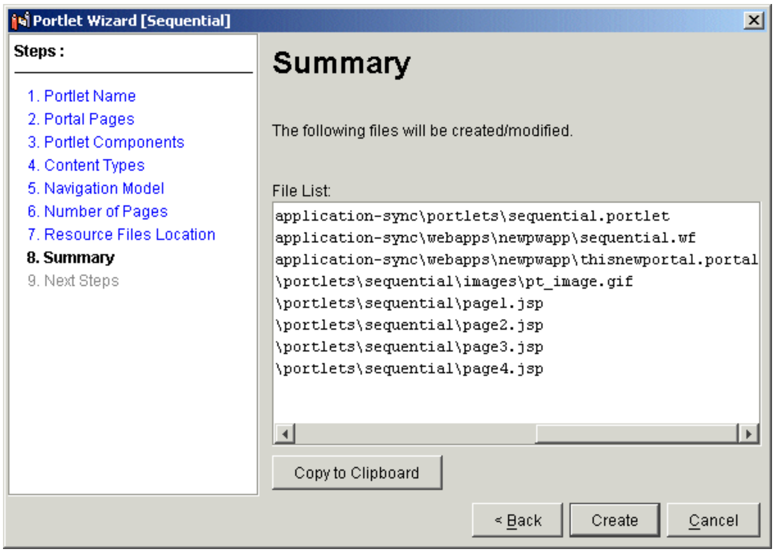
Portlet Resource Files Location:

NewDomain\beaApps\portalApp\NewPWApp\portlets

(e.g., C:\bea\weblogic700\portal\myWebApp\portlets\)

10. Confirm summary of files to be created, as shown in Figure 11-54, and click **Create**.

Figure 11-54 Confirm Summary



Summary

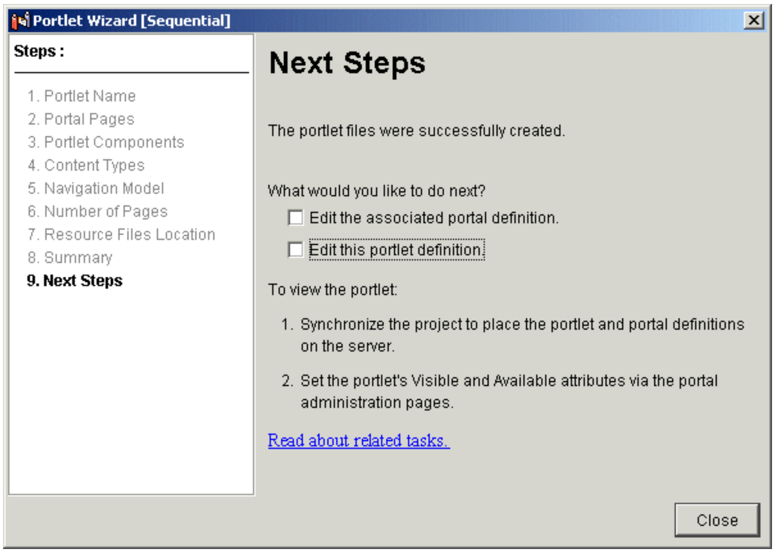
The following files will be created/modified.

File List:

```
application-sync\portlets\sequential.portlet
application-sync\webapps\newpwapp\sequential.wf
application-sync\webapps\newpwapp\thisnewportal.portal
\portlets\sequential\images\pt_image.gif
\portlets\sequential\page1.jsp
\portlets\sequential\page2.jsp
\portlets\sequential\page3.jsp
\portlets\sequential\page4.jsp
```

11. Confirm next steps, as shown in Figure 11-55, and click **Close**.

Figure 11-55 Confirm Next Steps



Next Steps

The portlet files were successfully created.

What would you like to do next?

- [ ] Edit the associated portal definition.
- [ ] Edit this portlet definition.

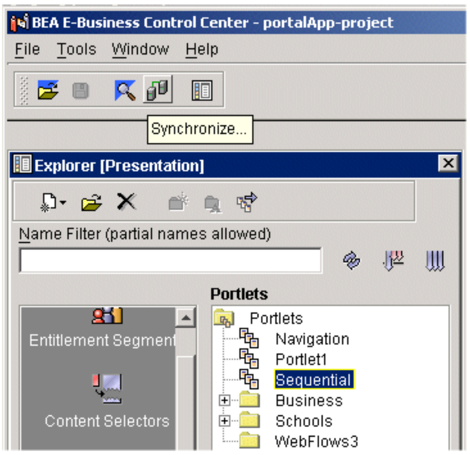
To view the portlet:

1. Synchronize the project to place the portlet and portal definitions on the server.
2. Set the portlet's Visible and Available attributes via the portal administration pages.

Read about related tasks.

12. Synchronize the project, as shown in Figure 11-56.

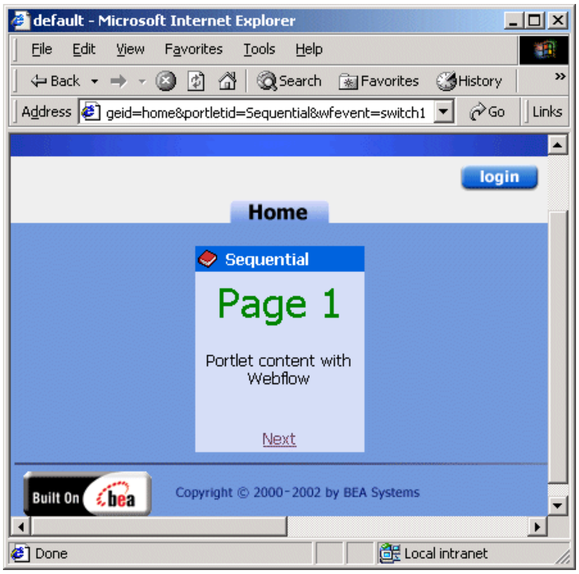
Figure 11-56 Synchronize project



13. Use the WebLogic Portal Administration Tools to make the portlet visible and available, as shown in the section called "Step 4: Make the New Portlet Visible and Available" on page 11-5.

14. View the portlet, as shown in Figure 11-57.

Figure 11-57 Viewing the Sequential Webflow Portlet

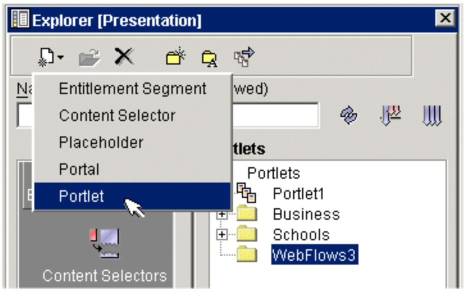

default - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address geid=home&portletid=Sequential&wfevent=switch1
Go Links
login
Home
Sequential
Page 1
Portlet content with Webflow
Next
Built On bea
Copyright © 2000-2002 by BEA Systems
Done
Local intranet

Creating a Hierarchical Webflow Portlet

This procedure shows how to use the portlet wizard to create a Hierarchical Webflow portlet.

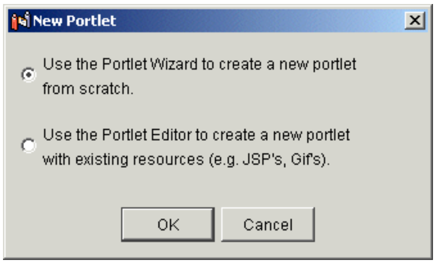
1. From the Presentation Tab in the E-Business Control Center, select New Portlet, as shown in Figure 11-58.

Figure 11-58 Starting the Portlet Wizard



2. Elect to use the portlet wizard, as shown in Figure 11-59, and click **OK**.

Figure 11-59 Elect to use the Portlet Wizard



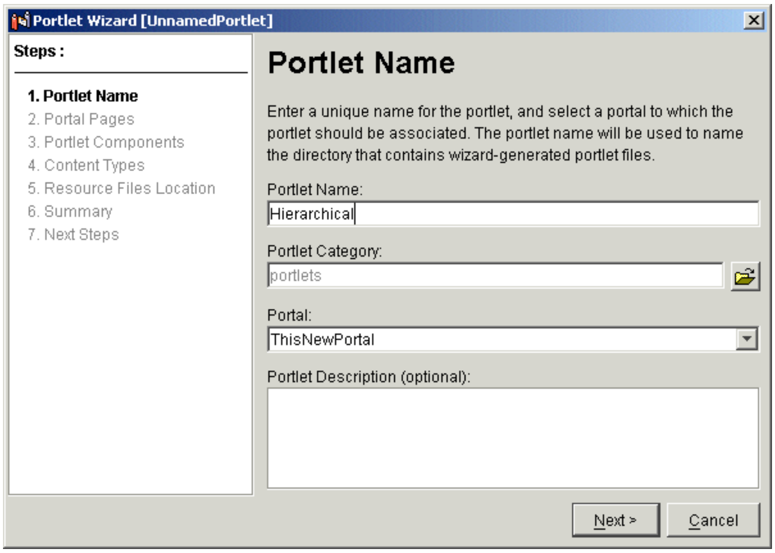
New Portlet

[✓ Use the Portlet Wizard to create a new portlet from scratch.]

Use the Portlet Editor to create a new portlet with existing resources (e.g. JSP's, Gif's).

3. Name the portlet Hierarchical, as shown in Figure 11-60, then click **Next**.

Figure 11-60 Naming the Hierarchical Portlet



Portlet Wizard [UnnamedPortlet]

Portlet Name

Enter a unique name for the portlet, and select a portal to which the portlet should be associated. The portlet name will be used to name the directory that contains wizard-generated portlet files.

Portlet Name: Hierarchical

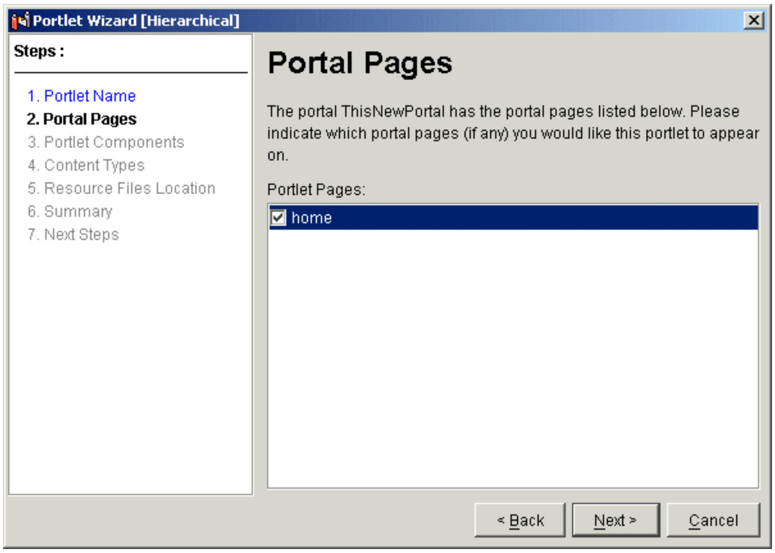
Portlet Category: portlets

Portal: ThisNewPortal

Portlet Description (optional):

4. Associate the new portlet with the Home page, as shown in Figure 11-61, and click **Next**.

Figure 11-61 Associate the portlet with the Home page



Portal Pages

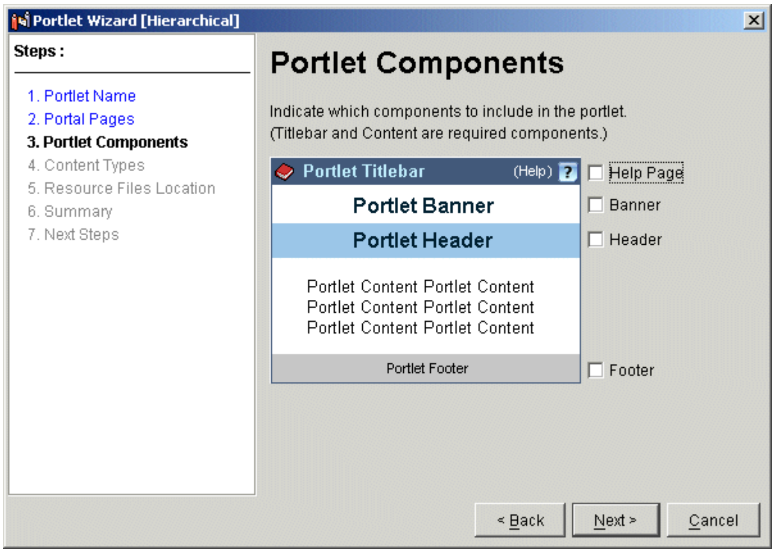
The portal ThisNewPortal has the portal pages listed below. Please indicate which portal pages (if any) you would like this portlet to appear on.

Portlet Pages:

[✓] home

5. Select portlet components, as shown in Figure 11-62, and click **Next**.

Figure 11-62 Select components



Portlet Components

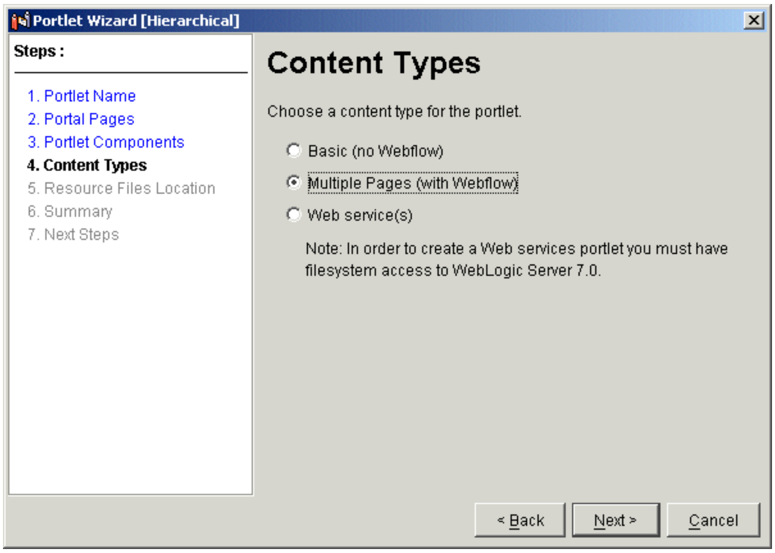
Indicate which components to include in the portlet. (Titlebar and Content are required components.)

Portlet Titlebar
(Help)
Portlet Banner
Portlet Header
Portlet Content Portlet Content
Portlet Content Portlet Content
Portlet Content Portlet Content
Portlet Footer

[ ] Help Page
[ ] Banner
[ ] Header
[ ] Footer

6. Select webflow content type, as shown in Figure 11-63, and click **Next**.

Figure 11-63 Select Webflow content type



Portlet Wizard [Hierarchical]

Steps :

1. Portlet Name
2. Portal Pages
3. Portlet Components
4. **Content Types**
5. Resource Files Location
6. Summary
7. Next Steps

Content Types

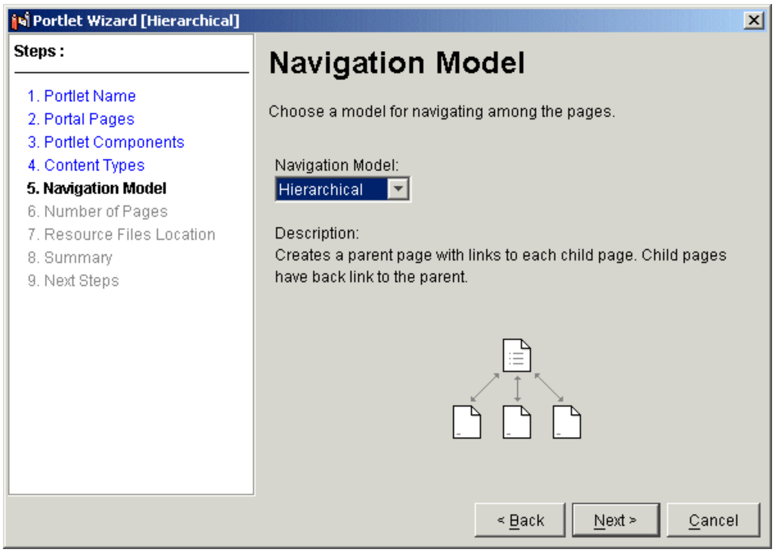
Choose a content type for the portlet.

Basic (no Webflow) [✓ Multiple Pages (with Webflow)] Web service(s)

Note: In order to create a Web services portlet you must have filesystem access to WebLogic Server 7.0.

7. Select Hierarchical model, as shown in Figure 11-64, and click **Next**.

Figure 11-64 Select Hierarchical Model


Portlet Wizard [Hierarchical]

Steps :

1. Portlet Name
2. Portal Pages
3. Portlet Components
4. Content Types
5. **Navigation Model**
6. Number of Pages
7. Resource Files Location
8. Summary
9. Next Steps

Navigation Model

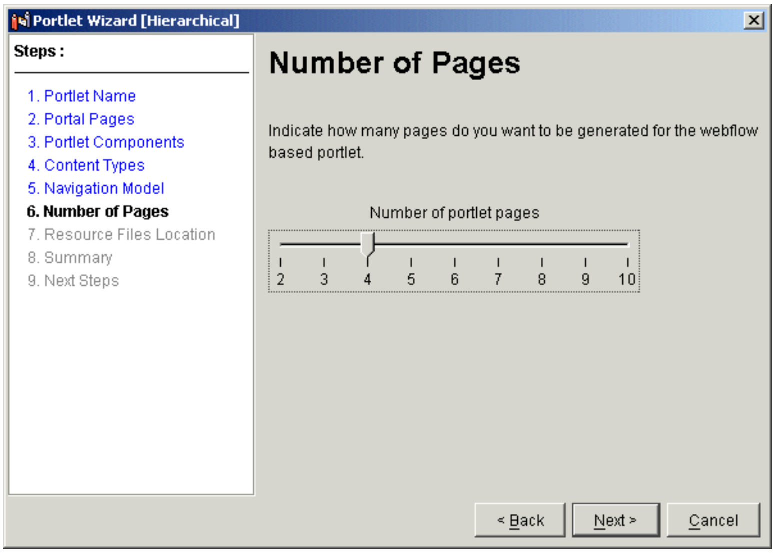
Choose a model for navigating among the pages.

Navigation Model: Hierarchical

Description:
Creates a parent page with links to each child page. Child pages have back link to the parent.

8. Select number of pages, as shown in Figure 11-65, and click **Next**.

Figure 11-65 Select number of pages


Portlet Wizard [Hierarchical]

Steps :

1. Portlet Name
2. Portal Pages
3. Portlet Components
4. Content Types
5. Navigation Model
6. **Number of Pages**
7. Resource Files Location
8. Summary
9. Next Steps

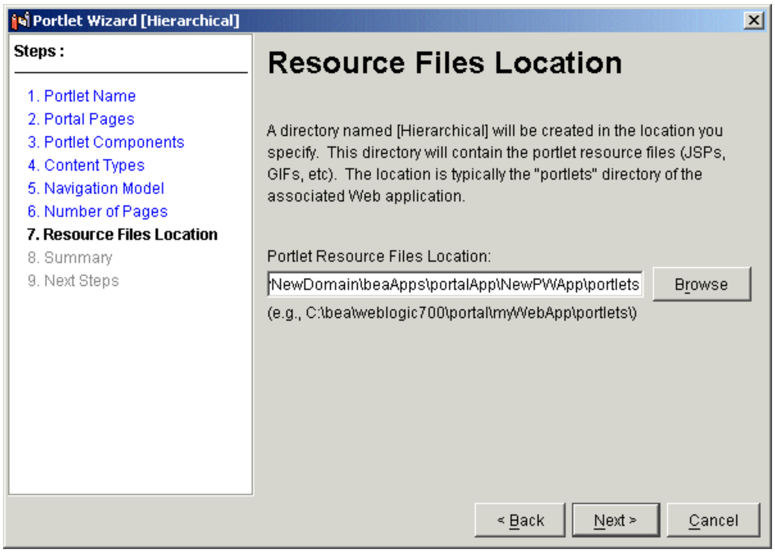
Number of Pages

Indicate how many pages do you want to be generated for the webflow based portlet.

Number of portlet pages: 4

9. Confirm resource files location, as shown in Figure 11-66, and click **Next**.

Figure 11-66 Confirm Resource Files Location



Resource Files Location

A directory named [Hierarchical] will be created in the location you specify. This directory will contain the portlet resource files (JSPs, GIFs, etc). The location is typically the "portlets" directory of the associated Web application.

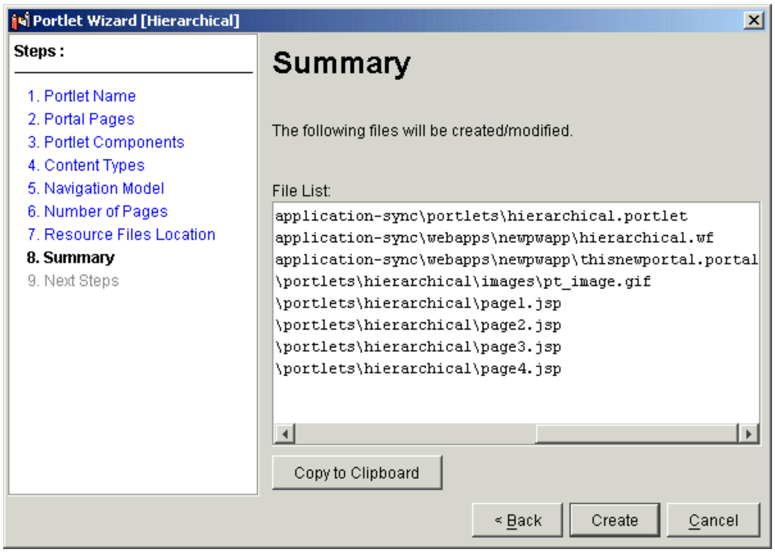
Portlet Resource Files Location:

NewDomain\beaApps\portalApp\NewPWApp\portlets

(e.g., C:\bea\weblogic700\portal\myWebApp\portlets\)

10. Confirm summary of files to be created, as shown in Figure 11-67, and click **Create**.

Figure 11-67 Confirm Summary



Summary

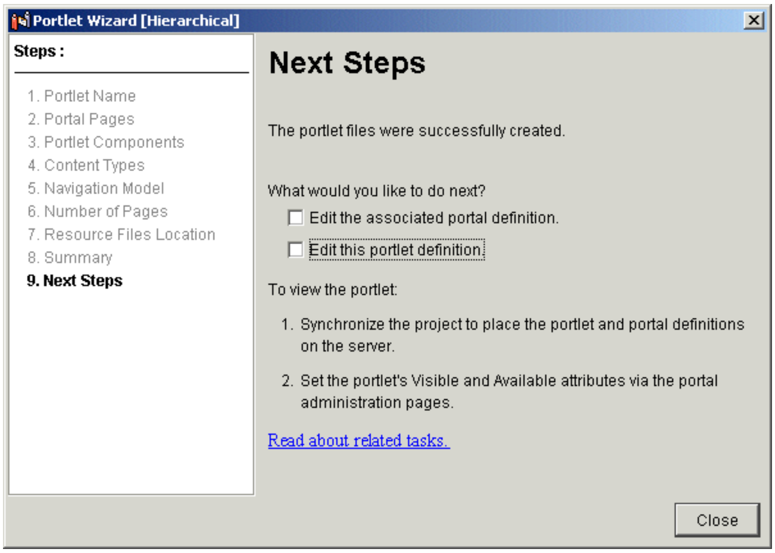
The following files will be created/modified.

File List:

```
application-sync\portlets\hierarchical.portlet
application-sync\webapps\newpwapp\hierarchical.wf
application-sync\webapps\newpwapp\thisnewportal.portal
\portlets\hierarchical\images\pt_image.gif
\portlets\hierarchical\page1.jsp
\portlets\hierarchical\page2.jsp
\portlets\hierarchical\page3.jsp
\portlets\hierarchical\page4.jsp
```

11. Confirm next steps, as shown in Figure 11-68, and click Close.

Figure 11-68 Confirm Next Steps



Next Steps

The portlet files were successfully created.

What would you like to do next?

☐ Edit the associated portal definition.

☐ Edit this portlet definition.

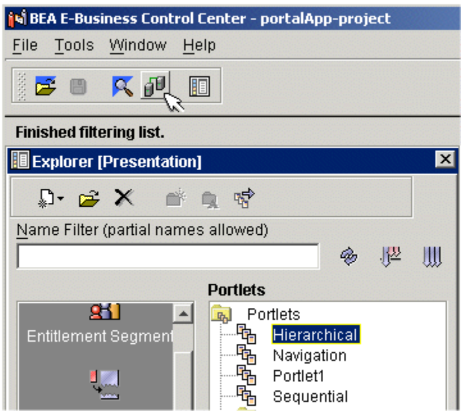
To view the portlet:

1. Synchronize the project to place the portlet and portal definitions on the server.
2. Set the portlet's Visible and Available attributes via the portal administration pages.

Read about related tasks.

12. Synchronize the project, as shown in Figure 11-69.

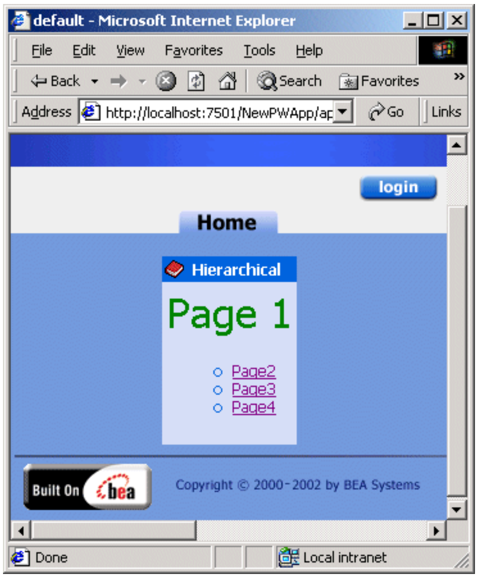
Figure 11-69 Synchronize project



13. Use the WebLogic Portal Administration Tools to make the portlet visible and available, as shown in the section called "Step 4: Make the New Portlet Visible and Available" on page 11-5.

14. View the portlet, as shown in Figure 11-70.

Figure 11-70 Viewing the Hierarchical Portlet


default - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:7501/NewPWApp/ap
Go Links
login
Home
Hierarchical
Page 1
Page2
Page3
Page4
Built On bea
Copyright © 2000-2002 by BEA Systems
Done
Local intranet

How a Portlet Handles a Refresh Event

To illustrate how portlets can use Webflow, observe how portlets handle the refresh event, as demonstrated in the bold code in Listing 11-5.

Listing 11-5 Adding refresh notification to a portlet

```
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%
System.out.println("Calling refresh on flowLet1 portlet.");
%>
<center>
<font size="6" color="green">Portlet 1</font><BR>
<p>
Portlet content with Webflow
<p>
<a href="<portlet:createWebflowURL event="switch1"/>">Next
Page</a>
<p>
</center>
```

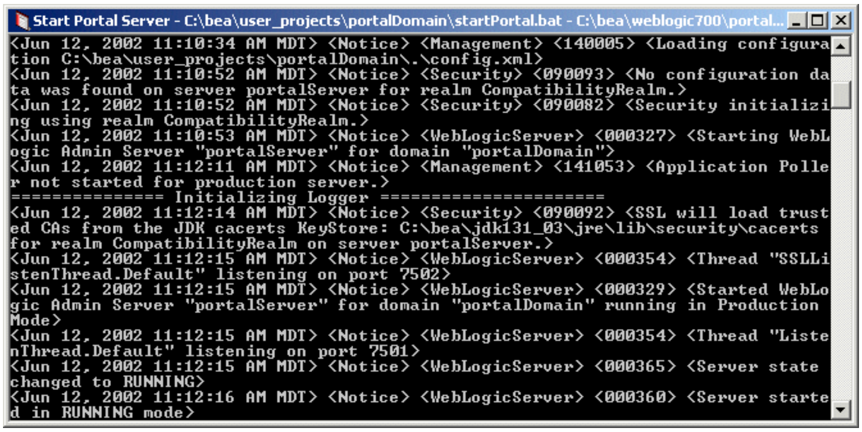
Continuing from the previous example, edit the text in the scriptDemo portlet as shown in the following steps:

1. Open the console by selecting it from the taskbar, as shown in Figure 11-71.

Figure 11-71 Open console window

   The console should resemble figure Figure 11-72.

Figure 11-72 Console window



Start Portal Server - C:\bea\user_projects\portalDomain\startPortal.bat - C:\bea\weblogic700\portal...

```
<Jun 12, 2002 11:10:34 AM MDT> <Notice> <Management> <140005> <Loading configura
tion C:\bea\user_projects\portalDomain\.\config.xml>
<Jun 12, 2002 11:10:52 AM MDT> <Notice> <Security> <090093> <No configuration da
ta was found on server portalServer for realm CompatibilityRealm.>
<Jun 12, 2002 11:10:52 AM MDT> <Notice> <Security> <090082> <Security initializi
ng using realm CompatibilityRealm.>
<Jun 12, 2002 11:10:53 AM MDT> <Notice> <WebLogicServer> <000327> <Starting WebL
ogic Admin Server "portalServer" for domain "portalDomain">
<Jun 12, 2002 11:12:11 AM MDT> <Notice> <Management> <141053> <Application Polle
r not started for production server.>
=============== Initializing Logger =====================
<Jun 12, 2002 11:12:14 AM MDT> <Notice> <Security> <090092> <SSL will load trust
ed CAs from the JDK cacerts KeyStore: C:\bea\jdk131_03\jre\lib\security\cacerts
for realm CompatibilityRealm on server portalServer.>
<Jun 12, 2002 11:12:15 AM MDT> <Notice> <WebLogicServer> <000354> <Thread "SSLLi
stenThread.Default" listening on port 7502>
<Jun 12, 2002 11:12:15 AM MDT> <Notice> <WebLogicServer> <000329> <Started WebLo
gic Admin Server "portalServer" for domain "portalDomain" running in Production
Mode>
<Jun 12, 2002 11:12:15 AM MDT> <Notice> <WebLogicServer> <000354> <Thread "Liste
nThread.Default" listening on port 7501>
<Jun 12, 2002 11:12:15 AM MDT> <Notice> <WebLogicServer> <000365> <Server state
changed to RUNNING>
<Jun 12, 2002 11:12:16 AM MDT> <Notice> <WebLogicServer> <000360> <Server starte
d in RUNNING mode>
```

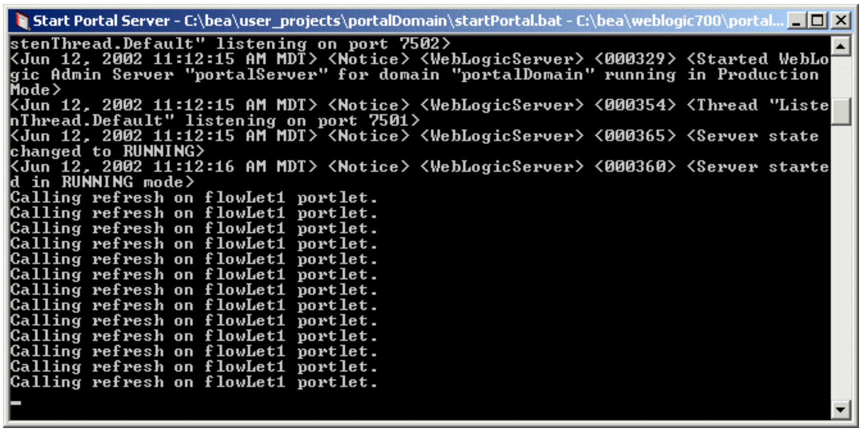
2. In a text editor, save the code from Listing 11-2 as `content.jsp` in the following directory:

   ```
   <BEA_HOME>\portalDomain\beaApps\portalApp\<PortalWebApp>\
   portlets\scriptDemo\
   ```

3. Refresh the browser, and verify that the portlets have not changed. Click the Webflows within each of the portlets, navigating back and forth a few times.

4. Return to the console and notice the output from flowLet1, as shown in Figure 11-73.

Figure 11-73 Refresh messages in Console


Start Portal Server - C:\bea\user_projects\portalDomain\startPortal.bat - C:\bea\weblogic700\portal...

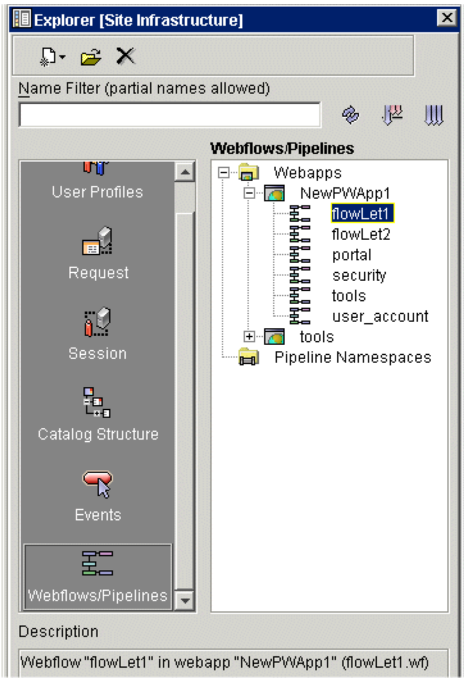
```
stenThread.Default" listening on port 7502>
<Jun 12, 2002 11:12:15 AM MDT> <Notice> <WebLogicServer> <000329> <Started WebLo
gic Admin Server "portalServer" for domain "portalDomain" running in Production
Mode>
<Jun 12, 2002 11:12:15 AM MDT> <Notice> <WebLogicServer> <000354> <Thread "Liste
nThread.Default" listening on port 7501>
<Jun 12, 2002 11:12:15 AM MDT> <Notice> <WebLogicServer> <000365> <Server state
changed to RUNNING>
<Jun 12, 2002 11:12:16 AM MDT> <Notice> <WebLogicServer> <000360> <Server starte
d in RUNNING mode>
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
Calling refresh on flowLet1 portlet.
_
```

Understanding Webflow Refresh Events in a Portlet

In Webflow, the Refresh event amounts to calling an entity called lastContentUrl, which merely allows you to trigger a portlet to refresh itself without specifying its name. By navigating through the various buttons and controls in the portal in this example while observing the console window for refresh messages, you can see when refresh events are caused by certain actions. To look more closely at the Webflow associated with flowLet1, take the following steps:

1. Looking at a Webflow: From the Site Infrastructure tab of the E-Business Control Center, select **WebFlows/Pipelines**.
2. Open **NewPWApp1** by double-clicking it, as shown in Figure 11-74.

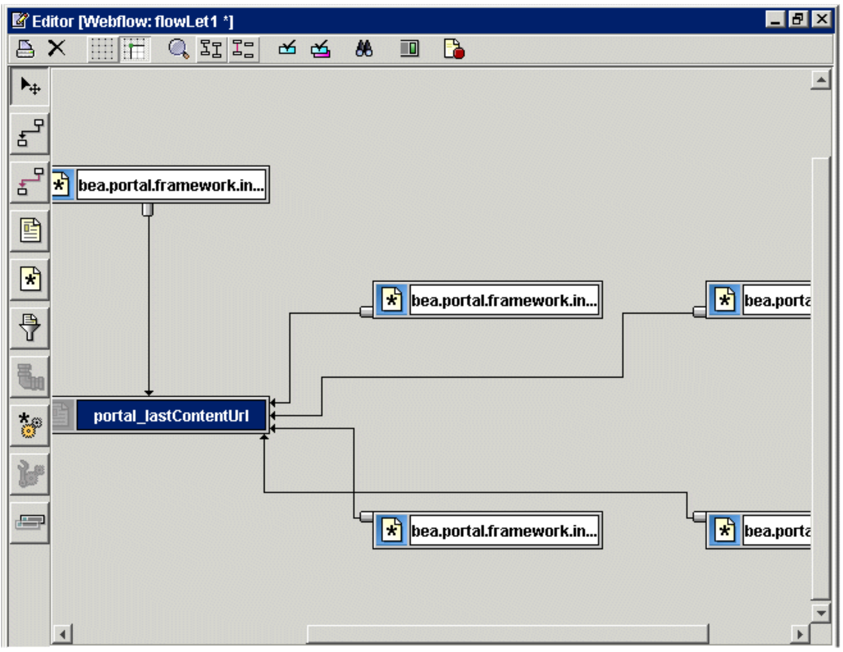
Figure 11-74 Opening Webflows within Portal Web Application



Webflow "flowLet1" in webapp "NewPWApp1" (flowLet1.wf)

3. Open the Webflow named **flowLet1** by double-clicking it, and examine the image that appears in the editor window. To the right side of the navigation mapping, notice that the minimize, maximize, unMinimize and unMaximize nodes are linked to `portal_lastContentUrl`, as shown in Figure 11-75.

Figure 11-75 Nodes linked to portal_lastContentUrl


Editor [Webflow: flowLet1 *]
bea.portal.framework.in...
bea.portal.framework.in...
bea.porta
portal_lastContentUrl
bea.portal.framework.in...
bea.porta

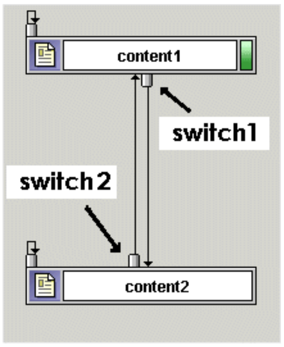
By default, refresh is always invoked on an entire portal, and can be called on every portlet. Any portlet with its own Webflow can be made to respond to all refresh events. To make this default behavior for a portlet, use the Webflow editor to open the portlet's Webflow and set *.refresh proxynode to `lastContentUrl`.

For an in-depth look at using Webflow, consult Chapter 9, "Setting Up Portal Navigation."

Making a Portlet Respond to a Custom Event

To explain custom events, consider the navigation mapping in the Webflow for flowLet1 portlet, shown in Figure 11-76.

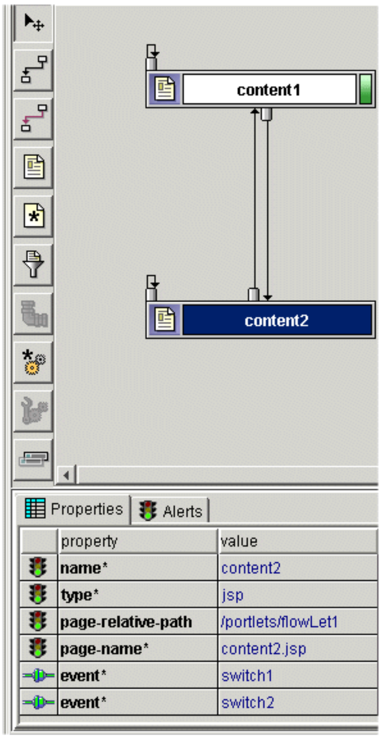
Figure 11-76 Navigation nodes in flowLet1 Webflow



The navigation between `content1.jsp` and `content2.jsp` is accomplished using pre-defined events called `switch1` and `switch2`.

By selecting one of the content nodes, you can examine the properties of the Webflow that links content1.jsp with content2.jsp in the flowLet1 portlet. The properties of the content2 presentation node are shown in detail in Figure 11-77.

Figure 11-77 Mappings for content2

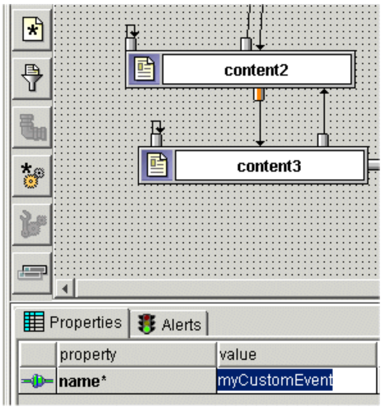


property	value
name*	content2
type*	jsp
page-relative-path	/portlets/flowLet1
page-name*	content2.jsp
event*	switch1
event*	switch2

Defining a Custom Event

Figure 11-78 shows how a custom event is assigned to a new presentation node. This event is named `myCustomEvent`, and the Event connector tool is used to link presentation node `content2` with presentation node `content3`.

Figure 11-78 Defining myCustomEvent



property	value
name*	myCustomEvent

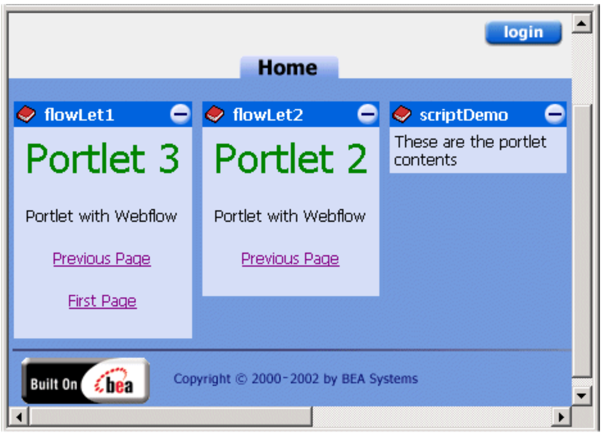
Invoking the Custom Event

To call the custom event in the portlet, include the following code in your JSP:

```
<a href="<portlet:createWebflowURL event="myCustomEvent"/>">Next
Page</a>
```

Figure 11-79 shows the third screen of flowLet1 after a new presentation node, called content3.jsp, has been added.

Figure 11-79 The third page of flowLet1 portlet



Sharing State from One Portlet to Another

Share state from one portlet to another by placing the arguments into the HttpSession object using an Input Processor. For instance, if `portletA` uses a form named "foo", use the IP to extract the “foo” form data, and `portletB` can get the form data from the Pipeline Session.

Note: For more information on Input Processors, consult Chapter 9, “Setting Up Portal Navigation.”

Web Service Portlets

Web Services are reusable software components that enable applications to interact efficiently and in a loosely-coupled manner; they are used internally for fast and low-cost application integration, or made available to customers, suppliers or partners over the Internet. Enabling portlets to consume Web Services, either internally or over the Internet, introduces a range of new benefits - and introduces some new developer issues as well.

Note: For the sake of simplicity, the examples shown in this section show portlets connecting to Web Services that are locally hosted.

This section contains information on the following topics:

- Using the Portlet Wizard to Create Web Services Portlets
- Creating a Simple Form-Driven Web Service Portlet
- Creating a Call-Generation Web Service Portlet
- Creating a Web Services Interface Portlet
- Deploying the Web Services Portlets
- Error Handling within Web Services Portlets
- Calling Web Services Asynchronously

Using the Portlet Wizard to Create Web Services Portlets

The Portlet Wizard can create portlets that consume Web Services in three different ways:

- **Simple Web Service Form**: This is the simplest mode of interacting with a Web Service, whereby a simple parameter is sent to the Web Service class using the httpSession, and a primitive value is printed into the resulting HTML.
- **Call Generation:** Slightly more complex is the Call Generation type portlet, in which classes hosted by the Web Service are listed in an object called the WSDL (Web Services Definition Language). The portlet makes remote calls to these classes and consumes the returned objects. In the Portlet Wizard, the Call Generation option creates a stubbed out portlet that calls into the designated Web service. You are required to set the __REPLACE_ME__ variables with output from sources such as the user profile, the request, and the session.
- **Web Service(s) Interface:** This option allows you to select multiple Web services and extract interface documentation into the portlet itself. Use this option when parameters or return values require complex data types.

  This option can be used to create a portlet that calls a Web Service asynchronously through client polling. The section "Calling Web Services Asynchronously" includes instructions on creating this type of portlet.

This section shows how to use the Portlet Wizard to create a portlet of each type.

Preparation

Start the WebLogic Workshop Examples Server by navigating to Programs →BEA WebLogic Platform 7.0 →WebLogic Workshop Example →Start Examples Server. Begin with the sample portal described in the section The scriptDemo Portlet.

Note: This example shows the scriptDemo, flowLet1 and flowLet2 portlets removed, but you complete these Web Service portlet examples without removing the old portlets from the portal.

Avoiding Namespace Collisions

Both the previous Call Generation and the Interface examples create instances of identical classes which run within the same portal namespace. To prevent namespace collision, the local instances of each class must be given unique names. In the second and third portlets, the implementation classes are re-named, as shown before the code samples.

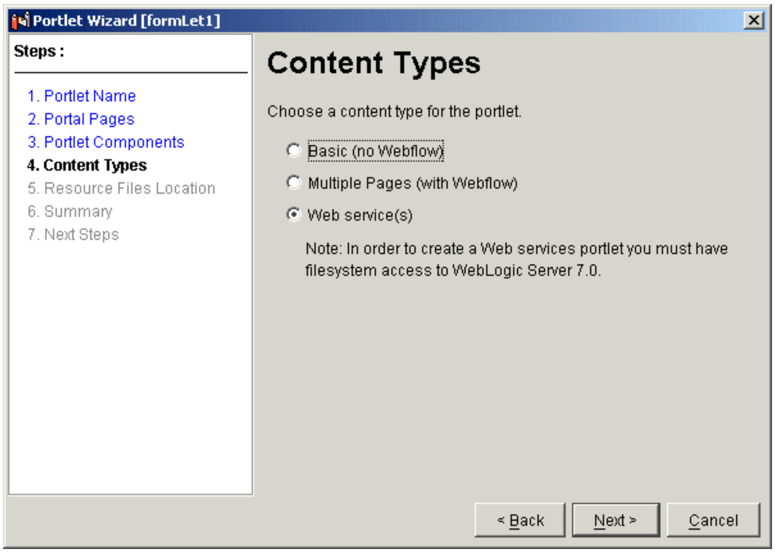
Creating a Simple Form-Driven Web Service Portlet

The first portlet sends simple form input to the `AccountEJBClient` Web Service, creating a simple bank account object hosted at the WebLogic Workshop examples server.

Create the Portlet Called formLet Using the Portlet Wizard

1. Follow the steps outlined in Figure 11-32 through Figure 11-36.
2. When the Content Types screen appears, select **Web Service** and click **Next**, as shown in Figure 11-80.

Figure 11-80 Select Generated Code type



Content Types

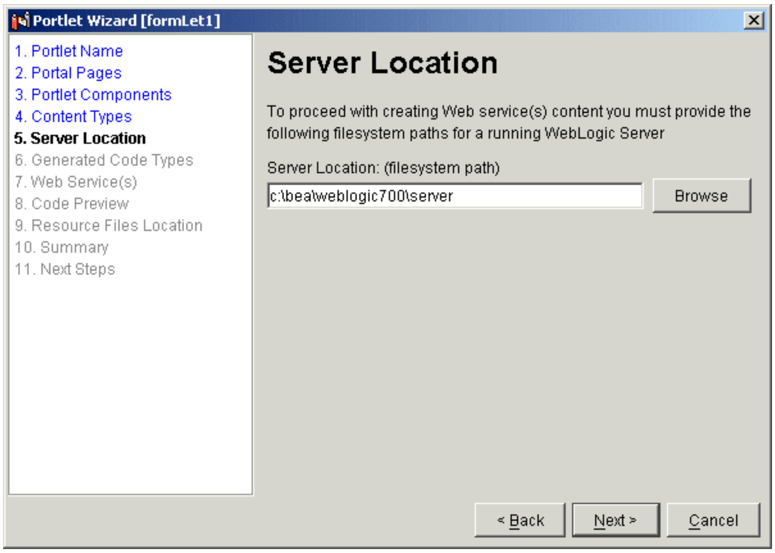
Choose a content type for the portlet.

( ) Basic (no Webflow)
( ) Multiple Pages (with Webflow)
(●) Web service(s)

Note: In order to create a Web services portlet you must have filesystem access to WebLogic Server 7.0.

3. When the Server Location screen appears, navigate to an instance of WebLogic Server, as shown in Figure 11-81, and click **Next**.

Figure 11-81 Select Server Location



Server Location

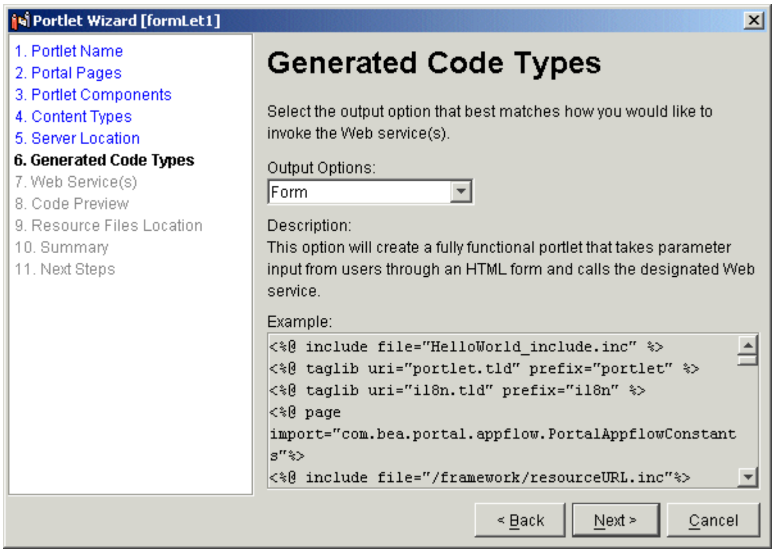
To proceed with creating Web service(s) content you must provide the following filesystem paths for a running WebLogic Server

Server Location: (filesystem path)

c:\bea\weblogic700\server

4. When the Generated Code Types screen appears, select **Form**, as shown in Figure 11-82, and click **Next**.

Figure 11-82 Select Generated Code Types



Generated Code Types

Select the output option that best matches how you would like to invoke the Web service(s).

Output Options: Form

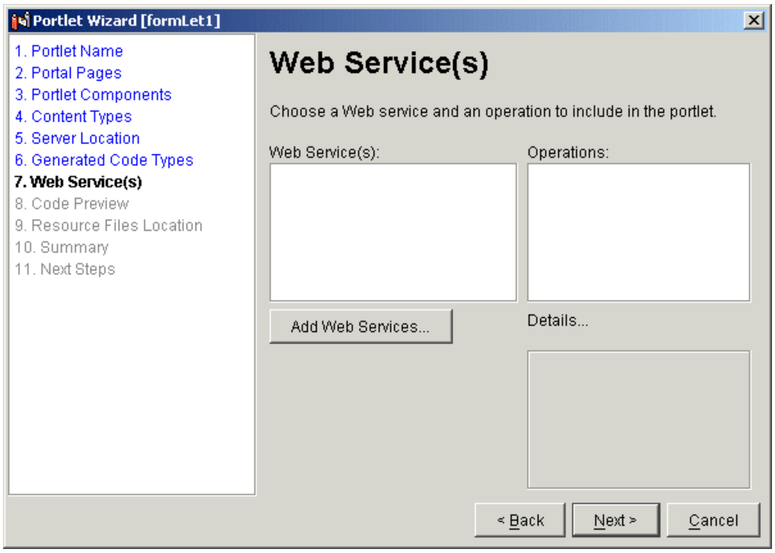
Description:
This option will create a fully functional portlet that takes parameter input from users through an HTML form and calls the designated Web service.

Example:

```
<%@ include file="HelloWorld_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page
import="com.bea.portal.appflow.PortalAppflowConstant
s"%>
<%@ include file="/framework/resourceURL.inc"%>
```

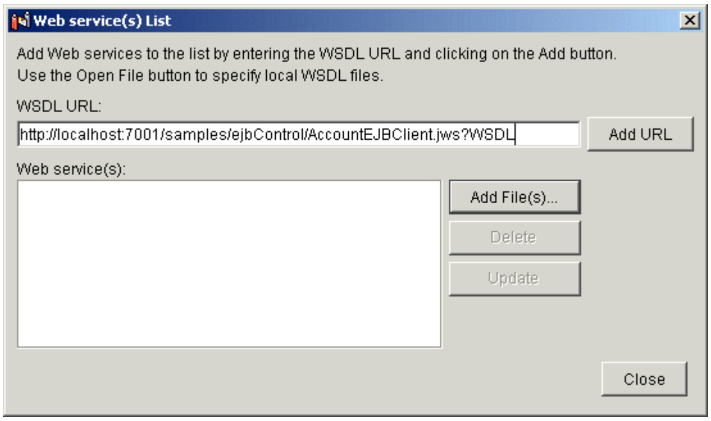
5. When the Select a Web Service screen appears, click **Add Web Services**, as shown in Figure 11-83.

Figure 11-83 Click Add Web Services



6. Type in the URL of the WSDL, as shown in Figure 11-84, then click **ADD URL**.

Figure 11-84 Enter the URL of the WSDL



Add Web services to the list by entering the WSDL URL and clicking on the Add button.
Use the Open File button to specify local WSDL files.

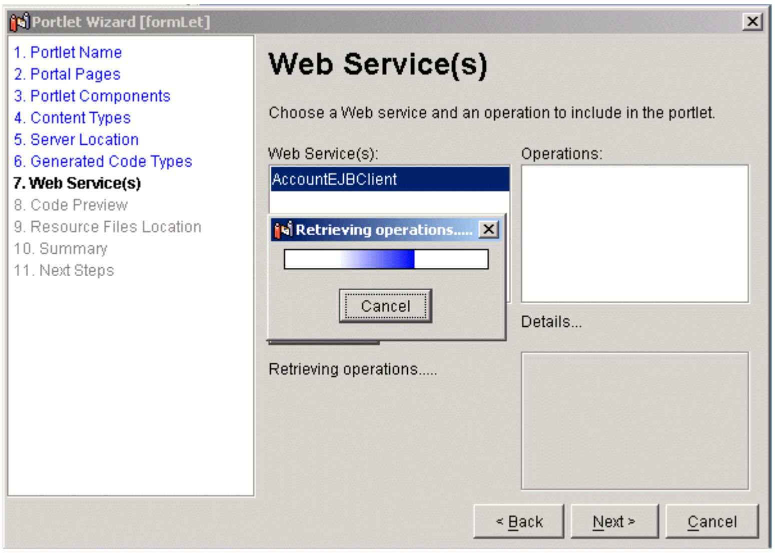
WSDL URL: http://localhost:7001/samples/ejbControl/AccountEJBClient.jws?WSDL

Web service(s):

7. When the `AccountEJBClient` Web Service appears in the list, click **Close**.

8. When the Web Services screen re-appears, click the newly added Web Service. A small window entitled “Retrieving Operations” appears in front of the Web Services screen, as shown in Figure 11-85.

Figure 11-85 Retrieving Operations



Choose a Web service and an operation to include in the portlet.

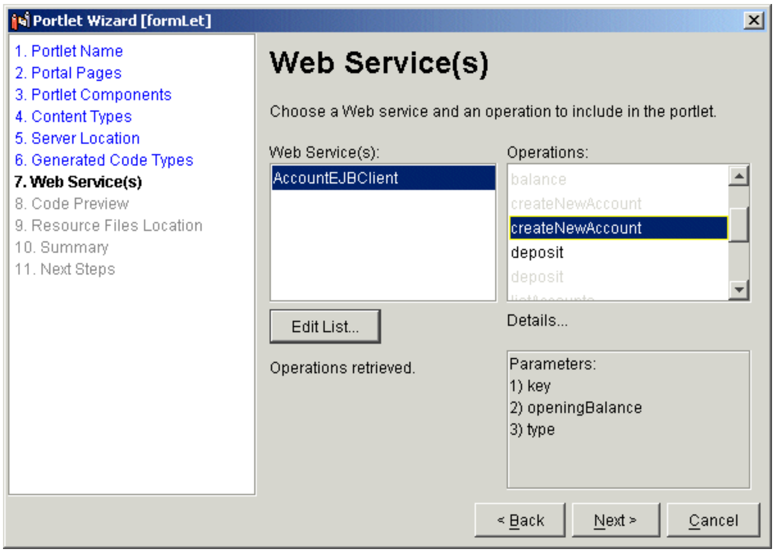
Web Service(s): AccountEJBClient

Operations:

Retrieving operations.....

9. When the Retrieving Operations screen disappears, a list of operations should be available in the listbox to the right. Select the `CreateNewAccount` operation from this list, as shown in Figure 11-86 and click **Next**.

Figure 11-86 Select the createNewAccount operation


Portlet Wizard [formLet]

1. Portlet Name
2. Portal Pages
3. Portlet Components
4. Content Types
5. Server Location
6. Generated Code Types
7. **Web Service(s)**
8. Code Preview
9. Resource Files Location
10. Summary
11. Next Steps

Web Service(s)

Choose a Web service and an operation to include in the portlet.

Web Service(s):
AccountEJBClient

Operations:
balance
createNewAccount
createNewAccount
deposit
deposit

Edit List...

Details...

Operations retrieved.

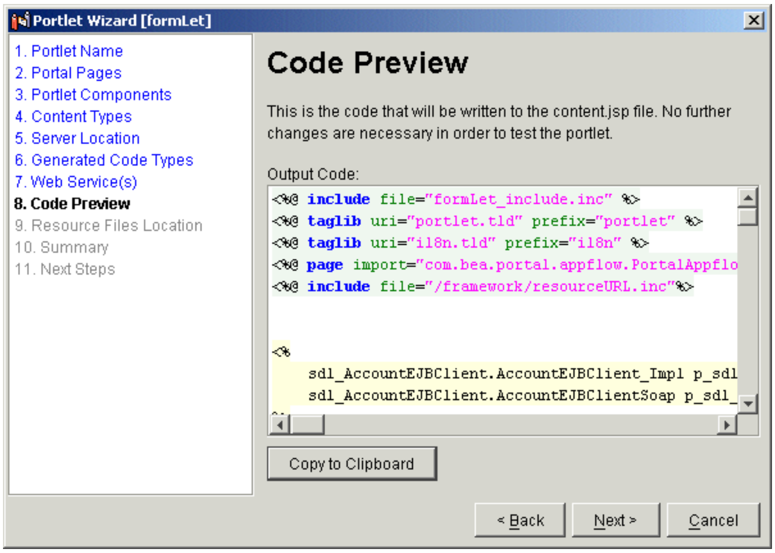
Parameters:
1) key
2) openingBalance
3) type

< Back | Next > | Cancel

10. When the Code Preview screen appears, scroll down in the generated code to begin to familiarize yourself with taglib includes and portal refresh events. You can also click **Copy to Clipboard** and paste the code into an editor for viewing.

11. Click **Next**, as shown in Figure 11-87.

Figure 11-87 Code Preview



Code Preview

This is the code that will be written to the content.jsp file. No further changes are necessary in order to test the portlet.

Output Code:

```
<%@ include file="formLet_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page import="com.bea.portal.appflow.PortalAppflo
<%@ include file="/framework/resourceURL.inc"%>


<%
    sdl_AccountEJBClient.AccountEJBClient_Impl p_sdl
    sdl_AccountEJBClient.AccountEJBClientSoap p_sdl_
```

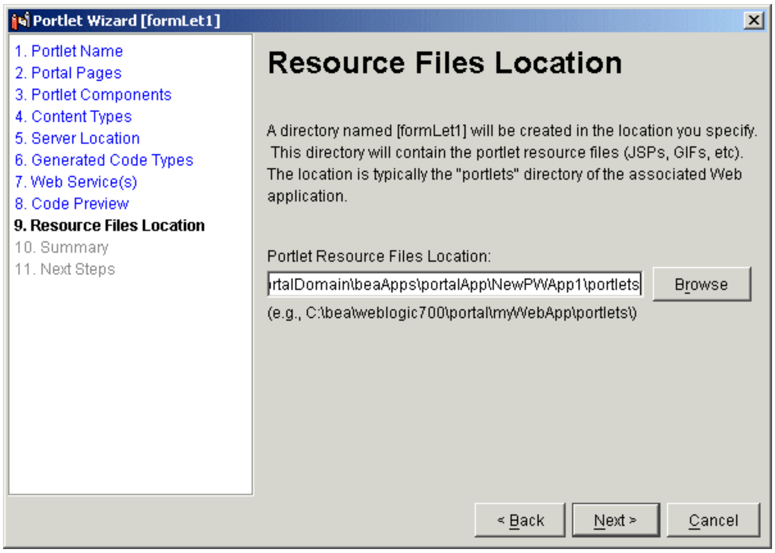
12. When the Resource Files location screen appears, as shown in Figure 11-88, confirm the portlet JSPs will be placed in the correct directory, then click **Next**.

Note: The directory specified in this screen will act as the parent directory for the newly created portlet files. This directory should exist under the WLP web application directory. A sub-directory with the same name as the new portlet will be created where users can find the files generated by this wizard.

In addition, a WEB-INF/lib directory will be created in this location. This directory will contain a .jar file that the portlet references at runtime.

If the directory specified in this screen was not under a WLP web application, users will need to copy the created portlets sub-directory containing the generated files and the .jar file to the portlets directory and the WEB-INF/lib directory, respectively, of the intended Portal Web application.

Figure 11-88 Select/Confirm Resource Files location



Resource Files Location

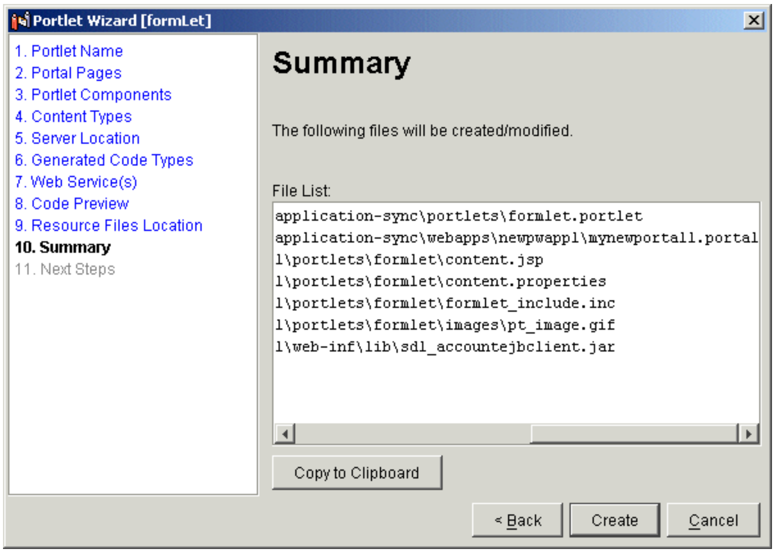
A directory named [formLet1] will be created in the location you specify. This directory will contain the portlet resource files (JSPs, GIFs, etc). The location is typically the "portlets" directory of the associated Web application.

Portlet Resource Files Location: rtalDomain\beaApps\portalApp\NewPWApp1\portlets

(e.g., C:\bea\weblogic700\portal\myWebApp\portlets\)

13. When the Summary page appears, as shown in Figure 11-89, verify the files to be created and click **Create**.

Figure 11-89 Confirm Summary of Files



Summary

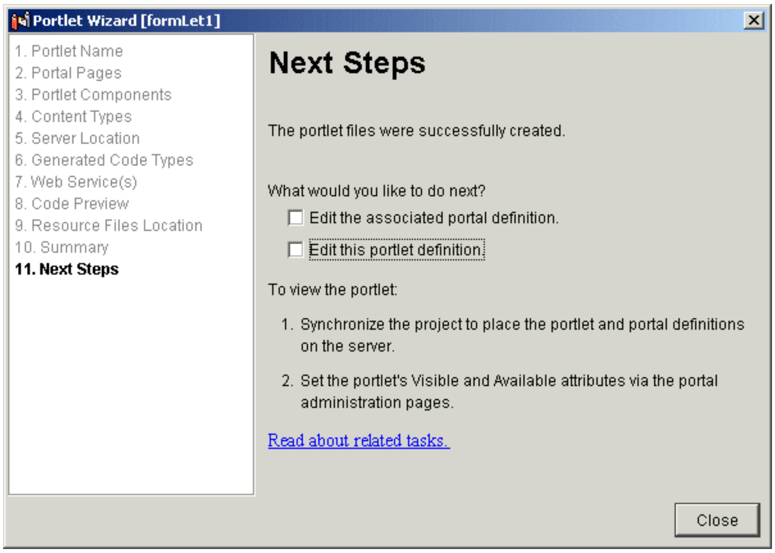
The following files will be created/modified.

File List:

```
application-sync\portlets\formlet.portlet
application-sync\webapps\newpwappl\mynewportall.portal
l\portlets\formlet\content.jsp
l\portlets\formlet\content.properties
l\portlets\formlet\formlet_include.inc
l\portlets\formlet\images\pt_image.gif
l\web-inf\lib\sdl_accountejbclient.jar
```

14. When the Next Steps screen appears, as shown in Figure 11-90, click **Close**.

Figure 11-90 Choose Next Steps



Next Steps

The portlet files were successfully created.

What would you like to do next?

- ☐ Edit the associated portal definition.
- ☐ Edit this portlet definition.

To view the portlet:

1. Synchronize the project to place the portlet and portal definitions on the server.
2. Set the portlet's Visible and Available attributes via the portal administration pages.

Read about related tasks.

Note: For this example, we'll create all three portlets first, then deploy them at once at the end.

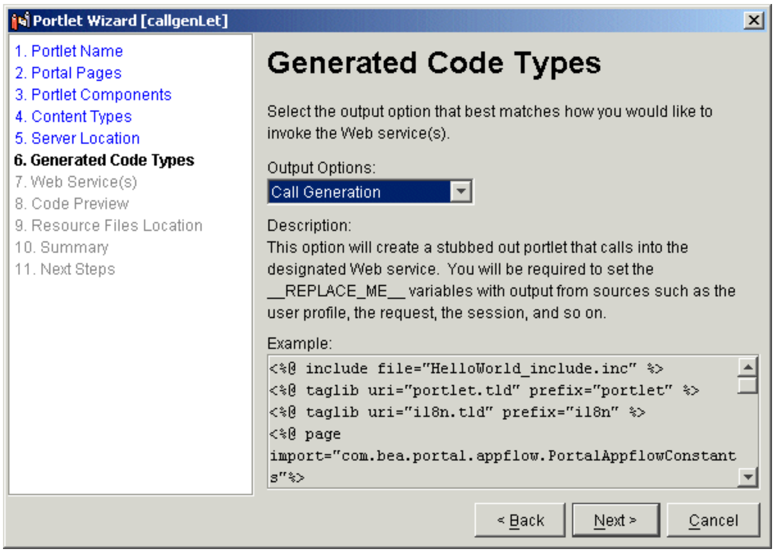
Creating a Call-Generation Web Service Portlet

The Call Generation portlet enables the user to list big accounts by remotely invoking classes hosted within the Web Service hosted on the WebLogic Workshop examples server.

Create the Portlet Called callgenLet Using the Portlet Wizard

1. Follow the steps outlined in Creating a Simple Form-Driven Web Service Portlet up to Step 4.
2. When the Generated Code Types screen appears, select **Call Generation**, as shown in Figure 11-91, and click **Next**.

Figure 11-91 Select Generated Code type



Generated Code Types

Select the output option that best matches how you would like to invoke the Web service(s).

Output Options: Call Generation

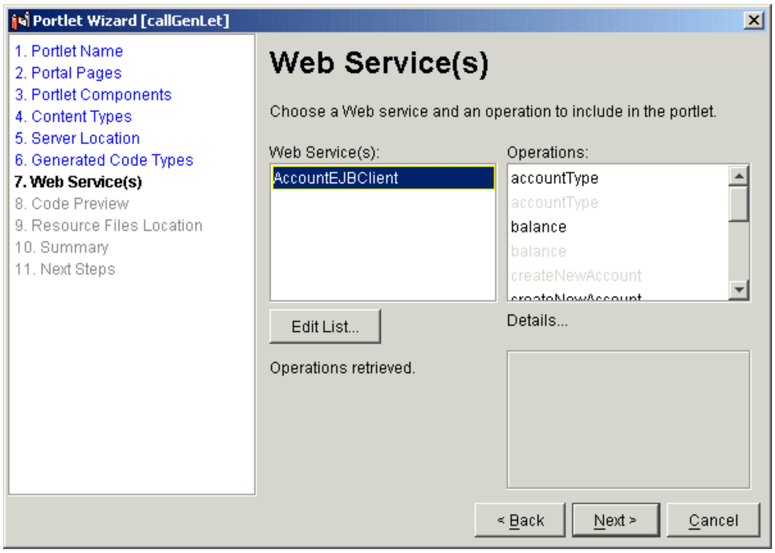
Description:
This option will create a stubbed out portlet that calls into the designated Web service. You will be required to set the __REPLACE_ME__ variables with output from sources such as the user profile, the request, the session, and so on.

Example:

```
<%@ include file="HelloWorld_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page
import="com.bea.portal.appflow.PortalAppflowConstant
s"%>
```

3. When the Web Service(s) List appears, click Edit List, as shown in Figure 11-92.

Figure 11-92 Web Service(s) screen



Web Service(s)

Choose a Web service and an operation to include in the portlet.

Web Service(s): AccountEJBClient

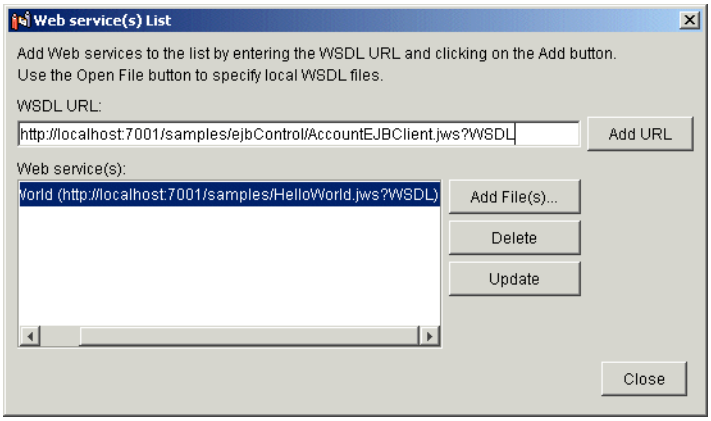
Operations: accountType, accountType, balance, balance, createNewAccount, createNewAccount

Operations retrieved.

4. When the Web Service(s) List appears, as shown in Figure 11-93, enter the following WSDL and click **Add URL**:

```
http://localhost:7001/samples/ejbControl/
AccountEJBClient.jws?WSDL
```

Figure 11-93 Adding another Web Service WSDL



Add Web services to the list by entering the WSDL URL and clicking on the Add button.
Use the Open File button to specify local WSDL files.

WSDL URL: http://localhost:7001/samples/ejbControl/AccountEJBClient.jws?WSDL

Web service(s): Vorld (http://localhost:7001/samples/HelloWorld.jws?WSDL)

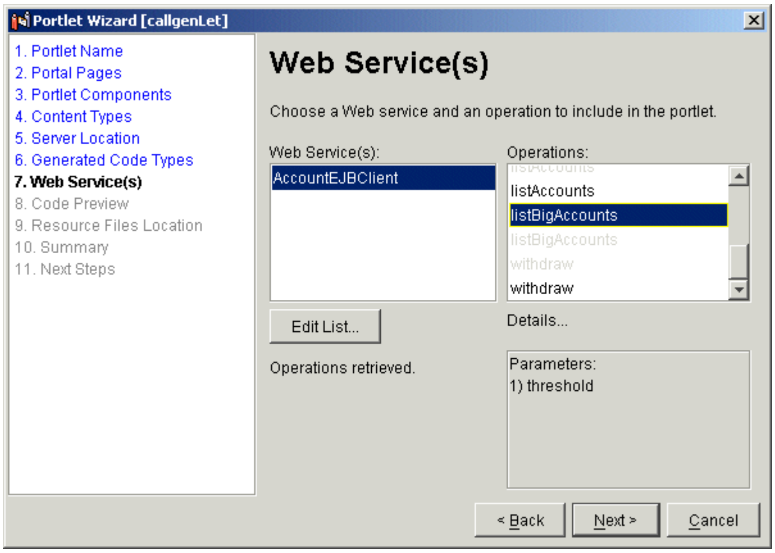
5. When the URL you entered has been added to the Web Services list, click **Close**. The Web Service(s) screen reappears, with the newly-added Web Service listed to the left, and Operation listed to the right.

6. Click the `AccountEJBClient` Web Service and wait for its operations to be retrieved and listed to the right.

7. In the list of operations for `AccountEJBClient`, click **listBigAccounts**, as shown in Figure 11-94, and click **Next**.

Note: Notice the parameter list below the operation.

Figure 11-94 Selecting the listBigAccounts operation



Web Service(s)

Choose a Web service and an operation to include in the portlet.

Web Service(s): AccountEJBClient

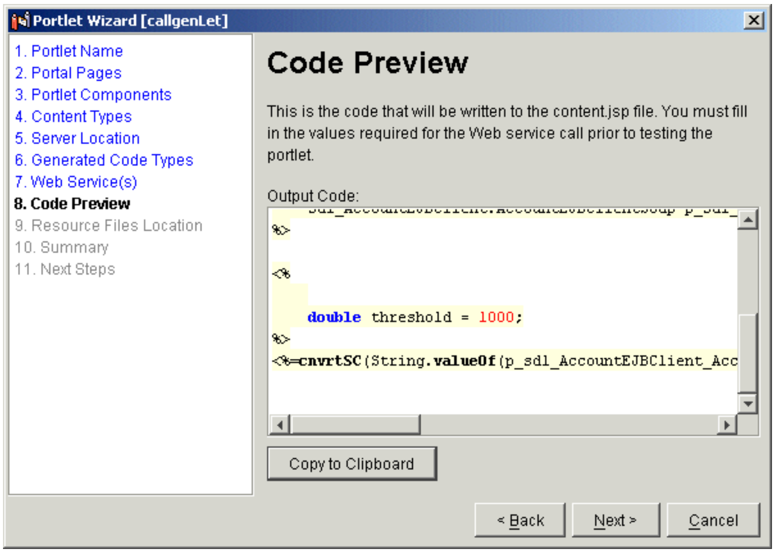
Operations: listAccounts, listAccounts, listBigAccounts, listBigAccounts, withdraw, withdraw

Operations retrieved.

Parameters:
1) threshold

8. When the Code Preview page appears, as shown in Figure 11-95, notice the string **_REPLACE ME_** occurring in place of the `threshold` parameter, visible in Listing 11-6. After replacing these parameter values with those in bold in Listing 11-6, click **Next**.

Figure 11-95 Code Preview



Code Preview

This is the code that will be written to the content.jsp file. You must fill in the values required for the Web service call prior to testing the portlet.

Output Code:

```
  sdl_AccountEJBClient.AccountEJBClientSoap p_sdl_
%>


<%

    double threshold = 1000;
%>
<%=cnvrtSC(String.valueOf(p_sdl_AccountEJBClient_Acc
```

Note: Within Listing 11-6, the following entries, in boldface within the code, have been shortened:

```
callImpl was originally p_sdl_AccountEJBClient_AccountEJBClient_Impl
callSoap was originally p_sdl_AccountEJBClient_AccountEJBClientSoap
```

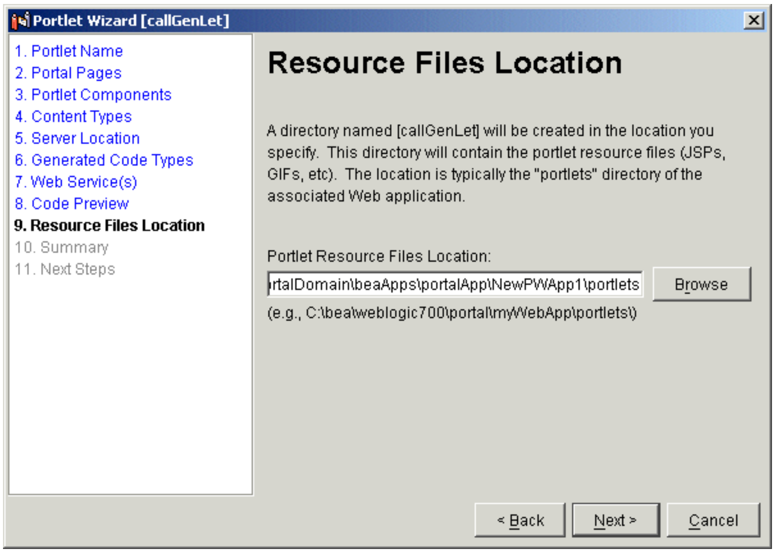
Listing 11-6 Call Generation Portlet code

```
<%@ include file="callgenlet1_include.inc" %>

<%@ taglib uri="portlet.tld" prefix="portlet" %>

<%@ taglib uri="i18n.tld" prefix="i18n" %>

<%@ page import="com.bea.portal.appflow.PortalAppflowConstants"%>

<%@ include file="/framework/resourceURL.inc"%>

<%

 sdl_AccountEJBClient.AccountEJBClient_Impl callImpl  = new
sdl_AccountEJBClient.AccountEJBClient_Impl();
```

```
 sdl_AccountEJBClient.AccountEJBClientSoap callSoap =
callImpl.getAccountEJBClientSoap();

%>

<%

    double threshold = 1000;

%>

<%=cnvrtSC(String.valueOf(callSoap.listBigAccounts(threshold))%>
```

9. When the Resource Files Location screen appears, as shown in Figure 11-96, ensure the files are going to be installed in the correct directory and click **Next**.

Figure 11-96 Select/Confirm Resource Files location



Resource Files Location

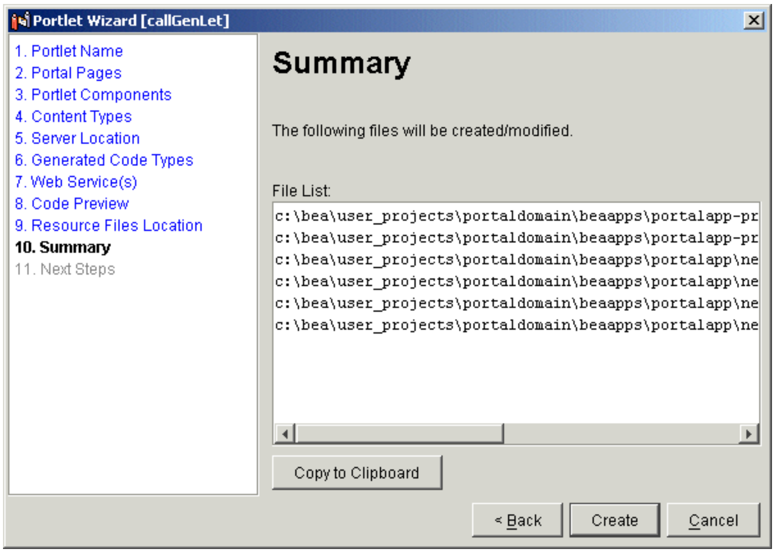
A directory named [callGenLet] will be created in the location you specify. This directory will contain the portlet resource files (JSPs, GIFs, etc). The location is typically the "portlets" directory of the associated Web application.

Portlet Resource Files Location: ırtalDomain\beaApps\portalApp\NewPWApp1\portlets

(e.g., C:\bea\weblogic700\portal\myWebApp\portlets\)

10. When the Summary files screen appears, as shown in Figure 11-97, verify the correct files are to be created and click **Create**.

Figure 11-97 Confirm Summary of Files



Portlet Wizard [callGenLet]

1. Portlet Name
2. Portal Pages
3. Portlet Components
4. Content Types
5. Server Location
6. Generated Code Types
7. Web Service(s)
8. Code Preview
9. Resource Files Location
10. Summary
11. Next Steps

Summary

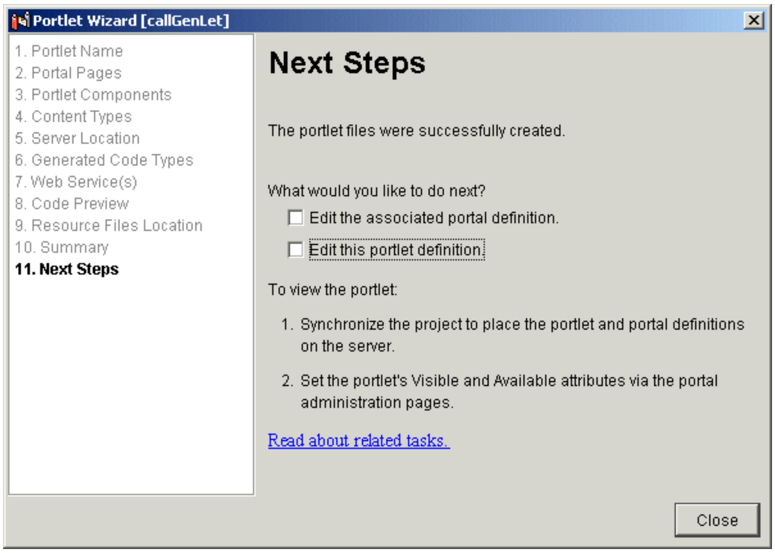
The following files will be created/modified.

File List:

```
c:\bea\user_projects\portaldomain\beaapps\portalapp-pr
c:\bea\user_projects\portaldomain\beaapps\portalapp-pr
c:\bea\user_projects\portaldomain\beaapps\portalapp\ne
c:\bea\user_projects\portaldomain\beaapps\portalapp\ne
c:\bea\user_projects\portaldomain\beaapps\portalapp\ne
c:\bea\user_projects\portaldomain\beaapps\portalapp\ne
```

11. When the Next Steps screen appears, as shown in Figure 11-98, click **Close**.

Figure 11-98 Choose Next Steps



Portlet Wizard [callGenLet]

1. Portlet Name
2. Portal Pages
3. Portlet Components
4. Content Types
5. Server Location
6. Generated Code Types
7. Web Service(s)
8. Code Preview
9. Resource Files Location
10. Summary
11. Next Steps

Next Steps

The portlet files were successfully created.

What would you like to do next?

☐ Edit the associated portal definition.

☐ Edit this portlet definition.

To view the portlet:

1. Synchronize the project to place the portlet and portal definitions on the server.
2. Set the portlet's Visible and Available attributes via the portal administration pages.

Read about related tasks.

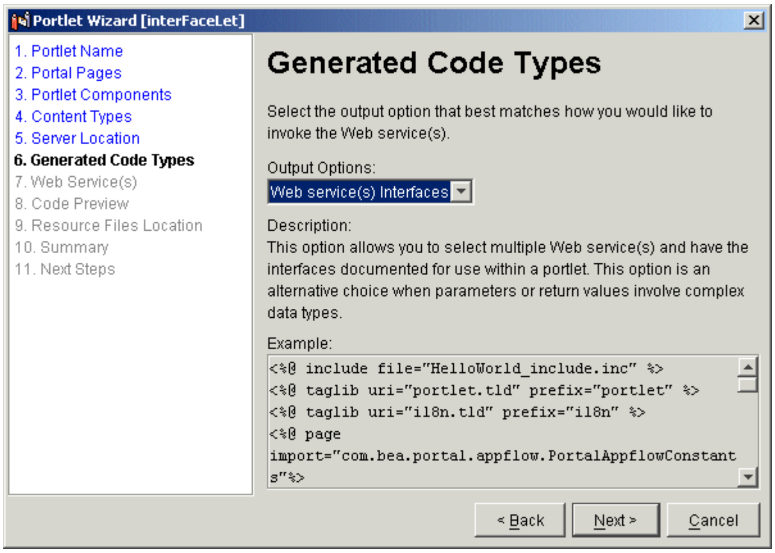
Creating a Web Services Interface Portlet

This portlet searches for accounts with a balance over a threshold set within the portlet, and lists them in the portlet. What is significant is the way in which this portlet communicates with the Web Service: the portlet uses classes implemented according to the self-describing WSDL called `listBigAccounts`.

Create the Portlet Called interFaceLet Using the Portlet Wizard

1. Follow the steps outlined in Creating a Simple Form-Driven Web Service Portlet up to Step 4.
2. When the Generated Code Types screen appears, select **Web service(s) Interface**, as shown in Figure 11-99, and click **Next**.

Figure 11-99 Select Generated Code Type



Generated Code Types

Select the output option that best matches how you would like to invoke the Web service(s).

Output Options: Web service(s) Interfaces

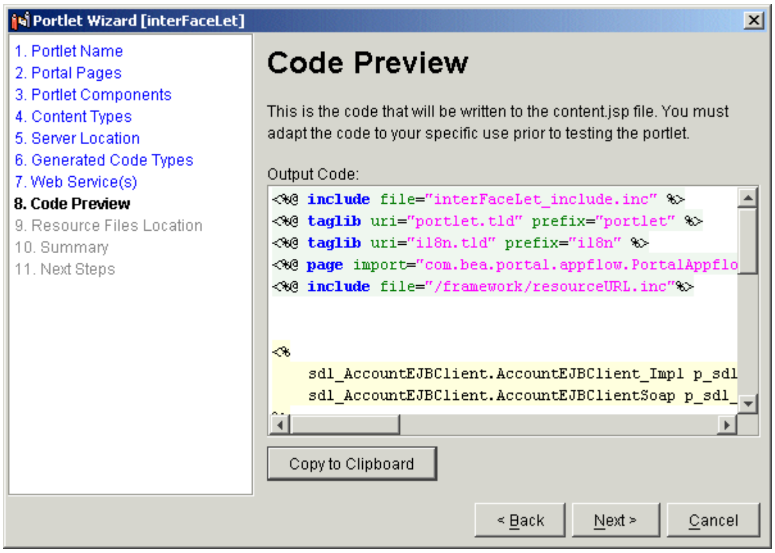
Description:
This option allows you to select multiple Web service(s) and have the interfaces documented for use within a portlet. This option is an alternative choice when parameters or return values involve complex data types.

Example:
```
<%@ include file="HelloWorld_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page
import="com.bea.portal.appflow.PortalAppflowConstant
s"%>
```

Note: In the Web Service(s) screen for Interface Generation portlets, notice that no operations are available. This is because you are being provided the interface description and must implement the required methods yourself.

3. Edit the portlet code within the Code Preview screen, as shown in Figure 11-100, and click **Next**.

Figure 11-100 Code Preview screen for interFaceLet



Code Preview

This is the code that will be written to the content.jsp file. You must adapt the code to your specific use prior to testing the portlet.

Output Code:

```
<%@ include file="interFaceLet_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page import="com.bea.portal.appflow.PortalAppflo
<%@ include file="/framework/resourceURL.inc"%>


<%
    sdl_AccountEJBClient.AccountEJBClient_Impl p_sdl
    sdl_AccountEJBClient.AccountEJBClientSoap p_sdl_
```

4. Implement interfaces required for listing big Accounts. Use the generated code as your starting point, shown in Listing 11-7.

   Note: Within Listing 11-7, the following entries, in boldface within the code, have been shortened:

   `intImpl` `was originally`
   `p_sdl_AccountEJBClient_AccountEJBClient_Impl`

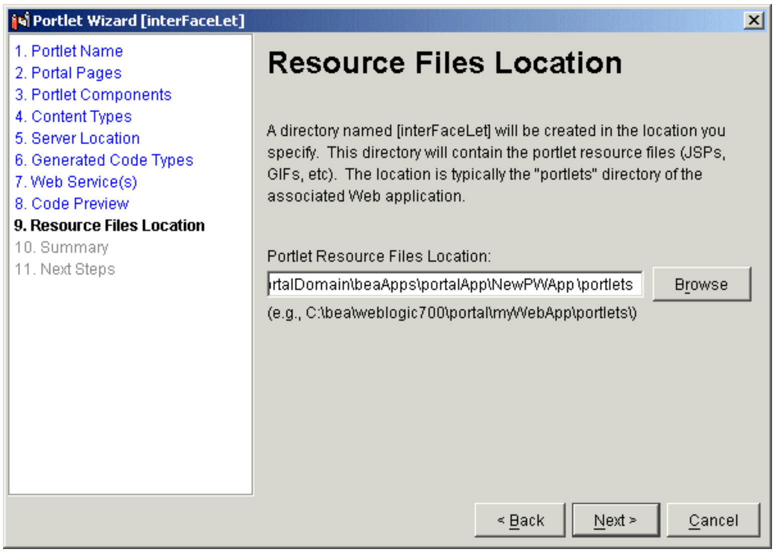
   `intSoap` `was originally`
   `sdl_AccountEJBClient.AccountEJBClientSoap`

Listing 11-7 Code for interFaceLet

```
<%@ include file="interFaceLet_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page import="com.bea.portal.appflow.PortalAppflowConstants"%>
<%@ include file="/framework/resourceURL.inc"%>
<%

sdl_AccountEJBClient.AccountEJBClient_Impl intImpl = new
    sdl_AccountEJBClient.AccountEJBClient_Impl();
sdl_AccountEJBClient.AccountEJBClientSoap intSoap =
    intImpl.getAccountEJBClientSoap();

...

*/%>
```

5. When the Resource Files Location screen appears, as shown in Figure 11-101, ensure the files are going to be installed in the correct directory and click **Next**.

Figure 11-101 Select/Confirm Resource Files location



Resource Files Location

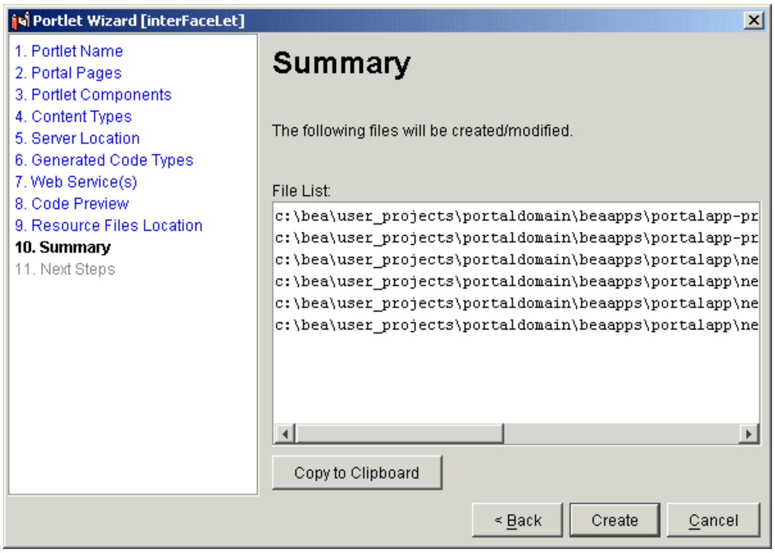
A directory named [interFaceLet] will be created in the location you specify. This directory will contain the portlet resource files (JSPs, GIFs, etc). The location is typically the "portlets" directory of the associated Web application.

Portlet Resource Files Location: rtalDomain\beaApps\portalApp\NewPWApp\portlets

(e.g., C:\bea\weblogic700\portal\myWebApp\portlets\)

6. When the Summary files screen appears, as shown in Figure 11-102, verify the correct files are to be created and click **Create**.

Figure 11-102 Confirm Summary of Files



Summary

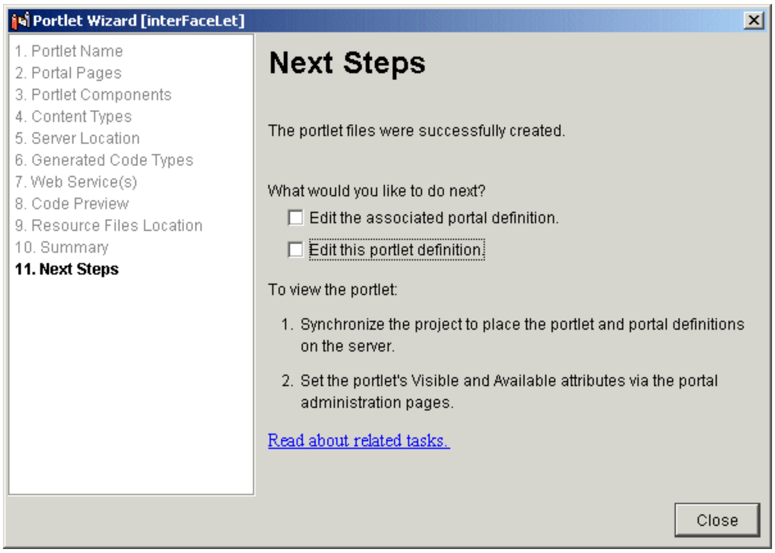
The following files will be created/modified.

File List:

```
c:\bea\user_projects\portaldomain\beaapps\portalapp-pr
c:\bea\user_projects\portaldomain\beaapps\portalapp-pr
c:\bea\user_projects\portaldomain\beaapps\portalapp\ne
c:\bea\user_projects\portaldomain\beaapps\portalapp\ne
c:\bea\user_projects\portaldomain\beaapps\portalapp\ne
c:\bea\user_projects\portaldomain\beaapps\portalapp\ne
```

7. When the Next Steps screen appears, as shown in Figure 11-103, click **Close**.

Figure 11-103 Choose Next Steps



Next Steps

The portlet files were successfully created.

What would you like to do next?

- ☐ Edit the associated portal definition.
- ☐ Edit this portlet definition.

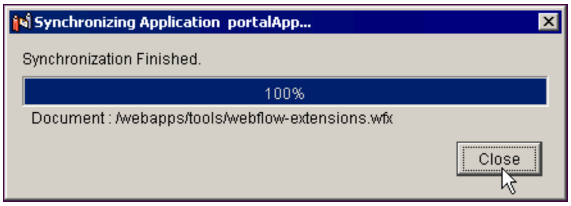
To view the portlet:

1. Synchronize the project to place the portlet and portal definitions on the server.
2. Set the portlet's Visible and Available attributes via the portal administration pages.

8. From the E-Business Control Center toolbar, click the **Synchronize** button.

9. When the Synchronizing Application window shows that synchronization is complete, click **Close**, as shown in Figure 11-104.

Figure 11-104 Synchronization Complete



Synchronization Finished.
Document : /webapps/tools/webflow-extensions.wfx

Deploying the Web Services Portlets

Now that they are all created and installed, what remains is to make them available to the WebLogic Server instance. This is done by simply re-deploying the Portal Web Application in which these portlets are to run, according to the following procedures:

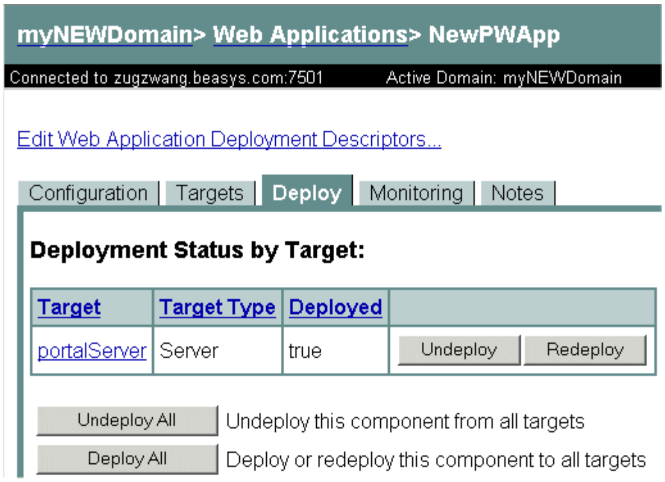
1. In your Web browser, navigate to the WebLogic Server console at the following URL:

   ```
   http://<host>:<port>/console
   ```

2. Login as weblogic/weblogic.

3. Select Deployments →Applications →NewPWApp, and click the **Deploy** tab in the right pane, as shown in Figure 11-105.

Figure 11-105 Left tab of console: Deployments →Applications →NewPWApp



myNEWDomain> Web Applications> NewPWApp

Connected to zugzwang.beasys.com:7501 Active Domain: myNEWDomain

Edit Web Application Deployment Descriptors...

Configuration | Targets | Deploy | Monitoring | Notes

Deployment Status by Target:

Target	Target Type	Deployed	
portalServer	Server	true	Undeploy Redeploy

Undeploy All — Undeploy this component from all targets

Deploy All — Deploy or redeploy this component to all targets

4. UnDeploy the NewPWApp by clicking the **Undeploy** button to the right of portalServer target. When the UnDeployment Activity status turns to Complete, click **Deploy**.

5. When the status on the Deployment Activity screen turns to Complete, the new portlets have been re-deployed.

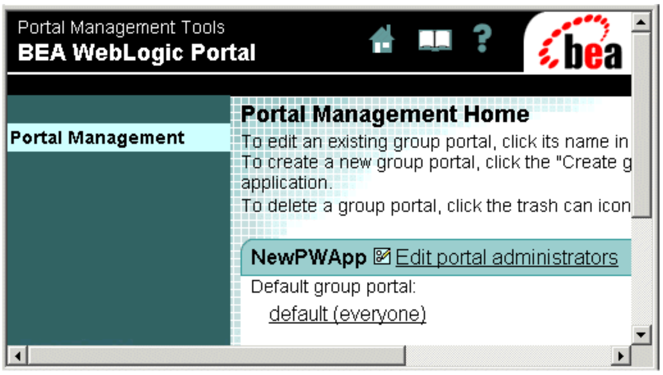
6. Make the new portlets visible and available using the WebLogic Portal Administration Tools: In your Web browser, navigate to the following URL: http://<hostname>:<port>/portalAppTools.

7. Login as administrator/password, and click **Portal Management**, as shown in Figure 11-106.

Figure 11-106 Go to Portal Management

8. From the Portal Management Home page, Click the **Default Portal**, as shown in Figure 11-107**.**

Figure 11-107 Select Default Group Portal



Portal Management Home

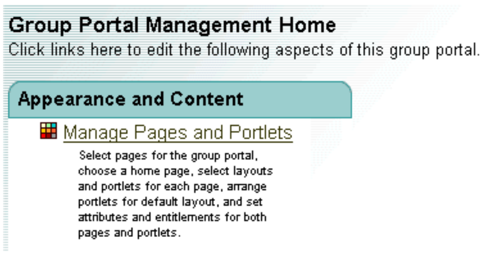
To edit an existing group portal, click its name in
To create a new group portal, click the "Create g
application.
To delete a group portal, click the trash can icon

NewPWApp Edit portal administrators

Default group portal:
default (everyone)

9. From the Group Portal Management Home page, click **Manage Pages and Portlets** as shown in Figure 11-108.

Figure 11-108 Manage Pages and Portlets



Group Portal Management Home

Click links here to edit the following aspects of this group portal.

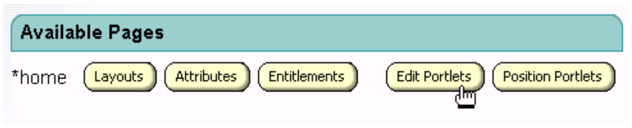
Appearance and Content

Manage Pages and Portlets

Select pages for the group portal, choose a home page, select layouts and portlets for each page, arrange portlets for default layout, and set attributes and entitlements for both pages and portlets.

10. Next to the portal page, click **Edit Portlets**, as shown in Figure 11-109.

Figure 11-109 Click Edit Portlets



Available Pages

*home

11. Set all three portlets attributes to Visible and Available, and click **Save**.

12. **Click Save:** The attributes of formLet, callGenLet and interFaceLet portlets are now correctly set.

Viewing the Web Services Portlets

Now that they are all deployed and placed within the same portal page, observe the functionality they provide by taking the following steps:

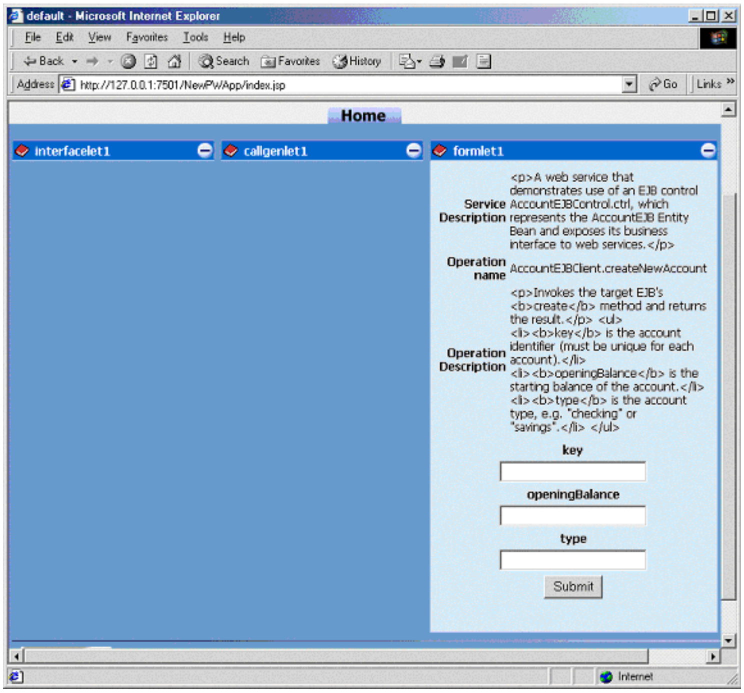
1. Verify the portlets can be accessed by navigating to the following URL:

```
http://<host>:<port>/NewPWApp/
```

The result should resemble that shown in Figure 11-110.

2. Logon as a valid user and two new accounts; one with a balance of 1001 threshold, one 751. Notice what appears in the List big accounts portlet.
3. Change threshold in the content.jsp of the callGenlet portlet to 750.
4. Create a new account with a balance of 749. List big accounts.

Figure 11-110 Web Services portlets before accounts have been entered


default - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://127.0.0.1:7501/NewPWApp/index.jsp
Go Links
Home
interfacelet1
callgenlet1
formlet1
Internet

Service Description	<p>A web service that demonstrates use of an EJB control AccountEJBControl.ctrl, which represents the AccountEJB Entity Bean and exposes its business interface to web services.</p>
Operation name	AccountEJBClient.createNewAccount
Operation Description	<p>Invokes the target EJB's <b>create</b> method and returns the result.</p> <ul> <li><b>key</b> is the account identifier (must be unique for each account).</li> <li><b>openingBalance</b> is the starting balance of the account.</li> <li><b>type</b> is the account type, e.g. "checking" or "savings".</li> </ul>

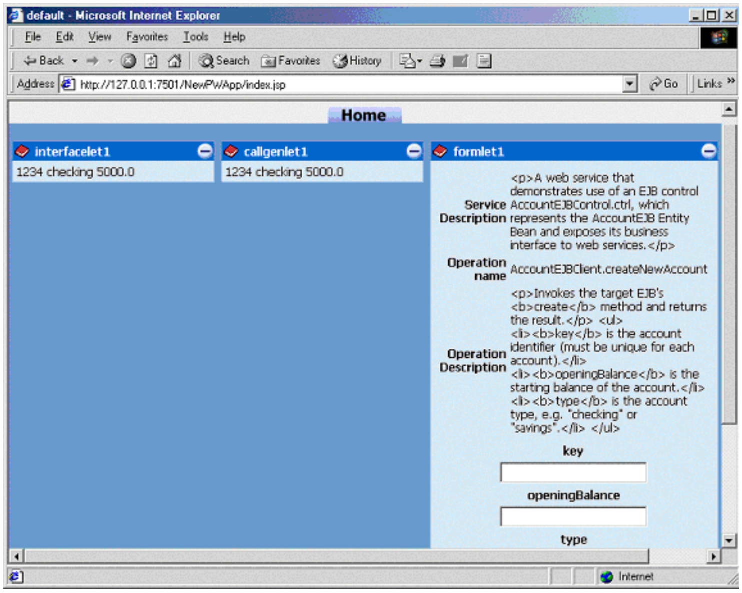
key

openingBalance

type

Submit

Figure 11-111 Web services portlets with some account activity


default - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://127.0.0.1:7501/NewPWApp/index.jsp
Go Links
Home
Internet

interfacelet1

1234 checking 5000.0

callgenlet1

1234 checking 5000.0

formlet1

Service Description <p>A web service that demonstrates use of an EJB control AccountEJBControl.ctrl, which represents the AccountEJB Entity Bean and exposes its business interface to web services.</p>

Operation name AccountEJBClient.createNewAccount

Operation Description <p>Invokes the target EJB's <b>create</b> method and returns the result.</p> <ul> <li><b>key</b> is the account identifier (must be unique for each account).</li> <li><b>openingBalance</b> is the starting balance of the account.</li> <li><b>type</b> is the account type, e.g. "checking" or "savings".</li> </ul>

key

openingBalance

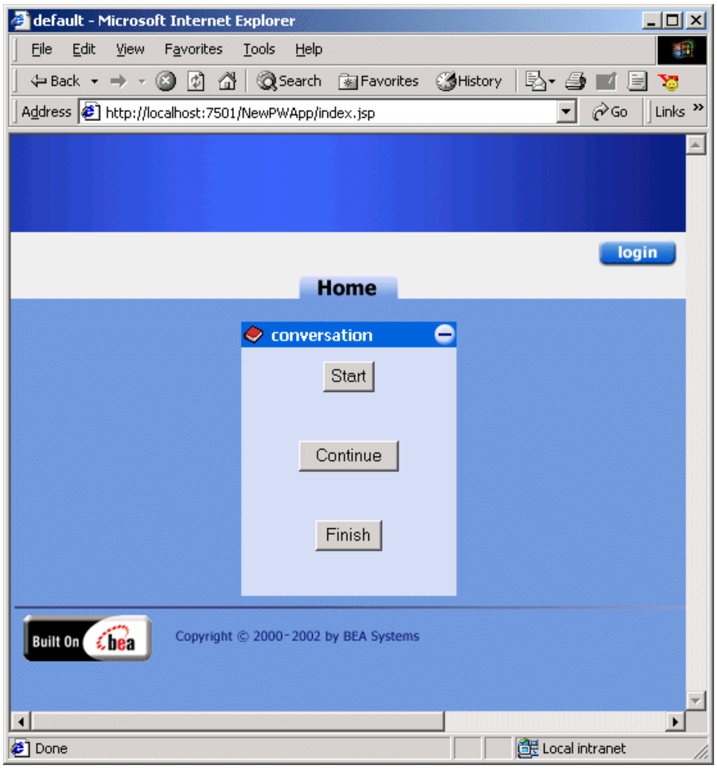
type

Note: For more information on Web Services, consult "Building Web Services on WebLogic Platform" at http://edocs.bea.com/platform/docs70/interm/webserv.html and "Introduction to WebLogic Workshop" at http://edocs.bea.com/workshop/docs70/index.html.

Calling Web Services Asynchronously

BEA WebLogic Portal 7.0 enables portlets to participate in asynchronous communication with Web Services such as conversations. This example shows how to create a simple conversation portlet that interacts with a Web Service hosted on the local server.

Figure 11-112 Conversation Web Services Portlet


default - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:7501/NewPWApp/index.jsp
Go Links
login
Home
conversation
Start
Continue
Finish
Built On bea
Copyright © 2000-2002 by BEA Systems
Done
Local intranet

About the Conversation Portlet

The portlet created in this example contains three button-activated actions:

- The **Start** button sends a session id to the Web Service, which holds this as a token for the conversation.
- The **Continue** button requests status on the token.
- The **Finish** button ends the conversation, causing the Web Service to relinquish the token and stop waiting for more messages.

Preparation

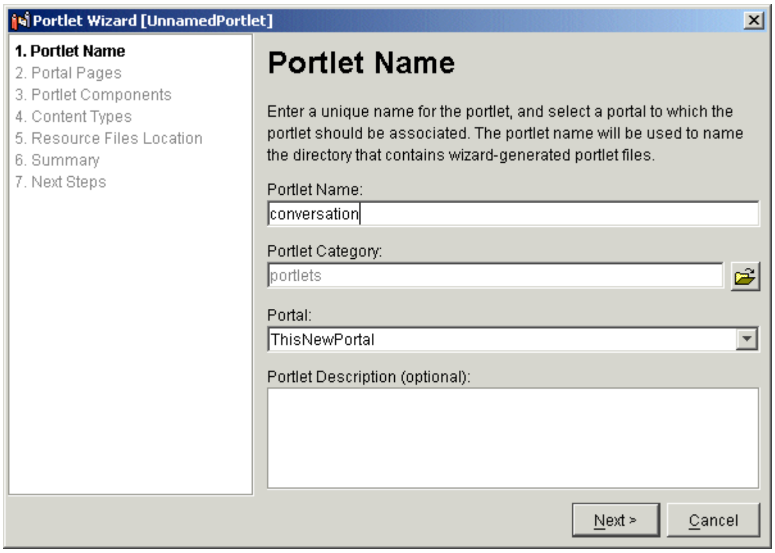
- Start the WebLogic Workshop Examples Server by navigating to Programs → BEA WebLogic Platform 7.0 → WebLogic Workshop Example → Start Examples Server. Begin with the sample portal described in the section The scriptDemo Portlet.
- Start the portal server for your domain. In this example, this is done by navigating to Programs → BEA WebLogic Platform 7.0 → User Projects → MyNewDomain → Start Portal Server.

Creating the Conversation Portlet

To create and deploy a sample Conversation portlet, follow these steps.

1. Use the Portlet Wizard to create a Web Services Interface portlet, as shown in the Creating a Web Services Interface Portlet section. Name this portlet "conversation", as shown in Figure 11-113, and click **Next**.

Figure 11-113 Creating the conversation portlet


Portlet Wizard [UnnamedPortlet]

Portlet Name

Enter a unique name for the portlet, and select a portal to which the portlet should be associated. The portlet name will be used to name the directory that contains wizard-generated portlet files.

Portlet Name: conversation

Portlet Category: portlets

Portal: ThisNewPortal

Portlet Description (optional):

2. Associate the new portlet with the portal page called "home" and click **Next**.

3. When the Select Portlet Components page appears, click **Next** without designating any extra components.
4. When the Content Types screen appears, select Web Service and click **Next**.
5. When the Server Location screen appears, navigate to an instance of WebLogic Server and click **Next**.
6. When the Generated Code Types screen appears, select Web Service(s) Interfaces, and click **Next**.
7. When the Select a Web Service screen appears, click Add Web Services, type in the following URL: `http://localhost:7001/samples/async/Conversation.jws?WSDL`. Click **Add URL**.
8. After the Conversation Web Service appears in the list, click **Close**.
9. Select the Conversation Web Service from the list on the left, and click **Next**.
10. When the Code Preview screen appears, click **Next**. This code will be replaced later in this procedure.
11. When the Resource Files location screen appears, confirm the portlet JSPs will be placed in the correct directory, then click **Next**.
12. When the Summary page appears, verify the files to be created and click **Create**.
13. When the Next Steps screen appears, make sure all the checkboxes are unselected, then click **Close**.
14. In a text editor, enter the code in Listing 11-8 and save it as `content.jsp` in the following directory: `myNewDomain\beaApps\portalApp\NewPWApp\portlets\conversation\`.

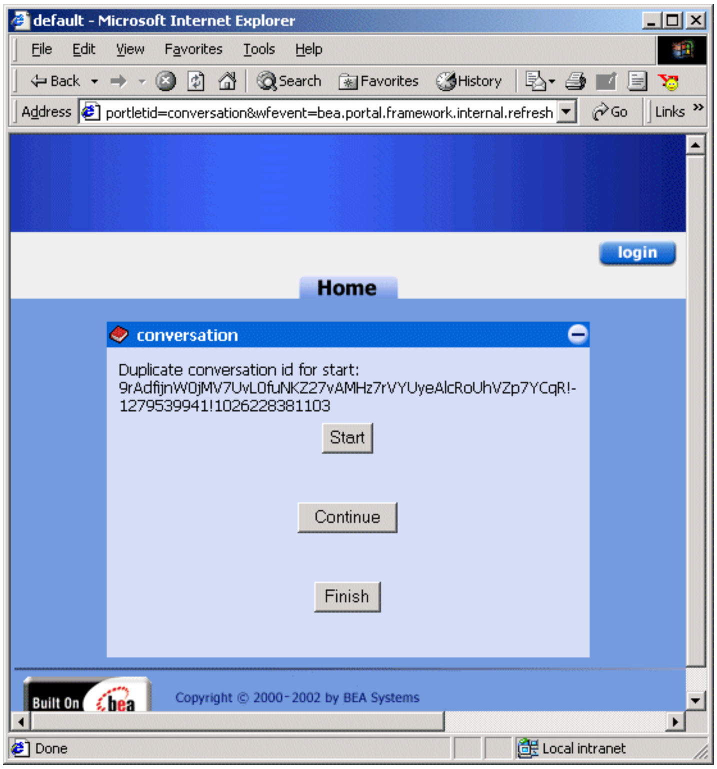
Listing 11-8 content.jsp for Conversation Web Services Portlet

```
<%@ include file="Conversation_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page import="org.openuri.www.StartRequest"%>
<%@ page import="org.openuri.www.GetRequestStatusResponse"%>
<%@ page import="org.openuri.www.x2002.x04.soap.conversation.StartHeader"%>
<%@ page import="org.openuri.www.x2002.x04.soap.conversation.ContinueHeader"%>
<%@ page import="weblogic.xml.schema.binding.internal.builtin.VoidType"%>
<%@ page import="com.bea.portal.appflow.PortalAppflowConstants"%>
<%@ page import="com.bea.portal.appflow.PortalAppflowConstants"%>
<%@ include file="/framework/resourceURL.inc"%>

<%
    DL_wsdl_Conversation.Conversation_Impl conversationImpl = new
DL_wsdl_Conversation.Conversation_Impl();
    DL_wsdl_Conversation.ConversationSoap soap =
conversationImpl.getConversationSoap();
%>

<%
    String target = request.getParameter("target");
    String conversationID = session.getId();
        if ( conversationID == null )
            conversationID = "";
%>

<portlet:form event="<%= PortalAppflowConstants.PORTLET_REFRESH %>">
    <table border="0" align="center">
        <tr>
            <td width="100%" align="center">
            <table border="0" align="left">
            <tr>
<%
                if ( target != null
                     && target.equals("start")
                     &&  true  )
                {
                    try
                    {
                     StartHeader startHeader = new StartHeader(conversationID,
"http://localhost:7001/samples/async/Conversation.jws");
            VoidType startResponse = new VoidType();

            StartRequest begin = new StartRequest(false);

            startResponse = soap.startRequest(begin, startHeader);
%>
            <td><%=cnvrtSC("Conversation started with ID: " +
String.valueOf(conversationID))%></td>
<%
               }
                catch (java.rmi.RemoteException e)
               {
%>
            <td><%=cnvrtSC("Duplicate conversation id for start: " +
String.valueOf(conversationID))%></td>
<%
```

```
            e.printStackTrace();
             }
                }
%>
            </tr>
            </table>
            </td>
        </tr>
        <tr>
            <td width="100%" align="center"><input type="submit" name="start"
value="Start"></td>
        </tr>
    </table>
    <br><br>
    <input type="hidden" name="target" value="start">
</portlet:form>

<portlet:form event="<%= PortalAppflowConstants.PORTLET_REFRESH %>">
    <table border="0" align="center">
        <tr>
            <td width="100%" align="center">
            <table border="0" align="left">
            <tr>
<%
                if ( target != null
                     && target.equals("continue")
                     &&  true  )
                {
                    try
                    {
            ContinueHeader continueHeader = new
ContinueHeader(conversationID);
            GetRequestStatusResponse status = soap.getRequestStatus(null,
continueHeader);
            String result = status.getGetRequestStatusResult();
%>
            <td><%=cnvrtSC("Response: " + String.valueOf(result))%></td>
<%
                }
                catch ( Exception e )
                {
                    e.printStackTrace();
                 }
             }
%>
            </tr>
            </table>
            </td>
        </tr>
        <tr>
            <td width="100%" align="center"><input type="submit" name="continue"
value="Continue"></td>
        </tr>
    </table>
    <br><br>
    <input type="hidden" name="target" value="continue">
</portlet:form>

<portlet:form event="<%= PortalAppflowConstants.PORTLET_REFRESH %>">
    <table border="0" align="center">
        <tr>
            <td width="100%" align="center">
            <table border="0" align="left">
            <tr>
<%
                if ( target != null
                     && target.equals("finish")
                     &&  true  )
                {
                    try
                    {
                     VoidType terminateResponse = new VoidType();
                     ContinueHeader finishHeader = new
ContinueHeader(conversationID);
                  terminateResponse = soap.terminateRequest(null, finishHeader);
%>
                <td><%=cnvrtSC("Conversation terminated.")%></td>
<%
                    }
                    catch ( java.rmi.RemoteException e )
                    {
%>
                <td><%=cnvrtSC("Conversation already terminated.")%></td>
<%
                        e.printStackTrace();
                    }
                }
%>
            </tr>
            </table>
            </td>
        </tr>
        <tr>
            <td width="100%" align="center"><input type="submit" name="finish"
value="Finish"></td>
        </tr>
    </table>
    <br><br>
    <input type="hidden" name="target" value="finish">
</portlet:form>
```

15. Deploy the conversation portlet according to the instructions in the section Deploying the Web Services Portlets.
16. Test the new portlet by clicking on each of the buttons and verifying the results, as shown in Figure 11-114.

Figure 11-114 Starting the conversation


default - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address portletid=conversation&wfevent=bea.portal.framework.internal.refresh
Go Links
login
Home
conversation

Duplicate conversation id for start:
9rAdfijnW0jMV7UvL0fuNKZ27vAMHz7rVYUyeAlcRoUhVZp7YCqR!-1279539941!1026228381103

Start
Continue
Finish
Built On bea
Copyright © 2000-2002 by BEA Systems
Done
Local intranet

Error Handling within Web Services Portlets

In a production scenario, the Web Services to which your portlets connect are typically hosted elsewhere, and are out of your control. The portal framework enables portlets to generate two errors specifically designed to handle Web Services problems:

JAXRPCException: If a Web service is unavailable at run-time, the portlet will cause the throw a `javax.xml.rpc.JAXRPCException`. Add error handling to your .JSP by catching the exception in a generated portlet.

Note: The `JAXRPCException` applies to cases where the connection is refused, not when there is a delay in service.

SOAPFaultException: When a Web service cannot handle the SOAP request generated from the Web Service Portlet Wizard, a `javax.xml.rpc.soap.SOAPFaultException` is thrown. You should catch this exception within your .JSP to protect from compile failures.

Portalizing an Existing Web Application

To move an existing non-portal Web application into the portal framework, certain modifications are necessary. This section outlines the process using an example provided with the WebLogic Platform installation.

Getting Started

One strategy for adding functionality to a portlet is to graft JSP code from an existing (non-portal) Web application into the JSPs that constitute a portlet. This tutorial uses refactored sample code from the Internationalization portlet included with the product such that the functionality is replicated within a portlet. Figure 11-115 and Figure 11-116 show the Internationalization sample application displaying language-specific content based on user input.

Note: For detailed instructions on launching and exploring this application, consult the Personalization Examples section of the WebLogic Platform documentation.

Figure 11-115 Internationalization Input

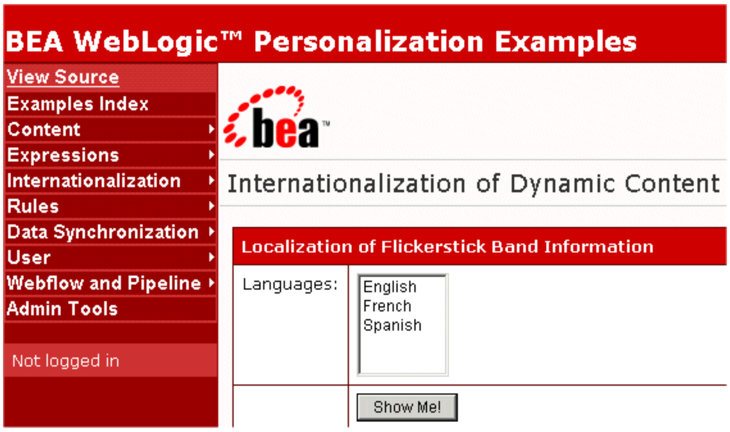
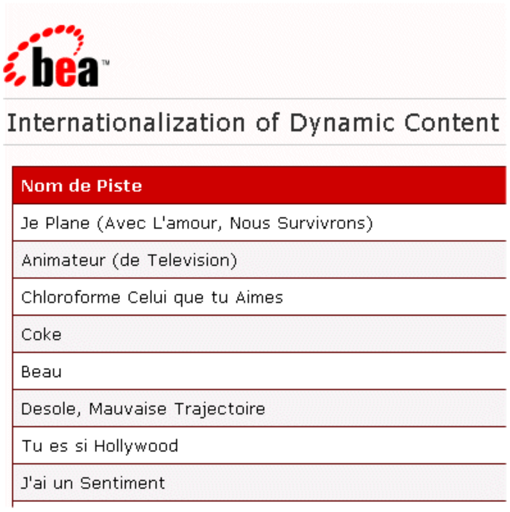


Figure 11-116 Internationalization Results



Nom de Piste
Je Plane (Avec L'amour, Nous Survivrons)
Animateur (de Television)
Chloroforme Celui que tu Aimes
Coke
Beau
Desole, Mauvaise Trajectoire
Tu es si Hollywood
J'ai un Sentiment

Requirements

WebLogic Portal 7.0 with Service Pack 1 must be successfully installed.

Process Overview

This process includes the following steps:

Step 1: Create a Portal Web Application

Step 2: Build a 2-page WebFlow Portlet

Step 3: Edit Portlet Code

Step 4: Load Content Resources

Step 5: Test the application

Step 1: Create a Portal Web Application

For instructions on creating a new portal Web application to use as the basic structure for this new application, consult the tutorial called Creating the New Portal in the WebLogic Portal 7.0 Development Guide.

For this example, the portal Web application will be called `NewPWApp`.

Note: If your application makes use of portal services such as Personalization, Internationalization, etc., you must add support for this functionality to the portal created using the portal Wizard. For detailed instructions on adding these features to your portal, consult the section called Adding All Portal Services to Your Domain in the Building Custom Templates chapter of the WebLogic Portal 7.0 Development Guide.

Step 2: Build a 2-page WebFlow Portlet

While the portal server created in the new domain is running, use the E-Business Control Center to launch the Portlet Wizard. Create a 2-page Webflow portlet, naming it **i18n**. For detailed instructions on creating Webflow portlets this way, consult the section Creating a Sequential Webflow Portlet.

Note: Don't forget to make the portlet visible and available using the WebLogic Portal Administration Tools.

Step 3: Edit Portlet Code

In this step, JSPs and properties files are edited to use the portlet Webflow and to invoke personalization.

Replace Portlet JSPs

First, the JSPs generated by the Portlet Wizard need are replaced with JSPs that make calls to Personalization services and act upon content. Save the contents of Listing 11-9 and Listing 11-10 in the following directory:

```
<BEA_HOME>beaApps\portalApp\NewPWApp\portlets\i18n
```

Listing 11-9 Page1.jsp

```
<%----------------------------------------------------------

Copyright (c) 2000-2002  BEA Systems, Inc.  All rights reserved.

---------------------------------------------------------%>


<%---------------------------------------------------------

File: Page1.jsp

Purpose: Gathers form input for I18N language of choice.

-----------------------------------------------------------%>


<%@ taglib uri="webflow.tld" prefix="webflow"%>

<%@ taglib uri="portlet.tld" prefix="portlet"%>


<%-----------------------------------------------------------

Declare html font styles for valid and invalid form entries, to be

used with webflow validated form.

----------------------------------------------------------%>
```

```
<% String validStyle = "background: white; color: black;
font-family: Arial"; %>

<% String invalidStyle = "background: white; color: red;
font-style: italic"; %>


<center>

<%-------------------------------------------------------

Using portlet validated form.

---------------------------------------------------------%>

<portlet:validatedForm event="switch2" applyStyle="message"

     messageAlign="right" validStyle="<%= validStyle %>"

     invalidStyle="<%= invalidStyle %>" unspecifiedStyle="<%=
validStyle %>">


<table border="0" cellspacing="0" cellpadding="0" width="100%">

  <tr>

    <td>

      <table border="0" cellpadding="6" cellspacing="1"
width="100%">

        <tr class="header">

          <td colspan="2">

            Localization of Flickerstick Band Information

          </td>

        </tr>

        <tr class="tablerow1">

         <td align="right" valign="top" width="1%">Languages:</td>

          <td>

<%----------------------------------------------------------
```

```
Using <webflow:select> and <webflow:option> in place of standard
html select and option to enable form validation.

----------------------------------------------------------%>

            <webflow:select name="language" size="5">

              <webflow:option value="en"/>English

              <webflow:option value="fr"/>French

              <webflow:option value="es"/>Spanish

            </webflow:select>

          </td>

        </tr>

        <tr class="tablerow1">

          <td align="right" valign="top" width="1%">&nbsp;</td>

          <td>

            <input type="submit" name="Submit" value="Show Me!">

          </td>

        </tr>

        <tr class="tablerow2">

          <td class="label" colspan="2">

            Select the language in which you would like to view
Flickerstick information.

          </td>

        </tr>

      </table>



      <input type="hidden" name="resultFile" value="Page2.jsp">

      <input type="hidden" name="sample" value="<%=
request.getParameter("sample") %>">

    </td>

  </tr>
```

```
</table>

</portlet:validatedForm>

</center>
```

Listing 11-10 Page2.jsp

```
<%-------------------------------------------------------------

Copyright (c) 2000-2002  BEA Systems, Inc.  All rights reserved.

-------------------------------------------------------------%>

<%-------------------------------------------------------------

File: Page2.jsp

Purpose: Gathers form input for I18N language of choice.

-----------------------------------------------------------%>

<%@ page import="com.bea.p13n.content.ContentHelper"%>

<%@ page import="com.bea.p13n.content.Content" %>

<%@ taglib uri="cm.tld" prefix="cm" %>

<%@ taglib uri="es.tld" prefix="es" %>

<%@ taglib uri="i18n.tld" prefix="i18n" %>

<%@ taglib uri="portlet.tld" prefix="portlet" %>


<%-----------------------------------------------------------

Contruct the query string.

Example: isTrackIdentifier='true' && bandName='Flickerstick' &&
language='en'

-----------------------------------------------------------%>

<%

    StringBuffer queryStr = null;

    String language = request.getParameter("language");
    if (language != null)

    {

       // Build the query string

       queryStr = new StringBuffer();

       queryStr.append("isTrackIdentifier = 'true' && bandName = 
'Flickerstick' && language = '");

       queryStr.append(language);

       queryStr.append("'");

    }


    //queryStr = new StringBuffer();

    //queryStr.append("bandName = 'Flickerstick'");


    if (queryStr != null)

    {

%>

<% System.out.println("\n\nqueryStr=" + queryStr + 
"--------------------------------------------\n\n"); %>

<br>Language is: <%= language %><br>

<br>Query String is: <%= queryStr %><br>

<br>ContentHelper.DEF_CONTENT_MANAGER_HOME is : <%= 
ContentHelper.DEF_CONTENT_MANAGER_HOME %><br>

<%------------------------------------------------------------

Localize the page with the selected language. Future invocations of

i18n tags in this request will default to this language.

------------------------------------------------------------%>

   <i18n:localize language="<%= language %>"/>
```

```
<%------------------------------------------------------------

Using the constructed query string retrieve the track names for

Flickerstick.

------------------------------------------------------------%>

   <cm:select contentHome="<%=
ContentHelper.DEF_CONTENT_MANAGER_HOME %>"

      sortBy="trackNum" query="<%= queryStr.toString() %>"
id="contentArray" failOnError="true"/>


    <table border="0" cellspacing="0" cellpadding="0" width="100%">

      <tr>

        <td>

          <table border="0" cellspacing="1" cellpadding="6"
width="100%">

            <tr class="tableheader">

<%---------------------------------------------------------

Retrieved localized messages for track name and track number.

---------------------------------------------------------%>

     <td><i18n:getMessage messageName="trackName"
bundleName="Page2"/></td>

     <td><i18n:getMessage messageName="trackNum"
bundleName="Page2"/></td>

            </tr>


            <% int row = 0; %>

        <br>contentArray length is : <%= contentArray.length %><br>

      <es:forEachInArray id="nextDoc" array="<%= contentArray %>"
type="Content">

    <tr class="<%= (row % 2 == 0) ? "tablerow1" : "tablerow2" %>">

<%------------------------------------------------------------
```

```
Get the bandName property using the cm:getProperty tag and use it to

construct the parameters to pass to the Webflow.

-----------------------------------------------------------%>

          <td>

   <cm:printProperty id="nextDoc" name="trackName" encode="html"/>

         </td>

         <td>

   <cm:printProperty id="nextDoc" name="trackNum" encode="html"/>

          </td>

         </tr>

         <% row++; %>

            </es:forEachInArray>

          </table>

        </td>

      </tr>

    </table>

<%

    }

    else

    {

%>

        <b>Please specify one language in your request!</b>

<%

    }

%>

<center>

<a href="<portlet:createWebflowURL event="switch1"/>">Previous
Page</a>
```

```
</center>
```

Save Properties Fies for Internationalization

Save the contents of the following listings in this directory:

`<BEA_HOME>beaApps\portalApp\NewPWApp\portlets\i18n`

For example, Listing 11-11 would be saved as `Page2_en.properties`.

Listing 11-11 Page2_en.properties

```
trackName=Track Name

trackNum=Track Number
```

Listing 11-12 Page2_fr.properties

```
trackName=Nom de Piste

trackNum=Numero do Piste
```

Listing 11-13 Page2_sp.properties

```
trackName=Nombre de la canción

trackNum=Número de la canción
```

Step 4: Load Content Resources

In this step, content resources are imported from the Personalization domain.

1. Replace the dmsBase folder in your portal domain by copying the entire dmsBase folder (including its contents) from `<BEA_HOME>weblogic700\samples\portal\p13nDomain` into the following directory:

   ```
   <BEA_HOME>\user_projects\myNewDomain
   ```

2. To make this content available to the Portal framework, the metadata must be loaded into the server. While the server is running, execute the `loaddata` script in the following directory:

   ```
   <BEA_HOME>\user_projects\myNewDomain
   ```

Step 5: Test the application

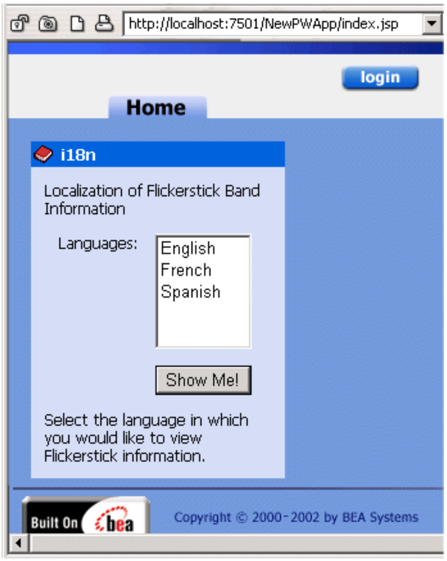
Now that the JSPs are edited, observe the functionality of the new portlet by taking the following steps:

1. Verify the portlet can be accessed by navigating to the following URL:

```
http://<host>:<port>/NewPWApp/
```

   The result should resemble that shown in Figure 11-117.

Figure 11-117 Verifying the i18n Portlet


http://localhost:7501/NewPWApp/index.jsp
login
Home
i18n
Show Me!
Built On bea
Copyright © 2000-2002 by BEA Systems

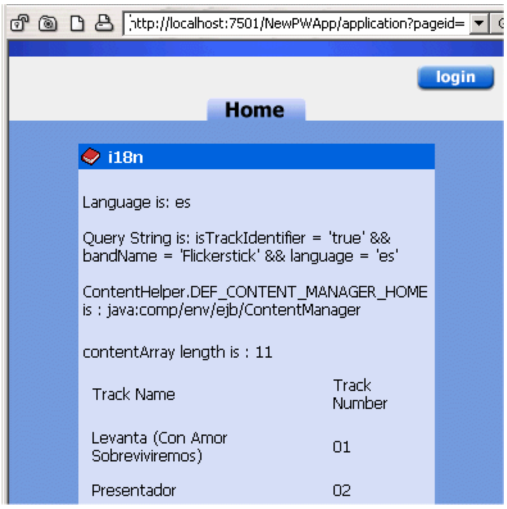
Localization of Flickerstick Band Information

Languages: English French Spanish

Select the language in which you would like to view Flickerstick information.

2. Select language and click `Show Me`. The results should resemble those shown in Figure 11-117.

Figure 11-118 Results page of i18n Portlet



Language is: es

Query String is: isTrackIdentifier = 'true' && bandName = 'Flickerstick' && language = 'es'

ContentHelper.DEF_CONTENT_MANAGER_HOME is : java:comp/env/ejb/ContentManager

contentArray length is : 11

Track Name	Track Number
Levanta (Con Amor Sobreviviremos)	01
Presentador	02

Performance Tuning

This section covers performance issues related specifically to WebLogic Portal, including JDBC and Thread settings and several cache settings. Many factors effecting the performance of your portal application are specific to WebLogic Server. For information on making those adjustments, consult the *WebLogic Server Performance and Tuning* guide at http://edocs.bea.com/wls/docs70/perform/index.htmll.

Using Caches to Tune Performance

To adjust caching for production Web site, examine the following factors:

- Adjust Caching for Content Management
- Property Caching in a Clustered Environment
- Adjust Caching for the Discount Service
- Adjusting the discountCache
- Adjusting the globalDiscountCache
- Discount-Service Caches in Clustered and Non-Clustered Environments
- Adjust Group Membership TTL in the Caching Realm
- Tuning Thread / Connection Parameters in JDBC

Adjust Caching for Content Management

To optimize content-management performance for your production Web site, the Content Manager uses the caching framework to configure and manage the following caches:

`documentContentCache`

`documentMetadataCache`

`documentIdCache`

The content management JSP tags provide an additional set of caches, which you can access by doing the following:

For the `cm:select`, `cm:selectById`, `pz:contentQuery`, and `pz:contentSelector` JSP tags, use the `useCache` attribute whenever possible. Doing so avoids a call to `DocumentManager` and, in the case of `pz:ContentSelector`, to the Rules Manager.

To clear cached content when user and/or document attributes change, use the remove method of com.bea.p13n.content.ContentCache. For more information, see the WebLogic Portal Javadoc. for com.bea.p13n.content.ContentCache.

For the cm:select, cm:selectById, pz:contentQuery, and pz:contentSelector JSP tags, set the cacheScope attribute to application whenever possible. This application scope applies to the Web application, not to the enterprise application, as shown in Listing 11-14.

Listing 11-14 Setting cacheScope to *application*

```
<cm:select id="myDocs" query="riskFactor = 'Low'"

useCache="true" cacheId="myDocs"

cacheScope="application"

max="10" cacheTimeout="300000" />
```

The application cache type is global instead of per-user and should speed up queries by avoiding a call to the DocumentManager EJB.

For `pz:contentSelector`, set the `cacheScope` attribute to application only when you want to select shared content. For example, you create an application that uses an application-scoped cache to select content for non-authenticated users. Because it uses the application scope, all non-authenticated users see the same content. For authenticated users, your application provides personalized content by switching to a session scoped cache.

Whenever you can predict the next document that users will view based on the document that they are currently viewing, load the next document into the cache before users request it. This “forward caching” will greatly improve the speed at which

WebLogic Portal responds to user requests (assuming that your prediction is correct; forward caching a document that no one requests will only degrade performance and scalability).

Listing 11-15 contains a snippet of a JSP with an example of forward caching a document:

Listing 11-15 Forward caching a document

```
<%-- Get the first set of content --%>

<cm:select id="myDocs" query="riskFactor = 'Low'"

useCache="true" cacheId="myDocs"

cacheScope="application"

max="10" cacheTimeout="300000" />

<%-- Generate a query from each content's relatedDocId --%>

<% String query = null; %>

<es:forEachInArray array="<%=myDocs%>" id="myDoc"
type="com.bea.p13n.content.Content">

<% String relId = (String)myDoc.getProperty("relatedDocId", null);
%>

<es:notNull item="<%=relId%>">

<%

if (query != null)

query += " || ";

else

query = "";

query += "identifier = '" +

ExpressionHelper.toStringLiteral(relId) + "'";

%>

</es:notNull>

</es:forEachInArray>
```

```
<%-- Load the related content into the cache via cm:select --%>

<es:notNull item="<%=query%>">

<cm:select query="<%=query%>" id="foo" useCache="true"
cacheId="relatedDocs"

cacheScope="session" max="10" cacheTimeout="300000" />

</es:notNull>
```

For more information on content management JSP tags, see “Personalization JSP Tags” in the *JavaServer Page Guide* at http://edocs.bea.com/wlp/docs70/jsp/p13njsp.htm.

Property Caching in a Clustered Environment

To decrease the amount of time needed to access user, group, and other properties data, the WebLogic Server Configurable Entity and Entity Property Manager use the cache framework to configure and manage the following caches:

`ldapGroupCache`

`ldapUserCache`

`entityPropertyCache`

`entityIdCache`

`unifiedProfiletypeCache`

`propertyKeyIdCache`

Note: By default, these property caches are enabled.

With property caching enabled in a clustered environment, each server in a cluster maintains its own cache; the cache is not replicated on other servers. In this environment, when properties that are stored in the caches change on one server, they may not change on another server in a timely fashion. In most cases, immediate or quick access to properties on another server is not necessary: user sessions are pinned to a single server, and even with caching enabled, users immediately see changes they make to their own settings on the server.

If a user and an administrator are pinned to different servers in the cluster and the administrator changes a user's properties, the user may not see the changes during the current session. You can mitigate this situation by specifying a small Time-To-Live (TTL) setting.

If you require multiple servers in a cluster to have immediate access to modified properties, disable property caching.

Adjust Caching for the Discount Service

To reduce the amount of time the Order and Shopping Cart services need to calculate order and price information that include discounts, the Discount Service uses the caching framework to create and manage the following caches:

- discountCache, which contains data for campaign discounts. Campaign discounts are targeted to specific customers or customer segments, and are available only in the context of a campaign.
- globalDiscountCache, which contains data for global discounts. Global discounts apply to all customers, regardless of customer properties or customer segments.

When a customer adds an item to the shopping cart, removes an item from the shopping cart, checks out, or confirms an order, the Pricing Service is responsible for determining the price of the items in the cart. To calculate the effect of discounts on the shopping cart, the Pricing Service requests the Discount Service to retrieve information about all global discounts and about any campaign discounts that apply to the current customer.

The first request for information about discounts requires a separate call to the database for each discount that applies. For example, if you have defined one global discount and if a customer is eligible for two campaign-related discounts, the Discount Service makes three calls to the database. To decrease the response time for any subsequent requests, the Discount Service uses the caches.

Adjusting the discountCache

The discountCache contains data for campaign discounts. For maximum performance, set the capacity to the number of campaign discounts that are currently deployed. A larger capacity will potentially use more memory than a smaller capacity.

The Time-To-Live (TTL) property determines the number of milliseconds that the Discount Service keeps the information in the cache. After the cache value times out, the next request for the value requires the Discount Service to call the database to retrieve the information and then cache the value. A longer TTL decreases the number of database calls made over time when requesting cached objects. In a clustered environment, the TTL is the maximum time required to guarantee that any changes to global discounts are available on all servers.

Adjusting the globalDiscountCache

The Maximum Number of Entries property for global caches does not need to be modified.

The time-to-live property determines the number of milliseconds that the Discount Service keeps information in the global-discount cache. After the Time-To-Live (TTL) expires, the next request for global discount information requires the Discount Service to call the database to retrieve the information and then cache the value. A longer TTL decreases the number of database calls made over time when requesting cached objects. In a clustered environment, the TTL is the maximum time required to guarantee that any changes to campaign discounts are available on all servers.

Discount-Service Caches in Clustered and Non-Clustered Environments

In either environment (clustered or non-clustered), when you change a discount priority, end date, or its active/inactive state, WebLogic Portal flushes the discount from the appropriate cache. Changes to a campaign discount flush only the specific discount from the campaign-discount cache. Changes to a global discount flush all discounts from the global-discount cache.

For example, you log in to a WebLogic Portal host named bread and deactivate a campaign discount named CampaignDiscount1. WebLogic Portal flushes the CampaignDiscount1 from the campaign-discount cache on bread.

In a clustered environment, other machines in the cluster continue to use their cached copy of the discount until the TTL for that discount expires.

Adjust Group Membership TTL in the Caching Realm

The WebLogic Server Caching realm stores the results of both successful and unsuccessful realm lookups. It does not use the WebLogic Portal caching framework.

The Caching realm manages separate caches for Users, Groups, permissions, ACLs, and authentication requests. It improves the performance of WebLogic Server by caching lookups, thereby reducing the number of calls into other security realms.

WebLogic Portal enables the Caching realm by default. While all of the caches in the Caching realm can improve performance, the Time-To-Live (TTL) value for the Group Membership Cache in particular can affect the performance of WebLogic Portal.

In addition, note that if you delete a user from the system without first removing the user from a group, then the system continues to recognize the user until the TTL for the Group Membership Cache expires.

For information on adjusting the Group Membership TTL, refer to the *WebLogic Server Administration Guide* at http://edocs.bea.com/wls/docs70/adminguide/index.html.

Tuning Thread / Connection Parameters in JDBC

Certain performance problems encountered in the portal may be corrected by changing config.xml entries to reduce to thread count for the default execute queue in WLS lower than the connection pool maximum capacity specified for the commercePool. The basic formula should make the number of connections in the connection pool equal to the number of threads + 1.

For information on adjusting threads and connection pools, consult “Tuning JDBC Connection Pool Maximum Capacity” at http://edocs.bea.com/wls/docs70/perform/WLSTuning.html#1117878.

CHAPTER

12 Setting Up Personalization and Interaction Management

WebLogic Portal comes with robust authentication and personalization features that allow administrators to determine what content a visitor can interact with and how that information will appear to the specific visitor. Visitors themselves can leverage WebLogic Portals personalization features to select their own content and create their own look and feel. A major component of the portal development process is to create the resources by using such tools as the Advisor, the Rules Framework, and content selectors to make such authorization and personalization possible.

This section includes information on the following subjects:

Using the Advisor to Personalize a Portal Application

The WebLogic Portal Advisor is an easy-to-use and flexible access point for personalization services-including personalized content, user segmentation, and the underlying rules engine. The Advisor delivers content to a personalized application based on a set of rules and user profile information. It can retrieve any type of content from a Document Management system and display it in a JSP.

The Advisor ties together all the services and components in the system to deliver personalized content. The Advisor component includes a JSP tag library and an Advisor EJB (stateless session bean) that access the WebLogic Portal's core personalization services including:

- User Profile Management
- Rules Manager
- Content Management
- Personalization Platform

The tag library and session bean contain personalization logic to access these services, sequence personalization actions, and return personalized content to the application. It is also possible to write your own Advisor plug-ins and access them with JSP tags you create.

This architecture allows the JSP developer to take advantage of the personalization services using the Advisor JSP tags. In addition, a Java developer can access the underlying WebLogic Portal personalization features via the public Advisor bean interface. For more information, see the WebLogic Portal Javadoc API documentation.

You can use the Advisor in one of two ways:

- **Using the JSP tags**. Developers will probably find it easiest to use the JSP tags when building typical pages. The tags provide ways to switch content on and off based on user classification, return content based on a static query, and match content to users based on rules that execute a content query. The JSP tags that perform these tasks are: `<pz:div>`, `<pz:contentSelector>`, and `<pz:contentQuery>`.

- **Using the Advisor session bean.** The page or application developer may use the Advisor session bean directly in place of the tags, if desired. The Advisor session beans provide ways to switch content on and off based on user classification, return content based on a static query, and match content to users based on rules that execute a content query.

Creating a Personalized Portal Application with Advisor JSP Tags

Table 12-1 describes the three JSP tags the Advisor provides to help developers create personalized applications. These tags provide a JSP view to the Advisor session bean and allow developers to write pages that retrieve personalized data without writing Java source code.

Table 12-1 Advisor JSP Tags

Tag	Description
`<pz:div>`	Turns user-provided content on or off based on the results of a classifier rule being executed. If the result of the classifier rule is `true`, it turns the content on; if `false`, it turns the content off.**The system turns on the content by inserting the content residing between the start and end `<pz:div>` tags in the JSP code. This content can include any valid JSP content, including HTML tags, other JSP tags, and scriptlets. If the classifier rule returns `false`, the system skips the content between the start and end `<pz:div>` tags.
`<pz:contentQuery>`	provides content attribute searching for content in a content management system. It returns an array of `Content` objects that a developer can handle in numerous ways.
`<pz:contentSelector>`	recommends content if a user matches the classification part of a content selector rule. When a user matches, the personalization engine executes a content query defined in the rule and returns the content back to the JSP page.

In addition to using JSP tags to create personalized applications, you can work directly with the Advisor bean. For more information about using the bean, see "Creating Personalized Applications with the Advisor Session Bean" on page 12-6.

Classifying Users with the JSP <pz:div> Tag

The `<pz:div>` tag to turns user-provided content on or off based on the results of a classifier rule being executed. If the result of the classifier rule is `true`, it turns the content on; if `false`, it turns the content off.

Note: Rules are created in the E-Business Control Center. The E-Business Control Center letsusers develop their own classifier rules. Because users are not exposed to the concept of rules, you will see classifier rules referred to as "customer segments."

Listing 12-1 shows how to use the `<pz:div>` tag to execute the *PremierCustomer* classifier rule and displays an alert to premier customers in the HTML page's output.

Listing 12-1 Using <pz:dev> to Execute a Classifier Rule

```
<%@ taglib URI="pz.tld" prefix="pz" %>
.
.
.
<pz:div
rule="PremierCustomer">
    <p>Please check out our new Premier Customer bonus program…</p>
</pz:div>
```

Selecting Content with the <pz:contentQuery> JSP Tag

Use the `<pz:contentQuery>` tag to provide content attribute searching of content in a content management system. It returns an array of `Content` objects that you can handle in numerous ways.

Listing 12-2 shows an example of how to execute a query against the content management system to find all content where the author attribute is *Hemingway* and then display the `Document` titles found:

Listing 12-2 Executing a Query Against a CMS to Find Specified Content

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib URI="pz.tld" prefix="pz" %>
.
.
.
<pz:contentQuery id="docs"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
query="author = 'Hemingway'" />

<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

Matching Content to Users with the <pz:contentSelector> JSP Tag

The `<pz:contentSelector>` recommends content if a user matches the classification part of a content selector rule. When a user matches based on a rule, the Advisor executes the query defined in the rule to retrieve content.

The example in Listing 12-3 asks the Advisor to return content specific to premier customers and then display the `Document` titles as the results.

Listing 12-3 Asking the Advisor to Display Specific Customers

```
<%@ page import="com.bea.p13n.content.ContentHelper" %>
<%@ taglib URI="cm.tld" prefix="cm" %>
<%@ taglib URI="pz.tld" prefix="pz" %>
<%@ taglib URI="es.tld" prefix="es" %>
.
.
.
<pz:contentSelector id="docs"
    rule="PremierCustomerSpotlight"
    contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>" />
<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

Creating Personalized Applications with the Advisor Session Bean

Java developers can work directly against the Advisor bean through a set of APIs to create personalized applications. This process provides an alternative to using the JSP tags to call into the bean.

Note: See the WebLogic Portal Javadoc for more information about using the session bean to create personalized applications.

The following steps provide a general overview of the process involved for an application to get content recommendations from the Advisor.

1. Look up an instance of the Advisor session bean.
2. Use the AdvisorFactory's static `createAdviceRequest` method to create an AdviceRequest object.

   Note: You must provide this method with the URI representing the request. The Advisor uses the URI prefix to determine which Advislet to invoke.
3. Set the required and optional attributes for the AdviceRequest object.
4. Call the Advisor's `getAdvice` method.

   The Advisor calls the best Advislet to make the recommendation. The Advislet determines the recommendations and the Advisor then passes the resultant `Advice` object back to the application.

   The Advisor uses the Advislet Registry to choose the Advislet to invoke.
5. The personalized application extracts the recommendation from the `Advice` object and uses it in the application.

When a personalized application requests advice from the Advisor, the Advisor bean delegates the request to a registered Advislet that can handle the request. The Advisor uses the URI prefix to determine which registered Advislet will receive the advice request. The Advislet then makes the recommendations and returns the `Advice` object back to the Advisor. This design encapsulates all of the advice logic into the Advislet and allows developers to create custom Advislets for more specialized purposes.

Attribute objects act as parameters for the request. Attribute objects can be set on the `AdviceRequest` object and are associated with a `String` object representing the name of the attribute.

Three Advislets are supplied with the system: Classifier Advislet, ContentQuery Advislet and ContentSelector Advislet. Names for the attributes that need to be set on the supplied Advislets are defined as static Strings in the `AdviceRequestConstants` interface.

Table 12-2 shows the logic the Advisor uses to determine how to map a recommendation request to an Advislet.

Table 12-2 Mapping recommendation requests to an Advislet

Uri Prefix	Inferred Advislet
`classifier`	Uses a rules-based inference engine to classify a user based on rules written using the Customer Segment tool in the E-Business Control Center.
`contentselector`	■ Uses a rules-based inference engine to classify a user. ■ Determines if the user matches the classification. ■ Uses a rules-based inference engine to obtain a content query for the classification. ■ Selects content based on the content query obtained.
`contentquery`	Performs a content attribute search on a specified content management system.

The following sections demonstrate how to directly access the Advisor to provide the same functionality as that provided by the JSP tags.

Classifying Users with the Advisor Session Bean

For classification requirements beyond what the JSP tags provide, or to use classification in a servlet, use the Advisor EJB directly.

To ask the Advisor for a classification, use this procedure. (See the Javadoc API documentation for API details.)

Note: Unless otherwise indicated, all classes used here reside in the `com.bea.p13n.advisor` package.

1. Look up and create an instance of the Advisor session bean. The `EJB_REF_NAME` constant found in the EJB Advisor Home interface may be used as the JNDI name of the Advisor EJB Home.
2. Use the AdvisorFactory's static `createAdviceRequest` method to create an `AdviceRequest` object. In this case, the URI argument should be "`classifier://`".
3. Set the required attributes on the `AdviceRequest` object (see `AdviceRequestConstants`). These include:
   - `HTTP_REQUEST` – the request object (retrieved from `com.bea.p13n.httpRequest.createP13NRequest(HttpServlet Request)`).
   - `HTTP_SESSION` – the session object (retrieved from `com.bea.p13n.httpSession.createP13NSession(HttpServletReques t)`).
   - `USER` – the user object (retrieved from `com.bea.p13n.usermgmt.SessionHelper.getProfile(HttpServletRe quest)`).
   - `TIME_INSTANT` – a `java.sql.Timestamp` object representing *now.*
   - `RULES_RULENAME_TO_FIRE` – (optional) the name of the segmentation rule to fire.
4. Call the `getAdvice` method on the Advisor, supplying the newly created `AdviceRequest`.

5. The Advisor returns an instance of `Advice`. The `getResult` method is called to obtain the classification object. If a classification object is returned, then the classification is considered to be `true`. If the return value is `null`, the classification is considered to be `false`.

Note: If the optional `AdviceRequest` parameter `RULES_RULENAME_TO_FIRE` is not supplied, there may be multiple classifications returned for the user.

Querying a Content Management System with the Advisor Session Bean

For content selection requirements beyond what the JSP tags provide, or to use content selection in a servlet, developers can use the Advisor EJB directly.

To ask the Advisor for a content, use this procedure. (See the Javadoc API documentation for API details.)

Note: Unless otherwise indicated, all classes used here reside in the `com.bea.p13n.advisor` package.

1. Look up and create an instance of the Advisor session bean. The `EJB_REF_NAME` constant found in the EJB Advisor Home interface may be used as the JNDI name of the Advisor EJB Home.

2. Use the AdvisorFactory's static `createAdviceRequest` method to create an `AdviceRequest` object. In this case, the URI argument should be "`contentquery://`"

3. Set the required attributes on the `AdviceRequest` object (see `AdviceRequestConstants`). These include:

   - `CONTENT_MANAGER_HOME` (required) – the JNDI name to find a content manager home interface.
   - `CONTENT_MANAGER` (optional) - the instance of a ContentManager remote interface that should be used. If this is set, then `CONTENT_MANAGER_HOME` does not need to be set.
   - `CONTENT_QUERY_STRING` (required) – the query to run against the system.
   - `CONTENT_QUERY_SORT_BY` (optional) – the order in which to sort the returned results.
   - `CONTENT_QUERY_MAX_ITEMS` (optional) – the maximum instances to return.

- `CONTENT_CONTEXT_PARAMS` (optional) - a map of name/value pairs to pass in the generated Search object to the ContentManager.

4. Call the `getAdvise` method on the Advisor, supplying the newly created `AdviceRequest`.

5. The Advisor returns an instance of `Advice`. The `getResult` method is called to obtain the array of `Content` objects representing the results of the content query.

Matching Content to Users with the Advisor Session Bean

For content selection requirements beyond what the JSP tags provide, or to use content selection in a servlet, developers can use the Advisor EJB directly.

To ask the Advisor for a content, use this procedure. (See the Javadoc API documentation for API details.)

Note: Unless otherwise indicated, all classes used here reside in the `com.bea.p13n.advisor` package.

1. Look up and create an instance of the Advisor session bean. The `EJB_REF_NAME` constant found in the EJB Advisor Home interface may be used as the JNDI name of the Advisor EJB Home.

2. Use the AdvisorFactory's static `createAdviceRequest` method to create an `AdviceRequest` object. In this case the URI argument should be "`contentselector://`"

3. Set the required attributes on the `AdviceRequest` object (see `AdviceRequestConstants`). These include:

   - `HTTP_REQUEST` – the request object (retrieved from `com.bea.p13n.httpRequest.createP13NRequest(HttpServletRequest)`).
   - `HTTP_SESSION` – the session object (retrieved from `com.bea.p13n.httpSession.createP13NSession(HttpServletRequest)`).
   - `USER` – the user object (retrieved from `com.bea.p13n.usermgmt.SessionHelper.getProfile(HttpServletRequest)`).
   - `TIME_INSTANT` – a `java.sql.Timestamp` object representing the time *now.*

- `RULES_RULENAME_TO_FIRE` – (optional) the name of the content selector rule to fire.
- `CONTENT_MANAGER_HOME` (required) – the JNDI name to find a content manager home interface.
- `CONTENT_MANAGER` (optional) - the instance of a ContentManager remote interface that should be used. If this is set, then `CONTENT_MANAGER_HOME` does not need to be set.
- `CONTENT_QUERY_STRING` (required) – the query to run against the system.
- `CONTENT_QUERY_SORT_BY` (optional) – the order in which to sort the returned results.
- `CONTENT_QUERY_MAX_ITEMS` (optional) – the maximum instances to return.
- `CONTENT_APPEND_QUERY_STRING` (optional) - the query to append to the query from the rules engine. If this query starts with "||" (2 vertical bars), it will be OR'ed to the rules query; otherwise it will be AND'ed.
- `CONTENT_CONTEXT_PARAMS` (optional) - a map of name/value pairs to pass in the generated Search object to the ContentManager.

4. Call the `getAdvise` method on the Advisor, which supplies the newly created `AdviceRequest`.
5. The Advisor returns an instance of `Advice`. The getResult method is called to obtain the array of `Content` objects representing the recommendation.

Personalizing Applications with HTTP Request and Session Properties

Attributes in the HTTP Request and Session can be used to personalize content. Use the E-Business Control Center to create Customer Segments, Content Selectors, or Campaigns that use HTTP Request and Session property sets. Once you have synchronized your Request or Session property sets to the WebLogic Portal server, you can personalize content with them.

- For more information on customer segments, see "Creating Customer Segments" at http://edocs.bea.com/wlp/docs70/admin/usrgrp.htm#1184110.

- For more information on content selectors, see "Personalization with Content Selectors" on page 12-19.
- For more information on Campaigns, see "Creating Campaigns" at http://edocs.bea.com/wlp/docs70/admin/campaign.htm.

HTTP Request-Based Personalization

This section shows a Customer Segment definition created in the E-Business Control Center. The Customer Segment, called `RequestPropertyDemo`, automatically makes a user a member of the Customer Segment when an HTTP request has a specific value. The HTTP request value being looked for is defined in an HTTP Request property called `RequestPropertyOne`, which is also defined in an HTTP Request property set in the E-Business Control Center. If the value of RequestPropertyOne is `success`, the user is a member of the `RequestPropertyDemo` Customer Segment and can be targeted with personalized content.

When **all** of these conditions apply:

an HTTP request has the following properties:

RequestPropertyOne is equal to 'success'

Consider the visitor a member of the **RequestPropertyDemo** segment.

The Customer Segment definition and HTTP Request property definition must be saved and synchronized to the WebLogic Portal server.

The following JSP code does the following:

1. The `request.setAttribute()` method sets an HTTP request property, called `RequestPropertyOne` to a value of `success`. Because of the Customer Segment definition, this HTTP request value makes the user a member of the `RequestPropertyDemo` Customer Segment.
2. The `<pz:div>` JSP tag has a `rule` attribute whose value is `RequestPropertyDemo`, the name of the Customer Segment defined in the E-Business Control Center.
3. The `<pz:div>` tag contains the content that is displayed only to members of the `RequestPropertyDemo` Customer Segment.

```
<%@ taglib uri="pz.tld" prefix="pz" %>

<%
```

```
    request.setAttribute("RequestPropertyOne", "success");

%>

<pz:div rule="RequestPropertyDemo">

    <p>--  the "RequestPropertyDemo" rule evaluated to "true"</p>

</pz:div>
```

While this example shows personalization based on a user's Customer Segment membership, you can trigger HTTP Request-based personalization directly in Content Selectors and Campaigns. When you define Content Selectors and Campaigns in the E-Business Control Center, you can trigger them with HTTP Request property values without using Customer Segments.

HTTP Session-Based Personalization

Personalization using HTTP Session-based Customer Segments is performed similarly to HTTP Request-based Customer Segments. The following JSP code assumes the E-Business Control Center has been used to create and synchronize a Customer Segment named `SessionPropertyDemo` and an HTTP Session property called `SessionPropertyOne`.

The text within the `<pz:div>` tags will be displayed only to members of the `SessionPropertyDemo` Customer Segment when the JSP is rendered.

```
<%@ taglib uri="pz.tld" prefix="pz" %>

<%

    session.setAttribute("SessionPropertyOne", "sessionValue");

%>

<pz:div rule="SessionPropertyDemo">

    <p>--  the "SessionPropertyDemo" rule evaluated to "true"</p>

</pz:div>
```

While this example shows personalization based on a user's Customer Segment membership, you can trigger HTTP Session-based personalization directly in Content Selectors and Campaigns. When you define Content Selectors and Campaigns in the E-Business Control Center, you can trigger them with HTTP Session property values without using Customer Segments.

Special Considerations

Request and Session attributes are not scoped to specific property sets. For example, you can create a `SessionPropertyOne` in two different Session property sets, then set an attribute of that name in the HTTP Session, and all rules based on that property will be evaluated without reference to the property set in which it is found.

To extend the previous Session-based example, create a second Session property set, also with a property called `SessionPropertyOne`, but make it a numeric property. Then create a second Customer Segment called `SessionPropertyDemoTwo` as follows:

When **all** of these conditions apply:

the HTTP session has the following properties:

SessionPropertyOne is equal to 3

Consider the visitor a member of the **SessionPropertyDemoTwo** segment.

The following JSP code contains two <pz:div> tags. One contains content that will be displayed to members of the `SessionPropertyDemo` Customer Segment, and the other contains content that will be displayed only to members of the `SessionPropertyDemoTwo` Customer Segment.

```
<%@ taglib uri="pz.tld" prefix="pz" %>

<%

    session.setAttribute("SessionPropertyOne", new Long(3));

%>

<pz:div rule="SessionPropertyDemo">

    <p>--  the "SessionPropertyDemo" rule evaluated to "true"</p>

</pz:div>

<pz:div rule="SessionPropertyDemoTwo">

   <p>--  the "SessionPropertyDemoTwo" rule evaluated to "true"</p>

</pz:div>
```

Because of the `SessionPropertyOne` value in the `session.setAttribute()` method, whose value defines the `SessionPropertyDemoTwo` Customer Segment, the `rule=SessionPropertyDemoTwo` will evaluate to "true," and the `rule=SessionPropertyDemo` will evaluate to "false", even though the property was defined in two different property sets. Normally this should not be a problem, as duplicate properties are not usually defined in different property sets. However, the E-Business Control Center does not alert you if you do this.

Triggering Campaign Actions with Session, Request, and Event Properties

Campaign scenario rules are evaluated only when a single event occurs for which the campaign listener is configured. When the event is triggered, the event takes a snapshot of the current session properties, the single request property (contained in the session), and the event properties (contained in the request). The snapshot taken by the event is in the form of a `Request` object, which the event passes to the Campaign service for evaluation. If the values in that snapshot evaluate to true against any Campaign action rules, those Campaign actions are triggered.

When campaign actions are not triggered as expected using session, request, and event properties, one or more of the following is usually to blame:

- No event was fired that the Campaign service was listening for.
- The session, request, or event properties contained in the Campaign rule were not part of the `Request` object snapshot taken when the event was fired.
- In Campaign rules that are defined so that *all* conditions must apply for the campaign action to be triggered, one or more of the conditions evaluated to false.

Consider the following Campaign action rules as created in the E-Business Control Center:

When **all** of these conditions apply:

an HTTP request has the following properties:

RequestPropertyOne is equal to 'success'

any of the following events has occurred:

SessionLoginEvent

Do the following… [Ad action, e-mail action, or discount action].

The rule will be evaluated only if an event for which the Campaign service is listening occurs. (This event need not be used directly in the Campaign rule.) For example, if the Campaign service is configured to listen for the BEA-provided

`UserRegistrationEvent` (which it is by default), then when a `UserRegistrationEvent` occurs the event takes a snapshot of the `Request` object and the Campaign rules are evaluated.

Here is how the previous Campaign action rules would be evaluated:

- Is there a request property called `RequestPropertyOne` with a value of `success`?
- Am I a `SessionLoginEvent`?
- Are *all* of these conditions true?

Because a UserRegistrationEvent woke up the campaign service and took a snapshot of the `request` object, the campaign action will not be triggered, because the rule requires that *all* of its conditions evaluate to true. The SessionLoginEvent rule is false (because it was the UserRegistrationEvent that woke up the campaign service).

If the rule was defined so that *any* of the conditions evaluating to true would trigger the action (rather than *all* conditions), the campaign action would have fired if the request property evaluated to true.

To use session or request properties to trigger campaign actions, make sure you do the following:

- In the JSP containing the event to be fired, get the request attribute through a variable or set it directly in the JSP.
- In the JSP containing the event to be fired, get/set any event properties you want to use.
- If you want to use session properties to trigger campaign actions, make sure the firing event is in the same session containing the session properties you want to use.

For more information on events and using them in Campaigns, see Chapter 15, "Event and Behavior Tracking," in particular "Writing the Custom Event Class," which contains information on getting the `Request` object with your event.

Working with the Rules Framework

Rules Management forms a key part of the personalization process by prescribing a flexible and powerful mechanism for expressing business rules. The business logic encompassed by these rules allows robust delivery of personalized content marketed specifically to each end user type.

The various components of the Rules Framework are configured with an external configuration file called `rules.properties`. This file resides in the `p13n_util.jar` file (within the `com/bea/p13n/rules` directory) that can be found in the root directory of any WebLogic Portal application. This section explains each of the configuration properties that can be set in this file.

Changes to the `rules.properties` file are only seen by the application in which the file resides. That is, this configuration file is scoped to the application. This makes it possible to configure the Rules Framework differently for different applications.

Validating Rules Expressions

If you want the Rules engine to validate all Rules expressions (both conditions and actions) exactly one time, set `rules.engine.expression.validation` to true. You can set this property to true during development and testing for additional expression validation, as shown in Listing 12-4.

Listing 12-4 Setting the `rules.engine.expression.validation` Property

```
##
# Rules engine expression validation:
#
# If this property is set to true, the rules engine
# will validate expressions the first time they are
# executed.
##
rules.engine.expression.validation=true
```

Rules Engine Error Handling and Reporting

The `rules.engine.throw.expression.exceptions` and `rules.engine.ignorable.exceptions` properties determine the type of exceptions that will be propagated to the user during Rules engine execution.

- To prevent exceptions from being propagated and any condition expression that generates an exception to evaluate to false, set the `rules.engine.throw.expression.exceptions` parameter to `false`.
- To propagate all exceptions to the user—except those listed with the `rules.engine.ignorable.exceptions` parameter—set the `rules.engine.throw.expression.exceptions` parameter to `true`.

Listing 12-5 shows an example of how these parameters are set.

Listing 12-5 Rules Engine Pattern Expression Execution Error Handling

```
##
# Rules engine pattern expression execution error handling:
#
# rules.engine.throw.expression.exceptions
#
# If this property is set to true, pattern expression
# execution exceptions will be thrown. Otherwise, a pattern
# expression exception will cause the pattern condition to
# evaluate to false.
#
# Defaults to true.
#
# rules.engine.throwable.exceptions (list of class names)
#
# If the previous property is set to true, expression exceptions
# with embedded exceptions of a type other than the listed classes
# will be thrown. If no class types are specified, all expression
# exceptions will be thrown.
#
# Defaults to all exception class types.
##
rules.engine.throw.expression.exceptions=true
rules.engine.ignorable.exceptions=java.lang.NullPointerException
```

Personalization with Content Selectors

A content selector is one of several mechanisms that WebLogic Portal provides for retrieving documents from a content management system. By using content selectors, you can personalize a portal or portlet by specifying conditions under which WebLogic Portal retrieves one or more documents. For example, you can personalize a portlet that displays data from a content management system by specifying such information as a date range, the status of the user, and the user's e-mail address. The content selector would only retrieve documents that fit the selection criteria.

Note: Before you can work with content selectors, customer segments must be created. Creating segments is an administrative task and is discussed in the *Administration Guide*. For more information, see "Creating Customer Segments" at http://edocs.bea.com/wlp/docs70/admin/usrgrp.htm#1184110.

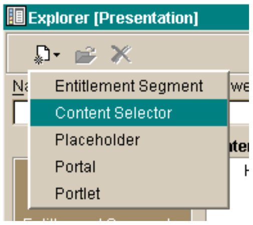
Use the E-Business Control Center to define the conditions that activate a content selector and to define the query the content selector uses to find and retrieve documents. Then, use the content selector JSP tags and a set of other JSP tags to retrieve and display the content targeted by the content selector.

To use the E-Business Control Center to define the conditions that activate a content selector and the query criteria, use this procedure:

1. Open the E-Business Control Center and display the Presentation tab.
2. Click the **Content Selector** icon in the left pane of the Explorer.

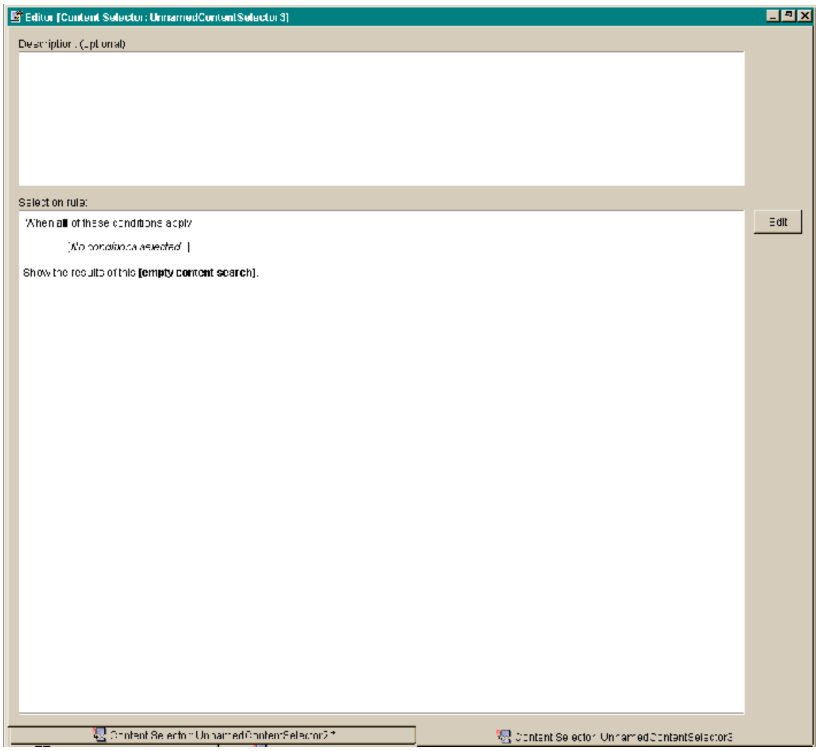
   Any existing content selectors will appear in the right pane of the Explorer.
3. Select **New** to display the New menu and select **Content Selector**.

Figure 12-1 E-Business Control Center New Menu with Content Selector Selected



The Content Selector Editor appears.

Figure 12-2 Content Selector Editor


Editor [Content Selector: UnnamedContentSelector3]
Description (optional)
Selection rule:
Edit
Content Selector: UnnamedContentSelector2 *
Content Selector: UnnamedContentSelector3

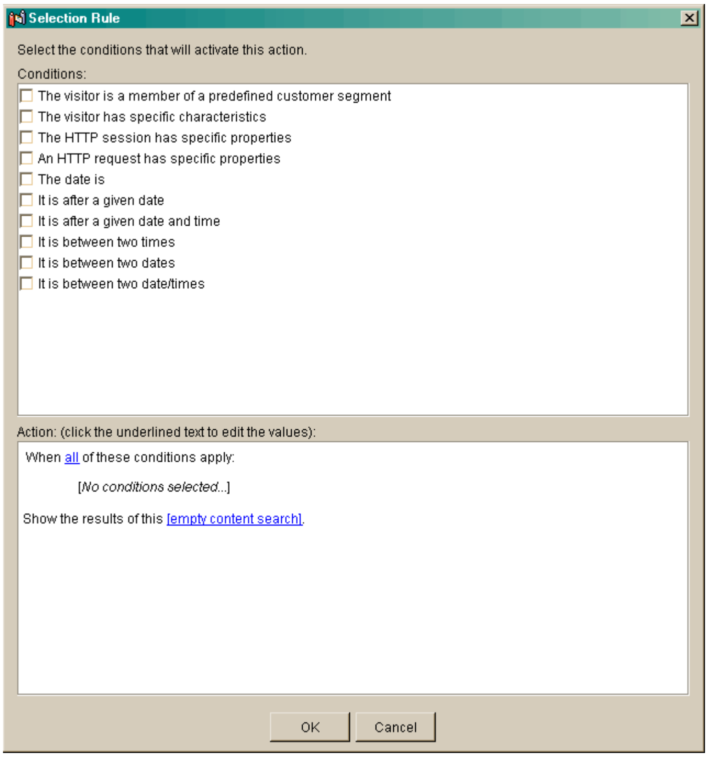
When all of these conditions apply

[No conditions selected]

Show the results of this **[empty content search]**.

4. Double-click anywhere on the **Selection rule** pane to open the Selection Rules editor:

Figure 12-3 Selection Rules Editor



Select the conditions that will activate this action.

Conditions:

- ☐ The visitor is a member of a predefined customer segment
- ☐ The visitor has specific characteristics
- ☐ The HTTP session has specific properties
- ☐ An HTTP request has specific properties
- ☐ The date is
- ☐ It is after a given date
- ☐ It is after a given date and time
- ☐ It is between two times
- ☐ It is between two dates
- ☐ It is between two date/times

Action: (click the underlined text to edit the values):

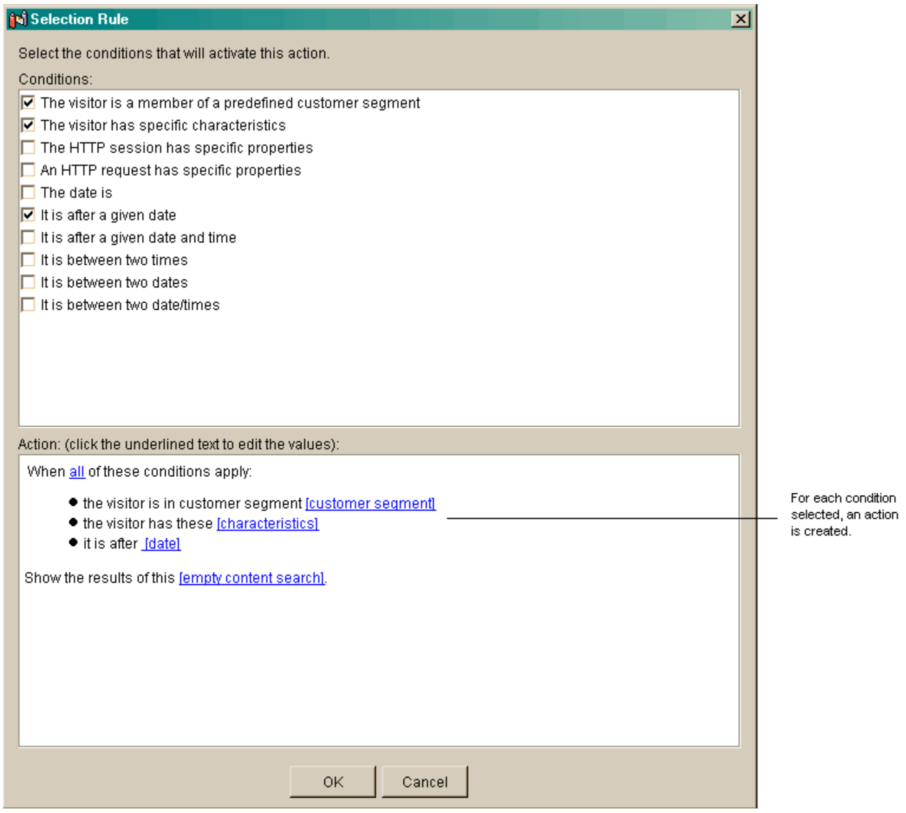
When all of these conditions apply:

[No conditions selected...]

Show the results of this [empty content search].

5. Click the checkbox next to each condition you want to activate the content selector. For each condition selected, a related action is added to the action pane.

Figure 12-4 Selection Rules Editor with Conditions Selected


Selection Rule

Select the conditions that will activate this action.

Conditions:

- [✓] The visitor is a member of a predefined customer segment
- [✓] The visitor has specific characteristics
- [ ] The HTTP session has specific properties
- [ ] An HTTP request has specific properties
- [ ] The date is
- [✓] It is after a given date
- [ ] It is after a given date and time
- [ ] It is between two times
- [ ] It is between two dates
- [ ] It is between two date/times

Action: (click the underlined text to edit the values):

When all of these conditions apply:

- the visitor is in customer segment [customer segment]
- the visitor has these [characteristics]
- it is after [date]

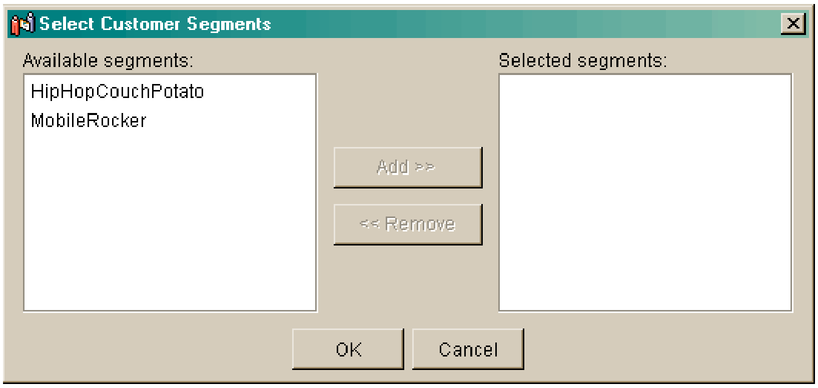
Show the results of this [empty content search].

6. In the Action pane, do the following:

   a. Determine how the conditions will apply. The default value is **all**, which means that all conditions must be true before the content selector is activated. Click the word **all** to toggle the value to **any**, which means that at least one of the conditions must be true to activate the content selector.

   b. Next, set the values for each condition by clicking the underlined text in the condition list; for example, if you selected the condition **The visitor is a member of a predefined customer segment**, the condition **The visitor is in customer segment [customer segment]** appears in the **Action** pane. Click **[customer segment]** to display the Select Customer Segments dialog box.

Figure 12-5 Select Customer Segments Dialog Box



c. Select the customer segments of which the visitor must be a member and click **Add** to move them to the **Selected segments** list. When you've added all of the segments necessary, click **OK**.

d. Repeat step b for each condition selected.

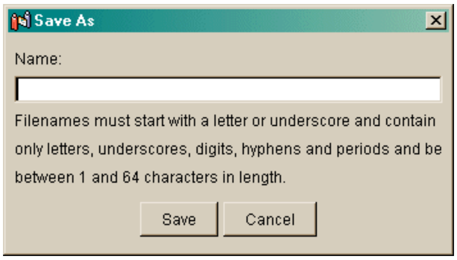
e. When the values for all selected conditions have been set, click **OK**.

The **Selection Rules** dialog box closes.

7. In the E-Business Control Center, open the **File** menu and select **Save as**.

The **Save as** dialog box appears.

Figure 12-6 Save As Dialog Box



Filenames must start with a letter or underscore and contain only letters, underscores, digits, hyphens and periods and be between 1 and 64 characters in length.

8. Type the name you want to call the content selector in **Name** and click **Save**.

The new content selector will appear in the content selector list in the Explorer.

9. Open the **Tools** menu and select **Synchronize**.

The new content selector is ready to use.

To use the content selector features on a given JSP, you must add calls to the content selector JSP tag and a set of associated tags. For more information, please refer to "Using Content-Selector Tags and Associated JSP Tags" on page 8-25.

Using an Edit .jsp to Personalize a Portlet

Visitors can personalize portlets by providing the necessary personalization attributes or preferences to an Edit JSP. An Edit JSP is a JSP that collects personalization data and uses it to render a personalized view of requested date.

Enabling a visitor to personalize a portlet by using an Edit JSP is a two-step process:

- Step 1. Create the Edit JSP
- Step 2. Enable Portlet Editing

Step 1. Create the Edit JSP

Create the Edit JSP (or JSPs, if a Webflow is required) as you would any other JSP. You can name it anything you want and give it any appropriate look and feel. The important features of the JSP are those that allow the visitor to enter and retrieve personalized content.

For example:

- The Edit JSP for a stock quote portlet might have a text box or boxes into which the visitor can enter stock symbols and an **OK** or **Set** button that launches an event that retrieves a quote for the respective symbols.
- The Edit JSP for an email portlet might include edit boxes into which the visitor sets a user name and password for retrieving email from a server, which is also specified on the Edit JSP.
- The Edit JSP for a portlet that displays columnar data might have checkboxes that let the visitor turn columns on or off and specify sorting preference.

As you can see, there are few restrictions as to what you need to include in an Edit JSP, so long as it meets visitor personalization requirements.

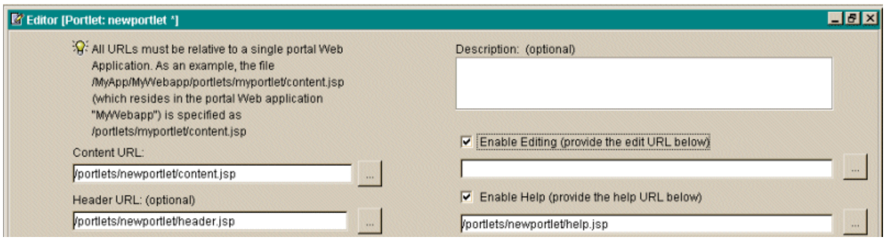
Step 2. Enable Portlet Editing

Next, you need to enable visitors to edit a portlet to add personalization information. To do this, use this procedure:

1. Start the E-Business Control Center. For instructions on starting the E-Business Control Center see "Starting the E-Business Control Center" in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/admintro.htm#1185814.
2. In the Explorer window, click the **Presentation** tab at the bottom of the Explorer window, then click the **Portlets** icon.
3. From the list of portlets in the Explorer, double-click the portlet you for which you want to enable editing.

   The Portlet Editor appears.
4. Select **Enable Editing**, as shown in Figure 12-7.

Figure 12-7 Portlet Editor with Enable Editing Checkbox Selected


Editor [Portlet: newportlet *]

All URLs must be relative to a single portal Web Application. As an example, the file /MyApp/MyWebapp/portlets/myportlet/content.jsp (which resides in the portal Web application "MyWebapp") is specified as /portlets/myportlet/content.jsp

Content URL: /portlets/newportlet/content.jsp

Header URL: (optional) /portlets/newportlet/header.jsp

Description: (optional)

[✓] Enable Editing (provide the edit URL below)

[✓] Enable Help (provide the help URL below) /portlets/newportlet/help.jsp

5. Note that by selecting this checkbox, you also enable the edit box below it. In this box, specify the relative URL of the Edit JSP you created.

For more information on using the Portlet Editor, see "Modifying Portlet Characteristics" in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/frmwork.htm#1199768.

Personalizing a Portal or Portlet by Using Placeholders

Placeholders are devices that represent an area in a portal or portlet to which content is provided when certain criteria are met that define what content can appear. A placeholder is a named entity that contains one or more queries. When a visitor requests a JSP that contains a placeholder tag, the placeholder selects a single query to run—usually based upon established rules or customer properties—and generates the HTML that the browser requires to display the results of the query.

This section includes information on the following subjects:

- How Placeholders are Used
- Placeholder JSP Tag: <ph:placeholder>
- Implementing the Placeholder
- Creating Placeholder Files

How Placeholders are Used

Placeholders are used primarily in campaigns to direct a visitor's attention to programs, merchandise, or other information in which the visitor's behavior has indicated an interest. For example, an online sporting goods store notices that a visitor has purchased a number of fishing lures from a specific manufacturer. A rule exists that tells the portal to display information about discounts available when the visitor has spent more than a certain amount of money on products by that particular manufacturer. The visitor will see that information in the area specified as the placeholder for that content. Another visitor has shown interest in camping equipment (measured by the number of times that visitor has accessed the "Camping" pages for the sporting goods store's catalog). Instead of fishing lure discounts, the camper will see information about camping gear in the area specified as the placeholder.

The above example demonstrates how ad placeholders are used to personalize information for a specific visitor. In the same manner, non-ad placeholders can offer the same level of personalization. For example, a doctor researching drugs at a pharmaceutical company Website shows a tendency for studying a certain family of drugs. The company's portal can track his behavior and display, in a placeholder, information about, or links to, related drugs that he hasn't yet researched.

Placeholders are created by using a JSP tag and defining the placeholder queries in the E-Business Control Center. This section describes how to use those features to set up placeholders to personalize portal and portlet content.

Placeholder JSP Tag: <ph:placeholder>

The placeholder tag `<ph:placeholder>` is a named location on a JSP. The tag identifies the placeholder to the JSP and describes the behavior established for it in the E-Business Control Center. See "Creating Placeholder Files" on page 12-29 for instructions on using the E-Business Control Center to set up the placeholder behavior.

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

The `<ph:placeholder>` tag (Table 12-3) implements a placeholder, which describes the behavior for a location on a JSP page.

Multiple placeholder tags can refer to the same placeholder. Each instance of a placeholder tag invokes its placeholder definition separately. If the placeholder definition specifies multiple queries, each placeholder tag instance can display different ads, even though each instance shares the same definition.

When WebLogic Portal receives a request for a JSP that contains an ad placeholder, the placeholder tag contacts the Ad Service, a session EJB that invokes business logic to determine which ad to display.

Table 12-3 <ph:placeholder>

Tag Attribute	Req'd	Type	Description	R/C
name	Yes	String	A string that refers to a placeholder definition.	R

Table 12-3 <ph:placeholder> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
height	No	int	Specifies the height (in pixels) that the placeholder uses when generating the HTML that the browser requires to display a document. The placeholder uses this value only for content types to which display dimensions apply and only if other attributes have not already defined dimensions for a given document. If you do not specify this value and other attributes have not already been defined, the browser behavior determines the height of the document.	R
width	No	int	Specifies the width (in pixels) that the placeholder uses when generating the HTML that the browser requires to display a document. The placeholder uses this value only for content types to which display dimensions apply and only if other attributes have not already defined dimensions for a given document. If you do not specify this value and other attributes have not already been defined, the browser behavior determines the height of the document.	R

Example

This example displays the ad specified in the `MainPageBanner` placeholder.

```
<%@ taglib uri="ph.tld" prefix="ph" %>
. . .
<ph:placeholder name="/placeholders/MainPageBanner.pla"/>
```

Implementing the Placeholder

To implement a placeholder, use this procedure:

1. Open the JSP template file that will contain the placeholder. JSPs reside in an applicaton folder under the portlets folder for an individual application; for example:

   ```
   <BEA_HOME>\weblogic700\samples\portal\<PORTAL_DOMAIN>\beaApps\
      <PORTAL_APP>\<PORTAL_APP>\portlets\campaigns
   ```

2. Import the tag library by including the following code in the JSP

   ```
   <%@ taglib uri="ph.tld" prefix="ph" %>
   ```

3. Add `<ph:placeholder>` within the JSP element where you want the placeholder to appear. Be sure to specify the placeholder name within the placeholder tag, as shown in Listing 12-6.

Listing 12-6 <ph:placeholder> Tag

```
<table class="homebackground" width="100%" height="100%"
   border="0" cellspacing="0" cellpadding="0">

   <tr>
      <td align="center">
         <ph:placeholder name="PrimaryCampaign"/>
      </td>
   </tr>

</table>
```

Creating Placeholder Files

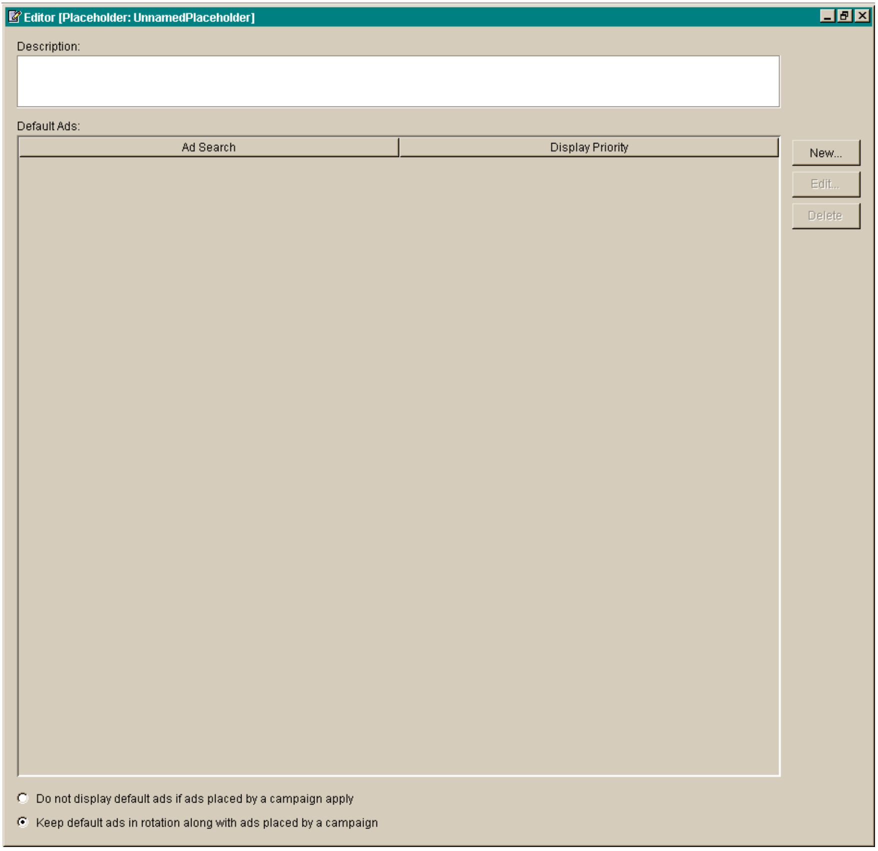
Use the E-Business Control Center to define placeholder files that match the placeholders in your site’s JSPs. The following procedure shows you how to create a placeholder file in the E-Business Control Center and how to set up default—not campaign—queries in that placeholder.

Note: Before beginning this procedure, you must define attributes for the documents in your content management system

To create a placeholder file:

1. Start WebLogic Server and open the the E-Business Control Center.
2. Open the application with which you want to work.
3. Choose **File →New →Presentation →Placeholder**. A new placeholder file opens in the Editor window, as shown in Figure 12-8.

Figure 12-8 Placeholder Editor


Editor [Placeholder: UnamedPlaceholder]

Description:

Default Ads:

Ad Search	Display Priority

○ Do not display default ads if ads placed by a campaign apply

[✓ Keep default ads in rotation along with ads placed by a campaign]

4. Enter a description for the placeholder in the Description area.

5. Click **New** to begin defining a default query. A placeholder file is considered incomplete if it does not have at least one default query (though you can still save the placeholder file).

   Note: Since the content you are trying to access is stored on the server, the Connection Setup window appears. Select an existing connection in the Display Name field, and enter your username and password. You only need to log in once per session when working with placeholders.

   You can create multiple default queries by repeating this step.

   If you do not create ad queries for default ads, the placeholder will display only ads that are generated by campaign queries. If there are no active campaigns, or if an active campaign contains no ad actions within scenarios to trigger an ad for a specific customer, then the placeholder remains empty to customers.

6. To change the priority of a default query, click the Display Priority column for the query and select a priority, as shown in Figure 12-8.

   The Display Priority determines the likelihood that the query runs relative to the priority of any other queries that are in the placeholder.

7. To prevent an ad placeholder from using default aqueries if it also contains campaign ad queries, select the option, **Do not display default ads if ads placed by a campaign apply**.

   If you want the placeholder to choose among default and campaign ad queries, select the option, **Keep default ads in rotation with ads placed by a campaign**. This selection potentially reduces the chance that the placeholder displays a given ad that is part of a campaign.

8. Save and name the placeholder. Be sure to use the name of a placeholder that already exists or will exist on a JSP.

   The new file is displayed in the list of placeholders.

CHAPTER

13 Setting Up Campaign Services

A campaign coordinates several WebLogic Portal services to create and track marketing goals on an e-commerce Web site. For example, your marketing organization can use campaigns to sell 100 ACME saws during the month of June. To reach this goal, Marketing can target advertising, e-mail, and discounted product pricing to customers who match a set of criteria, such as customers who have previously purchased ACME hardware from your site.

Your responsibility in setting up a campaign service is to develop the infrastructure to support the campaigns and modify that infrastructure as individual campaigns require. This activity can include building placeholders for campaigns, specifying display and clickthrough behavior, loading ads into a content management system, creating personalized e-mails for campaigns, and sending bulk mail to prospective and existing customers.

This section contains information on the following subjects:

- What are Campaign Services?
- Building Placeholders for Campaigns
- Using Attributes to Specify Display and Clickthrough Behavior
- Loading Ads Into Your Content Management System
- Creating Personalized E-mails for Campaigns
- Sending Bulk Mail

Note: Campaigns cannot be used with anonymous users.

What are Campaign Services?

Campaigns coordinate the following services:

- Events and Behavior Tracking services identify how a customer interacts with your site. By default WebLogic Portal tracks only a specific set of customer interactions (events), but you can add to this set by customizing the Event Service. For more information on event and behavior tracking, see Chapter 15, "Event and Behavior Tracking."
- Customer Segments categorize customers based on information in a customer's profile and other dynamic data. Each customer must create a profile to log in to your site. The profile includes information that the customer provides, such as shipping addresses, and information that WebLogic Portal provides, such as number of visits and total value of products the customer has purchased on the site. You can create customer segments in the E-Business Control Center.
- Scenarios which trigger actions if a specific event occurs or if a specific customer matches a customer segment. You can create scenarios in the E-Business Control Center.

Scenarios can engage any of the following services:

- *Ad Placeholders*, which query the content management system for an ad and display the query results on the Web site. For example, if a customer logs in and the customer's profile matches the SailingEnthusiast customer segment, then a scenario causes an ad for sailboats to appear in the Web site's top banner.
- *E-mail Service*, which uses a JSP to generate e-mail and provides a utility for sending the e-mail in batches. Because the Mail Service uses a JSP to generate e-mail, you can use JSP tags to personalize the e-mail.

Discounts These offer reduced prices for specific products or product categories. You can creates discount in the E-Business Control Center.

Building Placeholders for Campaigns

The ad placeholder is a container that generates the HTML that the browser requires to display the ad content and places it in the JSP at the location of the placeholder tag. An ad is a document in your content management system that an ad placeholder displays. Ads can be an integral part to a campaign. For example, campaigns can specify as a goal to record a specific number of ad clickthroughs.

Using Attributes to Specify Display and Clickthrough Behavior

You need to define the document attributes in your content management system that ad placeholders use to support the following features:

- Choosing a single document if a query returns multiple documents
- Making an image ad clickable
- Supplying movie preferences for a Shockwave file

For information about associating attributes with documents, refer to the documentation for your content management system. If you use the reference content management system supplied by BEA, refer to "Loading Ads into the Reference Content Management System" on page 13-4. Valid attributes are listed in Table 13-1, Table 13-2, and Table 13-3.

Loading Ads Into Your Content Management System

The queries you can define for ad placeholders search through the attributes that you attach to the documents in your content management system. WebLogic Portal places no restrictions on the set of attributes that you use to describe your ads. For example, you can create attributes that describe the name of the product that the document advertises, the name of the ad sponsor, and a product category that matches the categories in your e-commerce product catalog.

The method of loading ads into a content management system is dictated by the CMS.

- If you use the reference content management system supplied by BEA, use the procedures in this section.
- If you are using a third-party CMS, follow the load instructions provided by the vendor.

Loading Ads into the Reference Content Management System

WebLogic Portal provides a content management system for sites with limited content-management needs. If you use the reference content management system, you must load ads and ad attributes at the same time. You cannot add attributes to documents that have already been loaded.

When you install WebLogic Portal, the reference content management system (which uses the sample PointBase database) already contains a set of sample ads.

To load ads and ad attributes into the reference content management system, you must do the following:

- Step 1. Set Up Attributes in HTML Documents
- Step 2. Set Up Attribute Files for Image and Shockwave Documents
- Step 3. Move Files Into the dmsBase/Ads Directory Tree

- Step 4. Run the loadads Script

Step 1. Set Up Attributes in HTML Documents

For ads that contain only HTML, you must place document attributes in `<META>` tags within a document's `<HEAD>` element. Use the following syntax in the `<META>` tag:

```
<META name="attribute-name" content="attribute-value">
```

Use a separate `<META>` tag for each document attribute. For example:

```
<META name="attribute1-name" content="attribute1-value">
<META name="attribute2-name" content="attribute2-value">
<META name="attribute3-name" content="attribute3-value">
```

Listing 13-1 shows an HTML file that contains a simple ad with several attributes.

Listing 13-1 Attributes for an HTML Ad

```
<HTML>
<HEAD>

<META name="adWeight" content="3">
<META name="productCategory" content="hardware">
<META name="productSubCategory" content="electic drill">
<META name="productName" content="Super Drill">
<META name="Manufacturer" content="ACME">

</HEAD>

<BODY>

<P>Buy our Super Drill. It'll get the job done!</P>

</BODY>

</HTML>
```

Table 13-1 describes the `adWeight` attribute, which you can associate with XHTML, image, and Shockwave documents.

Table 13-1 Attributes for All Document Types

Attribute Name	Value Type	Description and Recommendations
`adWeight`	Integer	Provides an integer that is used to select a document if a query returns multiple documents. Assign a high number to ads that you want to have a greater chance of being selected. The default value for this attribute is `1`. **Note:** In the E-Business Control Center, you can assign a priority to a query for a scenario action. The priority, which bears no relation to the `adWeight` attribute, gives a greater or lesser chance that a placeholder runs a query. The `adWeight` attribute is used to choose an ad after a query has run.

Step 2. Set Up Attribute Files for Image and Shockwave Documents

For ads that are images or Shockwave movies, you must place attributes in a separate file. Each image or Shockwave file must be accompanied by a separate file that is named with the following convention:

`filename.extension``.md.properties`

Both files must be located in the same directory.

For example, for an image file named `superDrill.jpg`, you must place attributes in a file named `superDrill.jpg.md.properties`.

Within the *`filename.extension`*`.md.properties` file, use the following syntax to express attributes and values:

`attribute-name=attribute-value`

Listing 13-2 shows an example file that contains attributes for an image ad.

Listing 13-2 Syntax for the Attributes File

```
adWeight=5
adTargetUrl=AcmeAds/saws.jpg
adAltText=Buy ACME and save!
```

```
productCategory=hardware
productSubCategory=electic drill
productName=Super Drill
Manufacturer=ACME
```

Other Image File Attributes

Table 13-2 describes other attributes, in addition to `adWeight`, that you can associate with image files.

Table 13-2 Attributes for Image Files

Attribute Name	Value Type	Description and Recommendations
`adTargetUrl`	String	Makes an image clickable and provides a target for the clickthrough, expressed as a URL. The Event Service records the clickthrough. Use either `adTargetUrl`, `adTargetContent`, or `adMapName`, depending on how you want to identify the destination of the ad clickthrough.
`adTargetContent`	String	Makes an image clickable and provides a target for the clickthrough, expressed as the content management system's content ID. The Event Service records the clickthrough. Use either `adTargetUrl`, `adTargetContent`, or `adMapName`, depending on how you want to identify the destination of the ad clickthrough.
`adMapName`	String	Makes an image clickable, using an image map to specify one or more targets. The value for this attribute is used in two locations: ■ In the anchor tag that makes the image clickable, `<a href=`*`value`*`> <img> </a>` ■ In the map definition, `<map name=`*`value`*`>` Use either `adTargetUrl`, `adTargetContent`, or `adMapName`, depending on how you want to identify the destination of the ad clickthrough. If you specify a value for `adMapName`, you must also specify a value for `adMap`.

Table 13-2 Attributes for Image Files (Continued)

Attribute Name	Value Type	Description and Recommendations
`adMap`	String	Supplies the XHTML definition of an image map. If you specify a value for `adMap`, you must also specify a value for `adMapName`.
`adWinTarget`	String	Displays the target in a new pop-up window, using JavaScript to define the pop-up. The only value supported for this attribute is `newwindow`.
`adWinClose`	String	Specifies the name of a link that closes a pop-up window. The link appears at the end of the window content. For example, if you provide "Close this window" as the value for this attribute, then "Close this window" appears as a hyperlink in the last line of the pop-up window. If a customer clicks the link, the window closes.
`adAltText`	String	Specifies a text string for the `alt` attribute of the `<img>` tag. If you do not include this attribute, the `<img>` tag does not specify an `alt` attribute.
`adBorder`	Integer	Specifies the value for the `border` attribute of the `<img>` tag. If you do not include this attribute, the `border` attribute is given a value of `"0"`.

Other Shockwave Attributes

Table 13-3 describes other attributes, in addition to `adWeight`, that you can associate with Shockwave files. Ad placeholders and the `<ad:adTarget>` tag format these values as attributes of the `<OBJECT>` tag, which Internet Explorer on Windows uses to display the file, and the `<EMBED>` tag, which browsers that support the Netscape-compatible plug-in use to display the file.

For more information about these attributes, refer to your Shockwave developer documentation.

Table 13-3 Attributes for Shockwave Files

Attribute Name	Value Type	Description and Recommendations
`swfLoop`	String	Specifies whether the movie repeats indefinitely (`true`) or stops when it reaches the last frame (`false`). Valid values are `true` or `false`. If you do not define this attribute, the default value is `true`.
`swfQuality`	String	Determines the quality of visual image. Lower qualities can result in faster playback times, depending on the client's Internet connection. Valid values are `low`, `high`, `autolow`, `autohigh`, `best`.
`swfPlay`	String	Specifies whether the movie begins playing immediately on loading in the browser. Valid values are `true` or `false`. If you do not define this attribute, the default value is `true`.
`swfBGColor`	String	Specifies the background color of the movie. This attribute does not affect the background color of the HTML page. Valid value syntax is `#`*`RRGGBB`*.
`swfScale`	String	Determines the dimensions of the movie in relation to the area that the HTML page defines for the movie. Valid values are `showall`, `noborder`, `exact fit`.
`swfAlign`	String	Determines whether the movie aligns with the center, left, top, right, or bottom of the browser window. If you do not specify a value, the movie is aligned in the center of the browser. Valid values are `l`, `t`, `r`, `b`.
`swfSAlign`	String	Determines the movie's alignment in relation to the browser window. Valid values are `l`, `t`, `r`, `b`, `tl`, `tr`, `bl`, `br`.
`swfBase`	String	Specifies the directory or URL used to resolve relative pathnames in the movie. Valid values are `. (period)`, *`directory-name`*, *`URL`*.

Table 13-3 Attributes for Shockwave Files (Continued)

Attribute Name	Value Type	Description and Recommendations
`swfMenu`	String	Determines whether the movie player displays the full menu. Valid values are `true` or `false`.

Step 3. Move Files Into the dmsBase/Ads Directory Tree

To make the ads available to the campaign, place all HTML, image, and Shockwave files, and all attributes files into the Ads directory, which is located at the following path:

```
<BEA_HOME>/user_projects/<YOUR-APPLICATIONDOMAIN>/dmsBase/Ads
```

(where `<BEA_HOME>` is the directory in which you installed BEA WebLogic Platform and where `<YOUR-APPLICATIONDOMAIN>` is the directory of the particular domain).

You can place documents in subdirectories of the `Ads` directory, although the reference content management system does not use the subdirectories to organize documents.

If you use subdirectories to manage your source files, you must place the attributes files in the same directory as the files that they describe. For example, `superDrill.jpg` and `superDrill.jpg.md.properties` must be in the same directory.

Step 4. Run the loadads Script

The `loadads` script (`loadad.bat` for Windows user; `loadad.sh` for Unix users) runs the BulkLoader to load documents from the `dmsBase/Ads` directory to the content management system. It also attaches attributes to the documents.

To run `loadads`, do one of the following:

- Type `loadads` at the command line (or **Start** → **Run** in Windows NT or 2000). Be sure you are in the `<BEA_HOME>/user_projects/<YOUR-APPLICATIONDOMAIN>/dmsBase` directory.

OR

- Open Windows Explorer, locate `loadads` in the `<BEA_HOME>/user_projects/<YOUR-APPLICATIONDOMAIN>/dmsBase` directory, and double-click it in the file list.

For more information on running the BulkLoader, please see "Adding Content by Using the Bulk Loader" on page 8-1.

Creating Personalized E-mails for Campaigns

The E-mail service uses a JSP to generate e-mail and provides a utility for sending the e-mail in batches. Because the Mail service uses a JSP to generate e-mail, you can use JSP tags to personalize the e-mail. This section shows you how to create personalized e-mails for campaigns by following these steps:

- Step 1. Configure the E-mail Properties
- Step 2. Find Names of User Properties
- Step 3. Create E-mail JSPs

Step 1. Configure the E-mail Properties

Before a campaign can send e-mail, you must configure properties that the Campaign Service uses to send and receive mail. In a clustered environment, WebLogic Server propagates these properties to each node in the cluster.

To configure mail-related properties, do the following:

1. From the E-Business Control Center, open your application.
2. In the E-Business Control Center Explorer window, click the **Site Infrastructure** tab.
3. Click **User Profiles** and find the following:
   - The name of the property set and the property that defines customer e-mail addresses.
   - The name of the property set and the property that records a customer's preference for receiving campaign-related e-mail. The reference applications

store this preference in the Demographics property set in the `Email_Opt_In` property.

Step 2. Find Names of User Properties

1. Start your server and access the WebLogic Server Administration Console for the domain.
2. In the left pane of the WebLogic Server Administration Console, click **Deployments→ Applications → myApplication → Service Configuration → Campaign Service**.
3. On the Campaign Service Page, click the **Mail Action** tab.
4. On the **Mail Action** tab, enter the following values.

Table 13-4 Mail Action Tab Values

In this box...	Enter this value...
Default From Email Address	The default address that receives any replies from e-mail that the campaign sends. In a standard mail header, this is the From address. Each campaign scenario can specify its own From address that overrides this default property.
Email Address Property Name	The name of the property that contains customer e-mail addresses.
Property Set Name Containing Email Address Property	The name of the property set that contains customer e-mail properties.
Email Opt In Property Name	The name of the property that specifies whether customers want to receive campaign-related email. The reference applications store this preference in the Demographics property set in the Email_Opt_In property.
Property Set Name Containing Opt In Property	The name of the property set that contains the customer's opt-in property.

5. Click **Apply**.

All e-mail that the Campaign Service generates will now use these settings.

6. Configure the default SMTP host name for the Mail Service by clicking **Mail Service** in the left pane.

   Note: Any changes that you make on the Mail Service page affects all e-mails that you send using WebLogic Portal (whether or not they are generated by the Campaign Service).

Step 3. Create E-mail JSPs

The Mail Service requires that you place the content and formatting of your e-mails in a JSP file. In this JSP, you can use any of the JSP tags and APIs that are available to other JSPs in WebLogic Portal.

This step describes the following:

- E-mail Parameters
- Disabling Session Generation
- Sample E-mail JSP
- Saving E-Mail JSPs

E-mail Parameters

When a scenario action requests an e-mail JSP, it passes a `userid` parameter, which specifies the login name of the customer who triggered the scenario action. By using the `request.getParameter()` method, you can retrieve the user ID and pass it to JSP tags in the e-mail JSP.

In addition, the scenario passes the following parameters (you can also pass these parameters to JSP tags in the e-mail JSP):

- `scenarioId`, which specifies the ID of the scenario that triggered the e-mail.
- `scenarioName`, which specifies the name of the scenario that triggered the e-mail.
- `containerId`, which specifies the ID of the campaign to which the scenario belongs.

- `containerName`, which specifies the name of the campaign to which the scenario belongs.

Disabling Session Generation

The Java class that the Campaign Service uses to generate email from a JSP, `InternalRequestDispatcher`, also generates an HTTPSession object. Usually, generating this HTTPSession from an email JSP is extraneous because your application already generates an HTTPSession object when a customer accesses your site.

To disable the generation of an extraneous HTTPSession, add the following directive to the beginning of the JSPs that you use to generate email for campaigns:

```
<%@ page session="false" %>
```

Adding this directive is necessary only if your application generates HTTPSession objects when customers access your site (or log in) and only for email that is generated via the InternalRequestDispatcher.

Sample E-mail JSP

Listing 13-3 shows the e-mail JSP that is part of the sample Web application. The file is located at `<BEA_HOME>/weblogic700/samples/portal/wlcsDomain/beaApps/wlcsApp/wlcs/campaigns/emails/`

Listing 13-3 Sample E-mail JSP

```
Sample1.jsp
<%@ page session="false" %>
<%@ page contentType="text/plain" %>

(This sample e-mail was automatically sent out as part of a sample
campaign that you triggered while registering as a new user on the
BEA Commerce Templates.)

-------------------------------

Hello <%= request.getParameter("userId") %>,

Thank you for taking the time to become a registered member of our
site. We hope you took advantage of your $10 discount on a purchase
of $50 or more after you registered!
```

```
In addition, your registration entitles you to premium services
including:

   **Special "Members Only" discounts

   **Advance notice of new product releases

   **A personalized customer experience customized to
     your specific interests

Thanks again for becoming a registered member.

Best Regards
```

Saving E-Mail JSPs

You must save e-mail JSPs in a specific directory within a Web application so that E-Business Control Center user can browse and select the e-mail for a campaign.

By default, the directory is `myApp/`*`myWebApp`*`/campaigns/emails`.

(Where *`myWebApp`* is the name of a Web application containing the campaign)

For example, the Web application `wlcs` provides a sample e-mail in the following directory:

```
<BEA_HOME>/weblogic700/samples/portal/wlcsDomain/beaApps/wlcsApp/
   wlcs/campaigns/emails/Sample1.jsp
```

To choose this e-mail as part of a scenario action, do the following:

1. Open the wlcsApp application in the E-Business Control Center.
2. While creating an E-mail action, to browse through all e-mail JSPs that have been saved in `<BEA_HOME>/weblogic700/samples/portal/wlcsDomain/beaApps/wlcsApp/wlcs/campaigns/emails` and select `Sample1.jsp` for the scenario.

To change the default location in which you save e-mail JSPs, do the following:

1. From the WebLogic Server Administration Console, in the left pane, click **Deployments → Applications → myApplication → Service Configuration → Campaign Service**.
2. On the Campaign Service page, click the **Configuration** tab.

3. In the **Base Directory for Email Browsing** box, enter a pathname that is relative to the root directory of a Web application.

Sending Bulk Mail

You must periodically use a command to send the batched e-mail that the JSPs store in the WebLogic Portal data repository. You can also use `cron` or any other scheduler that your operating system supports to issue the send-mail command.

This section includes information on the following subsections:

- Sending Mail from a Remote Host or in a Clustered Environment
- Sending Bulk E-mail
- Scheduling Bulk E-mail Delivery

Sending Mail from a Remote Host or in a Clustered Environment

The send-mail wrapper script specifies the name and listen port of the WebLogic Portal host that processes the send-mail request. By default, the wrapper script specifies `localhost:7501` for the hostname and listen port. However, `localhost:7501` is valid only when you run the script while logged in to a WebLogic Portal host in a single-node environment (and only if you did not modify the default listen port).

Before you use the send-mail script from any other configuration, you must modify the script by doing one of the following tasks:

- Modify the Send-Mail Script to Work from a Remote Host
- Modify the Send-Mail Script to Work in a Clustered Environment

Modify the Send-Mail Script to Work from a Remote Host

If you want to run the send-mail script from a remote host (that is, a computer that is not a WebLogic Portal host), do the following:

1. Open the following file in a text editor:

   `<BEA_HOME>\weblogic700\portal\bin\win32\mailmanager.bat` (Windows)

   `<BEA_HOME>\weblogic700\portal\bin\win32\mailmanager.sh` (Unix)

2. In the mailmanager script, in the `SET HOST= line`, replace localhost with the name of a WebLogic Portal host.

3. If the host uses a listen port other than 7501, in the `SET PORT=` line, replace `7501` with the correct listen port.

4. Save the mailmanager script.

Modify the Send-Mail Script to Work in a Clustered Environment

If you work in a clustered environment, you must modify the send-mail wrapper script to specify the name of a host in the cluster. The default localhost value is not valid for the Mail Service in a clustered environment.

To use the send-mail script in a clustered environment, do the following on each host from which you want to run the script:

1. Open the following file in a text editor:

   `<BEA_HOME>\weblogic700\portal\bin\win32\mailmanager.bat` (Windows)

   `<BEA_HOME>\weblogic700\portal\bin\win32\mailmanager.sh` (Unix)

2. In the mailmanager script, in the `SET HOST=` line, replace localhost with the name of a WebLogic Portal host. Because each host in a cluster can access the data repository that stores the e-mail messages, you can specify the name of any host in the cluster.

3. If the host uses a listen port other than 7501, in the `SET PORT=` line, replace 7501 with the correct listen port.

4. Save the mailmanager script.

Sending Bulk E-mail

To send bulk e-mail, do the following from a shell that is logged in to a WebLogic Portal host:

1. To determine the names and contents of the e-mail batches in the data repository, enter the following command:

   `mailmanager.bat` *`appName`* `list` (Windows)

   `mailmanager.sh` *`appName`* `list` (Unix)

   where *`appName`* is the name of the enterprise application that generated the e-mail batch. The command prints to standard out. You can use shell commands to direct the output to files.

2. To send a batch and remove it from the data repository, enter the following command:

   `mailmanager.bat appName send-delete batch-name` (Windows)

   `mailmanager.sh appName send-delete batch-name` (Unix)

Scheduling Bulk E-mail Delivery

You can use a scheduling utility to send the e-mail batches in the data repository. Because you must specify the name of a batch when you use the `mailmanager` command to send mail, you must schedule sending mail for each campaign scenario separately. The name of a batch corresponds to the scenario's container ID. For information about the container ID, refer to "E-mail Parameters" on page 13-13.

For information in using a scheduling utility, refer to the documentation for your operating system.

Deleting E-mail Batches

You can delete e-mail batches as you send them (as described in Sending Bulk E-mail). You can also do the following to delete e-mail batches:

1. Determine the names and contents of the e-mail batches in the data repository by entering the following command:

   `mailmanager.bat` *`appName`* `list` (Windows)

   `mailmanager.sh` *`appName`* `list` (Unix)

   where *`appName`* is the name of the enterprise application that generated the e-mail batch. The command prints to standard out. You can use shell commands to direct the output to files.

2. Delete a batch by entering the following command:

   `mailmanager.bat` *`appName`* `delete batch-name` (Windows)

   `mailmanager.bat` *`appName`* `delete batch-name` (Unix)

14 Setting Up Commerce Services

Among the important commerce services available with WebLogic Portal are those that pertain to such business transaction services as taxation and payment and the product catalog service. Development tasks associated with these services include integrating both local and third-party taxation services and integrating local and third-party payment services. For the product catalog service, development tasks include loading data into the catalog database and creating and enhancing a custom catalog service.

This section includes information on the following subjects:

- Integrating a Portal with Business Transaction Services
- Supporting a Product Catalog

Integrating a Portal with Business Transaction Services

WebLogic Portal can be integrated with such business transaction services as taxation and payment services. Adding these services to a portal extends the functionality of the portal by allowing it to leverage external services for use locally. Integrating these services is a development function, requiring you to update specific EJBs in the enterprise application and URLs specific configuration files. Those modifications are described in this document.

This section includes information on the following subjects:

Integrating with a Taxation Service

The Tax Web service installed with WebLogic Portal provides a default framework for handling tax calculations on transactions received from the default `TaxCalculator` EJB. The business methods implement a standard workflow that is associated with the completion of order taxation. (The Tax Web service is itself a stateless session EJB wrapped in code that makes it a Web service.)

Integrating your enterprise applications with the Tax Web service involves modifying either the `TaxCalculator` EJB or the Tax Web service, depending on who will host the Web service: your organization or the third-party tax calculation vendor.

Important Notice The default Tax Web service that ships with WebLogic Portal automatically applies a 5% tax to an order. This default application of taxes is not designed for production use. You must integrate with your third-party vendor's tax service to calculate taxes properly.

If the Third-Party Vendor Hosts the Web Service

If the third-party vendor hosts the Tax Web service, the vendor will integrate the Web service with their program's API, and modify the `TaxWebService` EJB inside the Web service to translate the SOAP calls—the SOAP calls your enterprise application's `TaxCalculator` EJB sends it to the Web service—into messages their API can understand, and to create proper return SOAP calls to your `TaxCalculator` EJB.

To connect to the vendor-hosted Web service, use this procedure:

1. If the vendor has modified any of `AVS*.class` or `Tax*.class` files in the Web service's `tax.jar` file, duplicate those modifications in your enterprise application. You can find the source code for these classes in:

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\tax\
   ```

2. Compile the source files either by running `javac` from a command line or as directed by your Java editor.
3. Make any vendor-required modifications to the `TaxCalculator` EJB in your enterprise application so that it makes appropriate SOAP calls to the vendor's `TaxWebService` EJB. You can find the source code for the TaxCalculator EJB in:

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\tax\
   ```

4. Compile the source file either by running `javac` from a command line or as directed by your Java editor.
5. After you compile your source code, add the class files to the `wlcsSamples.jar` in your enterprise application folder. When you add them to the `.jar`, maintain their relative directory structure.
6. Run the EJB compiler (`ejbc`) on the `wlcsSample.jar` file.
7. In the `application-config.xml` file in the Meta-inf subdirectory of your application, locate the `<TaxServiceClient>` element, and modify the URL in the `TaxCalculatorWSDL` attribute to connect to the `TaxWebService WSDL` file on the vendor's server.

At startup, WebLogic Server reads the `application-config.xml` file, so it knows where to find the Web service.

If Your Organization Hosts the Web Service

If your organization hosts the Tax Web service, deploy the Web service on a separate Java Virtual Machine (JVM) than what your enterprise applications are running on. This way, if the Web service goes down and freezes the JVM it is running on, your enterprise application's JVM will continue to run.

To connect to a Tax Web service hosted by your organization, use this procedure:

1. Obtain your third-party vendor's tax calculation product API.

2. Modify the TaxWebService EJB (the Web service's EJB) so that it translates SOAP calls into the language of the third-party product's API. You can find the source code for the TaxWebService EJB in the following directory:

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\tax\
   ```

3. Compile the source file either by running `javac` from a command line or as directed by your Java editor.
4. After you have compiled the source code, add the class file to `tax.jar` in the `taxWSApp` directory. When you add the file to the `.jar`, maintain its relative directory structure.
5. Use the Web service generator (`servicegen`) on the `tax.jar` file to build a file called `tax-webservice.war`.

   For information on using wsgen, see "Programming WebLogic Server Web Services" at http://edocs.bea.com/wls/docs70/webserv/index.html.

6. Make any necessary modifications to the `TaxCalculator` EJB in your enterprise application so that it makes appropriate SOAP calls to the `TaxWebService` EJB. You can find the source code for the `TaxCalculator` EJB in:

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\tax\
   ```

7. Compile the source file either by running `javac` from a command line or as directed by your Java editor.
8. After you compile your source code, add the class file to wlcsSamples.jar in your enterprise application's root folder. When you add the file to the `.jar`, maintain its relative directory structure.
9. Run the EJB compiler (`ejbc`) on the `wlcsSample.jar` file.
10. In the `application-config.xml` file in the `META-INF` subdirectory of your application, locate the `<TaxServiceClient>` element, and modify the URL in the `TaxCalculatorWSDL` attribute to connect to the `TaxWebService WSDL` file on your Web service's server.

At startup, WebLogic Server reads the `application-config.xml` file, so it knows where to find the Web service.

Integrating with a Payment Service

The Credit Card Web service that is installed with WebLogic Portal provides a default framework for handling authorization, capture, and settlement of credit card transactions received from the default CreditCardService EJB. The business methods implement a standard workflow that is associated with the completion of credit card transactions. The current state of the transaction is maintained and each action is logged. (The Credit Card Web service is itself a stateless session EJB wrapped in code that makes it a Web service.)

Integrating your enterprise applications with the Payment Web service involves modifying either the CreditCardService EJB or the Credit Card Web service, depending on who will host the Web service: your organization or the third-party payment vendor.

Important Notice The default Payment Web service that ships with WebLogic Portal always sends payment information through without any errors, as if it were connected to and approved by a third-party payment service. This default processing of payment is not designed for production use. You must integrate with your third-party vendor's payment service to process payment correctly.

If the Third-Party Vendor Hosts the Web Service

If the third-party vendor hosts the Credit Card Web service, the vendor will integrate the Web service with their program's API, and modify the `CreditCardWebService` EJB inside the Web service to translate the SOAP calls—the SOAP calls your enterprise application's CreditCardService EJB sends to the Web service—into messages their API can understand, and to create proper return SOAP calls to your CreditCardService EJB.

To connect to the vendor-hosted Credit Card Web service, use this procedure:

1. If the vendor has modified any of `PS*.class` files in the Web service's `payment.jar` file, duplicate those modifications in your enterprise application. You can find the source code for these classes in:

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\payment
   ```

2. Compile the source files either by running `javac` from a command line or as directed by your Java editor.

3. Make any vendor-required modifications to the `CreditCardService` EJB in your enterprise application so that it makes appropriate SOAP calls to the vendor's `CreditCardWebService` EJB. You can find the source code for the `CreditCardService` EJB in:

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\payment
   ```

4. Compile the source file either by running `javac` from a command line or as directed by your Java editor.
5. After you compile your source code, add the class files to the `wlcsSamples.jar` in your enterprise application folder. When you add them to the `.jar`, maintain their relative directory structure.
6. Run the EJB compiler (`ejbc`) on the `wlcsSample.jar` file.
7. In the `application-config.xml` file in the Meta-inf subdirectory of your application, locate the `<PaymentServiceClient>` element, and modify the URL in the `PaymentWebServiceWSDL` attribute to connect to the `CreditCardWebService WSDL` file on the vendor's server.

At startup, WebLogic Server reads the `application-config.xml` file, so it knows where to find the Web service.

Important Security Information

Since a Web services is SOAP (that is, XML over HTTP), you are passing credit card information over the Web in plain text. Hackers listening in can track this information for nefarious purposes. BEA strongly recommends that you set up a dedicated line to the credit card processing provider and add a Secure Socket Layer (SSL) to the HTTP communication taking place. You should also choose a credit card provider that encrypts all communications.

If Your Organization Hosts the Web Service

If your organization hosts the Credit Card Web service, we strongly recommend that you deploy the Web service on a separate Java Virtual Machine (JVM) than what your enterprise applications are running on. This way, if the Web service goes down and freezes the JVM it is running on, your enterprise application's JVM will continue to run.

To connect to a Credit Card Web service hosted by your organization, use this procedure:

1. Obtain your third-party vendor's payment product API.
2. Modify the `CreditCardWebService` EJB (the Web service EJB) so that it translates SOAP calls into the language of the third-party product's API. You can find the source code for the `CreditCardWebService` EJB in the following directory:

   ```
   <BEA_HOME>\Weblogic700\samples\samples\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\payment
   ```
3. Compile the source file either by running `javac` from a command line or as directed by your Java editor.
4. After you have compiled the source code, add the class file to `payment.jar` in the `paymentWSApp` folder. When you add the file to the .jar, maintain its relative directory structure.
5. Use the Web service generator (`servicegen`) on the payment `.jar` file to build a file called `payment-webservice.war`.

   For information on using `servicegen`, see "Programming WebLogic Server Web Services" at http://edocs.bea.com/wls/docs70/webserv/index.html.
6. Make any necessary modifications to the `CreditCardService` EJB in your enterprise application so that it makes appropriate SOAP calls to the `CreditCardWebService` EJB. You can find the source code for the `CreditCardService` EJB in:

   ```
   <BEA_HOME>\Weblogic700\samples\samples\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\payment
   ```
7. Compile the source file either by running `javac` from a command line or as directed by your Java editor.
8. After you compile your source code, add the class file to `wlcsSamples.jar` in your enterprise application's root directory. When you add the file to the JAR, maintain its relative directory structure.
9. Run the EJB compiler (ejbc) on the `wlcsSample.jar` file.

10. In the application-config.xml file in the `META-INF` subdirectory of your application, locate the `<PaymentServiceClient>` element, and modify the URL in the `PaymentWebServiceWSDL` attribute to connect to the CreditCardWebService `WSDL` file on the your Web service's server.

At startup, WebLogic Server reads the application-config.xml file, so it knows where to find the Web service.

Guidelines for Modifying the Credit Card Web Service EJB

The Payment service EJB is a stateless session bean that provides services related to the authorization, capture, and settlement of credit card transactions. The Credit Card Web service EJB serves as an interface behind which integrations with various payment solutions can be implemented. The current state of each transaction is maintained and each action is journaled. General characteristics of transactions are described in the following list:

- Each transaction is initiated with a request to authorize. This authorization generally results in the creation of a persistent PaymentTransaction. The state of the payment and the key for that PaymentTransaction is returned in a TransactionResponse as well as service specific information. A handle for that PaymentTransaction can be obtained from the TransactionResponse.
- In the event that the initial authorization fails due to a failure to connect to the payment authorization service, it is possible to retry the authorization using the reauthorize method.
- An authorized transaction can be captured or settled depending on how the service is configured.

  By law you can only capture a transaction if the goods have been shipped. For example, if you buy a book online from a vendor and that book will take two days to ship, the vendor can only authorize the transaction but not capture it. Two days later, when the product ships, the vendor can capture the transaction. However, if you are buying software online and downloading it immediately, the vendor can authorize and capture the transaction then and there, since the order and the actual shipment download seconds apart.
- An entire transaction can be completed in a single AuthorizeAndCapture.
- Sites that experience a high volume of traffic should run the authorize or capture process offline. This process usually takes 3-8 seconds, and if you have thousands of users, this could slow down your site. If a site does run the

authorize or capture process offline, it should use separate machines for this process to ensure processing integrity.

Supporting a Product Catalog

This section describes the development tasks associated with supporting a product catalog. Some of the tasks discussed below will apply to product catalogs built with resources supplied by BEA (see "Creating and Administering a Catalog" in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/../admin/commerce.htm#1167188). Others apply to a custom catalog service that you can build by implementing the appropriate Stateless Session EJB service API. In addition, you will see how to display your catalog by using JSPs, and integrate a catalog service with a catalog cache.

This section includes information on the following subjects:

- Loading Your Product Data Into the Product Catalog Database Schema
- Showing a Catalog in a JSP
- Hooking Up a Catalog to a Shopping Cart
- Integrating Services With the Catalog Cache

Loading Your Product Data Into the Product Catalog Database Schema

The most important development tasks is to get information about your products into a form that WebLogic can understand. Use the DBLoader program to do this. The DBLoader lets you load, all at once, any data into any table in a database.

Creating your product catalog by using the DBLoader requires these steps:

- Step 1: Prepare to Use DBLoader
- Step 2: Edit the databaseload.properties File

- Step 3: Load Data by Running the DBLoader Program
- Step 4: Troubleshoot Using the DBLoader Log Files

Step 1: Prepare to Use DBLoader

Before you can add your product information to the database, consider the following database issues, and prepare your input files.

- Review Important Database Considerations
- Prepare Product Information Input Files to Load Into the Product Catalog Schema

Review Important Database Considerations

Some important database considerations that you should keep in mind while using the DBLoader program are:

Consider Referential Integrity and Constraints The schema for the Product Catalog enforces data and referential integrity between tables with the use of constraints. For example, the primary key constraint on `WLCS_PRODUCT` and `WLCS_CATEGORY`, or the foreign key constraint on `WLCS_PRODUCT_CATEGORY`.

Primary keys and unique indexes prevent the possibility of placing duplicate entries in the table. Foreign key constraints ensure referential integrity by making certain that the parent key already exists before allowing the child record to be written to the database.

Note: For every `WLCS_PRODUCT` and `WLCS_CATEGORY` table entry, a corresponding entry in the `CATALOG_ENTITY` table must also be made.

Consider Strings in Java All Strings in Java are represented as a series of Unicode 2.0 characters. Unicode 2.0 is a 16-bit character encoding that supports the world's major languages. Therefore, when reading text into and writing text out of the JVM, an encoding scheme must be used to convert the "native" encoding used by the operating system to or from Unicode 2.0. Data in text files is automatically converted to Unicode 2.0 when its encoding matches the default file encoding of the Java Virtual Machine (and that of the operating system).

Prepare Product Information Input Files to Load Into the Product Catalog Schema

Follow the rules in this section to create input files (text files) containing your product information that you want to use in your Web site. Create a separate file for every database table.

Verify Input File's File Structure The input data file is, by default, a pipe-separated value text file. The input file has the following structure:

- First row: header containing the table name
- Second row: column names for that table
- Third row: data types for the columns listed on the second row
- Fourth through *Nth* row: input data

Table 14-1 shows the input file structure.

Table 14-1 Input File Structure

Row	Content
First Row	The header of the file must identify: ■ The number of records to be loaded. DBLoader will use this number as a reference point only. It will process all the records in the file regardless of this indicator. ■ The name of the table to be loaded with data in the database.
Second Row	The second row identifies the table column names into which you are loading data. You must include the primary key column or columns in the input file. Preface each primary key column name with an asterisk (`*`). Apart from primary keys in tables, all other columns are defaulted as `NULL`. Thus, you may omit column names where `NULL` is an acceptable value, and specify only those with non-`NULL` values.
Third Row	The third row specifies the data type of each column being loaded.
Fourth Through N Rows	All subsequent lines in the input data file contain the data values.

Listing 14-1 shows an example of a simple input file.

Listing 14-1 Simple Input File

```
3|WLCS_PRODUCT
*SKU|NAME|IN_STOCK|EST_SHIP_TIME|SPECIAL_NOTES|CREATION_DATE
   VARCHAR|VARCHAR|VARCHAR|VARCHAR|VARCHAR|DATE
P123|CoolKid|N|Out of stock until further notice|Special order
   only|02-Oct-2000
P124|FastKid|Y|One week|No special order|02-Oct-2000
P125|RadSneakers|Y||regular stock|02-Oct-2000
```

Note: You can also view a sample input file at the following location: `<PORTAL_HOME>\db\data\sample\wlcs\hardware\PRODUCT.dat`

Empty input strings Empty input strings from the data file are inserted into database as empty strings. You must account for each unspecified column in the input record by including the delimiter character (by default, a comma) in the correct position (matching the position of the columns you listed in line 2, the column names). For example:

```
P125|RadSneakers|Y||regular stock|02-Oct-2000
```

Unspecified values for non-primary-key fields In the previous example a value for the fourth identified column (`EST_SHIP_TIME`) was not specified. This condition is acceptable because this column is not a primary key for the database record. The column's value is stored as an empty string.

Note: If you intend to store a null value in the database for a non-primary-key column, you should enter `NULL` in the correct position for the column in that record. Do not enclose `NULL` in quotes as that will cause the column to be stored as a string.

Step 2: Edit the databaseload.properties File

The DBLoader uses the information in the `databaseload.properties` file to determine information about the data and loading process, including what driver, database, or login to use. Open the `databaseload.properties` file in the `<PORTAL_HOME>\db` directory. Uncomment the lines for the database you want to use, and enter the correct settings. Be sure to comment out the Pointbase lines if you are using another database.

Any comment lines in the `databaseload.properties` file are prefixed with the `#` character. Both comment lines and blank lines are allowed.

Table 14-2 databaseload.properties File Settings and Possible Values

Property Name	Default Value	Description
`jdbcdriver`	See `databaseload.properties`.	Specify which JDBC driver to use to connect to your database. The default driver is the Pointbase JDBC driver that ships with WebLogic.
`connection`	See `databaseload.properties`.	Database connection string required for the driver to connect to your database.
`dblogin`	See `databaseload.properties`.	The database username. The login name must have read/write privileges on the affected tables.
`dbpassword`	See `databaseload.properties`.	The database user password.
delimiter	|	You can change the recognized delimiter character that is used to separate values in the input data file. Choose another character, such as the circumflex (^) as a delimiter.
`dateformat`	`dateformat=mm-YY-dd`	Identifies the format that will be used for date columns in the input data. Date format is locale specific. Other formats are commented out in the `databaseload.properties` file.
`timestamptable`	WLCS_CATEGORY, WLCS_PRODUCT	Identifies the database tables to which DBLoader will track updates (for these two tables). The column name is fixed in the schema provided by the commerce services. However, if you are using DBLoader for other tables (not `WLCS` tables), you can specify other column names of your own.

Property Name	Default Value	Description
`timestampfield`	MODIFIED_DATE	Specifies the column in the `WLCS_CATEGORY` and `WLCS_PRODUCT` tables that identifies the last time this record in the table was modified. The value of the column specified is used by DBLoader to learn when the most recent update was made in each record in the tables identified in the `timestamptable` property. The column name is fixed in the schema provided by the Commerce services. However, if you are using DBLoader for other tables (not `WLCS` tables), you can specify other column names of your own.
`commitTxn`	`50`	Sets how many records are loaded before committing the updates in the database. If the value is less than or equal to one, DBLoader will commit after loading each record.
encoding	Not specified in the `databaseload.properties` file; therefore, the default is the Java 2 SDK's platform default.	Sets the multibyte character encoding type. The property value supplied can be UCS2 or UTF8. When writing data into and reading data out of the database, Java will transparently convert from the native character encoding used by your systems and Unicode 2.0. There is nothing special that you must do. However, if you need to write/read data to/from the database that is encoded differently than your system's native encoding, you will have to explicitly perform the translation.

Step 3: Load Data by Running the DBLoader Program

Now that you have created the input files and set up the `databaseload.properties` file correctly, run the `databaseload` script to start the DBLoader program.

- Review databaseload Basics
- Non-Windows Environment – Prepare to Run the Script
- Run the databaseload Script

Review databaseload Basics

The script is located at:

- `<PORTAL_HOME>\db` (Windows)
- `<PORTAL_HOME>/db` (UNIX)

Note: `PORTAL_HOME` is the directory where you installed WebLogic Portal.

The `databaseload` script performs the tasks:

- Configures your environment for the duration of execution of this program
- Specifies where to find the data input file
- Launches the DBLoader program

Non-Windows Environment – Prepare to Run the Script

In a non-Windows environment, before you can run the `databaseload` script, make sure that the `set-environment` script specifies the same database as the `databaseload.properties` file. The `set-environment` script resides in the same directory as the `databaseload` script. For example, if the `databaseload.properties` file uses `'jdbc:pointbase:server://localhost:9092/wlportal'` connections, then `set-environment` script should have `SET DATABASE=POINTBASE`.

As mentioned earlier, DBLoader runs independently of the WebLogic Portal server. Therefore you do not need to stop the server if you are planning to run the loader.

If you are running the WebLogic Portal server with Oracle, then the drawback might be a slower performance for the time the data is being loaded into the database.

Note: You might want to back up the particular tables that you are about to update before running DBLoader. The DBLoader program does not keep history records in the database.

Run the databaseload Script

The command to run the script has the following format:

```
>> databaseload { -insert | -update | -delete } input-file.dat
```

For example:

```
>> databaseload -update product_categories.dat
```

In the previous example, the DBLoader program will update rows in the product catalog database that match the primary keys specified in the `category.dat` input file.

Selecting the type of operation You must select one of the three possible operations: `-insert, -update, or -delete.`

UNIX and privileges On UNIX systems, the `databaseload.sh` file needs to have its default protections set to include execute privilege. A typical way to do this is with the command:

```
$ chmod +x databaseload.sh
```

Loading data into several tables To insert, update, or delete data in several tables, run the `databaseload` script separately for each table, providing the corresponding input filename as a parameter. The order of tables being updated should use the same data integrity rules as all other SQL statements. For example, insert rows into the parent table with the primary key constraint before inserting rows into the child table with the foreign key constraint.

Step 4: Troubleshoot Using the DBLoader Log Files

You can determine errors and other issues that occurred during any particular DBLoader operation by using the two audit trail log files:

- `dbloader.log`
- `dbloader.err`

This section contains the following information:

- Determine When to Review the Files
- Review the dbloader.log File
- Review the dbloader.err File

Determine When to Review the Files

You must check the files immediately after each operation; the files are overwritten by each DBLoader operation. Both files are created in the same directory where you run the `databaseload` script.

Review the dbloader.log File

The `dbloader.log` file contains the following information:

- The input filename, and the action taken: insert, update, or delete.
- The number of records processed during the load operation.
- The start and end time of the database load processing.

Review the dbloader.err File

If any errors occurred during the attempted database load operation, the `dbloader.err` file captures the following information:

- The input filename, and the action taken: insert, update, or delete.
- The timestamp when the failure or exception occurred on the record.
- The index of the failed data record in the input file.
- The reason for the failure or exception and actual the input record's values.

The DBLoader program checks the number of columns affected by the load (as specified in the second line of the input data file) against the number of input columns in each record. Because the column delimiter is a comma (by default), this character is not allowed in a string input column. If extra commas are supplied inadvertently, such as punctuation in a `LONG_DESC` (Long Description) column, an error will result and is noted in the `dbloader.err` file. To avoid this type of error, carefully check the number of commas you are using to separate the input data column values. Or select a different delimiter character and specify it in the `databaseload.properties` file. For more information, see "Step 2: Edit the databaseload.properties File."

All errors and exceptions are displayed in the console where the DBLoader program is running. Records with errors in them will be skipped, and the processing continues until the end of the file. (The program does not roll back a transaction if an error has occurred.)

Showing a Catalog in a JSP

The JavaServer Page (JSP) templates and JSP tags included in the Commerce services allow you to easily create the presentation part of the Product Catalog. The templates provide the mechanism for your visitors to view a catalog's categories and product items; the JSP tags in the templates implement that functionality.

JSP tag libraries allow you to easily retrieve the attributes of items and categories in the Product Catalog. You can then format these attributes using HTML tags. Any HTML editor can be used to create custom layouts. You can also include custom Java code within the JSPs to display categories and items.

To use the Catalog JSP tags, you need to import the `cat.tld` tag library into your JSP file by including the following code in the JSP:

```
<%@ taglib uri="cat.tld" prefix="catalog" %>
```

Three basic tags are used in the JSP templates that compose the default product catalog supplied by BEA. These tags are:

- `<catalog:getProperty>`

  Retrieves a property for display from a specified ProductItem or Category. Either explicit or implicit properties may be retrieved.

- `<catalog:iterateViewIterator>`

  Iterates a specified ViewIterator. The ViewIterator may be iterated either by View (one View per iteration) or by contained Catalog item (one ProductItem or Category per iteration).

- `<catalog:iterateThroughView>`

  Iterates a specified ViewIterator through the ProductItems or Categories contained within a specified View.

You can add additional tags as necessary to meet your business needs.

Using the <catalog:getProperty> Tag

Use the `<catalog:getProperty>` tag (Table 14-3) to retrieve a property for display from either a `ProductItem` or `Category`. The property can either be an explicit property (a property that can be retrieved using a `get` method on the Catalog item) or an implicit property (a property available through the `ConfigurableEntity` `getProperty` methods on the Catalog item). The tag first checks to see if the specified property can be retrieved as an explicit property. If it cannot, the specified property is retrieved as an implicit property.

Table 14-3 `<catalog:getProperty>` Tag Attributes

Tag Attribute	Required	Type	Description	R/C
`getterArgument`	No	String	Denotes a reference to an object supplied as an argument to an explicit property getter method. May also be used to obtain implicit or custom properties that are defined using the property set framework, in which case the `getterArgument` would be the scope name for the property set (see second example below). The object must be presented in the form `<%= getterArgumentReference %>` and must be a run-time expression.	R
`id`	No	String	`id="newInstance"` If the `id` attribute is supplied, the value of the retrieved property will be available in the variable name to which `id` is assigned. Otherwise, the value of the property is inlined.	C
`object`	Yes	Catalog item	Denotes a reference to a `ProductItem` or `Category` object that must be presented in the form `<%= objectReference %>.`	R
`propertyName`	Yes	String	`propertyName="propertyName"` Name of the property to retrieve. If the property is explicit, it may be one of the values shown in Table 14-4.	C
`returnType`	No	String	`returnType="returnType"` If the `id` attribute is supplied, declares the type of the variable specified by the `id` attribute.	C

Table 14-4 propertyName Values

Property Name	Catalog Item Type
`"contributor | coverage | creationDate | creator | description | image | key | language | modifiedDate | name | publisher | relation | rights | source"`	Catalog Item (common properties)
`"jsp"`	Category
`"availability | currentPrice | format | jsp | msrp | shippingCode | taxCode | type | visible"`	ProductItem

Example 1: Retrieving the Detail JSP Information From an Item
Listing 14-2 retrieves the detail JSP information from an existing `ProductItem`:

Listing 14-2 Retrieving the Detail JSP Information From an Item

```
<%@ taglib uri="cat.tld" prefix="catalog" %>

<catalog:getProperty
object="<%= item %>"
   propertyName="Jsp"
   getterArgument=
   "<%= new Integer(ProductItem.DETAILED_DISPLAY_JSP_INDEX) %>"
   id="detailJspInfo"
returnType="com.beasys.commerce.ebusiness.catalog.JspInfo"
/>
```

Example 2: Using the getterArgument Attribute Listing 14-3 shows how to use the `getterArgument` attribute to obtain an implicit or custom property for a property set/schema with the following characteristics:

- Name: `MyCatalog`
- PropertyName: `color`

Note: Because the `getterArgument` must be a run-time expression, we assign `MyCatalog` to a String variable and use the variable as the value to the `getterArgument`.

Listing 14-3 Using the getterArgument Attribute

```
<%@ taglib uri="cat.tld" prefix="catalog" %>

<%
String myPropertySetName = "MyCatalog";
ProductItem myProductItem = .....; // reference to a ProductItem
%>
<catalog:getProperty
   object="<%=myProductItem%>
   propertyName="color"
   getterArgument="<%=myPropertySetName%>"
/>
```

Using the <catalog:iterateViewIterator> Tag

Use the `<catalog:iterateViewIterator>` tag (Table 14-5) to iterate through a `ViewIterator`. A `ViewIterator` is an iterator over a potentially large collection of remote data that is broken up into a series of fixed sized `Views`. `View Iterators` are returned from all Catalog service API methods that may potentially return a large set of `ProductItems` or `Categories`. This tag allows you to iterate the `ViewIterator` one item (`ProductItem` or `Category`) at a time (the default behavior) or by an entire `View` (fixed size set of `ProductItems` or `Categories`) at a time. It is important to note that this tag does not reset the state of the `ViewIterator` upon completion.

Table 14-5 `<catalog:iterateViewIterator>` Tag Attributes

Tag Attribute	Required	Type	Description	R/C
`id`	Yes	String	`id="newInstance"` The value of the current iterated object will be available in the variable name to which the `id` is assigned.	C
`iterator`	Yes	ViewIterator	Denotes a reference to a `ViewIterator` object. Must be presented in the form `<%= iteratorReference %>`.	R

Table 14-5 `<catalog:iterateViewIterator>` Tag Attributes (Continued)

Tag Attribute	Required	Type	Description	R/C
`iterateByView`	No	String	`iterateByView="{true|false}"` Specifies whether to iterate the `ViewIterator` by View or by Catalog item. If not specified, the `ViewIterator` will be iterated by Catalog item.	C
`returnType`	No	String	`returnType="returnType"` Declares the type of the variable specified by the `id` attribute. Defaults to `java.lang.Object`. If `iterateByView` is `true`, the type is assumed to be `com.beasys.commerce.ebusiness.catalog.View`.	C

Example 1: Displaying Keys of Categories in a ViewIterator Listing 14-4 displays the keys of all categories in a `ViewIterator`:

Listing 14-4 Displaying Keys of Categories in a `ViewIterator`

```
<%@ taglib uri="cat.tld" prefix="catalog" %>

<catalog:iterateViewIterator
   iterator="<%= myIterator %>"
   id="category"
   returnType="com.beasys.commerce.ebusiness.catalog.Category">
   <%= category.getKey().toString() %>
</catalog:iterateViewIterator>
```

Example 2: Displaying all Views in a ViewIterator Listing 14-5 displays all the `Views` contained within a `ViewIterator`:

Listing 14-5 Displaying all Views in a ViewIterator

```
<%@ taglib uri="cat.tld" prefix="catalog" %>
```

```
<catalog:iterateViewIterator
   iterator="<%= myIterator %>"
   id="view"
   returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator"
   iterateByView="true">
   <%= view.toString() %>
</catalog:iterateViewIterator>
```

Using the <catalog:iterateThroughView> Tag

The `<catalog:iterateThroughView>` tag (Table 14-6) iterates through a `View` of a specified `ViewIterator`. The tag will iterate the `View` one Catalog item at a time until the end of the view is reached. If you do not specify a specific view (by index) through which to iterate, the current `View` of the `ViewIterator` is used. It is important to note that this tag does not reset the state of the `ViewIterator` upon completion.

Table 14-6 `<catalog:iterateThroughView>` Tag Attributes

Tag Attribute	Required	Type	Description	R/C
`id`	Yes	String	`id="newInstance"` The value of the current iterated object will be available in the variable name to which the `id` is assigned.	C
`iterator`	Yes	ViewIterator	Denotes a reference to a `ViewIterator` object that must be presented in the form `<%= iteratorReference %>`	R
`returnType`	No	String	`returnType="returnType"` Declares the type of the variable specified by the `id` attribute. Defaults to `java.lang.Object`.	C
`viewIndex`	No	Integer	Specifies the index of the View (relative to the start of the `ViewIterator`) through which to iterate. The referenced object must be presented in the form `<%= viewIndexIntegerReference %>`.	R

Example 1: Displaying Keys of All ProductItems in Current View of ViewIterator Listing 14-6 displays the keys of all the `ProductItems` contained in the current `View` of a specified `ViewIterator`:

Listing 14-6 Displaying Keys of All ProductItems in Current View of ViewIterator

```
<%@ taglib uri=”cat.tld” prefix=”catalog” %>

<catalog:iterateThroughView
   iterator="<%= myIterator %>"
   id="item"
   returnType="com.beasys.commerce.ebusiness.catalog.ProductItem">
<%= item.getKey().toString() %>
</catalog:iterateThroughView>
```

Example 2: Displaying Keys of All ProductItems in First View of ViewIterator Listing 14-7 displays the keys of all the `ProductItems` contained in the first `View` of a specified `ViewIterator`:

Listing 14-7 Displaying Keys of All ProductItems in First View of ViewIterator

```
<%@ taglib uri=”cat.tld” prefix=”catalog” %>

<catalog:iterateThroughView
   iterator="<%= myIterator %>"
   id="item"
   returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
   viewIndex=”new Integer(0)”>
   <%= item.getKey().toString() %>
</catalog:iterateThroughView>
```

Hooking Up a Catalog to a Shopping Cart

To hook up a catalog to a shopping cart, you need to implement the `shoppingcart.jsp` template in the catalog. This template contains code that manages a shopping cart service, including implementations of the necessary input processors and pipeline components used to manage much of the business logic and back-end tasks required to successfully execute the shopping cart service. You can either implement this template as BEA provides it or you can tailor it to meet your specific needs.

Implementing shoppingcart.jsp

To implement `shopppingcart.jsp`, first, ensure that this JSP is stored in the appropraite portlets folder for the application. Then, do one of the following:

- Directly link `shoppingcart.jsp` to another JSP by providing a URL from the referencing page. Depending upon the functionality of the referencing page, you might need to add a **View Cart** button or icon to launch the forwarding event.

OR

- Include in the referencing page code a Webflow that calls `shoppingcart.jsp`. Depending upon the functionality of the referencing page, you might need to add a **View Cart** button or icon to launch the Webflow.

How shoppingcart.jsp Works

Customers can arrive at `shoppingcart.jsp` template from any product catalog page by clicking **View Cart** (or its equivalent) on that page. The `shoppingcart.jsp` template displays the items currently in a customer’s shopping cart. For each item the customer added to their cart (that is still actively part of the current purchase), the `shoppingcart.jsp` template displays the quantity, the item name, the list price, the actual price, a savings amount, and a subtotal. Following this information, a total price for the order is displayed.

The item quantity is shown in an editable field, allowing customers to change the quantity of the item simply by typing a new quantity and clicking **Update**. For your customers’ convenience, the item name is hyperlinked back to its description in the product catalog. For each item in the shopping cart, there is also a **Delete** button and a **Buy Later** button. Clicking **Delete** removes the item from the shopping cart, while clicking **Buy Later** causes the item to be moved from the Shopping Cart to the Saved Items list. For each item shown in the Saved Items list, the hyperlinked item name and a brief description are displayed. Additionally, **Delete** and **Add to Cart** in this section allows your customers to remove the item altogether or to move it back to their active Shopping Cart.

If customers click a link to an individual product item to review detailed information about that product item, the next page is the appropriate product catalog page. If they click on **Update**, **Empty Cart**, **Delete**, or **Save for Later**, they are returned to the shopping cart page (`shoppingcart.jsp`) after the appropriate input processor or pipeline has been executed to record the modification.

If the customer is satisfied with the contents of their shopping cart as shown on this page, the customer can initiate the checkout process by clicking **Check Out**. If this is the case, the next page is the shipping information page (`shipping.jsp`).

Notes: If the customer has not yet logged into the site and clicks **Check Out**, the customer will be prompted to log in at the `login.jsp` template (prior to loading the `shipping.jsp` template).

To be able to use the features of the Saved Items list, a customer must have first logged in.

If there are no items in a customer's shopping cart, the Empty Cart, Update, and Check Out buttons will not be available.

If the customer is satisfied with the contents of their shopping cart, the customer can click the Check Out button to begin the checkout process.

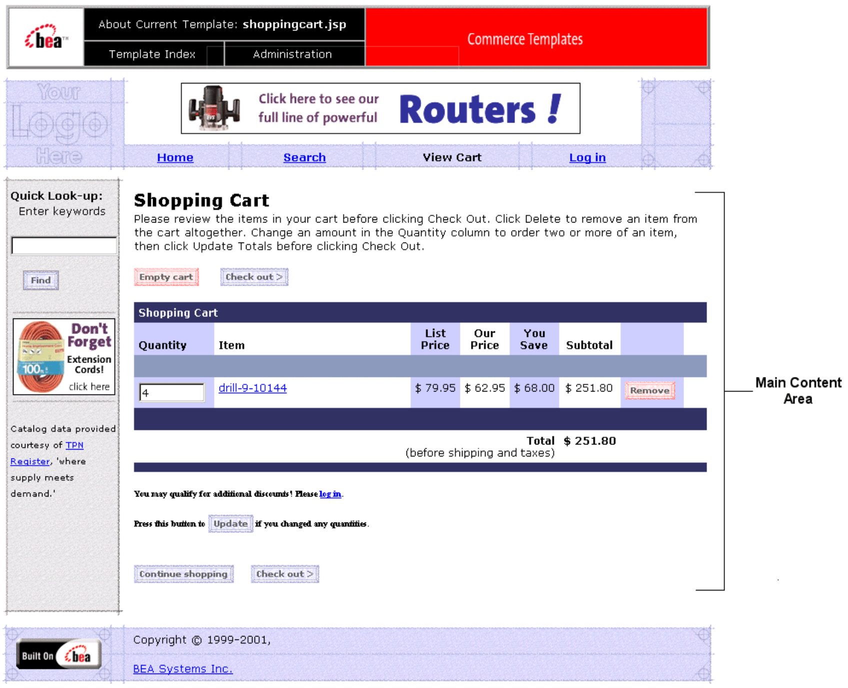
Note: If the customer is not logged into your e-commerce site, they will be prompted to do so before continuing to the next part of the checkout process.

If your customer wants to start over, the customer can click the Empty Cart button to empty the entire contents of the shopping cart (both active and saved). If your customer wants to continue shopping, the customer can click the Continue Shopping button to return to the product catalog.

Description

Figure 14-1 and Figure 14-2 show annotated versions of the `shoppingcart.jsp` template. Figure 14-1 shows the page for a customer who has not logged in:

Figure 14-1 `shoppingcart.jsp`-Formatted Web Page for a Visitor Who Has Not Logged In



Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Shopping Cart

Please review the items in your cart before clicking Check Out. Click Delete to remove an item from the cart altogether. Change an amount in the Quantity column to order two or more of an item, then click Update Totals before clicking Check Out.

Shopping Cart

Quantity	Item	List Price	Our Price	You Save	Subtotal	
4	drill-9-10144	$ 79.95	$ 62.95	$ 68.00	$ 251.80	Remove

Total $ 251.80
(before shipping and taxes)

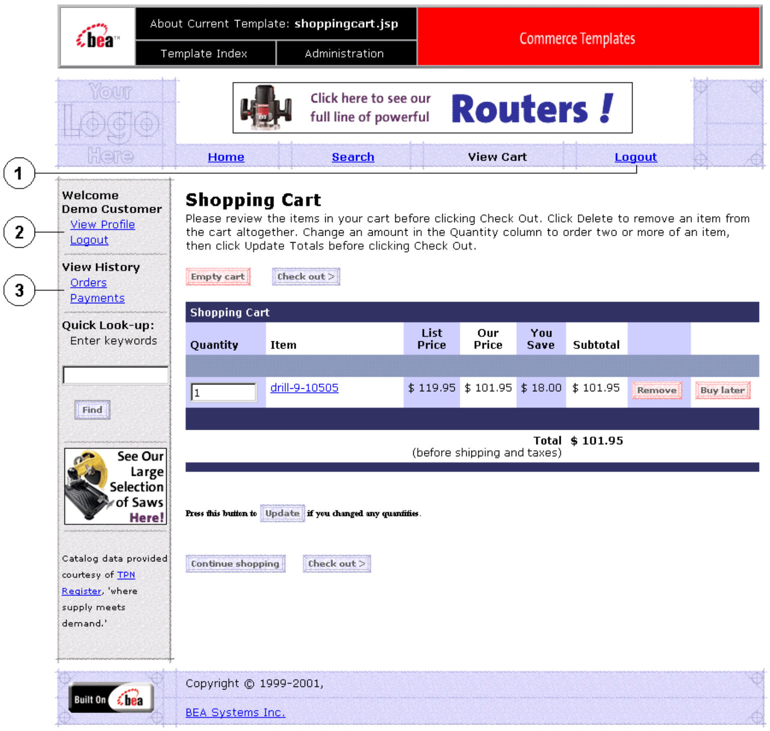
You may qualify for additional discounts! Please log in.

Press this button to Update if you changed any quantities.

Copyright © 1999-2001,
BEA Systems Inc.

Figure 14-2 shows the page for a customer who has logged in:

Figure 14-2 `shoppingcart.jsp`-Formatted Web Page for a Visitor Who Has Logged In



Shopping Cart

Please review the items in your cart before clicking Check Out. Click Delete to remove an item from the cart altogether. Change an amount in the Quantity column to order two or more of an item, then click Update Totals before clicking Check Out.

Empty cart | Check out >

Shopping Cart

Quantity	Item	List Price	Our Price	You Save	Subtotal		
1	drill-9-10505	$ 119.95	$ 101.95	$ 18.00	$ 101.95	Remove	Buy later

Total $ 101.95
(before shipping and taxes)

Press this button to Update if you changed any quantities.

Continue shopping | Check out >

Catalog data provided courtesy of TPN Register, 'where supply meets demand.'

Copyright © 1999-2001, BEA Systems Inc.

The main content area of the template contains both dynamically generated data and static content. The dynamic content on `shoppingcart.jsp` is generated using WebLogic Server and pipeline JSP tags, which obtain and display the contents for both the active shopping cart and Saved Item list. For the `shoppingcart.jsp` template, the form posts include Empty Cart, Check Out, Remove, Update, and Continue.

The following changes occur after the user has logged in:

1. The **Login** link changes to `Logout`.
2. A welcome section appears that shows the customer's name, a link to view that customer's profile, and a link to logout.

3. A view history section appears that shows the customer's order and payment history.

Key components of the template are shown in Table 14-7.

Table 14-7 Template Components

Type of Component	Components
Included templates	■ `admin.inc` ■ `stylesheet.inc` ■ `header.inc` ■ `leftside.inc` ■ `footer.inc`

Tag libraries	`<%@ taglib uri="weblogic.tld" prefix="wl" %>`
	`<%@ taglib uri="webflow.tld" prefix="webflow" %>`
	`<%@ taglib uri="i18n.tld" prefix="i18n" %>`

Imported Java packages	`java.util.*`
	`java.text.*`
	`com.beasys.commerce.axiom.units.*`
	`com.beasys.commerce.ebusiness.shoppingcart.*`
	`com.bea.commerce.ebusiness.price.service.DiscountPresentati on`
	`com.bea.commerce.ebusiness.price.quote.OrderAdjustment`
	`com.bea.commerce.ebusiness.price.quote.AdjustmentDetail`
	`com.beasys.commerce.webflow.HttpRequestConstants`
	`com.beasys.commerce.webflow.PipelineSessionConstants`
	`com.bea.p13n.appflow.webflow.WebflowJSPHelper`

Location in Default Webflow

Customers can arrive at `shoppingcart.jsp` template from any product catalog page by clicking the View Cart button. If the customer is satisfied with the contents of their shopping cart as shown on this page, the customer can initiate the checkout process by clicking the Check Out button. If this is the case, the next page is the shipping information page (`shipping.jsp`).

Note: If the customer has not yet logged into the site and clicks **Check Out**, the customer will be prompted to log in at the `login.jsp` template (prior to loading the `shipping.jsp` template).

If customers click a link to an individual product item to review detailed information about that product item, the next page is the appropriate product catalog page. If they click on the Update, Empty Cart, Delete, or Save for Later buttons, they are returned to the shopping cart page (`shoppingcart.jsp`) after the appropriate input processor or pipeline has been executed to record the modification.

This JSP is in the `sampleapp_order` namespace.

Events

Every time a customer clicks a button to manage the contents of their shopping cart, it is considered an event. Each event triggers a particular response in the default Webflow that allows the customer to continue. While this response can be to load another JSP, it is usually the case that an input processor and/or pipeline is invoked first. Table 14-8 provides information about these events and the business logic they invoke.

Table 14-8 `shoppingcart.jsp` Events

Event	Webflow Response(s)
`--`	`InitShoppingCartIP`
`--`	`RefreshSavedList`
`button.checkout`	`InitShippingMethodListIP`
`button.deleteItemFromShoppingCart`	`DeleteProductItemFromShoppingCartIP`
`button.deleteItemFromSavedList`	`UpdateSkuIP` `DeleteProductItemFromSavedList`
`button.emptyShoppingCart`	`EmptyShoppingCartIP`
`button.moveItemToSavedList`	`UpdateSkuIP` `MoveProductItemToSavedList`
`button.moveItemToShoppingCart`	`UpdateSkuIP` `MoveProductItemToShoppingCart`

Table 14-8 `shoppingcart.jsp` Events

Event	Webflow Response(s)
`button.updateShoppingCartQuantities`	`UpdateShoppingCartQuantitiesIP`

Table 14-9 briefly describes each of the pipelines from Table 14-8.

Table 14-9 Shopping Cart Pipelines

Pipeline	Description
`RefreshSavedList`	Contains `RefreshSavedListPC` and is not transactional.
`DeleteProductItemFromSavedList`	Contains `DeleteProductItemFromSavedListPC` and `PriceShoppingCartPC`, and is transactional.
`MoveProductItemToSavedList`	Contains `MoveProductItemToSavedListPC` and `PriceShoppingCartPC`, and is transactional.
`MoveProductItemToShoppingCart`	Contains `MoveProductItemToShoppingCartPC` and `PriceShoppingCartPC`, and is transactional.

Notes: Although the `InitShoppingCartIP` and `RefreshSavedList` pipeline are associated with the `shoppingcart.jsp` template, they are not triggered by events on the page. Rather, both are executed before the `shoppingcart.jsp` is viewed. The `InitShoppingCartIP` input processor creates an empty shopping cart in preparation for the customer's shopping experience, while the `RefreshSavedList` pipeline retrieves a customer's list of previously saved shopping cart items..

How shoppingcart.jsp Displays Data

One purpose of the `shoppingcart.jsp` template is to display the data specific to a customer's shopping experience for their review. This is accomplished on `shoppingcart.jsp` using a combination of WebLogic Server and pipeline JSP tags and accessor method and /attributes.

First, the `getProperty` JSP tag retrieves the `SHOPPING_CART` and `SAVED_SHOPPING_CART` attributes from the pipeline session. Table 14-10 provides more detailed information on these attributes.

Table 14-10 shoppingcart.jsp Pipeline Session Attributes

Attribute	Type	Description
`PipelineSessionConstants .SAVED_SHOPPING_CART`	`com.beasys.commerce.ebusiness .shoppingcart.ShoppingCart`	The saved shopping cart (source of the saved items).
`PipelineSessionConstants .SHOPPING_CART`	`com.beasys.commerce.ebusiness .shoppingcart.ShoppingCart`	The currently active shopping cart.

Listing 14-8 illustrates how these attributes are retrieved from the pipeline session using the `getProperty` JSP tag.

Listing 14-8 Retrieving Shopping Cart Attributes

```
<webflow:getProperty id="shoppingCart"
   property="<%=PipelineSessionConstants.SHOPPING_CART%>"

type="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart"
   scope="session" namespace="sampleapp_main" />

<webflow:getProperty id="savedShoppingCart"
   property="<%=PipelineSessionConstants.SAVED_SHOPPING_CART%>"
   type="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart"
   scope="session" namespace="sampleapp_main" />
```

The data stored within the pipeline session attributes is accessed by using accessor methods/attributes within Java scriptlets. Table 14-11 provides more detailed information about these methods for `ShoppingCart` (also `savedShoppingCart`), while Table 14-12 provides this information for `ShoppingCartLine`.

Table 14-11 ShoppingCart Accessor Methods/Attributes

Method/Attribute	Description
`getShoppingCartLineCollection()`	A collection of the individual lines in the shopping cart (that is, `ShoppingCartLine`).

Table 14-11 ShoppingCart Accessor Methods/Attributes (Continued)

Method/Attribute	Description
`getTotal`	In this instance, the total tax specified by the `OrderConstants.LINE_TAX` parameter. **Note:** The `getTotal()` method also allows you to combine different total types. For more information, see the *Javadoc*.

Because the `getShoppingCartLineCollection()` method allows you to retrieve a collection of the individual lines within a shopping cart, there are also accessor method and attributes you can use to break apart the information contained within each line. Table 14-12 provides information about these methods and attributes.

Table 14-12 ShoppingCartLine Accessor Methods/Attributes

Method/Attribute	Description
`getQuantity()`	The quantity of the item.
`getProductItem()`	The product item in the shopping cart line.
`getUnitPrice()`	The current price for the item at the time it was added to the shopping cart. May be different from MSRP.
`getBaseTotal(int totalType)`	The total before discounts.

Listing 14-9 provides an example of how these accessor methods/attributes are used within Java scriptlets.

Listing 14-9 Using Accessor Methods Within `shoppingcart.jsp` Java Scriptlets

```
<<td align="right" valign="top" bgcolor="#CCCCFF"><div class="tabletext" nowrap>
<%-- The i18n tag allows the "currency.properties" file to substitute a display
--%>
<%-- currency value (e.g "$") for the returned 3 letter ISO4217 code (e.g. "USD").
--%>

         <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
            shoppingCartLine.getProductItem().getMsrp().getCurrency() %>"/>
         <%= WebflowJSPHelper.priceFormat(shoppingCartLine.getProductItem().
            getMsrp().getValue() ) %></div>

      </td>

      <td align="right" valign="top"><div class="tabletext" nowrap>

          <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
            shoppingCartLine.getUnitPrice().getCurrency() %>"/>

         <%= WebflowJSPHelper.priceFormat( shoppingCartLine.getUnitPrice().
            getValue() ) %></div>

      </td>
      <td align="right" valign="top" bgcolor="#CCCCFF"><div class="tabletext"
         nowrap>

         <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
            shoppingCartLine.getBaseSavings().getCurrency() %>"/>

         <%= WebflowJSPHelper.priceFormat( shoppingCartLine.getBaseSavings().
            getValue() ) %></div>

      </td>

      <td align="right" valign="top"><div class="tabletext" nowrap>

         <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
            shoppingCartLine.getBaseTotal().getCurrency() %>"/>

         <%= WebflowJSPHelper.priceFormat( shoppingCartLine.getBaseTotal().
            getValue() ) %>

         </div>
      </td>
```

shoppingcart.jsp Form Fields

Another purpose of the `shoppingcart.jsp` template is to allow customers to make changes to their shopping cart using various HTML form fields. These form fields are also used to pass needed information to the Webflow.

The form fields used in the `shoppingcart.jsp` template, and a description for each of them, are listed in Table 14-13.

Table 14-13 `shoppingcart.jsp` Form Fields

Parameter Name	Type	Description
`"event"`	Hidden	Indicates which event has been triggered. It is used by the Webflow to determine what happens next.
`"origin"`	Hidden	The name of the current page (`shoppingcart.jsp`), used by the Webflow.
`HttpRequestConstants.CATALOG_ITEM_SKU`	Hidden	SKU of the item that the event is to operate on.
`NewQuantity_<SKU>` Where <*SKU*> is replaced with the SKU of the item on the shopping cart line.	Textbox	The new quantity for the item in the shopping cart. It is the only form field on this page that requires input from the customer.

Note: Parameters that are literals in the JSP code are shown in quotes, while non-literals will require scriptlet syntax (such as `<%= HttpRequestConstants.CATALOG_ITEM_SKU %>`) for use in the JSP.

shoppingcart.jsp Input Processorss

`shoppingcart.jsp` uses Webflow components called input processors and pipelines to execute much of its necessary business logic. This section describes the key input processors you can use. These input processors represent Java classes invoked to carry out more complex shopping cart service tasks when invoked by the Webflow mechanism. **[what .wf file(s) are we referring to here with these components?]**

This section includes information on these input processors:

- DeleteProductItemFromShoppingCartIP
- EmptyShoppingCartIP
- InitShoppingCartIP

- UpdateShoppingCartQuantitiesIP
- UpdateSkuIP

Note: See “Setting Up Portal Navigation” for information about using, creating, or modifying a Webflow and using input processors.

DeleteProductItemFromShoppingCartIP

This input processor (all input processor names end in the letters “IP”)removes an item from the shopping cart.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.webflow.DeleteProductItemFromShoppingCartIP`
Required `HTTPServletRequest` Parameters	`HttpRequestConstants.CATALOG_ITEM_SKU`
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.CATALOG_ITEM`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	`ProcessingException`, thrown if the required request parameters or required Pipeline session attributes are not available.

EmptyShoppingCartIP

This input processor creates a new shopping cart and stores it in the pipeline session and discards the old shopping cart.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.webflow.EmptyShoppingCartIP`
Required `HTTPServletRequest` Parameters	None
Required Pipeline Session Attributes	None

Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
	`PipelineSessionConstants.UPDATED_PRODUCT_ITEMS`

Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

InitShoppingCartIP

This input processor Initializes the active shopping cart prior to loading the shoppingcart.jsp template. If the shopping cart already exists, this input processor does nothing.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.webflow.InitShoppingCartIP`
Required `HTTPServletRequest` Parameters	None
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.`UPDATED_QUANTITY_DELTAS

Removed Pipeline Session Attributes	None
Validation	None
Exceptions	None

UpdateShoppingCartQuantitiesIP

This input processor validates the quantity fields for each line and sets those quantities in the shopping cart. If the quantity is zero, it will delete the item from the shopping cart.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.webflow.` `UpdateShoppingCartQuantitiesIP`
Required `HTTPServletRequest` Parameters	`NewQuantity_<SKU>` Where <*SKU*> is replaced with the SKU of the item on the shopping cart line.
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`

Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
	`PipelineSessionConstants.UPDATED_PRODUCT_ITEMS`

Removed Pipeline Session Attributes	None
Validation	Verifies that the quantity fields only contain positive integers.
Exceptions	`ProcessingException`, thrown if the required request parameters or required Pipeline session attributes are not available.

UpdateSkuIP

This input processor reads the SKU from the HTTP request and places it into the Pipeline session.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.webflow.UpdateSkuIP`
Required `HTTPServletRequest` Parameters	`HttpRequestConstants.CATALOG_ITEM_SKU`
Required Pipeline Session Attributes	None
Updated Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM_SKU`
Removed Pipeline Session Attributes	None
Validation	None
Exceptions	`ProcessingException`, thrown if the required request parameters are not available.

shoppingcart.jsp Pipeline Components

This section provides a brief description of each pipeline component associated with the `shoppingcart.jsp` template. These Pipelines are processor nodes a Webflow invokes to initiate the execution of specific tasks related to the Shopping Cart service.

Note: Some pipeline components extend other, base pipeline components. For more information on the base classes, see the *Javadoc*.

For more information on pipeline components, see "Types of Nodes" on page 9-3.

This section contains information on these pipeline components:

- .DeleteProductItemFromSavedListPC
- MoveProductItemToSavedListPC
- MoveProductItemToShoppingCartPC
- RefreshSavedListPC

- PriceShoppingCartPC
- AddToCartTrackerPC
- RemoveFromCartTrackerPC

Note: See "Setting Up Portal Navigation" for information about using, creating, or modifying a Webflow and using pipeline components.

DeleteProductItemFromSavedListPC

This pipeline component (all pipeline component names end in the letters "PC") removes the item from the saved list and updates the `WLCS_SAVED_ITEM_LIST` table in the database.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.pipeline.DeleteProductItemFromSavedListPC`

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM_SKU`
	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.USER_NAME`

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.DeleteProductItemFromSavedListPC`
Exceptions	`PipelineException`, thrown if the required Pipeline session attributes are not available.

MoveProductItemToSavedListPC

This pipeline component moves an item from the shopping cart, adds it to the saved list. It then updates the `WLCS_SAVED_ITEM_LIST` table in the database.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.pipeline.` `MoveProductItemToSavedListPC`

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM_SKU`
	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.USER_NAME`

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.CATALOG_ITEM`
	`PipelineSessionConstants.QUANTITY`

Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.` `MoveProductItemToSavedListPC`
Exceptions	`PipelineException`, thrown if the required Pipeline session attributes are not available.

MoveProductItemToShoppingCartPC

This pipeline component removes the item from the saved list, adds it to the shopping cart with a quantity of one. It then updates the `WLCS_SAVED_ITEM_LIST` table in the database.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.pipeline.` `MoveProductItemToShoppingCartPC`

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM_SKU`
	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.USER_NAME`

Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.CATALOG_ITEM`

Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`examples.wlcs.sampleapp.shoppingcart.` `pipeline.MoveProductItemToShoppingCartPC`
Exceptions	`PipelineException`, thrown if the required Pipeline session attributes are not available.

RefreshSavedListPC

This pipeline component queries the `WLCS_SAVED_ITEM_LIST` table and refreshes the saved shopping cart in the pipeline session. The saved list is only refreshed if the saved shopping cart does not exist in the pipeline session.

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.pipeline.` `RefreshSavedListPC`
Required Pipeline Session Attributes	`PipelineSessionConstants.USER_NAME`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SAVED_SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Session bean
JNDI Name	`examples.wlcs.sampleapp.shoppingcart.pipeline.` `RefreshSavedListPC`
Exceptions	`PipelineException`, thrown if the required Pipeline session attributes are not available.

PriceShoppingCartPC

This pipeline component invokes the Pricing Service to compute the line totals, discounts, shopping cart total and shopping cart discounts

Class Invoked	`examples.wlcs.sampleapp.shoppingcart.pipeline.PriceShoppingCartPC`
Required Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.USER_NAME`
Updated Pipeline Session Attributes	`PipelineSessionConstants.SHOPPING_CART`
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	`PipelineException`, thrown if the Pricing Service fails in any way

AddToCartTrackerPC

This pipeline component fires an `AddToCartEvent` describing which item was just added to the cart. For more information about this event, see "Event and Behavior Tracking".

Class Invoked	`examples.wlcs.sampleapp.tracking.pipeline.AddToCartTrackerPC`
Description	

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM`
	`PipelineSessionConstants.HTTP_SESSION_ID`
	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.STOREFRONT`
	`PipelineSessionConstants.CUSTOM_REQUEST`

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	None

RemoveFromCartTrackerPC

This pipeline component fires a RemoveFromCartEvent describing which item was just added to the cart. For more information about this event, see "Event and Behavior Tracking".

Class Invoked	`examples.wlcs.sampleapp.tracking.pipeline.RemoveFromCartTrackerPC`

Required Pipeline Session Attributes	`PipelineSessionConstants.CATALOG_ITEM`
	`PipelineSessionConstants.HTTP_SESSION_ID`
	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.STOREFRONT`
	`PipelineSessionConstants.CUSTOM_REQUEST`

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	None

UpdateShoppingCartQuantitiesTrackerPC

For each shopping cart line, this pipeline component does one of the following:

- If more items in the line were selected, it fires an `AddToCartEvent`.
- If fewer items in that line were selected, it fires a `RemoveFromCartEvent`.
- If the number of items in that line is the same as before, does not fire an event.

Class Invoked	`examples.wlcs.sampleapp.tracking.pipeline.` `UpdateShoppingCartQuantitiesTrackerPC`

Required Pipeline Session Attributes	`PipelineSessionConstants.UPDATED_PRODUCT_ITEMS`
	`PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
	`PipelineSessionConstants.HTTP_SESSION_ID`
	`PipelineSessionConstants.USER_NAME`
	`PipelineSessionConstants.STOREFRONT`
	`PipelineSessionConstants.CUSTOM_REQUEST`

Updated Pipeline Session Attributes	None
Removed Pipeline Session Attributes	None
Type	Java object
JNDI Name	None
Exceptions	None

Integrating Services With the Catalog Cache

The catalog architecture includes a powerful caching mechanism for items and categories within the Product Catalog. You can choose between integrating services in front of the cache or behind the cache. Currently the ProductItemManager and CategoryManager benefit from the caching architecture

Replacing the JNDI name of a bean in the CatalogManager's deployment descriptor will replace a service in front of the cache. The service will have to implement its own caching mechanism or forgo the benefits of caching.

The services defined by BEA—specified in the deployment descriptor for the CatalogManager—implement the caching for access to items and categories. The following beans query the cache and returned cached data if available; otherwise they delegate to the beans specified in their deployment descriptors:

- `com.beasys.commerce.ebusiness.catalog.service.item.ProductItemManager`
- `com.beasys.commerce.ebusiness.catalog.service.category.CategoryManager`

You can extend the functionality of the Product Catalog behind the cache by editing the deployment descriptors for the ProductItemManager and CategoryManager beans. This enables you to concentrate on the persistence model for the catalog without worrying about the caching architecture. As Listing 14-10 shows, you need to replace the current delegate service provider class (`JdbcCategoryManager`) in the `ejb-jar.xml` deployment descriptor with the name of a new session bean that implements the CategoryManager interface. You would also need to make the same change to the `weblogic-ejb-jar.xml` file.

Listing 14-10 CategoryManager Deployment Descriptor (`ejb-jar.xml`)

```
<session>
   <ejb-name>
      com.beasys.commerce.ebusiness.catalog.service.
         category.CategoryManager
   </ejb-name>
   <home>
      com.beasys.commerce.ebusiness.catalog.service.
         category.CategoryManagerHome
   </home>
   <remote>
      com.beasys.commerce.ebusiness.catalog.service.
         category.CategoryManager
   </remote>
   <ejb-class>
      com.beasys.commerce.ebusiness.catalog.service.
         category.CategoryManagerImpl
   </ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>
   <!--  one specifies the delegateName to tell the Bridge component
   (the one used by the catalog manager which ejb to delegate to.
   That way, one can change delegates by changing the env-entry...
   -->

  <env-entry>
      <env-entry-name>delegateName</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>ejb/JdbcCategoryManager</env-entry-value>
   </env-entry>

   <ejb-ref>
      <ejb-ref-name>ejb/JdbcCategoryManager</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <home>
         com.beasys.commerce.ebusiness.catalog.service.category.
         JdbcCategoryManagerHome
      </home>
      <remote>
         com.beasys.commerce.ebusiness.catalog.service.
            category.JdbcCategoryManager
      </remote>
   </ejb-ref>

   .
   .
   .
</session>
```

`ejb-jar.xml` and `weblogic-ejb-jar.xml` file are packaged in the `ebusiness.jar` file, which is in the `<BEA_HOME>\weblogic700\common\templates\domains\shared\bea\portal\apps\jars` directory (where <BEA_HOME> is the directory in which you installed your BEA applications).

CHAPTER

15 Event and Behavior Tracking

The Event system provides you with the ability to identify the interactions that visitors have with your portal or Web site.

Customer interactions One of the primary uses of events is in customer interactions such as promotions or campaigns. A simple example of using events in a campaign is triggering the display of an ad for a related product when a customer places an item in a shopping cart.

Behavior Tracking Another primary use of events is to track visitor behavior by recording events. Recording events is more than just navigation logging, which tells you only what pages were visited. Behavior Tracking allows you to know what the visitor saw and responded to, or equally important, *ignored* on a page. Recording events in a database allows leading e-analytics and e-marketing systems to use event data for data mining. Analyzing event information helps you create or enhance the rules that customize the content of your site to each visitor and evaluate the effectiveness of your promotional campaigns

Custom events The Event system allows you to create your own events. For example, you could create an event that records who visits each portlet and how often each portlet is accessed. When you create a custom event, the event can either be recorded to a database using the WebLogic Portal persistence mechanism or not be recorded using this mechanism. Events that are persisted are called Behavior Tracking events. If your custom event will be not persisted, follow the instructions in "Writing the Custom Event Class" on page 15-13. If you are creating a custom Behavior Tracking event, follow the instructions in "Writing a Behavior Tracking Event Class" on page 15-20.

The subject matter in this section is primarily intended for J2EE experts. It includes information about the following subjects:

- How Events Work in Campaigns—Contains a brief introduction about the relationship between events and campaigns.
- How the Event Service Works—Provides insight into the Event service so you can take full advantage of the Event and Behavior Tracking capabilities.
- How to Use Standard Events—Tells you what events are shipped with WebLogic Portal and how to generate them.
- Creating Custom Events—Guides you through the process of creating Event and Behavior Tracking classes and listeners.
- How to Enable Behavior Tracking—Tells you how to enable Behavior Tracking as a service.
- Debugging the Event Service—How to activate debugging for events.
- Registering Custom Events—How to register a custom event so it can be used by a campaign.
- Activating Behavior Tracking—How to turn on Behavior Tracking.

How Events Work in Campaigns

An event is generated by a visitor action, such as viewing a product. The Event service notifies all event listeners that it has detected the event. The event listener for the Campaign service activates a campaign scenario. Using a set of rules that match users with content, the campaign initiates an action. The available actions are listed below:

- Displaying specific content on a Web page, such as an ad.
- Sending a promotional email to a customer.
- Offering the customer a discount.

Note: For more information about campaigns, see the E-Business Control Center online help.

How the Event Service Works

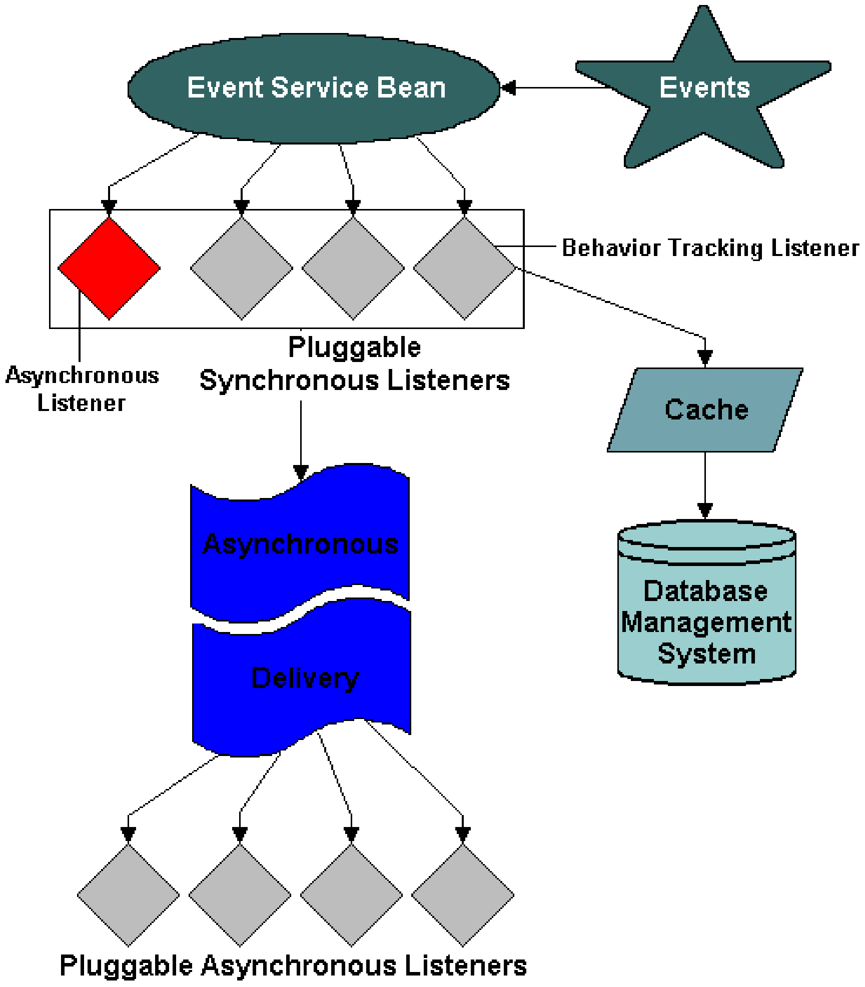
Understanding how the Event service works helps you use events and provides information you need to generate events and design a custom event.

About the Event service The Event service is an extensible, general purpose, event construction and propagation system. As shown in Figure 15-1, an event is generated by a trigger, such as a JSP tag, which creates the event object, locates the Event service bean, and passes the event object to the Event service. The Event service works with plug-in listeners that disseminate events to listeners interested in receiving the events. At creation time, each event listener returns the list of event types that it wants to receive. When the Event service receives an event, it checks the type of the event and sends the event to all listeners that are subscribed to receive that event's type.

Listener types The Event service has two sets of listeners: those that respond to events synchronously and those that respond to events asynchronously. The synchronous listeners use the thread of execution that created and transmitted the event to perform actions in response to that event. Behavior Tracking listeners use only the synchronous listeners. The asynchronous listeners receive the event from the thread where it was created and some time later, handles the event in a different thread of execution. The asynchronous service exists so that long-running event handlers can execute without delaying the application from a Web site visitor's perspective.

Whether a particular plug-in listener is installed on the synchronous or the asynchronous side of the Event service is based on the requirements of the application. Configuration of the Event service is done using the WebLogic Server Administration Console.

Figure 15-1 Event Mechanism


Event Service Bean
Events
Behavior Tracking Listener
Asynchronous Listener
Pluggable Synchronous Listeners
Cache
Database Management System
Asynchronous
Delivery
Pluggable Asynchronous Listeners

Event listeners implement the `com.bea.p13n.events.EventListener` interface. The interface defines signatures for two public methods:

- `public String[] getTypes()`
- `public void handleEvent( Event theEvent )`

The first method returns a list of event types that the listener is interested in receiving from the Event service. For example, if a listener is designed to receive events of type *Foo*, the listener returns *Foo* as an item in the array returned from invoking `getTypes()` on the listener. The second method is invoked when an event is passed to the listener. A listener has no knowledge of whether it is synchronous or asynchronous.

If you wish to create a listener interested in only campaign events, you would add the listener's fully-qualified classname in the WebLogic Server Administration Console. The listener would implement the `EventListener` interface and return the following event types:

```
{"ClickCampaignEvent","DisplayCampaignEvent","CampaignUserActiv
ityEvent" }
```

when its `getTypes()` method is invoked.

After the listener is installed, events of one of these three types arrive through the listener's `handleEvent( Event theEvent )` interface.

The Asynchronous Delivery graphic in Figure 15-1 indicates that the asynchronous event handler receives events transmitted asynchronously from the synchronous side of the Event service. It then dispatches events to the pluggable asynchronous listeners based on the event types each listener is subscribed to receive.

How Event Sequences Work

Figure 15-2 and Figure 15-3 provide a sample of the generation of events. These figures are intended to give you a sense of the order in which events fire, not a comprehensive examination of event sequencing. The intent is to show you how events provide insight into the visitor life cycle and how and when you can use events in your application.

Figure 15-2 Event Sequence Sample—Part 1

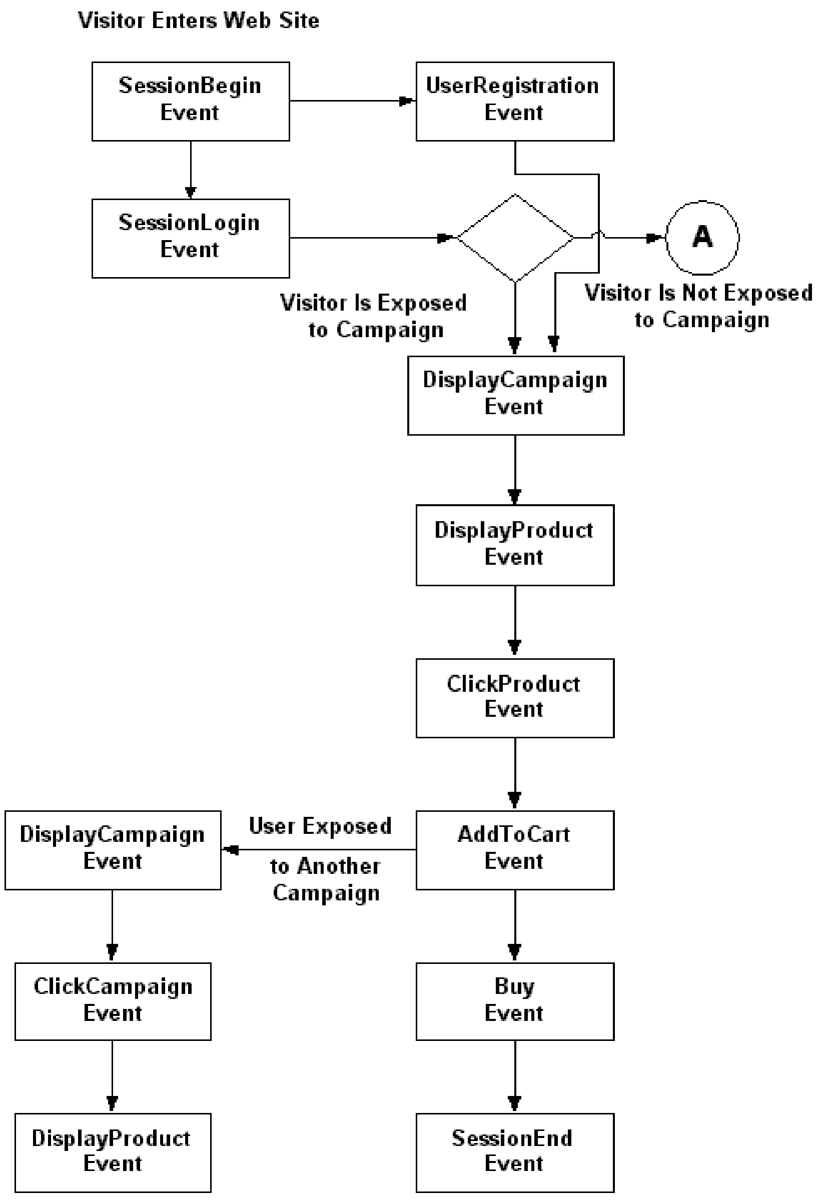
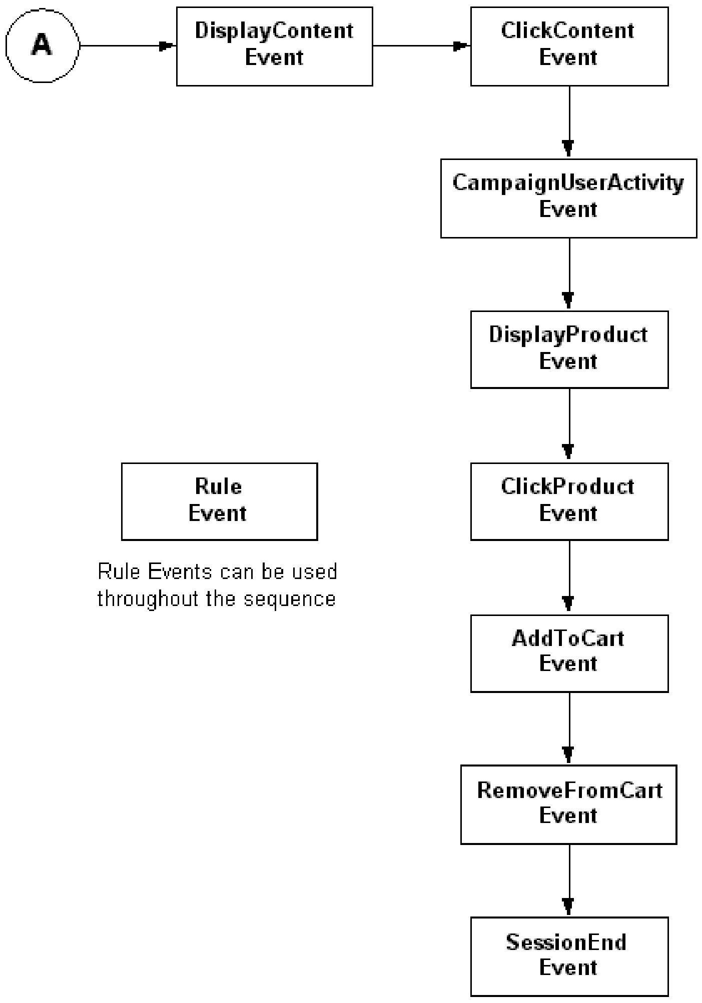


Figure 15-3 Event Sequence Sample—Part 2



How to Use Standard Events

This section provides general information about how to use the standard events provided with WebLogic Portal. For specific information about each event, see Appendix A, “Event Descriptions.” Appendix A contains a description of each kind of event, what generates the event, the class where event generation occurs, an example of usage, and the type of data within each event object.

All WebLogic Portal standard events contain the following basic information:

- Application from which the event was generated.
- Time of event.
- Type of event.
- Session ID.
- User ID (Null, if visitor not logged on.).
- Event specific information.

WebLogic Portal events are organized into categories. The following list presents each type of event category along with a brief description of what actions generates the event:

- **Session Events**: The start time, end time, and login time of a visitor’s session.
  - SessionBeginEvent
  - SessionEndEvent
  - SessionLoginEvent
- **User Registration Event:** The visitor registers on the Web site.
  - UserRegistrationEvent
- **Product Events**: The visitor is presented with a product or clicks (selects) the presented product.
  - ClickProductEvent
  - DisplayProductEvent

- **Content Events**: The visitor is presented some content, such as an ad, or clicks (selects) the presented content.
  - ClickContentEvent
  - DisplayContentEvent
- **Cart Events**: An item is added, removed, or updated to the visitor's shopping cart. These events are also generated when an entire order is purchased.
  - AddToCartEvent
  - RemoveFromCartEvent
  - PurchaseCartEvent
- **Buy Event**: The visitor completes the purchase of one or more items.
  - BuyEvent
- **Rules Event**: The rules that are fired as a visitor navigates a Web site.
  - RuleEvent
- **Campaign Events**: The events generated within the context of a campaign.
  - CampaignUserActivityEvent
  - DisplayCampaignEvent
  - ClickCampaignEvent

Servlet Lifecycle Events and Servlet Filter Events

These events are defined as part of the Servlet 2.3 API lifecycle events:

- SessionBeginEvent
- SessionEndEvent

They are listeners on the session `Created()` and session `Destroyed()` events, which are generated by the servlets defined in the `web.xml` file. One `web.xml` file exists for each application. For example, in wlcsApp E-Commerce Application, this file is located at:

```
<BEA_HOME>\weblogic700\portal\applications\wlcsApp\wlcs\WEB-INF
```

The following events are generated by JSP tags and filtered by the Servlet 2.3 `<filter>` element:

- ClickContentEvent
- ClickProductEvent
- ClickCampaignEvent

For each Web page displayed, the Web Application servlet checks for the presence of a click event in the `HttpServletRequest`. Each page click is then filtered by a Web Application servlet as defined by the Servlet 2.3 `<filter>` element. The click events are generated automatically when the `<filter>` element is called on each invocation of the servlet. The `ClickThroughFilter` determines which type of event is generated by checking the event type in the `HttpServletRequest`. The valid types are defined at the following locations:

- `<BEA_HOME>\weblogic700\portal\classes\campaign\campaign-app\*.properties`
- `<BEA_HOME>\weblogic700\portal\classes\commerce\commerce-app\*.properties`
- `<BEA_HOME>\weblogic700\portal\classes\p13n\p13n-app\*.properties`

Generating Login and Creation Events

This section describes different methods you can use to generate login and user registration events.

SessionLoginEvent You can generate the `SessionLoginEvent` in either of the following ways:

- If you are manually using the `<um:login>` tag or `weblogic.servlet.security.ServletAuthentication` to handle login, use the `com.bea.p13n.tracking.TrackingEventHelper.dispatchSessionLoginEvent()` method.
- If you are directly using `j_security_check FORM`-based login, register the `com.bea.p13n.servlets.P13NAuthFilter` as the `<auth-filter>` in your Web Application's `WEB-INF\weblogic.xml` file. You do not need to code a JSP or Webflow Processor.

UserRegistrationEvent Use the `com.bea.p13n.tracking.TrackingEventHelper.dispatchUserRegistrationEvent()` method to generate the UserRegistrationEvent. You should generate this event after the SessionLoginEvent (which should occur during user creation). You can use either an Input Processor or in a JSP.

Webflow If you are using the Portal Webflow framework, the SessionLoginEvent and the UserRegistrationEvent are generated automatically from the `com.bea.portal.appflow.processor.security.PostLoginProcessor` in the security Webflow as needed.

Adding or Customizing Event Generators

Standard events are generated at important points in an e-commerce site. The components that enable events include Java APIs, JSP tags, JSP scriptlets, Webflow input processors, Pipeline components, content selectors, and classification advislets. You can add or customize generators for each of the following events:

- `DisplayContentEvent`
- `DisplayProductEvent`
- `ClickContentEvent`
- `ClickProductEvent`

Each event is generated by JSP tags. You can use the JSP tags that initiate these events to specify which products and what content generate these events. For example, in the wlcsApp E-Commerce Application, the JSP tag for the `DisplayProductEvent` is located in the `details.jsp`.

The tag shown in Listing 15-1 generates an event for any product displayed on a catalog detail page. If you want to generate an event for one particular product, you can write a scriptlet that keys off the SKU for that product.

Listing 15-1 JSP Tag

```
<%-- once the product is displayed, fire off a displayProductEvent --%>
<productTracking:displayProductEvent documentId="<%= item.getName() %>"
        documentType="<%= DisplayProductEvent.ITEM_BROWSE %>"
```

```
            sku="<%= item.getKey().getIdentifier() %>" />
```

When you add a JSP tag for an event, you should include a reference to the tag library descriptor, as shown below:

```
<%@ taglib uri="productTracking.tld" prefix="productTracking" %>
```

The `details.jsp` is located in the following directory:

```
<BEA_HOME>\weblogic700\portal\applications\wlcsApp\wlcs\commerc
e\catalog
```

Creating Custom Events

This section provides the information necessary to write a custom event. You can create a custom event for anything you wish to track. If you want your event to be recorded using the WebLogic Portal persistence mechanism, create a Behavior Tracking event, as described in "Writing a Behavior Tracking Event Class" on page 15-20.

Idea for a custom event You could create an event that would tell you which pages are displayed for each visitor. You could then use the information to determine how many pages are viewed on average per session and which pages are the most popular. Additionally, marketing professionals could use this event when developing promotional campaigns that are based on the display of particular pages.

To demonstrate how to write a custom event, a simple example is provided. Each section references and expands the example.

The creation of a custom event is a multiple-step process. The following list provides an overview of the process and references the information not covered in this topic:

1. Write the code that defines the event.
2. Write the code that defines the event listener.
3. Install the listener class in the Event Service.

4. Write the code to generate the event with a JSP tag or an API call.
5. Register the event, if it is to be used in a campaign.
6. To record the event data to the `EVENT` table, create an entry for the event in the `EVENT_TYPE` table. For more information, see “Persisting Behavioral Tracking Data” in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/sysadmin.htm#1194894.

Writing the Custom Event Class

To create a custom event, take the following steps:

1. **Write the event class** This class encapsulates all the necessary information for correctly interpreting and handling the event when it arrives at a listener.

   All custom events must subclass the `com.bea.p13n.events.Event` class. This base class handles setting and retrieving an event’s timestamp and type and provided access to the custom event’s attributes. Two `Event` class methods set and retrieve attributes:

   ```
   setAttribute( String theKey, Serializable theValue )
   getAttribute( String theKey )
   ```

   These methods can be called from the custom event’s constructor to set attributes specific to the new event. Keep in mind that all objects set as values in the `Event` object *must* be Java serializable.

   The `getTimeStamp()` method returns the date of the event’s creation in milliseconds. The type of an event is accessed using the `Event` class’s `getType()` method. The timestamp and type of an `Event` object instance can be set only at creation time in the `Event` constructor. If not specified, the event is timestamped automatically when it is created. The application attribute is set automatically, either from the application in which the event was created or from the Event service EJB (Enterprise JavaBean) application.

   Example To illustrate the process of creating a custom event, a simple example is presented here, called `TestEvent`. The example is a basic demonstration of how to create an event subclass. An actual custom event would probably be more elaborate.

2. **Create the type** A custom event must first have a type. This type should be passed to the superclass constructor (for example, in the `Event` class). This type is returned at `getType()` invocations on custom-event object instances. For example:

   ```
   /** Event Type */
   public static final String TYPE = "TestEvent";
   ```

   To properly initialize the `Event` base class of the custom event object, the value `TYPE` is passed to the event constructor. The type of all events must be a simple Java string object.

3. **Define the keys** After defining the type, you must define the keys that access the attributes stored in the custom event. These attributes can be given values in the constructor. For example, the `TestEvent` class has two properties, `description` and `Zip Code`; the type of the value associated with `description` is a `String` and `Zip Code` is an `Integer`. The keys are defined as follows:

   ```
   /**
    * Event attribute key name for the first user defined property
    * Attribute value is a String
    */
   public static final String DESCRIPTION = "description";

   /**
    * Recall that all attribute values must be serializable
    * Event attribute key name for the second user defined
    * property
    * Attribute value is an Integer
    */
   public static final String ZIP_CODE = "Zip Code";
   ```

   Finally, a constructor brings the event type and the process of setting attributes together to create an event object. The constructor looks like:

   ```
   /**
    * Create a new TestEvent
    *
    *
    * @param desc The description of this event
    * @param zip The Zip Code
    */
    public TestEvent( String desc, Integer zip )
    {
      /* calls the Event class constructor with this event's
         type */
      super( TYPE );
   ```

```
            if( desc != null )
                setAttribute( DESCRIPTION, desc );

            if( zip != null )
                setAttribute( ZIP_CODE, zip );
          }
```

All the parts put together The entire custom event class is shown in Listing 15-2.

Listing 15-2 TestEvent Class

```
/* Start TestEvent class */

public class TestEvent
  extends com.bea.p13n.events.Event
{
  /** Event Type */
  public static final String TYPE = "TestEvent";

  /**
   * Event attribute key name for the first user defined property
   * Attribute value is a String
   */
   public static final String DESCRIPTION = "description";

  /**
   * Event attribute key name for the second user defined property
   * Attribute value is an Integer
   */
   public static final String ZIP_CODE = "Zip Code";

  /**
   * Crate a new TestEvent
   *
   *
   * @param desc The description of this event
   * @param zip The Zip Code
   */
   public TestEvent( String desc, Integer zip )
   {
     /* calls the Event class constructor with this event's type */
     super( TYPE );

     if( descriptionValue != null )
         setAttribute( DESCRIPTION, desc );
     if( ZipCodeValue != null )
         setAttribute( ZIP_CODE, zip );
   }
}
/* End TestEvent class */
```

About the example The example in Listing 15-2 shows you how to use the fundamental aspects of the `Event` base class and the Event service. An actual custom event constructor would probably be more complex. For example, it might check for default values or disallow null attributes. Additionally, the custom-event object might have more methods or member data.

Note: In order for a custom event to be used by a campaign, it must contain the following objects as attributes:

- `Request` (key = “`request`”) – `Request` of type `com.bea.p13n.http.Request`. It should be created with a constructor that takes the `HttpServletRequest` as a parameter.
- `Session` (key = “`session`”) – `Session` of type `com.bea.p13n.http.Session`. Created with a constructor that takes the `HttpServletRequest` as its parameter.
- `UserId` (key = “`userId`”) – A string that contains the username.

Saving the file You can save the file anywhere you like as long as it is in the enterprise application classpath used by WebLogic Server.

Writing the Custom Event Listener

In order to listen for an event, you must define an event listener. All event listeners must implement the `com.bea.p13n.events.EventListener` interface and have a no arguments (default) constructor. This interface specifies two methods that are fundamental to transmitting events of a given type to interested listeners:

```
public String[] getTypes()

public void handleEvent( Event ev )
```

The first method returns the types, in a string array, that the listener is interested in receiving. The Event service dispatches events of a given type to listeners that return the event's type in the types array. When the Event service has determined that a given listener has registered to receive the type of the current event, an event of that type is dispatched to the listener using the `handleEvent( Event ev )` call.

Implement both event listener methods When writing a custom event listener, both methods must be implemented from the `EventListener` interface. Continuing with the `TestEvent` example, the `TestEventListener` listens for instances of `TestEvent` that are sent through the Event service. This can be specified as follows:

```
/** The types this listener is interested in */
private String[] eventTypes = {"TestEvent"};

/**
  The method invoked by the event service to determine the
  types to propagate to this listener.
 */
public String[] getTypes()
{
  return eventTypes;
}
```

To handle the event, the `handleEvent( Event evt )` method is implemented as follows:

```
/**
 * Handle events that are sent from the event service
 */
public void handleEvent( Event ev )
{
  System.out.println("TestListener::handleEvent " +
                     " -> received an event" +
                     " of type: " + ev.getType() );

  /* Do the work here */

}
```

Putting all of these pieces together with a constructor, Listing 15-3 shows a simple event listener that registers to receive `TestEvent` objects.

Listing 15-3 Event Listener

```
import com.bea.p13n.events.EventListener;
import com.bea.p13n.events.Event;
```

```
    /**
     * TestListener to demonstrate the ease with which listeners can be plugged
     * into the behavior tracking system.
     *
     * This class should be added to the property eventService.listeners
     * in order to receive events.  The fully qualified classname must be added
     * to this property; don't forget to add the ",\" at the end of the previous
     * line or the properties parser will not find the new classname.
     *
     * The types of events that are heard are listed in the eventTypes
     * String array.  Add and remove strings of that type as necessary.
     *
     * @author Copyright (c) 2001 by BEA Systems, Inc. All Rights Reserved.
     */
    public class TestListener
        implements EventListener
    {

    private String[] eventTypes = {"TestEvent"};

    public TestListener()
    {
    }

    public String[] getTypes()
    {
        return eventTypes;
    }

    public void handleEvent( Event ev )
    {
         System.out.println("TestListener::handleEvent -> received an event" +
                              " of type: " + ev.getType() );

         return;
    }
}
```

Event listeners should be generic As with writing a simple event, writing a simple `EventListener` is also straightforward. Any event listener's internals should be generic; the same `TestEventListener` instance may not handle all `TestEvent` objects. Therefore `TestEventListener` should be entirely stateless and should operate on data that is contained in the event object or stored externally in a database.

Note: Multiple instances of any listener may execute concurrently.

Installing the Listener Class in the Event Service

Notes: This section provides information on how to add a listener class in the Sample Portal. For your application, you would use similar steps.

If the Event service does not exist as a service for your application, use WebLogic Server Administration Console to add it.

To enable Behavior Tracking, you must add Behavior Tracking as a synchronous listener to the Event service.

Add a listener To add a synchronous or asynchronous listener, take the following steps:

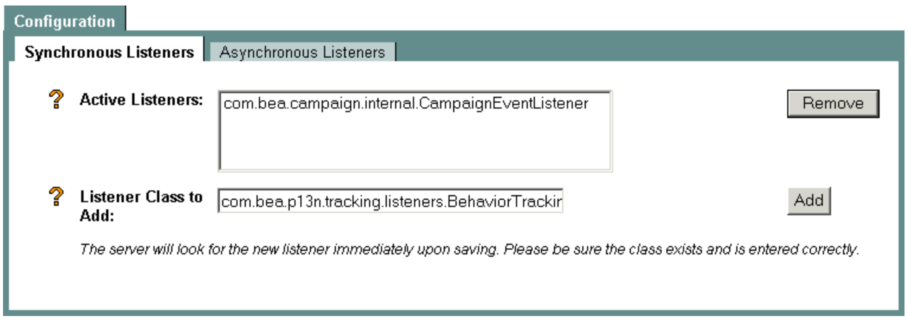
Note: Behavior Tracking listeners can only be implemented as synchronous listeners.

1. In the WebLogic Server Administration Console, navigate to Synchronous or Asynchronous Listeners tab in the node tree for sampleportalDomain as follows:

   http://<hostname>:<port>/console →sampleportalDomain →Deployments → sampleportal →Service Configurations →Event Service →Configuration Tab → Synchronous Listeners or Asynchronous Listeners

2. Add the synchronous or asynchronous listener to the corresponding fields, as shown in Figure 15-5.

Figure 15-4 WebLogic Server Administration Console—Event Service



Active Listeners: com.bea.campaign.internal.CampaignEventListener

Listener Class to Add: com.bea.p13n.tracking.listeners.BehaviorTrackir

The server will look for the new listener immediately upon saving. Please be sure the class exists and is entered correctly.

Writing a Behavior Tracking Event Class

A Behavior Tracking event is a special type of event that tracks a visitor's interactions with a Web site. E-analysis systems use the data gathered from Behavior Tracking events to evaluate visitor behavior. The evaluation is primarily used for campaign development and optimizing visitor experience on a Web site.

Example A Behavior Tracking event and its listeners are created in much the same way as the `TestEvent` class and `TestEventListener` examples. A simple example is also presented here. The example tracking event is called `TestTrackingEvent`. All Behavior Tracking events persisted (recorded) to a database for use with BEA Behavior Tracking are handled by the `com.bea.p13n.tracking.listeners.BehaviorTrackingListener`. The `BehaviorTrackingListener` extends the `com.bea.p13n.events.EventListener` class.

Redeploy After the `BehaviorTrackingListener` is defined as a listener on the Event service, you need to redeploy the application before it can receive and persist Behavior Tracking events.

About the buffer This listener receives events from the Event service and adds them to a buffer that is intermittently persisted to the Event tables in the database. The frequency of sweeping of events from the buffer is controlled by the following properties on the Behavior Tracking service.

- `MaxBufferSize` – Sets the maximum size of the event buffer. Setting this to 0 means all events are persisted as they are received.
- `SweepInterval` – Sets the interval, in seconds, at which to check the buffers to see whether events in the buffer must be persisted. Events are persisted when either the maximum buffer size (`MaxBufferSize`) is reached or the maximum time to wait in the buffer (`SweepMaxTime`) has been exceeded.
- `SweepMaxTime` – Set the time, in seconds, to wait before forcing a flush to the database. This is the longest amount of time that an event can exist in any cache.

Optimizing Tune these properties to optimize performance. A buffer sweep should be performed often enough that writing to the database is not too time consuming but not so frequent that the operation is wasteful.

Configuring Events Buffer Sweeping

Notes: This section provides information on configuring buffer sweeping in the Sample Portal. For your application, you would use similar steps.

If the Event service does not exist as a service for your application, use WebLogic Server Administration Console to add it.

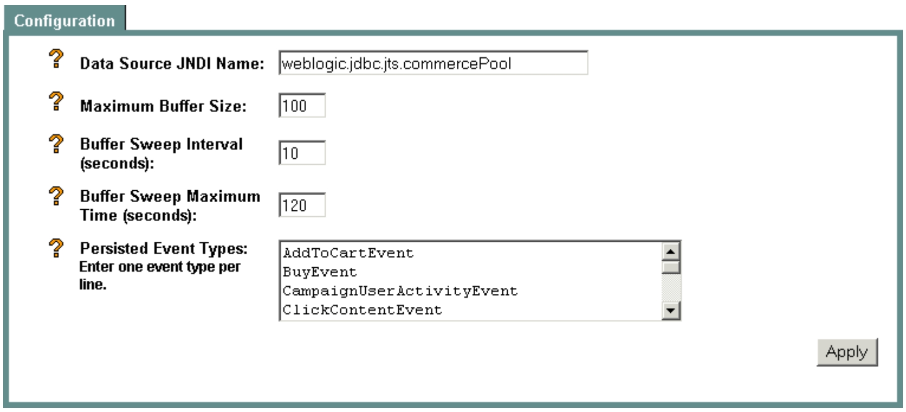
To configure the sweeping of the events buffer, take the following steps:

1. In the WebLogic Server Administration Console, navigate to Behavior Tracking in the node tree for sampleportalDomain as follows:

   http://<hostname>:<port>/console →sampleportalDomain →Deployments → sampleportal →Service Configurations →Behavior Tracking Service

2. Enter the new buffer values in the appropriate fields, as shown in Figure 15-5.

Figure 15-5 WebLogic Server Administration Console—Behavior Tracking



Data Source JNDI Name:	weblogic.jdbc.jts.commercePool
Maximum Buffer Size:	100
Buffer Sweep Interval (seconds):	10
Buffer Sweep Maximum Time (seconds):	120
Persisted Event Types: Enter one event type per line.	AddToCartEvent BuyEvent CampaignUserActivityEvent ClickContentEvent

Facilitating OffLine Processing

Behavior Tracking events are designed to be persisted to a table in the database, called the `EVENT` table. Part of the process of recording data from Behavior Tracking events is creating an XML representation of the data, which is stored in the `xml_definition` column of the `EVENT` table. If you are planning to use the BEA Behavior Tracking event persistence mechanism, you *must* persist events in this location. Therefore, to persist events in the provided `EVENT` table, your custom event must conform to the descriptions in this section so that it is created and persisted properly.

XML-XSD schema To formally specify the data contained in a Behavior Tracking event, you need to develop an XML-XSD schema for the new event. While XSDs are not used internally to verify the creation of XML, the XML that is created represents the event's data in the database. If the event class is properly developed and used, it will conform to the XML-XSD schema. With an XSD document, development of the constructor and attribute keys for a Behavior Tracking event follows easily. The specific data elements for each standard event are shown in the `XML_DEFINITION` Data Elements table in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/sysadmin.htm#1195110.

Association between files To correctly turn a Behavior Tracking event into an XML representation, the Behavior Tracking event must have several pieces of member data that fully describe an XML instance document for the schema associated with the event type. This data describes the namespace and XSD file associated with the event. For example, Listing 15-4 and Listing 15-5 show the association between the following files:

`com.bea.campaign.tracking.events.ClickCampaignEvent` and

`/lib/schema/tracking-click-campaign-1_0_1.xsd` in `<BEA_HOME>\weblogic700\portal\\lib\campaign\ejb\campaign.jar`.

For more examples, look at the existing XSD files.

Listing 15-4 ClickCampaignEvent.java

```
/**
   Event for tracking click of campaign
 */
public class ClickCampaignEvent
    extends ClickEvent
{
    /** The event type */
    public static final String TYPE = "ClickCampaignEvent";

    /**
       The XML namespace for this event
    */
    private static final String XML_NAMESPACE =
       "http://www.bea.com/servers/commerce/xsd/tracking/click-campaign/1.01";
```

```
/**
   The XSD file containing the schema for this event
*/
private static final String XSD_FILE = "tracking-click-campaign-1_0_1.xsd";

/**
 * Event attribute key name for the campaign id
 * Attribute value is a String
 */
public static final String CAMPAIGN_ID = "campaign-id";

/**
 * Event attribute key name for the scenario id
 * Attribute value is a String
 */
public static final String SCENARIO_ID = "scenario-id";

/**
 * Event attribute key name for storefront (aka application-name)
 * Attribute value is a String
 */
public static final String APPLICATION_NAME = "application-name";

/**
 * Event attribute key name for item category id
 * Attribute value is a String
 */
 public static final String PLACEHOLDER_ID = "placeholder-id";

/**
  Suggestions for entry into the documentType data passed to the constructor
   Attribute value is a String
*/
public static final String BANNER_AD_PROMOTION = "bannerAdPromotion";

/**
   These are the keys and their order for elements that
   will be present in the XML representing this object
*/
private static final String localSchemaKeys[] =
{
    APPLICATION, SESSION_ID, USER_ID, DOCUMENT_TYPE, DOCUMENT_ID,
    CAMPAIGN_ID, SCENARIO_ID, APPLICATION_NAME, PLACEHOLDER_ID
};

/**
 * Create a new ClickCampaignEvent.
 *
```

```
     * @param theSessionId from HttpSession.getId()
     * @param theUserId from HttpServletRequest.getRemoteUser() or
     * equivalent (null if unknown)
     * @param theRequest the http servlet request object
     * @param aDocumentType Document Type for the clicked content (optionally
     * null)
     * @param aDocumentId Document ID for the clicked content (optionally null)
     * @param aCampaignId campaign id for the campaign from which the item was
     * clicked
     * @param aScenarioId scenario id for the scenario (within the campaign)
     * for which the item was clicked
     * @param aApplicationName application name (aka storefront) (optionally
     * null)
     * @param aPlaceholderId a placeholder id
     */
    public ClickCampaignEvent( String theSessionId,
                               String theUserId,
                               HttpServletRequest theRequest,
                               String aDocumentType,
                               String aDocumentId,
                               String aCampaignId,
                               String aScenarioId,
                               String aApplicationName,
                               String aPlaceholderId )
    {
        super( TYPE,
               theSessionId,
               theUserId,
               XML_NAMESPACE,
               XSD_FILE,
               localSchemaKeys,
               theRequest,
               aDocumentType,
               aDocumentId);

        if( aCampaignId != null ) setAttribute( CAMPAIGN_ID, aCampaignId );
        if( aScenarioId != null ) setAttribute( SCENARIO_ID, aScenarioId );
        if( aApplicationName != null ) setAttribute( APPLICATION_NAME,
            aApplicationName );
        if( aPlaceholderId != null ) setAttribute( PLACEHOLDER_ID,
            aPlaceholderId );
    }
}
```

Event and XSD cross-referenced Notice the cross-reference between `ClickCampaignEvent` and the XSD schema. This association allows the Behavior Tracking data to be properly recorded in the database.

Listing 15-5 Corresponding XSD Schema

```
<xsd:schema

targetNamespace="http://www.bea.com/servers/commerce/xsd/tracking
/click-campaign/1.0.1"

xmlns="http://www.bea.com/servers/commerce/xsd/tracking/click-cam
paign/1.0.1"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/XMLSchema
                        http://www.w3.org/2001/XMLSchema.xsd"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsd:element name="ClickCampaignEvent">
        <xsd:complexType>
            <xsd:sequence>
               <xsd:element ref="application"/>
               <xsd:element ref="event-date"/>
               <xsd:element ref="event-type"/>
               <xsd:element ref="session-id"/>
               <xsd:element ref="user-id" minOccurs="0"/>
               <xsd:element ref="document-type" minOccurs="0"/>
               <xsd:element ref="document-id" minOccurs="0"/>
               <xsd:element ref="campaign-id"/>
               <xsd:element ref="scenario-id"/>
               <xsd:element ref="application-name" minOccurs="0"/>
               <xsd:element ref="placeholder-id" minOccurs="0"/>
            </xsd:sequence>
            <!-- types = banner-ad-promotion -->
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="application" type="xsd:string"/>
    <xsd:element name="event-date" type="xsd:string"/>
    <xsd:element name="event-type" type="xsd:string"/>
    <xsd:element name="session-id" type="xsd:string"/>
    <xsd:element name="user-id" type="xsd:string"/>
    <xsd:element name="document-type" type="xsd:string"/>
    <!-- types = banner-ad-promotion -->
    <xsd:element name="document-id" type="xsd:string"/>
    <xsd:element name="campaign-id" type="xsd:string"/>
    <xsd:element name="scenario-id" type="xsd:string"/>
    <xsd:element name="application-name" type="xsd:string"/>
    <xsd:element name="placeholder-id" type="xsd:string"/>
</xsd:schema>
```

List the keys The source code for your Behavior Tracking event should also list the keys and their order for creating an XML instance document from an event object. For an example, see Listing 15-4. The structure of an XSD document and details on XML namespaces can be found at http://www.w3.org/XML/Schema. Several XSD schemas for BEA Behavior Tracking events can be found in `/lib/schema` at the following location:

```
<BEA_HOME>\weblogic700\portal\lib\p13n\ejb\events.jar
```

Specify namespace and schema The namespace and schema are specified as:

```
/**
 The XML namespace for this event
 */
private static final String XML_NAMESPACE=
    "http://<your URI>/testtracking";

/**
  The XSD file containing the schema for this event
 */
private static final String XSD_FILE="TestTrackingEvent.xsd";
```

Note: These values are used when creating an instance document to populate the fields.

The `schemaKeys` are a list of strings which are keys to the event class's `getAttribute` and `setAttribute` methods. These keys are used to extract the data that populate elements in the XML instance document which represent the Behavior Tracking event. The keys should be listed in an array that consists of string-typed objects. Their order specifies the order in which they appear in the XML instance document. In the XSD files that the Behavior Tracking system generates, the order of the elements is important; an XML file will not validate with an XSD file if elements are out of order. Elements can be omitted by using the XML `numOccurs` keyword and setting the value to zero. For examples of how this is done, see the XSD schemas for BEA Behavior Tracking events in `/lib/schema`, at the following location:

```
<BEA_HOME>\weblogic700\portal\\lib\p13n\ejb\events.jar
```

Structuring the array An example array for the Behavior Tracking version of the `TestEvent` described above might appear as:

```
/**
 These are the keys and their order for elements that
 will be present in the XML representing this object.
 */
private static final String localSchemaKeys[] =
{
    SESSION_ID, USER_ID, USER_PROPERTY_ONE_KEY,
        USER_PROPERTY_TWO_KEY
};
```

Data elements The `SESSION_ID` and the `USER_ID` are data elements in the `localSchemaKeys` array that are useful in implementing a tracking event. The `SESSION_ID` is the WebLogic Server session ID that is created for every session object. (For more information, see the WebLogic Server documentation at http://edocs.bea.com/wls/docs70/index.html.) The `USER_ID` field (which may be null) is the username of the Web site visitor associated with the session from which the event was generated. For some events, a user may not be associated with an event; as previously mentioned, the `numOccurs` for the `USER_ID` field in an XSD file should be zero. To persist events in the EVENT table, the `SESSION_ID` must be non-null.

Other attributes All Behavior Tracking events must extend the `com.bea.p13n.tracking.events.TrackingEvent` class. This class defines three keys that are useful for setting attributes for all tracking events, as follows:

- `TrackingEvent.SESSION_ID`
- `TrackingEvent.USER_ID`
- `TrackingEvent.REQUEST`

These keys are used in `setAttribute` calls made in the `TrackingEvent` constructor when setting the `SESSION_ID`, `USER_ID`, and `REQUEST` (an `HttpServletRequest` object), respectively. They should also be used to retrieve values associated with each key when invoking `Event.getAttribute(String Key)` on event objects that extend `TrackingEvent`.

Writing a TrackingEvent Base Class Constructor

The `TrackingEvent` base class has a constructor that is more complicated than the `Event` class's constructor. The `Event` constructor is invoked by the `super( String eventType )` call in the `TrackingEvent` constructor. The `TrackingEvent` constructors are shown in Listing 15-6 and Listing 15-7.

Listing 15-6 Tracking Event Constructor—Example 1

```
/**
 * Create a new TrackingEvent.
 *
 * @param theEventType the event's type
 * @param theSessionId from HttpSession.getId()
 * @param theUserId from HttpServletRequest.getRemoteUser() or equivalent
 * (null if unknown)
 * @param theXMLNamespace the namespace for an XML representation of this event
 * type
 * @param theXSDFile the file that contains the schema which specifies and
 * enforces typing on the data in the XML file
 * @param theSchemaKeys the list of keys (in their order in the XSD schema)
 * representing the data to be persisted in this event's XML
      */
public TrackingEvent( String theEventType,
                      String theSessionId,
                      String theUserId,
                      String theXMLNamespace,
                      String theXSDFile,
                      String[] theSchemaKeys )
```

The `TrackingEvent` constructor shown in Listing 15-7 takes an `HttpServletRequest` object.

Listing 15-7 Tracking Event Constructor—Example 2

```
/**
 * Create a new TrackingEvent.
 *
 * @param theEventType the event's type
 * @param theSessionId from HttpSession.getId()
 * @param theUserId from HttpServletRequest.getRemoteUser() or equivalent
 * (null if unknown)
 * @param theXMLNamespace the namespace for an XML representation of this event
 * type
 * @param theXSDFile the file that contains the schema which specifies and
 * enforces typing on the data in the XML file
 * @param theSchemaKeys the list of keys (in their order in the XSD schema)
 * representing the data to be persisted in this event's XML
 * @param theRequest the http servlet request object
 */
public TrackingEvent( String theEventType,
                      String theSessionId,
                      String theUserId,
                      String theXMLNamespace,
                      String theXSDFile,
                      String[] theSchemaKeys,
                      HttpServletRequest theRequest )
```

About the constructors In the first constructor, shown in Listing 15-6, the only data that can be null is `theUerId`; all other data is required so that the tracking event is correctly persisted to the `EVENT` table. In the second constructor, shown in Listing 15-7, the `HttpServletRequest` object can be passed in from generating locations where the `HttpServletRequest` object is available. This object provides the data needed to fire rules against event instances.

Note: In order to fire rules on a custom Behavior Tracking event, the `HttpServletRequest` and the `USER_ID` must be non-null. Generally, a non-null `USER_ID` means that a visitor is logged into a Web site. Rules cannot be fired on an event with a null user.

The `TestTrackingEvent` constructor is shown in Listing 15-8.

Listing 15-8 TestTrackingEvent Constructor

```
/**
 * Create a new TestTrackingEvent
 *
 * @param theSessionId from HttpSession.getId()
 * @param theUserId from HttpServletRequest.getRemoteUser() or equivalent
 * (null if unknown)
 * @param userPropertyOne some user defined property typed as a String
 * @param userPropertyTwo another user defined property typed as a Double
 */
public TestTrackingEvent( String theSessionId,
                          String theUserId,
                          String userPropertyOneValue,
                          Double userPropertyTwoValue )
{
    super( TYPE, theSessionId, theUserId, XML_NAMESPACE, XSD_FILE,
           localSchemaKeys );

    if( userPropertyOneValue != null )
        setAttribute( USER_PROPERTY_ONE_KEY, userPropertyOneValue );

    if( userPropertyTwoValue != null )
```

```
        setAttribute( USER_PROPERTY_TWO_KEY, userPropertyTwoValue );

}
```

This constructor calls the `TrackingEvent` constructor to populate the required values, then sets the attributes necessary for this particular Behavior Tracking event type.

The entire `TestTrackingEvent` is shown in Listing 15-9.

Listing 15-9 TestTracking Event

```
import com.bea.p13n.tracking.events.TrackingEvent;

/**
 * Test, user-defined behavior tracking event.
 *
 * This event can be persisted to the database.
 *
*/
public class TestTrackingEvent
    extends TrackingEvent
{

    /** Event type */
    public static final String TYPE = "TestTrackingEvent";

    /**
      The XML namespace for this event
     */
    private static final String XML_NAMESPACE="http://<your URI>/testtracking";

    /**
      The XSD file containing the schema for this event
     */
    private static final String XSD_FILE="TestTrackingEvent.xsd";

    /**
     * Event attribute key name for the first user defined property
     * Attribute value is a String
     */
    public static final String USER_PROPERTY_ONE_KEY = "userPropertyOne";

    /**
     * Event attribute key name for the second user defined property
     * Attribute value is a Double
     */
    public static final String USER_PROPERTY_TWO_KEY = "userPropertyTwo";

    /**
      These are the keys and their order for elements that
      will be present in the XML representing ths object.
     */
    private static final String localSchemaKeys[] =
    {
        SESSION_ID, USER_ID, USER_PROPERTY_ONE_KEY, USER_PROPERTY_TWO_KEY
    };

    /**
     * Create a new TestTrackingEvent
     *
     * @param theSessionId from HttpSession.getId()
     * @param theUserId from HttpServletRequest.getRemoteUser() or equivalent
     * (null if unknown)
     * @param userPropertyOne some user defined property typed as a String
     * @param userPropertyTwo another user defined property typed as a Double
     */
    public TestTrackingEvent( String theSessionId,
                              String theUserId,
                              String userPropertyOneValue,
                              Double userPropertyTwoValue )
    {
        super( TYPE, theSessionId, theUserId, XML_NAMESPACE, XSD_FILE,
               localSchemaKeys );

        if( userPropertyOneValue != null )
            setAttribute( USER_PROPERTY_ONE_KEY, userPropertyOneValue );

       if( userPropertyTwoValue != null )
           setAttribute( USER_PROPERTY_TWO_KEY, userPropertyTwoValue );
    }
}
```

The `TestTrackingEvent`, shown in Listing 15-9, correctly sets its own attributes and sets the attributes in its instantiation of `TrackingEvent`. This enables correct population of the XML instance document at the time of its creation. Recall that the XML instance document represents the `TestTrackingEvent` in the database's `EVENT` table.

Persisting to database If you want the custom Behavior Tracking event type to be persisted in the database, the event must be added to the `behaviorTracking.persistToDatabase` property in the `application-config.xml` file. If you are not persisting the event, you do not need to add the event type to this property.

How to Enable Behavior Tracking

Note: If the Event service does not exist as a service for your application, use WebLogic Server Administration Console to add it.

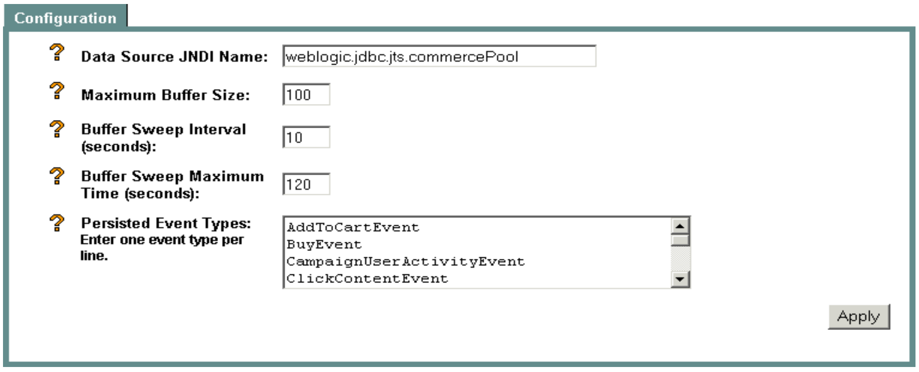
The following steps describe how to enable Behavior Tracking as a service for the Sample Portal. For your application, you would use similar steps:

1. In the WebLogic Server Administration Console, navigate to Behavior Tracking in the node tree for sampleportalDomain as follows:

   http://<hostname>:<port>/console →sampleportalDomain →Deployments → sampleportal →Service Configurations →Behavior Tracking

2. Enter the name of the event in the Persisted Event Types field, as shown in Figure 15-6.

Figure 15-6 WebLogic Server Administration Console—Behavior Tracking



Data Source JNDI Name: weblogic.jdbc.jts.commercePool
Maximum Buffer Size: 100
Buffer Sweep Interval (seconds): 10
Buffer Sweep Maximum Time (seconds): 120
Persisted Event Types: Enter one event type per line.
```
AddToCartEvent
BuyEvent
CampaignUserActivityEvent
ClickContentEvent
```

Converting Behavior Tracking Events to XML

When persisting Behavior Tracking events to the `EVENT` table, the bulk of the data must be converted to XML. The XML document should conform to an XML XSD schema that you create which specifies the order of the XML elements in the XML instance document. Additionally, the schema must include the types of elements and their cardinalities. The process of creating XML from an event object is handled by a helper class that utilizes variables and constants in a Behavior Tracking event's class file. All schema documents use the namespace: `http://www.w3.org/2000/10/XMLSchema` and all instances of Behavior Tracking schemas use the namespace: `http://www.w3.org/2000/10/XMLSchema-instance`. The XML created in Listing 15-10 will conform to the XSD schema.

Listing 15-10 XSD Document Example

```
<xsd:schema
    targetNamespace="http://www.bea.com/servers/commerce/xsd/tracking/buy/1.0.1"
        xmlns="http://www.bea.com/servers/commerce/xsd/tracking/buy/1.0.1"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2001/XMLSchema
                            http://www.w3.org/2001/XMLSchema.xsd"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
        <xsd:element name="BuyEvent">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="application"/>
                    <xsd:element ref="event-date"/>
                    <xsd:element ref="event-type"/>
                    <xsd:element ref="session-id"/>
                    <xsd:element ref="user-id" minOccurs="0"/>
                    <xsd:element ref="sku"/>
                    <xsd:element ref="quantity"/>
                    <xsd:element ref="unit-price"/>
                    <xsd:element ref="currency" minOccurs="0"/>
                    <xsd:element ref="application-name" minOccurs="0"/>
                    <xsd:element ref="order-line-id"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="application" type="xsd:string"/>
        <xsd:element name="event-date" type="xsd:string"/>
        <xsd:element name="event-type" type="xsd:string"/>
       <xsd:element name="session-id" type="xsd:string"/>
       <xsd:element name="user-id" type="xsd:string"/>
       <xsd:element name="sku" type="xsd:string"/>
       <xsd:element name="quantity" type="xsd:double"/>
       <xsd:element name="unit-price" type="xsd:double"/>
       <xsd:element name="currency" type="xsd:string"/>
       <xsd:element name="application-name" type="xsd:string"/>
       <xsd:element name="order-line-id" type="xsd:long"/>
</xsd:schema>
```

Constructing the XML Creation of an event's representation in XML takes place generically relative to the event's type. Consequently, to create an accurate XML instance document, each event must specify the namespace, event type, elements, and order of its elements. Using the `TestTrackingEvent` example, you would construct the XML representing an instance of the `TestTrackingEvent` as follows:

Note: Assume that `testTrackingEvent` is a well-formed instance of a `TestTrackingEvent`.

1. Get the event's type with the `testTrackingEvent.getType()` call.
2. Get the event's namespace with the `((TrackingEvent)testTrackingEvent).getXMLNamespace()`call.
3. Get the event's XSD filename with the `((TrackingEvent)testTrackingEvent).getXSDFile()` call.

Using the schema keys from the `TestTrackingEvent` class, values are inserted into the XML document. Schema key/attribute value pairs correspond to XML elements in this way:

```
<schema Key>value</schema Key>
```

The helper class that creates XML for Behavior Tracking assumes that the elements inserted into an XML instance document are not deeply nested. Additionally, the `toString()` method is used to create a representation of the value object that is retrieved through the `Event` classes's `getAttribute( String Key )` call. The contents of the string returned by invoking `toString()` on the value object must match the type specified in the event's schema document. The `TestTrackingEvent` retrieves values using the following keys in the order specified in the `schemaKeys` array:

- `SESSION_ID`
- `USER_ID`
- `USER_PROPERTY_ONE_KEY`
- `USER_PROPERTY_TWO_KEY`

The values for these keys are retrieved using the `testTrackingEvent.getAttribute( <schema Key> )` call. The order in which the XML formatted key/value pairs are inserted into the instance document is specified by the constant `schemaKeys` array, which is defined and populated in the `TestTrackingEvent` class.

The steps assembled to create an XML instance document for the `TestTrackingEvent` are presented in Listing 15-11.

Listing 15-11 XML Instance Document Example

```
<TestTrackingEvent
    xmlns="http://<your URI>/testtracking"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
    xsi:schemaLocation="http://<your URI>/testtracking
TestTrackingEvent.xsd"
    >
<event_date>XML time instant formatted event date</event_date>
<event_type>TestTrackingEvent</event_type>
     <application>wlcsApp</application>
     <session_id>theSessionIdValue</session_id>
     <user_id>theUserIdValue</user_id>
     <userPropertyOne>userPropertyOneValue</userPropertyOne>
     <userPropertyTwo>userPropertyTwoValue</userPropertyTwo>
</TestTrackingEvent>
```

The XML creation is performed automatically when events arrive at the `com.bea.p13n.tracking.listeners.BehaviorTrackingListener`, which enables Behavior Tracking in WebLogic Portal. The Behavior Tracking listener is installed by adding it to the `<EventService Listeners="...">` property in the `application-config.xml` file. For information on how to install a Behavior Tracking listener, see“How to Enable Behavior Tracking” on page 15-32.

Caution: You must be careful when defining the namespaces, XSD documents, and schema keys variables in custom Behavior Tracking event classes, especially if they will be persisted to the `EVENT` table. The method for creating and storing XML presented in this discussion exactly follows the variables and constants specified in the event class. You are free to develop other ways of creating and storing XML; this section is directed only at the process of persisting XML Behavior Tracking representations in the BEA `EVENT` table.

Note: The Event's date is retrieved using the `Event` class's `getTimeStamp()` call, which returns a Java primitive `long` typed value. That `long` must be converted into the type specified for the `event_date` element in the XSD schema document. The type in this case is time instant. Event date and event type the first two elements in all XML instance documents created through the `BehaviorTrackingListener`.

Creating Custom Behavior Tracking Event Listeners

To create a custom Behavior Tracking listener, in addition to or instead of the default `BehaviorTrackingListener`, follow the example presented in "Writing the Custom Event Listener" on page 15-16. Add the new event types to the custom listener's `eventTypes` array (for example, `TestTrackingEvent`). A given listener can listen for any number of event types that may or may not be Behavior Tracking events. The custom Behavior Tracking listener can be installed on either the synchronous or asynchronous side of the Event service, whichever is appropriate.

Writing Custom Event Generators

Once events are created, you must set up a mechanism for generating events in the application. Events may be generated from Pipeline components, input processors, JSP scriptlets, or JSP tags. Some Behavior Tracking events are generated from within WebLogic Portal software.

After determining the mechanism for generating events, Behavior Tracking events can be sent to the event system using the `com.bea.p13n.tracking.TrackingEventHelper` class. This class defines helper methods that pass events to the Event service. Listing 15-12 shows an example of passing the `TestTrackingEvent`.

Listing 15-12 Dispatching an Event

```
/*
 * Create the event
 */
Event theEvent = new TestTrackingEvent( "<some session id>",
                                        "<some user id> ",
                                        new String("userPropertyOneValue"),
                                        new Double( 3.14 ) );

/*
 * Dispatch the event
 */
EventService eventService = TrackingEventHelper.getEventService();
TrackingEventHelper.dispatchEvent( eventService, theEvent );
```

Dispatching events Because the Event service is an EJB, before dispatching events, the Event service must be running in a WebLogic Server instance.

If dispatching multiple events, it is best to get an instance of the Event service and save it as an attribute in your class for reuse, as shown in the following code:

```
/**
    * Access and start Event service
*/
private EventService eventService =
com.bea.p13n.tracking.TrackingEventHelper.getEventService ( );
```

Note: There are three APIs for this. To decide which one to use, see the Javadoc at http://edocs.bea.com/wlp/docs70/javadoc/index.html.

Now use that instance of the Event service to dispatch events, as follows:

```
/**
    * Dispatch the event
*/
```

```
EventService eventService = TrackingEventHelper.getEventService();
TrackingEventHelper.dispatchEvent ( eventService, theEvent )
```

Debugging the Event Service

To debug the Event service, create a `debug.properties` file in the following directory:

```
<BEA_HOME>\weblogic700\portal\config\<YourDomain>\debug.propertie
s
```

The contents of this file are shown in Listing 15-13.

Listing 15-13 Debugging the Event Service

```
usePackageNames: on

# Turns on debug for all classes under events
com.bea.p13n.events: on
# com.bea.p13n.events.internal.EventServiceBean: on

# Turns on debug for all classes under tracking
com.bea.p13n.tracking: on

# Or you can selectively turn on classes
com.bea.p13n.tracking.internal persistence: on
com.bea.p13n.mbeans.BehaviorTrackingListener: on
com.bea.p13n.tracking.listeners.BehaviorTrackingListener: on
com.bea.p13n.tracking.SessionEventListener: on
```

Registering Custom Events

This section contains basic information about registering custom events including background information about custom events, how to register events using the Events Editor in the BEA E-Business Control Center, and what you need to do when you make changes to custom events.

Note: You cannot change any of the standard events supplied with WebLogic Portal.

The creation of a custom event is a multiple-step process. The following list provides an overview of the process:

Note: You should have already completed steps 1 and 2.

1. Create the code that defines the event and event listener.
2. Create the code to trigger the event with a JSP tag or an API call.
3. Register the event using the instructions in this topic.
4. To record the event data for Behavior Tracking analysis, add the event to the Event service with the WebLogic Server Administration Console and create an entry for the event in the `EVENT_TYPE` table.

When to Register an Event

When you create a custom event to use in a campaign, you must register the event. If your event is not used in a campaign, you do not need to register it. Registering a custom event lets the E-Business Control Center know that the custom event exists. Registering permits campaign developers using the E-Business Control Center to create scenario actions that refer to the event. Registering also identifies the event's properties.

Caution: Whenever you change the event code, you must update the event registration. Conversely, whenever you change the event registration, you must also update the event code. A possible ramification of event modification is that the scenario actions that refer to the event's properties may need to be modified.

Event Properties

The Event editor in the E-Business Control Center allows you to easily register a custom event. For the purpose of registering an event, you can consider an event property as a name-value pair. During the registration of a custom event, you specify the event's name, description, and one or more properties. Each property has a range, type of permissible value, and default value. The information you need to register for an event should be available from your Commerce Business Engineer (CBE) or Java developer.

The properties for a custom event includes the following information:

- **Data type**: Specifies the data type for your property. The possible values are Text, Numeric, Floating Point Number, Boolean, and Date/Time.
- **Selection mode**: Specifies whether a property has a single default value or a collection of default values.
- **Value range**: Specifies whether the defaults are restricted to one specific value, one or more specific values, or any value.

Note: When you set property values, you are not guaranteed that the property will adhere to these restrictions at run time. Events are not checked by the `SchemaManager` for adherence to a property schema. Therefore, you need to keep the event type definition and the event registration synchronized.

As the previous list suggests, a combination of property values are possible. The possible combinations of properties are listed here:

- **Boolean**: The values for this type of property are either True or False. You can choose the default. The default value is displayed only in the Enter Property Values Window, not in the Edit Event Property window. When this data type is selected, the Selection mode and Value range are unavailable.
- **Single, Unrestricted**: This type of property has only one value, which is also the default value.
- **Single, Restricted**: This type of property has multiple values and a single default value. You can select which value is the default.
- **Multiple, Restricted**: This type of property has multiple values. You can select one or more values as defaults values.

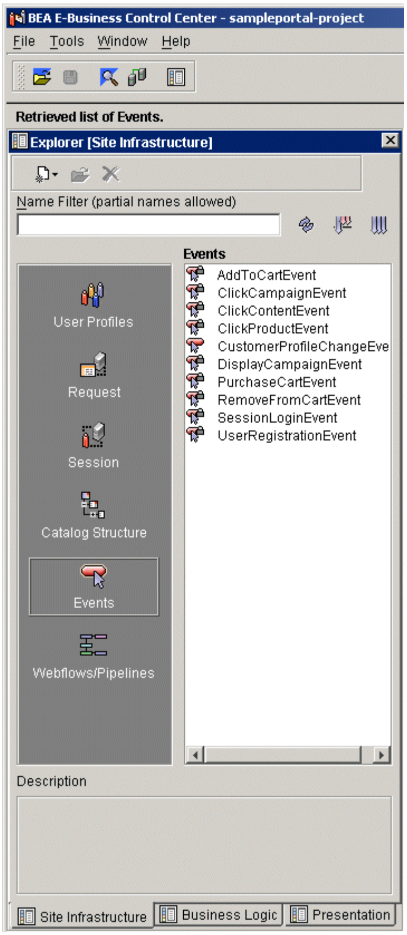
- **Multiple, Unrestricted**: This type of property has multiple values. You cannot select any defaults; all values are defaults.

Instructions for Registering a Custom Event

To register a custom event, complete the following steps:

1. Start the E-Business Control Center. The Explorer window opens as shown in Figure 15-7.

Figure 15-7 E-Business Control Center Window

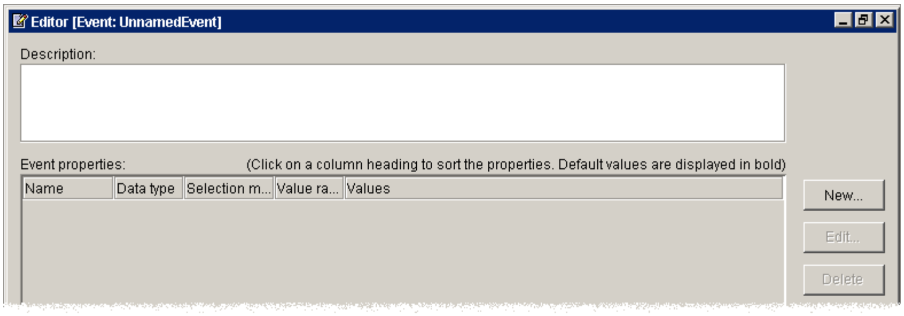


2. Open your project. For more information see the E-Business Control Center online help.

3. In the Explorer window, select the Site Infrastructure tab, and then click the Event icon. A list of events appears in the Events field.

4. Click the **New** icon, and then select Event. The Event Editor window appears as shown in Figure 15-8.

Figure 15-8 Event Editor Window


Editor [Event: UnnamedEvent]

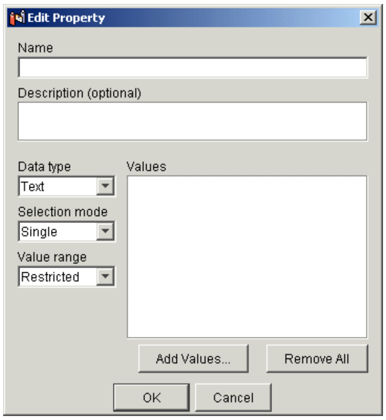
Description:

Event properties: (Click on a column heading to sort the properties. Default values are displayed in bold)

Name	Data type	Selection m...	Value ra...	Values

5. In the Event Editor window, click the **New** button. The Edit Property window opens, as shown in Figure 15-9.

Figure 15-9 Edit Property Window


Edit Property

Name

Description (optional)

Data type: Text

Selection mode: Single

Value range: Restricted

Values

6. In the Edit Property window, complete these steps:

   a. In the Name field, enter a unique name for the event no longer than 100 characters (required).

   b. In the Description field, enter a description for the event no longer than 254 characters (optional).

   c. Select the Data Type, Selection mode, and Value range for the property value from the drop lists.

   d. Click the **Add Values** button. The dialog box that appears depends on the properties.

   e. Enter the appropriate values and select the defaults (if needed).

   f. After you have completed entering the property values for the event, click the **OK** button.

7. Save the event (E-Business Control Center menu →File →Save).

Updating a Registered Custom Event

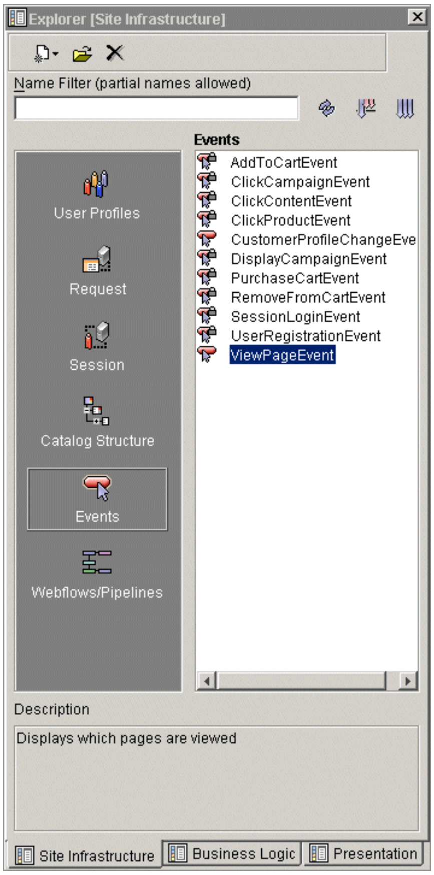
Whenever you make changes to a custom event's code, you should update that event's registration. Updating the registration lets the E-Business Control Center know about the changes in the custom event and aids campaign developers using the E-Business Control Center to modify any scenario actions (in campaigns) that refer to the event.

To update a custom event, complete the following steps.

1. Start the E-Business Control Center. The Explorer window opens.
2. Open your project. For more information, see the E-Business Control Center online help.
3. In the Explorer window, select the Site Infrastructure tab, and then click the **Event** icon. A list of events appears in the Events field as shown in Figure 15-10.

   Note: You cannot edit standard events.

Figure 15-10 Explorer Window



Events

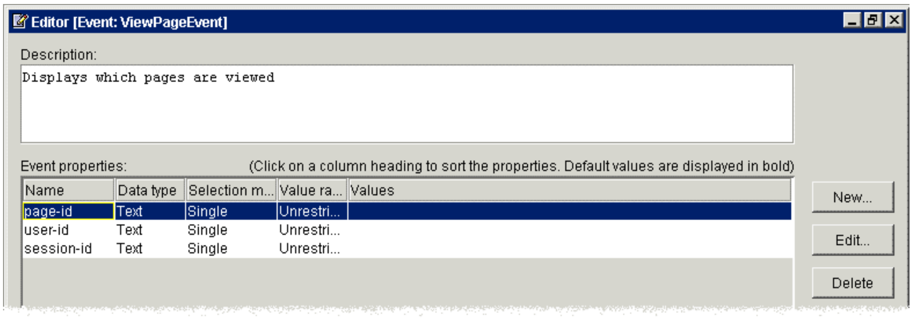
AddToCartEvent
ClickCampaignEvent
ClickContentEvent
ClickProductEvent
CustomerProfileChangeEve
DisplayCampaignEvent
PurchaseCartEvent
RemoveFromCartEvent
SessionLoginEvent
UserRegistrationEvent
ViewPageEvent

Description

Displays which pages are viewed

4. Double-click the custom event that you wish to edit. The Event Editor window opens as shown in Figure 15-11. The Event properties field displays a list of existing properties.

Figure 15-11 Event Editor Window



Editor [Event: ViewPageEvent]

Description:

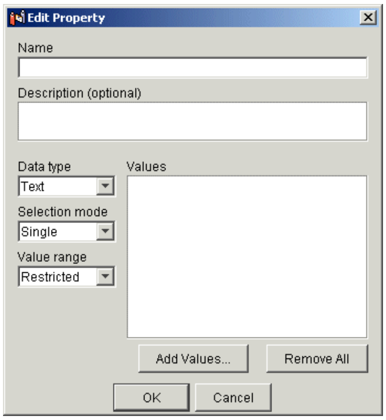
```
Displays which pages are viewed
```

Event properties: (Click on a column heading to sort the properties. Default values are displayed in bold)

Name	Data type	Selection m...	Value ra...	Values
page-id	Text	Single	Unrestri...	
user-id	Text	Single	Unrestri...	
session-id	Text	Single	Unrestri...	

5. Select the property you want to edit, and then click the **Edit** button. The Edit Properties window opens, as shown in Figure 15-12.

Figure 15-12 Edit Property Window



Edit Property

Name:

Description (optional):

Data type: Text

Selection mode: Single

Value range: Restricted

Values:

6. Make the appropriate changes, and then click the **OK** button.

7. Save the event (E-Business Control Center menu →File →Save).

Activating Behavior Tracking

To record how online visitors are interacting with your Web site, you can record event information in a database. These kinds of events are called Behavior Tracking events. E-analytics and e-marketing systems can then analyze these events offline to evaluate visitor behavior and transactional data.

Note: For information about how to configure a database for recording event data, see "Persisting Behavioral Tracking Data" in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/sysadmin.htm#1194894.

This sections contains information on the following subjects:

- Procedure for Activating Behavior Tracking
- Configuring the Behavior Tracking Service in WebLogic Server
- Configuring a Data Source

Procedure for Activating Behavior Tracking

Before Behavior Tracking events can be recorded to a database, you must enable the Behavior Tracking listener. This is accomplished by adding a listener class.

Note: If the Event service does not exist as a service for your application, use WebLogic Server Administration Console to add it.

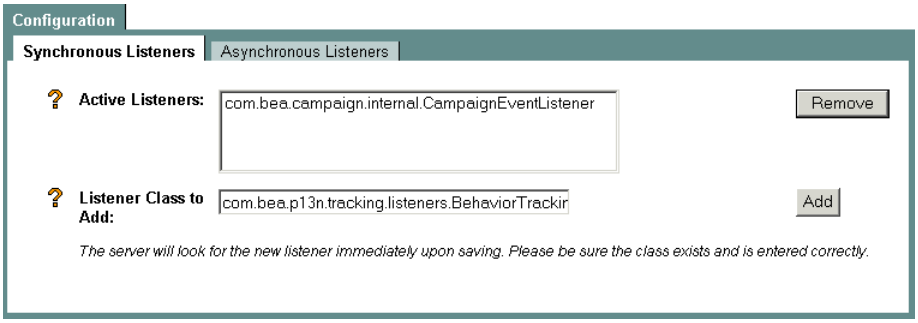
The following steps describe how to add a listener class in the Sample Portal. For your application, you would use similar steps.

1. In the WebLogic Server Administration Console, navigate to the Synchronous or Asynchronous Listeners tab in the node tree for sampleportalDomain as follows:

   http://<hostname>:<port>/console → sampleportalDomain → Deployments → Applications → sampleportal → Service Configurations → Event Service → Configuration Tab → Synchronous Listeners

2. Add the Behavior Tracking listener (`com.bea.p13n.tracking.listeners.BehaviorTrackingListener`) to the Listen Class to Add field, and then click the **Add** button. See Figure 15-13.

Figure 15-13 WebLogic Server Administration Console—Event Service



Active Listeners: com.bea.campaign.internal.CampaignEventListener

Listener Class to Add: com.bea.p13n.tracking.listeners.BehaviorTrackir

The server will look for the new listener immediately upon saving. Please be sure the class exists and is entered correctly.

Note: You must configure your database before activating Behavior Tracking. For information on how to do this, see "Persisting Behavior Tracking Data" in the *Administration Guide* at http://edocs.bea.com/wlp/docs70/admin/sysadmin.htm#1194894.

Configuring the Behavior Tracking Service in WebLogic Server

Behavior Tracking events are placed in a buffer and then intermittently persisted to the Event tables in the database where they can be analyzed offline. An asynchronous service is used so that long-running event handlers can execute without delaying the application from a Web site visitor's perspective.

Note: Each Behavior Tracking event property must be configured in the WebLogic Server Administration Console.

Connection pool The buffered Behavior Tracking events are swept into the database using a pool of data connections. The default Data Source is `weblogic.jdbc.jts.commercePool`. You can use a different Data Source. To do this, create and configure the new Data Source (see "Configuring a Data Source" on page 15-50) and substitute the name of the default Data Source with the name of the new Data Source in the WebLogic Server Administration Console.

Properties The particular events that are persisted to the database are specified in the `PersistEventTypes` property. You can view and alter the list of the persisted events in the WebLogic Server Administration Console. The types in this list must match the type specified in the event; for example, the `SessionBeginEvent` has as its type the string "`SessionBeginEvent`".

Optimize performance The frequency of the sweeping of events from the buffer is controlled by the following properties the Behavior Tracking service:

- `MaxBufferSize`
- `SweepInterval`
- `SweepMaxTime`

You should tune these properties to optimize performance. A buffer sweep should be performed often enough that writing to the database is not too time consuming but not so frequent that the operation is wasteful.

Steps To configure the Behavior Tracking Service, take the following steps:

Notes: These steps provide information on how to optimize performance in the Sample Portal. For your application, you would use similar steps.

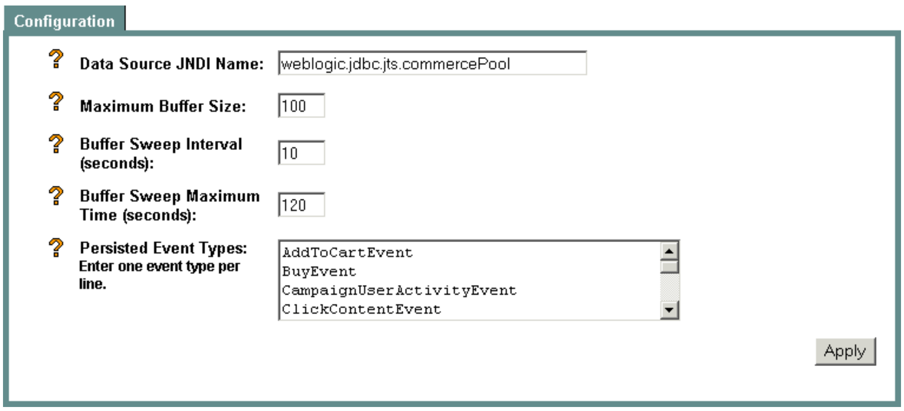
If the Event service does not exist as a service for your application, use WebLogic Server Administration Console to add it.

Note: If the Behavior Tracking and Event services do not exist for your application, use the WebLogic Server Administration Console to add them.

1. In the WebLogic Server Administration Console, navigate to the Behavior Tracking Service (shown in Figure 15-13) in the node tree for sampleportalDomain, as follows:

   http://<hostname>:<port>/console →sampleportalDomain →Deployments → Applications →sampleportal →Service Configurations →Behavior Tracking Service

Figure 15-14 WebLogic Server Administration Console—Behavior Tracking Service



Data Source JNDI Name:	weblogic.jdbc.jts.commercePool
Maximum Buffer Size:	100
Buffer Sweep Interval (seconds):	10
Buffer Sweep Maximum Time (seconds):	120
Persisted Event Types: Enter one event type per line.	AddToCartEvent BuyEvent CampaignUserActivityEvent ClickContentEvent

2. To change the Data Source, enter the fully-qualified name of the Data Source in the Data Source JNDI Name field.

3. To change the sweeping of events from the buffer, enter the new buffer values in the appropriate fields.

4. To specify whether a particular event is persisted, add or remove the event from the Persisted Event Types list box.

Configuring a Data Source

This section provides a brief description about configuring a new Data Source for a connection pool used for persisting events in the Sample Portal. For your application, you would use similar steps

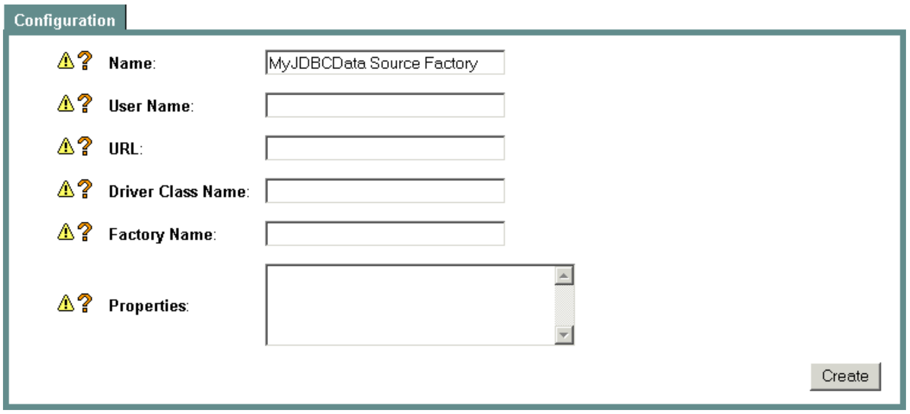
To configure a new Data Source, take the following steps.

Note: For more information on using the WebLogic Server Administration Console, see the WebLogic Server documentation at http://edocs.bea.com/wls/docs70/index.html.

1. In the WebLogic Server Administration Console, navigate to the Behavior Tracking Service (shown in Figure 15-13) in the node tree for sampleportalDomain, as follows:

http://<hostname>:<port>/console →sampleportal →Services →JDBC →Data Sources →JDBCData Source Factories

Figure 15-15 WebLogic Server Administration Console—JDBC Data Sources



Name:	MyJDBCData Source Factory
User Name:	
URL:	
Driver Class Name:	
Factory Name:	
Properties:	

2. In the right pane, click **Configure a new JDBC Data Source Factory**.
3. Enter the appropriate values for the new Data Source in the appropriate tabs and fields.

CHAPTER

16 Using the Expression Package

This topic illustrates how to use the services of the Expression package. The Expression package is part of the Personalization and Interaction Management features in WebLogic Portal. The Expression package allows you to externalize calculations, business policies, decision trees, and other operations from your Java code.

This section includes information on the following subjects:

What Is the Expression Package?

As previously mentioned, the Expression package allows you to externalize business logic or formulas from your Java code. Using the Expression package, any arithmetic, boolean, relational or conditional statement can be represented. You can use the Expression package to dynamically assemble and evaluate your own business logic.

An example of Expression package use is a rental car agency using it for calculating rental costs, which may change frequently. Rather than expressing the calculation using Java statements, the calculation can be externalized from the Java code into an XML document and interpreted at run time.

WebLogic Portal provides an Expression example. To see this example, take the following steps:

1. Start the Personalization server, as follows:

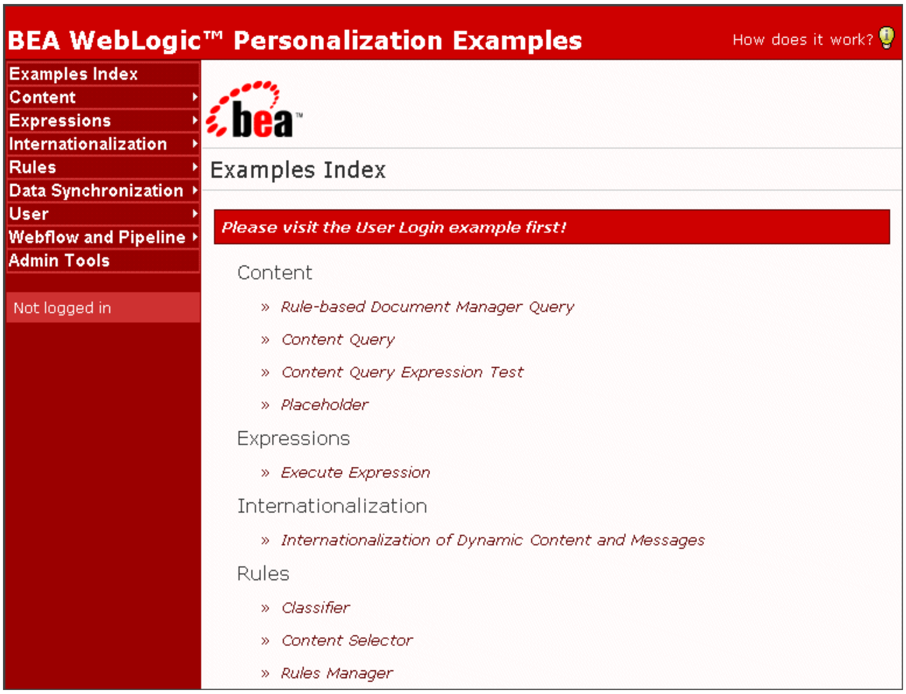
   Start →BEA WebLogic Platform 7.0 →WebLogic Portal 7.0 →Portal Examples →Personalization Examples →Launch Personalization Server

2. After the Personalization server is running, start the Personalization Examples, as follows:

   Start →BEA WebLogic Platform 7.0 →WebLogic Portal 7.0 →Portal Examples →Personalization Examples →Start Personalization Examples

   A browser window opens showing the Personalization Examples index, as shown in Figure 16-1.

Figure 16-1 Personalization Examples Index



Examples Index

Please visit the User Login example first!

Content

» *Rule-based Document Manager Query*

» *Content Query*

» *Content Query Expression Test*

» *Placeholder*

Expressions

» *Execute Expression*

Internationalization

» *Internationalization of Dynamic Content and Messages*

Rules

» *Classifier*

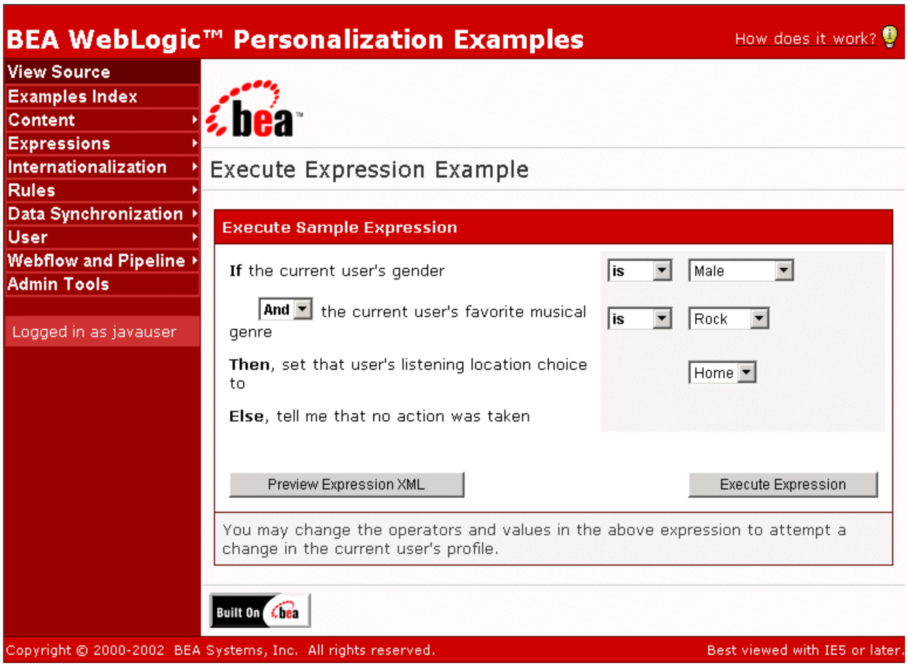
» *Content Selector*

» *Rules Manager*

3. If you are not logged in or have not created a user, click **Please visit the User Login example first**. A page opens where you can either log in or go to another page to create a user.

4. After you have logged in or created a user, in the left column of the page, select Expressions →Execute Expression. The Execute Expression Example page opens, as shown in Figure 16-8.

Figure 16-2 Execute Expression Example



Execute Expression Example

Execute Sample Expression

If the current user's gender — is — Male

And the current user's favorite musical genre — is — Rock

Then, set that user's listening location choice to — Home

Else, tell me that no action was taken

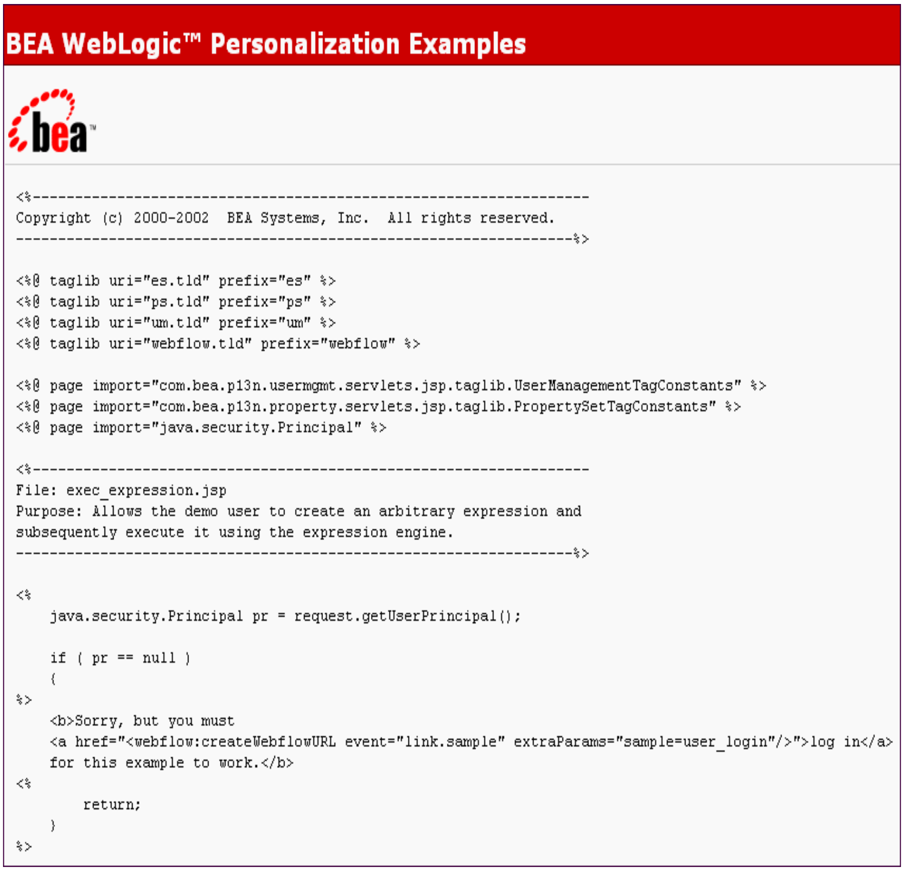
You may change the operators and values in the above expression to attempt a change in the current user's profile.

Copyright © 2000-2002 BEA Systems, Inc. All rights reserved. Best viewed with IE5 or later.

This window shows a simple application of the Expression package, where the parameters are set using the drop-down lists.

5. Click **View Source** in the left column. A new page opens showing the JSP source, as shown in Figure 16-3.

Figure 16-3 JSP Source



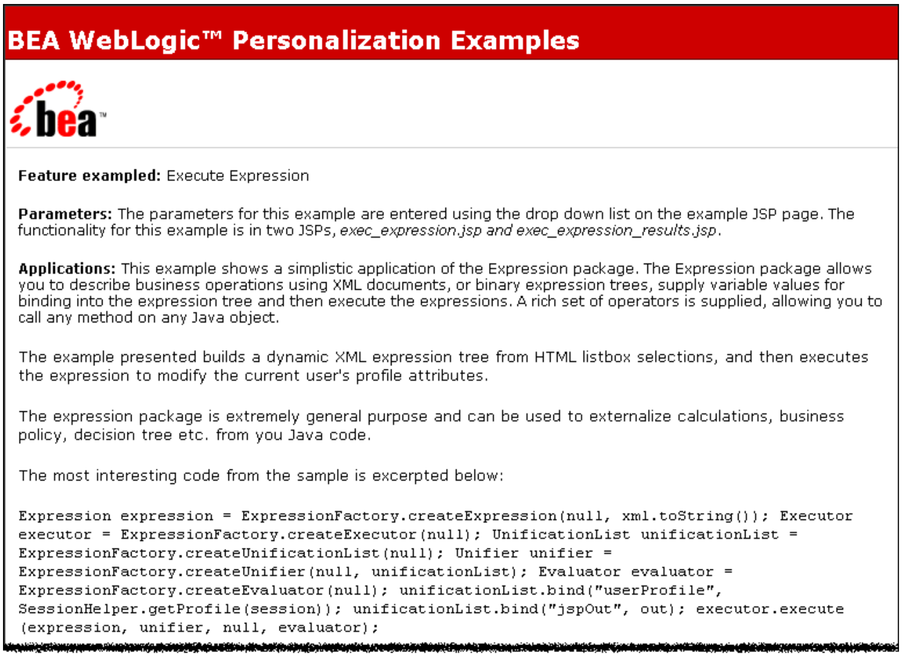
```
<%-----------------------------------------------------------------
Copyright (c) 2000-2002  BEA Systems, Inc.  All rights reserved.
-----------------------------------------------------------------%>

<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="ps.tld" prefix="ps" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="webflow.tld" prefix="webflow" %>

<%@ page import="com.bea.p13n.usermgmt.servlets.jsp.taglib.UserManagementTagConstants" %>
<%@ page import="com.bea.p13n.property.servlets.jsp.taglib.PropertySetTagConstants" %>
<%@ page import="java.security.Principal" %>

<%-----------------------------------------------------------------
File: exec_expression.jsp
Purpose: Allows the demo user to create an arbitrary expression and
subsequently execute it using the expression engine.
-----------------------------------------------------------------%>

<%
    java.security.Principal pr = request.getUserPrincipal();

    if ( pr == null )
    {
%>
    <b>Sorry, but you must
    <a href="<webflow:createWebflowURL event="link.sample" extraParams="sample=user_login"/>">log in</a>
    for this example to work.</b>
<%
        return;
    }
%>
```

This page shows the JSP source (`exec_expression.jsp`) for the Expression Example. After the expression is executed (and a parameter is changed), the results are contained in the `exec_expression_results.jsp`. Both JSP files are located in the `<BEA_HOME>\weblogic700\samples\portal\p13nDomain\beaApps\p13nApp\p13n` directory.

The next step provides information about how the expression works.

6. On the **Execute Expression Example** page, click the **How does it work?** link. A a new browser window opens that describes the example, as shown in Figure 16-4.

Figure 16-4 How Does It Work? Execute Expression



BEA WebLogic™ Personalization Examples

Feature exampled: Execute Expression

Parameters: The parameters for this example are entered using the drop down list on the example JSP page. The functionality for this example is in two JSPs, *exec_expression.jsp and exec_expression_results.jsp*.

Applications: This example shows a simplistic application of the Expression package. The Expression package allows you to describe business operations using XML documents, or binary expression trees, supply variable values for binding into the expression tree and then execute the expressions. A rich set of operators is supplied, allowing you to call any method on any Java object.

The example presented builds a dynamic XML expression tree from HTML listbox selections, and then executes the expression to modify the current user's profile attributes.

The expression package is extremely general purpose and can be used to externalize calculations, business policy, decision tree etc. from you Java code.

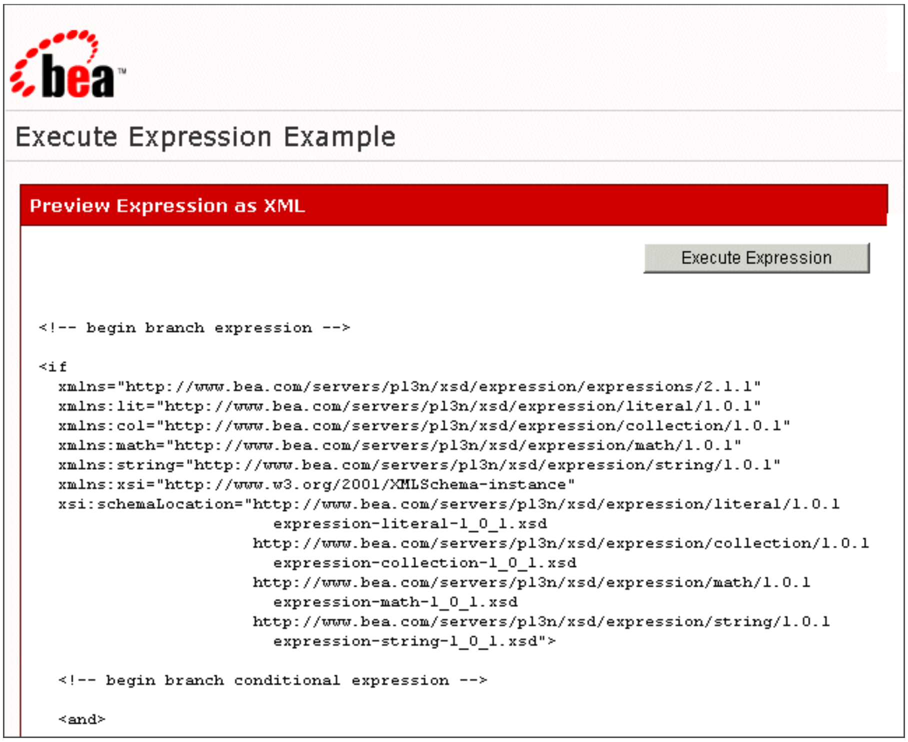
The most interesting code from the sample is excerpted below:

```
Expression expression = ExpressionFactory.createExpression(null, xml.toString()); Executor
executor = ExpressionFactory.createExecutor(null); UnificationList unificationList =
ExpressionFactory.createUnificationList(null); Unifier unifier =
ExpressionFactory.createUnifier(null, unificationList); Evaluator evaluator =
ExpressionFactory.createEvaluator(null); unificationList.bind("userProfile",
SessionHelper.getProfile(session)); unificationList.bind("jspOut", out); executor.execute
(expression, unifier, null, evaluator);
```

This page describes how the Expression Example works. It also suggests an exercise to further your understanding of expressions.

7. On the **Execute Expression Example** page, click the **Preview Expression XML** button. A page showing the XML appears, as shown in Figure 16-5.

Figure 16-5 Preview Expression as XML



```
<!-- begin branch expression -->

<if
  xmlns="http://www.bea.com/servers/p13n/xsd/expression/expressions/2.1.1"
  xmlns:lit="http://www.bea.com/servers/p13n/xsd/expression/literal/1.0.1"
  xmlns:col="http://www.bea.com/servers/p13n/xsd/expression/collection/1.0.1"
  xmlns:math="http://www.bea.com/servers/p13n/xsd/expression/math/1.0.1"
  xmlns:string="http://www.bea.com/servers/p13n/xsd/expression/string/1.0.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.bea.com/servers/p13n/xsd/expression/literal/1.0.1
                        expression-literal-1_0_1.xsd
                      http://www.bea.com/servers/p13n/xsd/expression/collection/1.0.1
                        expression-collection-1_0_1.xsd
                      http://www.bea.com/servers/p13n/xsd/expression/math/1.0.1
                        expression-math-1_0_1.xsd
                      http://www.bea.com/servers/p13n/xsd/expression/string/1.0.1
                        expression-string-1_0_1.xsd">

  <!-- begin branch conditional expression -->

  <and>
```

This page shows the Expression Example XML before it is executed.

The next section discusses the differences between the Expressions package and the Rules Framework.

Using Rules or Expressions

One of the applications of the Rules Manager is to use business rules to match users and groups with appropriate content. The Rules Manager, like the Expressions package, is part of WebLogic Portal Personalization and Interaction Management.

The most important difference between the Expression package and the Rules Engine is that the Expression package uses named variables, while the Rules Engine does not. Additionally, the Rules Manager uses rule sets, where one rule can trigger another, that is, the rules can cascade.

In general, you use expressions when you want to bind variables to values (usually only one) and rules where you need pattern detection and want to evaluate all possible bindings to variables.

The Rule Engine has extremely powerful pattern matching and inferencing capabilities. However, these capabilities may come with a performance penalty. If you find yourself repeatedly executing a named rule, consider converting the rule to an expression. If you do not leverage the inferencing capabilities of the Rule Engine or rarely have more than one potential variable-value binding, then use expressions.

You should carefully evaluate the performance differences between using expressions, supplying an explicit binding between variables and values through a UnificationList or custom Unifier, and using the Rule Engine to explore all potential bindings.

Table 16-1 shows some examples of when to use rules and expressions.

Table 16-1 Expressions vs. Rules

Feature	Expressions	Rules
Externalize business logic from Java code	Yes	Yes
Rapid deployment of business logic independent of application code	Yes	Yes
Non programmers can assemble business logic using JSP or Swing GUI	Yes	Yes
Inferencing capability	No	Yes One rule firing can cause another rule to be fired.
Explicit binding of value to variables	Yes	No Values are bound to variables using class type. All possible bindings are automatically tested.
Long-lived persistence of business logic	Yes	Yes Business logic is persisted as XML documents. The XML Schema can provide independence from the Java code.

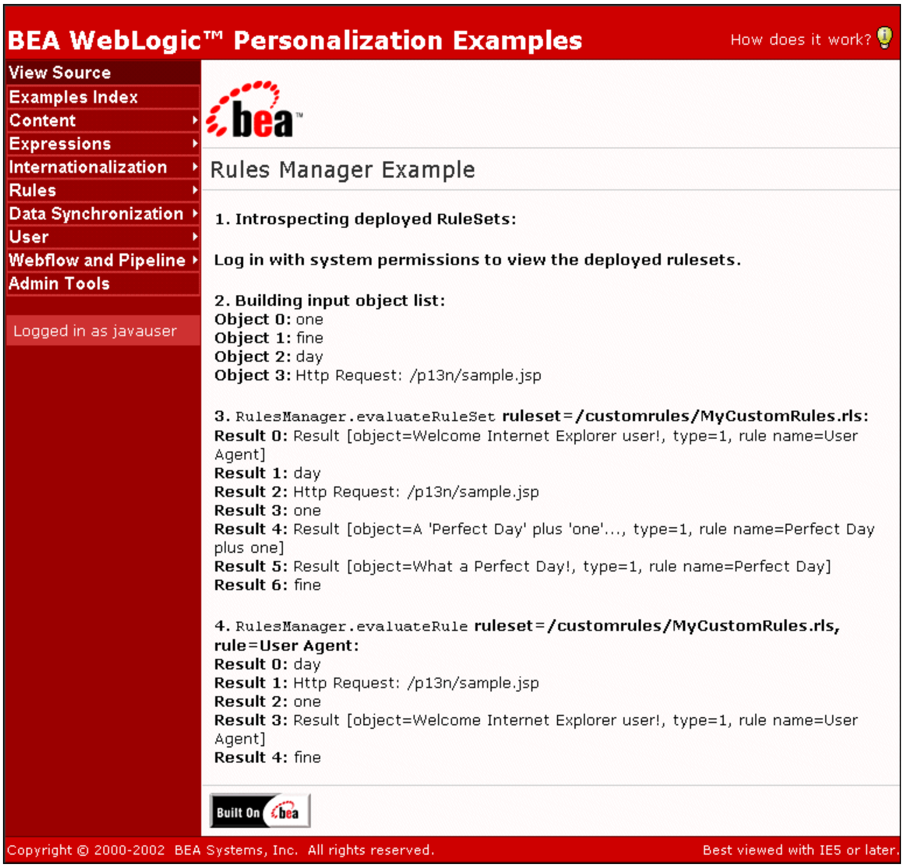
Table 16-1 Expressions vs. Rules

Feature	Expressions	Rules
Business logic can be passed between processes	Yes	Yes XML documents defining business logic can be serialized or passed between Web Services.
XML Parsing cache	No	Yes RulesManager implements a TTL cache for ruleset documents.
Expressions cache and optimization	Yes	Yes The Rule Engine uses the Expression package internally, and hence leverages many of its underlying optimizations.

To see an example of rules, take the following steps:

1. In the Personalization Examples window, select Rule →Rules Manager. The Rules Manager Example appears, as shown in Figure 16-6.

Figure 16-6 Rules Manager Example



Rules Manager Example

1. Introspecting deployed RuleSets:

Log in with system permissions to view the deployed rulesets.

2. Building input object list:
Object 0: one
Object 1: fine
Object 2: day
Object 3: Http Request: /p13n/sample.jsp

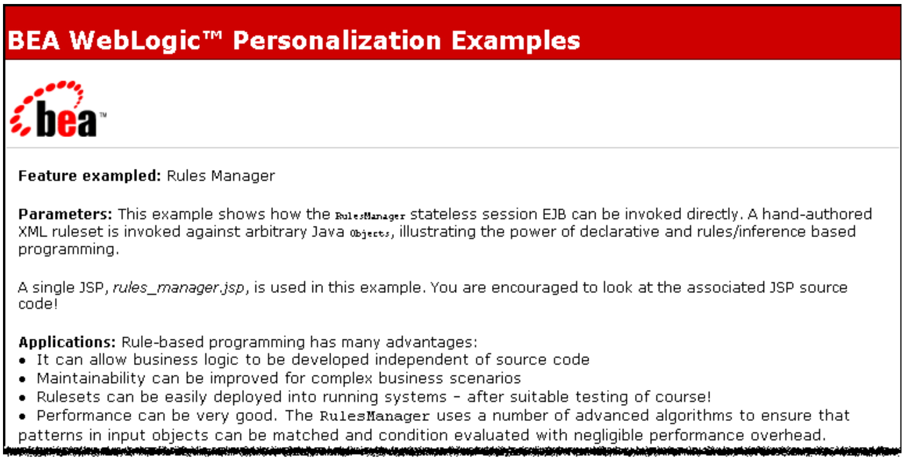
3. `RulesManager.evaluateRuleSet` **ruleset=/customrules/MyCustomRules.rls:**
Result 0: Result [object=Welcome Internet Explorer user!, type=1, rule name=User Agent]
Result 1: day
Result 2: Http Request: /p13n/sample.jsp
Result 3: one
Result 4: Result [object=A 'Perfect Day' plus 'one'..., type=1, rule name=Perfect Day plus one]
Result 5: Result [object=What a Perfect Day!, type=1, rule name=Perfect Day]
Result 6: fine

4. `RulesManager.evaluateRule` **ruleset=/customrules/MyCustomRules.rls, rule=User Agent:**
Result 0: day
Result 1: Http Request: /p13n/sample.jsp
Result 2: one
Result 3: Result [object=Welcome Internet Explorer user!, type=1, rule name=User Agent]
Result 4: fine

This page shows an example of rules. This example demonstrates that the action of one rule will cause the condition of another rule to become satisfied. This ability is not present in the Expression package.

2. For more explanation, click the **How does it work?** link. The explanation appears, as shown in Figure 16-7.

Figure 16-7 How Does It Work? Rules Manager


BEA WebLogic™ Personalization Examples

Feature exampled: Rules Manager

Parameters: This example shows how the `RulesManager` stateless session EJB can be invoked directly. A hand-authored XML ruleset is invoked against arbitrary Java `Objects`, illustrating the power of declarative and rules/inference based programming.

A single JSP, *rules_manager.jsp*, is used in this example. You are encouraged to look at the associated JSP source code!

Applications: Rule-based programming has many advantages:
- It can allow business logic to be developed independent of source code
- Maintainability can be improved for complex business scenarios
- Rulesets can be easily deployed into running systems - after suitable testing of course!
- Performance can be very good. The `RulesManager` uses a number of advanced algorithms to ensure that patterns in input objects can be matched and condition evaluated with negligible performance overhead.

Expression Package Classes

The Expression package allows users to dynamically assemble and execute XML-based expressions. The package defines a set of Java classes that represent various types of expression operators, and contains services for evaluating expressions consisting of instances of these operators.

The Expression package includes a base Expression class, a Variable class, and the following operator classes for operating on Expressions and Variables:

- Basic language operators (object creation, method call, and so on.)
- Logical operators
- Comparative operators
- Collection operators
- Mathematical operators
- String operators

The Expression package also includes the following services for operating on expressions:

- `Unifier`—prepares an expression for evaluation.
- `Validator`—validates that an expression is well-formed before evaluation.
- `Optimizer`—optimizes the structure of an expression before evaluation.
- `Evaluator`—evaluates an expression and returns the result of evaluation.
- `Executor`—an aggregate service that combines the unification, validation, and evaluation processes.

Unlike an expression written directly in Java and executed from within a Java program, the Expression package allows you to dynamically assemble and modify expressions from within your Java programs. An expression may be modified any number of times both before and after evaluation. When you assemble expressions using the Expression package you can also take advantage the advanced features of the Expression package, such as expression caching, validation, and optimization.

The Expression package serves as the foundation of the BEA Rules Engine. The Rules Engine leverages the package in order to represent and evaluate rule condition and action expressions. Likewise, you can use the Expression package to dynamically assemble and evaluate your own business logic.

The Package Structure for the Expression Package

The Expression package interfaces and abstract classes can be found in the following package: `com.bea.p13n.expression`

The Expression package operators are organized in the following packages:

Basic language operators—`com.bea.p13n.expression.operators`

Logical operators—`com.bea.p13n.expression.operators.logical`

String operators—`com.bea.p13n.expression.operators.string`

Mathematical operators—`com.bea.p13n.expression.operators.math`

Comparative operators—`com.bea.p13n.expression.operators.comparative`

Collection operators—`com.bea.p13n.expression.operators.collection`

The Expression package related classes are packaged in the `p13n_util.jar` archive.

Assembling and Managing Expressions

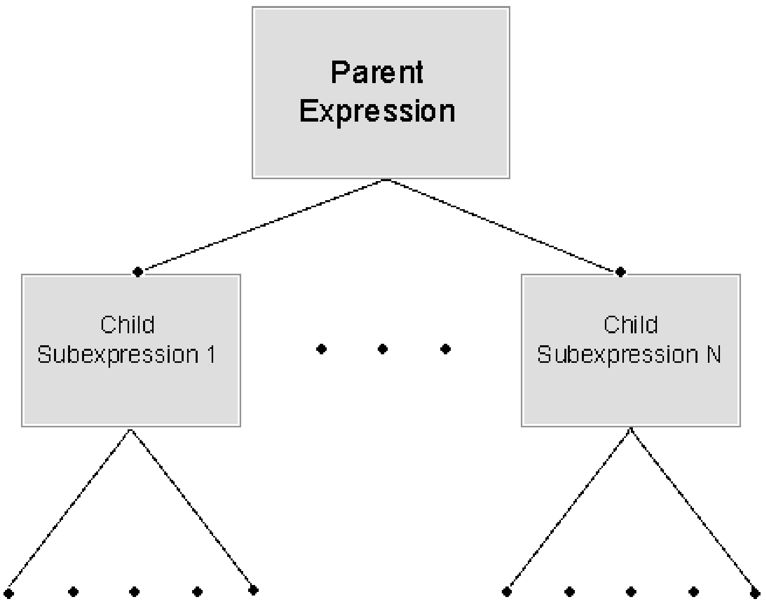
Before you can begin using expressions, you must first learn how to programmatically assemble them using the various operator classes provided in the Expression package.

An expression is represented as a tree, where each node is another expression itself or a plain Java object. Expression trees are assembled in a bottom-up manner; a child expression or Java object is first created, and then added to a parent expression.

Figure 16-8 illustrates the steps required to build an expression tree.

- The first step in the expression assembly process is to create one or more child operators or Java objects.
- Next, a parent operator is created by supplying the child operators or Java objects to the parent operator's constructor.
- This process of creating subexpressions continues until the entire expression is assembled.

Figure 16-8 Building an Expression Tree



Maintaining Parent-child Relationships

Each of the operator classes defined in the Expression package extends a common base class that contains the necessary logic for maintaining parent-child relationships; therefore, you do not have to worry about maintaining these relationships while assembling expressions. However, it is possible to modify the structure of an expression after it has been created.

Table 16-2 shows the operators provided in the `Expression` interface for adding, modifying, or removing subexpressions in an expression.

Table 16-2 Methods for Building an Expression Tree

Java Method	Description
`addSubExpression`	Adds a child (can be a subexpression) to an expression object.

Table 16-2 Methods for Building an Expression Tree

Java Method	Description
`removeSubExpression`	Removes an object (can be a subexpression) of the expression object.
`setSubExpression`	Replaces existing object (of an expression) by the given object (can be a subexpression).
`getSubExpression`	Can be used to access the children of an expression object.
`getParent`	Can be used to access the parent expression of an expression object.

For more information about the `Expression` interface, see the *Javadoc*.

Managing the Expression Cache

The expression interface also includes methods to manage the caching of results. The result of evaluating an expression may be cached in each expression object. When the cache is enabled for an expression, trying to evaluate the same expression a second time will return the cached value.

Note: By default, caching is turned off. You may want to keep the cache turned off for some operators, such as `MethodCall`.

Table 16-3 shows the methods provided in the `Expression` interface to manage the caching of results.

Table 16-3 Methods to Manage Caching of Results

Java Method	Description
`setCacheEnabled`	Can be used to enable or disable the cache for an expression.
`isCacheEnabled`	Can be used to check if the cache is enabled for an expression.

Table 16-3 Methods to Manage Caching of Results

Java Method	Description
`isCached`	Can be used to check if a result is currently cached for an expression.
`getCachedValue`	Can be used to get the current cached result of evaluating the expression.

For more information about the `Expression` interface, see the *Javadoc*.

Working with Expressions

After you have assembled an expression, you are ready to work with it using the various Expression package services. These services allow you to prepare an assembled expression for evaluation, validate that the expression is well-formed, optimize its structure, and finally, evaluate the expression.

The following information is presented in this section:

- The Expression Factory
- Expression Package Services
  - Unification Service
  - Optimization Service
  - Validation Service
  - Evaluation Service
  - Execution Service

The Expression Factory

The ExpressionFactory provides methods to create the various Expression package services and data structures used by these services.

For example, the following method will create an instance of the `Validator` service:

```
ExpressionFactory.createValidator(null);
```

For more detail on how to construct the various Expression package services, see the *Javadoc*.

Expression Package Services

The Expression package offers services which can be used on any expression that is built using the operators in the Expression package.

Unification Service

The `Unifier` is used to unify variables (assign values to variables) present in an expression. The `Unifier` uses a data structure known as a `UnificationList` that stores the variable name and the corresponding value of the variable. Like the `Unifier`, the `UnificationList` instances are created via the `ExpressionFactory`. The `Unifier` gets the value from the list for a particular variable using the variable name as a key to search the `UnificationList`, and binds the retrieved value to the variable.

For more information about the `Unifier` interface and the `ExpressionFactory` class, see the *Javadoc*.

Optimization Service

The `Optimizer` is used to optimize an expression. The default optimization algorithm used by the `Optimizer` is shown below.

- Traverse an expression tree and add each unique subexpression to a list.
- If a subexpression is equal to an expression present in the list, then replace it with a proxy expression. The proxy expression delegates to the original expression.

For more information about the `Optimizer` interface and the `ExpressionFactory` class, see the *Javadoc*.

Validation Service

The `Validator` is used to validate an expression. The default validation algorithm used by the `Validator` is as follows:

For each operand of an operator:

- Get the required type of the operand.
- If the operand is an expression, evaluate the expression and compare the type of the result with the required type; otherwise, assert that the operand is of the required type.
- If the type does not match or an error occurs during the evaluation of an operand expression, the `Validator` throws an `InvalidExpressionException` . An `UnboundVariableException` is thrown if any variables in an expression are not bound to a value.

The `Validator` can be used in a stateless or stateful mode. In stateless mode, any expression evaluations necessary to perform validation will be executed in stateless mode.

For more information about stateless and stateful evaluation modes, see the "Evaluation Service" section below.

For more information about the `Validator` interface and the `ExpressionFactory` class, see the *Javadoc*.

Evaluation Service

The `Evaluator` is used to evaluate an expression. An expression can be evaluated in stateful or stateless mode:

Stateful mode

In this mode, the value of each variable that appears in the expression is determined by retrieving the value set within the variable.

In other words, stateful mode relies upon the expression having been previously unified by a `Unifier`.

When an expression is evaluated in stateful mode and results caching is turned on, the results of evaluation will be cached within the expression.

Stateless mode

In this mode, the value of each variable that appears in the expression is determined by looking up a value that is bound to the name of the variable in an external data structure.

In other words, the evaluation process does not rely upon state associated with the expression, and as such, does not require the expression to be unified before evaluation.

The data structure that contains the name-value mappings for variables is known as a `UnificationList` and is associated with the `Evaluator`. Like the `Evaluator`, the `UnificationList` instances can be created using the `ExpressionFactory`.

A side effect of stateless mode is that expression evaluation cannot take advantage of results caching.

You can use of stateful mode in a situation where an expression need only be evaluated within a single thread of execution. In the case of multithreaded evaluation of a single expression, you *must* use stateless mode.

Note: If an expression does *not* contain variables, then there is no difference between the two evaluation modes.

For more information about the `Evaluator` interface and the `ExpressionFactory` class, see the *Javadoc*.

Execution Service

The `Executor` aggregates the Unification Service, Validation Service and Evaluation Service. The `execute` method on an `Executor` takes a `Unifier`, a `Validator` and an `Evaluator` to execute a cycle of unification-validation-evaluation operations.

The algorithm used by the `Executor` is shown below:

Unification

- If the `Unifier` is not null, unify the expression.
- If the `Unifier` is null, do not unify the expression.

Note: The `Unifier` should be null in the case where the expression passed to the `Executor` is already unified, or the expression is to be evaluated in stateless mode.

Validation

- If the `Validator` is not null, validate the given expression.
- If the `Validator` is null, ignore validation.

Evaluation

- If the `Evaluator` is not null, evaluate the expression in stateful or stateless mode. (depending on the type of evaluator passed.)
- If the `Evaluator` is null, the `Executor` throws an `IllegalArgumentException`.
- Return the result.

Note: If the `Evaluator` passed is stateless, then the `Unifier` should be null.

For more information about the `Executor` interface and the `ExpressionFactory` class, see the *Javadoc*.

Code Examples

This section contains examples that illustrate how to construct expressions programmatically and use the Expression package services.

This section contains the following four code examples:

- Stateful Evaluation of a Simple Expression
- Stateful Evaluation of an Expression Containing Variables
- Stateless Validation and Evaluation of an Expression Containing Variables
- Stateful Validation and Evaluation of an Expression Containing Variables

Stateful Evaluation of a Simple Expression

A logical expression is constructed and executed in stateful mode. The expression does not contain any variables.

Listing 16-1 Example

The source code for creating and executing the expression is shown below:

```
Expression expression = new LogicalAnd(Boolean.TRUE,
Boolean.FALSE);

// Prepare for creating an executor by creating a stateful
// evaluator. Since the expression does not contain variables,
// we are not using a validator or a unifier in this example,
// so we will not create them.

// null is passed for the environment Map.

Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// null is passed for the environment Map.

Executor executor = ExpressionFactory.createExecutor(null);

// Execute the above expression by passing null for both the unifier
// and validator parameters.

Object result = executor.execute(expression, null, null,
evaluator);

// The result should be Boolean.FALSE.
```

Stateful Evaluation of an Expression Containing Variables

An expression containing variables is constructed and evaluated in stateful mode.

Listing 16-2 Example

The source code for creating and executing the expression in stateful mode is shown below.

```
// Create a variable that can store an object of type Boolean
// and whose name is "?booleanVariable".

Variable booleanVariable = new Variable("?booleanVariable",
Boolean.class);

// Now, we will use the variable that we created in the above step.
```

```
Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);

// Next, we’ll unify the expression by binding any variables
// present in the expression. In the above case, there is one
// variable in the expression so the variable needs to be assigned a
// value. This is shown below.

// Create a UnificationList to store the variable name and value as
// key-value pairs.

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject(“?booleanVariable”, Boolean.FALSE);

// Create a unifier.

Unifier unifier = ExpressionFactory.createUnifier(null,
unificationList);

// Prepare for creating an executor by creating a stateful
// evaluator. We are not using a validator in this example,
// so we will not create one.

// null is passed for the environment Map.

Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// null is passed for environment Map.

Executor executor = ExpressionFactory.createExecutor(null);

// Execute the above expression by passing a unifier and a null
// validator.

Object result = executor.execute(expression, unifier, null,
evaluator);

// The result should be Boolean.FALSE.
```

Note: The expression can be unified before calling the `execute` method by calling the `unify` method on the `Unifier`. Once the expression is unified there is no need to pass a unifier to the `execute` method of the executor.

Stateless Validation and Evaluation of an Expression Containing Variables

An expression containing variables is constructed and evaluated in stateless mode. The `Validator` service is also used to validate the expression.

Listing 16-3 Example

The source code for creating and executing the expression in stateless mode is shown below.

```
// Create a variable that can store an object of type Boolean
// and whose name is “?booleanVariable”.

Variable booleanVariable = new Variable(“?booleanVariable”,
Boolean.class);

// Now we will use the variable that we created in the above step.

Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);

// Next, we’ll unify the expression by binding any variables
// present in the expression. In the above case there is one
// variable in the expression, so the variable needs to be assigned
// a value. This is shown below.

// Create a UnificationList to store the variable name and value as
// key-value pairs.

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject(“?booleanVariable”, Boolean.FALSE);

// Prepare for creating an executor by creating a stateless
// evaluator. We are not using a unifier in this example,
// so we will not create one.

// Creating a stateless evaluator by passing null for the
// environment Map and the UnificationList.

Evaluator evaluator = ExpressionFactory.createEvaluator(null,
unificationList);

// Creating a stateless validator.
```

```
Validator validator = ExpressionFactory.createValidator(null,
evaluator);

// Creating an executor.

Executor executor = ExpressionFactory.createExecutor(null);

// Execute the above expression by passing null for the unifier and
// a non-null validator.

Object result = executor.execute(expression, null, validator,
evaluator)

// The result should be Boolean.FALSE.

// After calling execute method, the given expression will not be
// modified by any services that were used above.

// The stateless execution mode is useful if an expression is shared
// between multiple threads.
```

Stateful Validation and Evaluation of an Expression Containing Variables

An expression containing variables is constructed and evaluated in stateful mode. The `Validator` service is also used to validate the expression.

Listing 16-4 Example

The source code for creating and executing the expression in a stateful mode is shown below.

```
// Create a variable that can store an object of type Boolean
// and whose name is “?booleanVariable”.

Variable booleanVariable = new Variable(“?booleanVariable”,
Boolean.class);

// Now we will use the variable that we created in the above step.

Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);

// Next, we’ll unify the expression by binding any variables
// present in the expression. In the above case, there is one
// variable in the expression, so the variable needs to be assigned
// a value. This is shown below.
// Create a UnificationList to store the variable name and value
// as key-value pairs.

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject("?booleanVariable", Boolean.FALSE);

// Create a unifier.

Unifier unifier = ExpressionFactory.createUnifier(null,
unificationList);

// Prepare for creating an executor by creating a stateful
// evaluator and validator.

// null is passed for the environment Map.

Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// null is passed for the environment Map.

// Creating a validator.

Validator validator = ExpressionFactory.createValidator(null);

// Creating an executor.

Executor executor = ExpressionFactory.createExecutor(null);

// Execute the above expression by passing a unifier and a non-null
// validator.

Object result = executor.execute(expression, unifier, validator,
evaluator);

// The result should be Boolean.FALSE.
```

Note: The expression can be unified before calling the `execute` method by calling the `unify` method on the `Unifier`. Once the expression is unified there is no need to pass a unifier to the execute method of the `Executor`. The validation service can be used directly by calling the `validate` method. The `validate` method throws an `InvalidExpressionException` if the given expression is invalid.

Configuring the Expression Package

The `expression.properties` file contains configuration settings for the Expression package and should be modified with care.

This file is archived in `p13n_util.jar` under the package `com.bea.p13n.expression`.

```
##
# Expression Comparator null handling
#
# If the following property is set to true the Expression
# Comparator will return false as the result of comparing
# any non-null value to a null, regardless of the
# comparison being performed.
#
# Defaults to true.
##

expression.comparator.nullcheck=true

##

# Expression Comparator equality epsilon.
#
# The following property determines the epsilon value for
# numeric equality comparisons.
#

# Defaults to 0.
##

expression.comparator.epsilon=0.00001

##
# Expression Introspector Method Array Caching
#
# If the following property is set to true the Expression
# Introspector will cache the array of Methods implemented by a
# Java Class.
#

# Defaults to true.
##

expression.introspector.method.array.cache=true
```

```
##
# Expression Introspector Method Caching
#
# If the following property is set to true the Expression
# Introspector will cache Methods by signature.
#
# Defaults to true.
##

expression.introspector.method.cache=true

##
# Expression Parser Node Support Classes
#
# This property supports a comma-delimited list of classes
# extending the base AST NodeSupport class. Such classes
# provide node creation support for expression-schema namespaces
# required for constructing the intermediate AST representing a
# given Expression instance.
#
# All NodeSupport subclasses must co-exist peacefully with the
# required CoreNodeSupport instance.
##

parser.node.support.list=\

com.bea.p13n.expression.internal.parser.expression.ExpressionNode
Support

##
# Expression Parser Transform Visitor Class
#
# This property specifies the ExpressionTranformVisitor or
# subclass to be used for intermediate AST-to-Expression
# transformations.
#
##

parser.transform=\

com.bea.p13n.expression.internal.parser.expression.ExpressionTran
sformVisitor
```

A Event Descriptions

This appendix provides information about the standard events provided with WebLogic Portal. Specifically, it contains a description of each kind of event, what generates the event, the class where event generation occurs, an example of usage, and the type of data within each event object.

WebLogic Portal Events are organized in the following categories:

- Session Events
- User Registration Event
- Product Events
- Content Events
- Cart Events
- Buy Event
- Rules Event
- Campaign Events

Session Events

Session events fire at the start time, end time, and if executed, the login time of a visitor's session.

SessionBeginEvent

Description	Occurs when a visitor begins interacting with a Web or portal site.
Class	`com.bea.p13n.tracking.events.SessionBeginEvent`
Generator	See "Servlet Lifecycle Events and Servlet Filter Events" on page 15-9.

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`

SessionEndEvent

Description	Occurs when a visitor leaves a site, or when the visitor's session has timed out.
Class	`com.bea.p13n.tracking.events.SessionEndEvent`
Generator	See "Servlet Lifecycle Events and Servlet Filter Events" on page 15-9.

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`

SessionLoginEvent

Description	Occurs when a visitor logs on a Web or portal site.

Class	`com.bea.p13n.tracking.events.SessionLoginEvent`
Generator	`TrackingEventHelper.dispatchSessionLoginEvent()`, `P13NAuthFilter`, and/or Input Processor. See "Generating Login and Creation Events" on page 15-10.

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`

User Registration Event

Only one registration event exists. It is described in the following table.

UserRegistrationEvent

Description	Occurs when visitor registers on a Web or portal site.
Class	`com.bea.p13n.tracking.events.UserRegistrationEvent`
Generator	`TrackingEventHelper.dispatchUserRegistrationEvent()` and/or Input processor.
Example Usage	`examples.wlcs.sampleapp.customer.webflow.LoginCustomerIP` located in `<BEA_HOME>\weblogic700\portal\samples\portal\wlcsDomain\wlcsApp\wlcs\WEB-INF\src`

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`

Product Events

These events occur when visitor is presented with a product or clicks (selects) the presented product.

ClickProductEvent

Description	Occurs when a visitor clicks a product link.
Class	`com.bea.commerce.ebusiness.tracking.events.ClickProductEvent`
Generator	JSP Tag. Also see "Servlet Lifecycle Events and Servlet Filter Events" on page 15-9.

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`
	`sku`
	`category-id`
	`application-name` (name of storefront, not portal application)

DisplayProductEvent

Description	Occurs when a product is displayed to the visitor.
Class	`com.bea.commerce.ebusiness.tracking.events.DisplayProductEvent`
Generator	JSP Tag

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`
	`sku`
	`category-id`
	`application-name` (name of storefront, not portal application)

Content Events

These events occur when the visitor is presented some content, such as an advertisement, or clicks the presented content.

ClickContentEvent

Description	Occurs when a visitor clicks some Web site content, such as a link or banner.
Class	`com.bea.p13n.tracking.events.ClickContentEvent`

Generator	JSP Tag. Also see "Servlet Lifecycle Events and Servlet Filter Events" on page 15-9.

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`

DisplayContentEvent

Description	Occurs when content is presented to a visitor, usually any content from a content management system.
Class	`com.bea.p13n.tracking.events`
Generator	JSP Tag

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`

Cart Events

These events indicate that one or more items are added or removed from a visitor's shopping cart.

AddToCartEvent

Description	Occurs when an item is added to a visitor’s shopping cart.
Class	`com.bea.commerce.ebusiness.tracking.events.AddToCartEvent`
Generator	Pipeline component. Located in `<BEA_HOME>\weblogic700\portal\applications\wlcsApp-project\application-sync\pipelines`.
Example Usage	`examples.wlcs.sampleapp.tracking.pipeline.AddToCartTrackerPC` located in `<BEA_HOME>\weblogic700\portal\samples\portal\wlcsDomain\beaApps\wlcsApp\src`

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`sku`
	`quantity`
	`unit-list-price`
	`currency`
	`application-name` (name of storefront, not portal application)

RemoveFromCartEvent

Description	Occurs when an item is removed from a visitor’s shopping cart.
Class	`com.bea.commerce.ebusiness.tracking.events.RemoveFromCartEvent`
Generator	Pipeline component. Located in `<BEA_HOME>\weblogic700\portal\applications\wlcsApp-project\application-sync\pipelines`

Example Usage	`examples.wlcs.sampleapp.tracking.pipeline.RemoveFromCartTrackerPC` located in `<BEA_HOME>\weblogic700\portal\samples\portal\wlcsDomain\beaApps\wlcsApp\src`

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`sku`
	`quantity`
	`unit-price`
	`currency`
	`application-name` (name of storefront, not portal application)

PurchaseCartEvent

Description	Occurs once for an entire order, unlike the `BuyEvent`, which occurs for each line item. This event is useful for campaigns. You can use it when writing scenario actions to know when your visitor makes a purchase with specific characteristics, such as an order greater than $100 or the purchase of a particular product.
Class	`com.bea.commerce.ebusiness.tracking.events.PurchaseCartEvent`
Generator	Pipeline component. Located in `<BEA_HOME>\weblogic700\portal\applications\wlcsApp-project\application-sync\pipelines`.
Example Usage	`examples.wlcs.sampleapp.tracking.pipeline.PurchaseTrackerPC` located in `<BEA_HOME>\weblogic700\portal\samples\portal\wlcsDomain\beaApps\wlcsApp\src`

Elements	`application`
	`session-id`
	`user-id`
	`event-date`
	`event-type`
	`total-price`
	`order-id`
	`currency`
	`application-name` (name of storefront, not portal application)

Buy Event

Only one buy event exists. It is described in the following table.

BuyEvent

Description	Occurs when a visitor completes the purchase. A `BuyEvent` occurs for each line item. A purchase may consist of one or more line items. A line item may consist of one or more items. For example, although a particular line item may have quantity of four items, only one `BuyEvent` occurs.
Class	`com.bea.commerce.ebusiness.tracking.events.BuyEvent`
Generator	Pipeline component
Example Usage	`examples.wlcs.sampleapp.tracking.pipeline.PurchaseTrackerPC` located in `<BEA_HOME>\weblogic700\portal\applications\wlcsApp\src`

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`sku`
	`quantity`
	`unit-price`
	`currency`
	`application-name` (name of storefront, not portal application)
	`order-line-id`

Rules Event

Only one rule event exists. It is described in the following table.

RuleEvent

Description	Indicates the rules that were fired as a visitor navigates a Web site.
Class	`com.bea.p13n.tracking.events.RuleEvent`
Generator	Fired internally from advislets

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`ruleset-name`
	`rule-name`

Campaign Events

These events occur when a visitor participates in a campaign.

CampaignUserActivityEvent

Description	Occurs when a visitor participates in a campaign. Specifically, this event is fired whenever one or more scenario actions are true and the campaign service is activated. You can limit this event to a single occurrence for a particular scenario. This event is intended for use by analytic software.
Class	`com.bea.campaign.tracking.events.CampaignUserActivityEvent`
Generator	Fired internally from the campaign service.

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`campaign-id`
	`scenario-id`

DisplayCampaignEvent

Description	Occurs when campaign content, such as an ad, is presented to the visitor. Specifically, this event is fired whenever a campaign placeholder displays an ad placed in the ad bucket by a campaign. You can use this event to trigger another campaign. Analytic software uses this event to determine if a visitor saw an ad as a result of a campaign.

Class	`com.bea.campaign.tracking.events.CampaignUserActivityEvent`
Generator	Fired internally from the campaign service.

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`
	`campaign-id`
	`scenario-id`
	`application-name` (name of storefront, not portal application)
	`placeholder-id`

ClickCampaignEvent

Description	Occurs when a campaign item, such as an ad, is clicked on by the visitor. Specifically, this event is fired whenever a visitor clicks a campaign ad that was placed in the ad bucket by a campaign. You can use this event to trigger another campaign. Analytic software uses this event to determine if a visitor clicked on an ad as a result of a campaign.
Class	`com.bea.campaign.tracking.events.ClickCampaignEvent`
Generator	Fired internally from campaign service. Also see "Servlet Lifecycle Events and Servlet Filter Events" on page 15-9.

Elements	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`
	`campaign-id`
	`scenario-id`
	`application-name` (name of storefront, not portal application)
	`placeholder-id`

Index