BEA WebLogic Portal®

Best Practices Guide

Version 8.1
July 2003
Revised: July 2003

// Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

What You Need to Know i
Product Documentation on the dev2dev Web Site i
Related Information ii
Contact Us! ii
Documentation Conventions iii

Development Strategies

Developing for WebLogic Portal 8.1 1-1
About Split Configuration 1-1
Adding Visitor Tools to a Custom Application 1-1
Authenticating Users 1-2
Using the Anonymous User Profile 1-3
Content Placeholders 1-3
Page Flows and Portlets 1-3
Processors 1-4
Varying Content for Mobile Devices 1-5
Samples of MultiChannel Functionality 1-9
Working with Look and Feel Elements 1-9
Look and Feel Files 1-9
Creating a Look and Feel File 1-9
Creating Skins and Skin Themes 1-11

Creating a Skin 1-11
Creating a Skin Theme 1-12
Skeletons and Skeleton Themes. 1-13
When should you create a skeleton?. 1-14
Creating a Skeleton. 1-14
Creating a Skeleton Theme. 1-15
Layouts. 1-18
Creating a Layout 1-19
Navigation Menus 1-22
Modifying the Navigation Menu File 1-23
Modifying The Skeleton JSP File. 1-23
Shells. 1-24
Creating a Shell. 1-24

Configuration and Administration

Configuring WebLogic Portal 8.1. 2-1
Split Configuration. 2-1
Deploying an EAR Versus an Exploded Application 2-1
How to Use Active Directory. 2-2
Single Signon Between Two Portal Web Applications 2-2

Performance

Performance Tuning 3-1
Managing Caches. 3-1
How to Use Caching Tables. 3-2
Notes on Threads 3-14
About the 8.1 Forkable Portlet Feature. 3-14

About This Document

This document includes best practices information on BEA WebLogic Portal 8.1.

This document covers the following topics:

- Chapter 1, "Development Strategies," introduces some concepts for developers and architects building portal Web applications.
- Chapter 2, "Configuration and Administration," includes instructions for content management, deployment and configuration of WebLogic Portal 8.1.
- Chapter 3, "Performance," includes information on caching and threading in WebLogic Portal 8.1 applications.

What You Need to Know

This document is intended for new or existing BEA customers interested in WebLogic Portal 8.1.

Product Documentation on the dev2dev Web Site

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

http://dev2dev.bea.com

To view the documentation for a particular product, select that product from the list on the dev2dev page; the home page for the specified product is displayed. From the menu on the left

side of the screen, select Documentation for the appropriate release. The home page for the complete documentation set for the product and release you have selected is displayed.

Related Information

Readers of this document may find the following documentation and resources especially useful:

- For helpful information about programming with the Studio client, see the following books in the WebLogic Integration document set:
  - *Using the WebLogic Integration Studio*
  - *BEA WebLogic Integration Javadoc*
- For general information about Java applications, go to the Sun Microsystems, Inc. Java Web site at `http://java.sun.com`.
- For general information about XML, go to the O'Reilly & Associates, Inc. `XML.com` Web site at `http://www.xml.com`.

Contact Us!

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. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Portal documentation.

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

If you have any questions about this version of BEA WebLogic Portal, or if you have problems installing and running BEA WebLogic Portal, contact BEA Customer Support at **http://support.bea.com**. You can also contact Customer Support by using the contact information provided on the quick reference sheet titled "BEA Customer Support," which is included in the product package.

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

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using

- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.
`monospace text`	Indicates *user input*, as shown in the following examples: • Filenames: `config.xml` • Pathnames: *`BEAHOME`*`/config/examples` • Commands: `java -Dbea.home=BEA_HOME` • Code: `public TextMsg createTextMsg(`
	Indicates *computer output*, such as error messages, as shown in the following example: `Exception occurred during event dispatching:java.lang.ArrayIndexOutOfBoundsException: No such child: 0`
`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `java utils.MulticastTest -n` *`name`* `[-p` *`portnumber`*`]`
		Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. *Example*: java weblogic.deploy [list|deploy|update]

<table>
<tr><th>Convention</th><th>Item</th></tr>
<tr><td><code>. . .</code></td><td>Indicates one of the following in a command line:
That an argument can be repeated several times in a command line
That the statement omits additional optional arguments
That you can enter additional parameters, values, or other information
The ellipsis itself should never be typed.

Example:
<code>buildobjclient [-v] [-o name] [-f "file1.cpp file2.cpp file3.cpp . . ."</code></td></tr>
<tr><td>.
.
.</td><td>Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.</td></tr>
</table>

CHAPTER 1

Development Strategies

Developing for WebLogic Portal 8.1

This section includes some guidelines and solutions to problems developers often face working with WebLogic Portal Platform Edition, and includes information on the following topics:

- About Split Configuration
- Adding Visitor Tools to a Custom Application
- Varying Content for Mobile Devices
- Working with Look and Feel Elements

About Split Configuration

WebLogic Portal does not support a split configuration, where servlet and EJB containers run on separate server instances.

Adding Visitor Tools to a Custom Application

WebLogic Portal Platform Edition installs with a pre-built portal application called sampleportal, which includes, among other things, a set of JSPs that enable visitors to set properties on personalized view of the portal. Using the following procedure, these visitor tools can be added to a new portal application and then customized.

1. In the native file system, copy the following directories inclusively:

– Copy the entire **com/bea/jsptools/portal** directory from the **WEB-INF/classes** directory in **sampleportal** into the **WEB-INF/classes** directory of the new Web application.

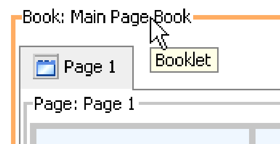
– Copy the entire **com/bea/jsptools/portal** directory from the **WEB-INF/classes** directory in **sampleportal** to the **WEB-INF/classes** directory of the new Web application.

– Copy the entire **visitorTools** directory from the **sampleportal** directory in **portalApp** to the **Portal Project** directory of the new Web application.

2. In WebLogic Workshop Platform Edition, open the new Portal application.

3. In the Portal Designer, select the main book.

Figure 0-1 Selecting Main Book



4. In the Properties Designer, set the Editable property to **Edit in Menu**. The **Mode Properties** heading is added to the available properties.

5. Click on the Content URI and browse to the **/visitorTools/visitorTools.portion** file.

6. When this portal runs, (not in development mode, but in a live desktop) the "edit" icon will appear on the main book. Clicking this icon will take the user to the Visitor Tools.

Note: The Visitor Tools JSPs only work when the portal server is running, and must be accessed by a user logged into the desktop.

Authenticating Users

The PortalServletFilter will guarantee that the correct user profile is configured in the session, so you no longer need to write custom code to do so.

<um:login>

By default, this tag handles the post-user-login process of authenticating the user, updating the profile in the session to the user's profile, and firing the UserRegistrationEvent. Additionally, the PortalServletFilter will handle the post-user-login process on the next request if the user is authenticated in another fashion (e.g. form-based, j_security_check,ServletAuthentication.weak()).

<um:createUser>

By default, this tag handles the post-user-registration process of saving the anonymous profile properties to the user's profile, logging in the user (which will authenticate the user, update the profile in the session to the user's profile and fire a SessionLoginEvent) and firing the UserRegistrationEvent.

Using the Anonymous User Profile

Take advantage of the anonymous user profile. Set properties in it. All user-based features can operate against an anonymous profile (for example, <pz:div>, <pz:contentSelector>, <ph:placeholder>), so you don't need to require a login for those features. Those properties will get persisted to the user's profile via <um:createUser>. When writing self-registration pages, it's best to set all the properties in the (anonymous) profile via <um:setProperty> before calling <um:createUser>. This is easier than trying to do so after creating the user, and will make sure the properties are set on the user's profile prior to firing the SessionLoginEventandUserRegistrationEvent.

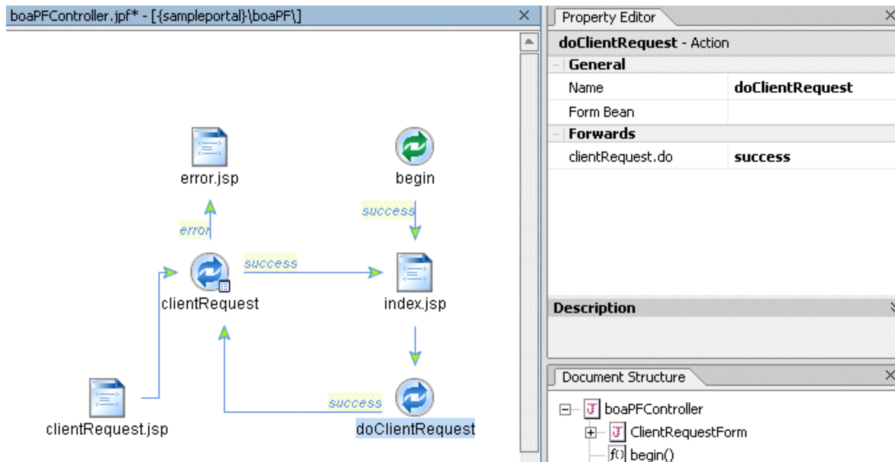
Content Placeholders

Content Placeholders now support queries that refer to user, request, and session properties. You no longer need a campaign to just put customized queries in a placeholder for a user. Additionally, content search queries in <cm:search> and <pz:contentQuery> can also refer to user, request, and session properties (e.g. " color = userProperty('myPropSet', 'favoriteColor') ").

Page Flows and Portlets

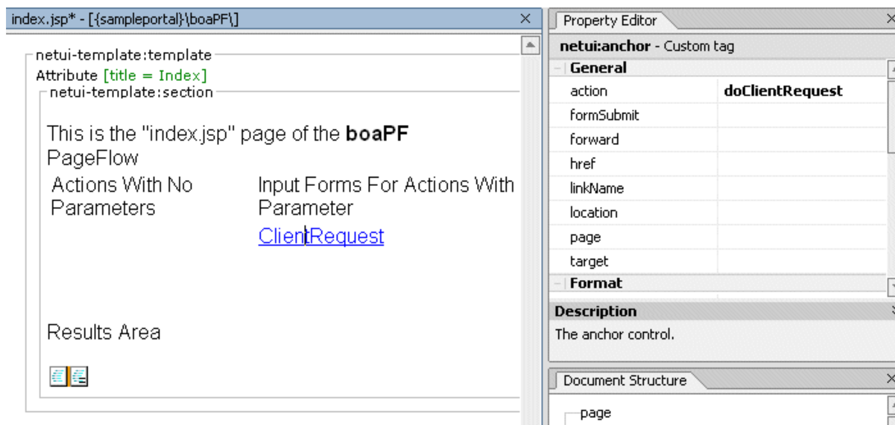
When using Page Flows, it is recommended that you not use hrefs within the JSPs. Instead, use an action by dragging an action into the flow and give it a name such as 'doClientRequest'. Link from this to the page you wish to go to, so that the flow looks something like Figure 0-1:

Figure 0-1 pageflow1



On the JSP where you call the other page, which is **index.jsp** in Figure 0-1, open the JSP and edit the anchor tag. Change it from an href to an action, as in Figure 0-2:

Figure 0-2 pageflow2



Processors

If you need to write a Processor (such as those used in a webflow) which adds data in the request, you need to consider refresh of the page. For example, if a Processor gets data and stores in the request, and then a JSP reads this data in the request to display information, an event refresh link

to the **lastcontenturl** of the master webflow will lose data stored by Input Processor in the request.

As discussed in the section "Page Flows and Portlets," you should never use HREFs with Page Flows. You can react to the refresh event (one of the standard portlet webflow events) to invoke an Input Processor or Pipeline Component. That way, the data is available for the JSP.

Although it is possible to write Pipeline Components that store the 'output data' in the pipeline session at session scope, this can be inflexible as a design pattern. Pipelines should be agnostic to the presentation; it is the webflow (Input Processors) that should prepare the rendering data and therefore they should determine when to persist to session scope, e.g. to retain presentation data after refreshes, etc.

Although it is a small overhead, it is recommended that Pipeline Components write output data to request scope unless it is for "internal" state use so that the contract between the pipelines and the webflows is the (pipeline session) request. This helps to ensure more portable/reusable portal components.It is also good practice to actively maintain the data held at session scope, to prevent session "memory leaks" and reduce the overhead on cluster replication.

Varying Content for Mobile Devices

There are many types of Web-enabled mobile devices that can access your portals. Since these devices have different interfaces and different-sized viewing areas, each has a unique requirement for the type of content they display.

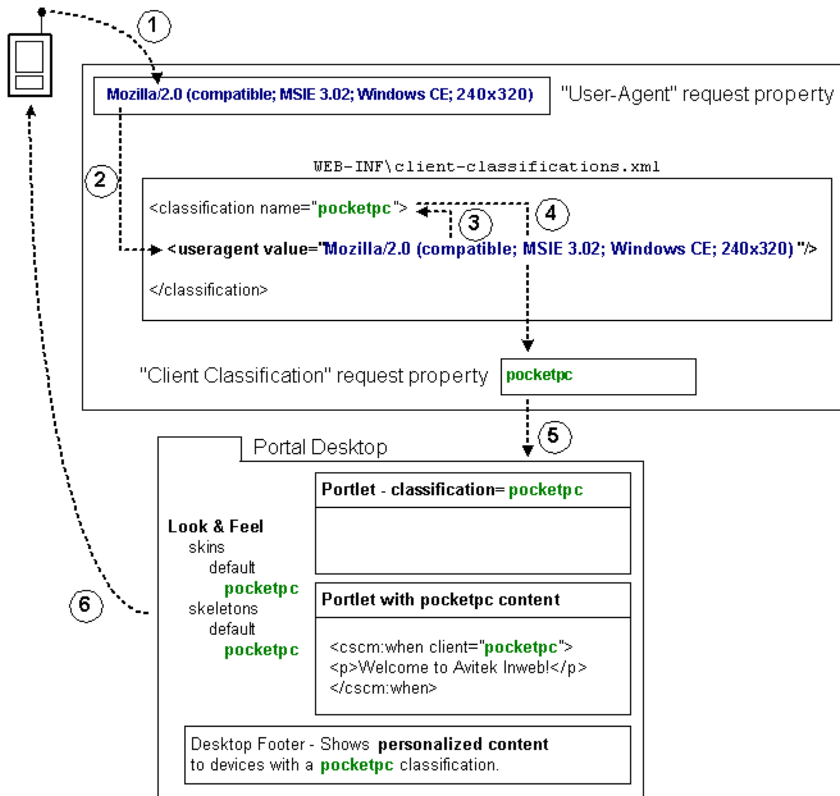
With the multichannel framework provided in WebLogic Workshop Portal Extensions, you can extend your portals to include support for mobile device access. This flexible framework lets you create a single portal that serves content to Web-capable devices seamlessly and simultaneously. You can also serve different content to different browsers, such as Mozilla, Netscape, Opera, and Internet Explorer.

The multichannel framework allows the following processes to occur: You can build specific content and look and feel elements for specific devices. When a device accesses a portal, the portal knows what kind of device it is and automatically serves the device the content you created for it.

When a device (whether a PC or a handheld) accesses a portal, it sends information about itself to the portal in the HTTP header—information such as the type of browser being used and the type of device. This combination of information defines a "client," which is equivalent to the model of a device. Clients, in turn, can be grouped into "classifications." For example, there are many models of Palm handheld devices, but they all fall under the classification of "Palm." Classifications are the key element in enabling multichannel support in portals.

Figure 0-3 illustrates the multichannel framework and provide instructions for building content and presentation for mobile devices. This illustration is annotated by the following numbered section.

Figure 0-3 Multi-Channel Request Cycle



1. When a device accesses a portal-enabled server with a URL, the device sends a user-agent string in the HTTP header that tells what kind of client it is. The server stores this user-agent string in the "User-Agent" request property for the portal application.

   The "User-Agent" request property is automatically included with any portal application you create in WebLogic Workshop Platform Edition. To view this property, open the following file in WebLogic Workshop:
   <PORTAL_APP>\data\request\DefaultRequestPropertySet.req.

Portal developer tasks: None. This happens automatically.

2. To enable multichannel support for devices, a portal Web project must be able to map the user-agent string stored in the "User-Agent" property to a classification. This mapping must be created before portals are accessed by mobile devices.

   Portal developer tasks: You must map clients to classifications in your portal Web project WEB-INF\client-classifications.xml file. The default client-classifications.xml file contains default client mappings.

   For each client entry that maps to a classification, you can enter either an explicit user-agent string that maps exactly to what a device sends, or you can enter a regular expression that can encompass multiple user-agent strings.

   The following example of a client classification mapping in client-classifications.xml shows explicit mappings (with the <useragent> tag) and a regular expression mapping (with the <useragent-regex> tag).

```
<classification name="pocketpc" description="For the PocketPC">

<useragent value="Mozilla/2.0 (compatible; MSIE 3.02; Windows CE;
240x320)"/>

<useragent value="Mozilla/2.0 (compatible; MSIE 3.02; Windows CE; PPC;
240x320)"/>

<useragent-regex value=".*PDA; Windows CE.*NetFront/3.*" priority="1"/>

</classification>
```

   An explicit <useragent> value can be used for only one classification. If you use more than one <useragent-regex> tag to map with regular expressions, it is possible that a device accessing a portal could map to more than one classification. To determine which classification the device is mapped to, use the priority attribute, as shown above. The value "1" is the highest priority. Enter any whole number for the priority value.

Note: For portlets that are assigned client classifications, the classification "description" value is used in the WebLogic Administration Portal to show which classifications the portlet is assigned to. Write descriptions that are easily understood by portal administrators.

3. Because of the client-classification.xml mappings you defined, the user-agent string stored in the "User-Agent" property is mapped to the classification name you provided. In the example mapping above, the name is "pocketpc".

   Portal developer tasks: None. This happens automatically.

4. After the client is successfully mapped to a classification, the classification name is stored in the "Client Classification" property in the DefaultRequestPropertySet.

Portal developer tasks: None. This happens automatically.

5. The portal uses that client classification name stored in the DefaultRequestPropertySet throughout the portal framework to identify the content and presentation tailored to the device.

   Portal developer tasks: The portal is where you develop and enable specific content and presentation to be used for different mobile devices. The portal framework includes the following touchpoints for creating device-specific content and presentation:

   Portlet Development - When you create a portlet with the WebLogic Workshop Portal Extensions, you can assign the portlet to be used by different devices (client classifications). With the portlet open in the Portlet Designer, in the Property Editor window, do the following:

   a. Click the ellipsis icon **[...]** in the Client Classifications field to launch the Manage Portlet Classifications dialog box.

   b. In the dialog box, select whether you want to enable or disable classifications for the portlet. (If you disable classifications, the portlet is automatically enabled for the classifications you do not select for disabling.)

   c. Move the classifications you want to enable/disable from the Available Classifications list to the Selected Classifications list, and click **OK**. The list of classifications is read from the client-classifications.xml file.

   JSP Tags - The WebLogic Workshop Portal Extensions include a set of JSP tags for creating device-specific inline content in JSPs. Only the content that meets the device criteria defined by the JSP tag is delivered to the device.

   The JSP tags have a required "client" attribute for mapping the JSP content to classifications. For that client value in the JSP tag, you must use the exact value used for the name in the client-classification.xml file (the value being stored in the "Client Classification" property in the DefaultRequestPropertySet).

   Look & Feel Development - The Look & Feels (skins and skeletons) provided with the WebLogic Workshop Portal Extensions include support for a few mobile devices (nokia, palm, and pocketpc). These skins and skeletons are included as subdirectories of the main skins and skeletons in your portal Web projects. For example, a pocketpc skin is included as part of the "default" skin in <project>\framework\skins\default\pocketpc.

   You can also develop your own skins and skeletons to support different devices. When a Look & Feel is selected for a desktop, the portal framework reads the "Client Classification" property in the DefaultRequestPropertySet and uses the Look & Feel logic to find skin and skeleton directories matching the name of the client classification.

Interaction Management Development - With the client classification name being stored in the "Client Classification" property of the DefaultRequestPropertySet, you can build and trigger personalization and campaigns for devices based on that property value.

6. Based on the mapping you set up to match user-agent (client) strings in the HTTP request to classification names, the portal sends the device-specific content and presentation you developed to the different devices that access the portal.

   Portal developer tasks: None. This happens automatically.

Samples of MultiChannel Functionality

The Tutorial Portal, one of the Portal Samples provided with the WebLogic Workshop Portal Extensions, includes examples of multichannel functionality. Also, when you create a portal Web project, a WEB-INF\client-classifications.xml file is created automatically with default settings.

Any portal Web project you create also includes a default set of multichannel Look & Feels located in skin and skeleton subdirectories (<project>\framework\skins and <project>\framework\skeletons).

Working with Look and Feel Elements

This section includes instructions on creating Look and Feel elements to vary the presentation of a portal at runtime. For a description of how these elements work together, consult the Presentation Framework section of the *WebLogic Portal Upgrade Guide*.

Look and Feel Files

A Look and Feel is described by a simple XML file that determines the skin and skeleton used for the Look and Feel. When you create a Look and Feel file, you can select the new Look and Feel in the Portal Designer for your portal desktops.

The following topics describe the Look and Feel architecture and show you how to create and use Look and Feels in your portals.

Creating a Look and Feel File

1. In WebLogic Workshop, with your portal application open, use the Application window to open an existing Look and Feel file: **`<project>\framework\markup\lookandfeel*.laf`**.

2. Choose File-->Save As and rename the file. Be sure to keep the .laf extension and store it in the same directory as the other Look and Feel files.

3. In the <netuix:lookAndFeel> element, change the following attribute values to the name of your Look and Feel: definitionLabel, title, description, and markupName. Each Look and Feel must have a unique markupName. Do not modify the markupType value.
4. For the skin attribute value, enter the directory name of the skin you want to use.
5. For the skinPath attribute value, enter the relative path (relative to the project folder) of the skin you want to use, up to the skin's parent folder.

   For example, if you created a skin stored in <project>/framework/skins/modern (the name of the skin is "**modern**"), your skin attributes would look like this:

   ```
   skin="modern" skinPath="/framework/skins/"
   ```

   The /framework/skins/ path is the default path used by the portal framework. If you did not enter a value for skinPath, the default path would be used.
6. For the skeleton attribute value, enter the directory name of the skeleton you want to use.
7. For the skeletonPath attribute value, enter the relative path (relative to the project folder) of the skeleton you want to use, up to the skeleton's parent folder.

   For example, if you created a skeleton stored in <project>/framework/skeletons/modern (the name of the skeleton is "**modern**"), your skeleton attributes would look like this:

   ```
   skeleton="modern" skeletonPath="/framework/skeletons/"
   ```

   The /framework/skeletons/ path is the default path used by the portal framework. If you did not enter a value for skeletonPath, the default path would be used.

   In many cases, you may just want to use the "default" skeleton stored in <project>/framework/skeletons/default.

   If you do not provide skeleton attributes, the skeleton identified in the skin.properties file is used. See Creating Skins and Skin Themes.
8. You can also set the default icon to be used in portlet titlebars by setting the defaultWindowIcon and defaultWindowIconPath attributes. For example, if the icon you want to use is located at <project>/images/window-icon.gif, set the attributes like this:

   ```
   defaultWindowIcon="window-icon.gif" defaultWindowIconPath="images/"
   ```
9. Save the file.
10. To use the Look and Feel for a portal, open the portal in WebLogic Workshop Platform Edition, select the Desktop icon in the Document Structure window, and select the Look and Feel in the Property Editor Look and Feel field.

Selecting a Look and Feel for a desktop in the Portal Designer simply gives the portal a default Look and Feel setting. Portal administrators and end users can change the Look and Feel used for a desktop.

The real key to Look and Feels working properly is in the correct creation and storage of your skins and skeletons. Skin and skeleton property files must be set up correctly and include all necessary paths, skeleton JSPs must be valid, and and skin resources (such as images, CSS files, and JavaScript files) must be referenced correctly in your skeleton files. For example, the icons for the portlet title bars must use the correct graphics names.

Creating Skins and Skin Themes

Skins are the graphics, cascading style sheets (CSS), and JavaScript behaviors that define button graphics, text styles, mouseover actions, and other elements in the way a portal looks. Skins, combined with skeletons, make up a portal desktop's Look and Feel. When you select a Look and Feel for a portal desktop, the Look and Feel points to the skins and skeletons to use.

A skin is a unified collection of graphics, CSS files, and JavaScript files stored under a parent skin directory. You can create as many skins as you need. Skins can also contain subdirectories for mobile device-specific skins.

Skins can also contain themes. A skin theme is a subset of graphics, CSS styles, and/or JavaScript behavior that can be used on books, pages, and portlets to give them a different look than the rest of the portal desktop.

Creating a Skin

1. With your portal application open in WebLogic Workshop platform edition, duplicate an existing skin directory in your Portal Web Project. For example, right-click <project>\framework\skins\default and choose Duplicate.

   The new skin directory appears with a number appended to the end of the name.

2. Rename the new skin directory.

3. Delete any skin subdirectories you do not plan to use.

4. In the images subdirectories, modify the button icons as desired. Do not rename any of the files. The filenames are used in skeleton JSP and class files to render the buttons.

5. In the css subdirectories, modify the CSS files as desired. Do not rename any of the styles in the CSS files. The style names are used in places like skeleton JSPs and layout files to render the styles.

To support Multi Level Menus in mobile devices, delete the line display: none; in the book.css file.

6. In the js subdirectories, modify the JavaScript as desired.

   Do not put business logic in skins. Create separate JSPs for business logic and surface those JSPs either in the portal shells (for the desktop headers or footers) or in portlets.

7. Modify the skin.properties file for the root skin and any sub skins.

   The skin.properties file contains references to images, themes, stylesheet links, JavaScript script entries, skeleton dependencies, and other information. The is self-documented to guide you through the file modification process.

   Entering all references in skin.properties is important because these references are inserted into the HTML head when the portal is rendered. Missing references will cause the skin to render incorrectly.

8. Open or create a Look and Feel file and associate the skin with the Look and Feel. See Creating Look and Feel Files.

9. To use the skin for a portal, open the portal in WebLogic Workshop Platform Edition, select the Desktop icon in the Document Structure window, and select the Look and Feel in the Property Editor Look and Feel field.

Selecting a Look and Feel for a desktop in the Portal Designer simply gives the portal a default Look and Feel setting. Portal administrators and end users can change the Look and Feel used for a desktop.

Creating a Skin Theme

A theme is represented by a single .theme file that is shared between skins and skeletons. For example, if you select a theme called "alert" for a portlet, the portal framework looks for skin and skeleton subdirectories called "alert." If a theme already exists that you want to simply create a skin for, start with step 5 of this procedure.

1. With your portal application open in WebLogic Workshop platform edition, duplicate an existing theme file in your Portal Web Project. For example, right-click <project>\framework\markup\theme\alert.theme and choose Duplicate.

   The new theme file appears with a number appended to the end of the name.

2. Rename the new theme file. Be sure to retain the .theme extension.

3. With the new theme file open, modify the following attributes in the <netuix:theme> element: name, title, description, markupName. The title attribute provides the name for selecting the theme in a drop-down menu; the markupName must be unique among the other themes.
4. Save the theme file.
5. In a skin directory, create a subdirectory with the same name as the theme.
6. Copy the appropriate skin directories and files from an existing skin into the new theme directory.
7. Modify the skin files in the skin theme. Do not modify filenames.
8. Associate the new theme with the skins you want to use the theme. In the skin.properties file of each skin, do the following:
   a. Reference the theme in the "THEME IMAGES DIRECTORIES" section of the file.
   b. If the theme includes unique stylesheets, reference those in the "LINK ENTRIES" section of the file.
   c. If the theme includes unique scripting, reference the scripts in the "SCRIPT ENTRIES" section of the file.
9. Save the skin.properties file.
10. Modify the skin theme's skin.properties file.
11. Copy the skin theme directory as a subdirectory to other skin directories.

All available themes (identified by the .theme files) are selectable for books, pages, and portlets regardless of whether or not a skin contains them. If the Look and Feel selected for the desktop references a skin that does not contain the selected theme in its skin.properties file, as outlined in the previous steps, no theme is used.

Skeletons and Skeleton Themes

A portal desktop is a collection of portal components, such as books, pages, and portlets, that have a hierarchical relationship to each another. (Books contain pages, pages, contain portlets, and so on.) Since portal components are largely XML files, rendering them in a browser requires a conversion to HTML. That rendering is the function of skeletons.

Each portal component has one or more corresponding skeleton JSP files. When a portal desktop is rendered, the skeleton JSPs for each portal component (in conjunction with any related classes)

perform their logic and insert the resulting HTML into the correct hierarchical locations of the HTML file.

Skeletons can also contain themes. A skeleton theme is a subset of skeleton JSPs that can be used on books, pages, and portlets to give them a different feel than the rest of the portal desktop. Skeletons, combined with skins, make up a portal desktop's Look and Feel. When you select a Look and Feel for a portal desktop, the Look and Feel points to the skeletons and skins to use.

When should you create a skeleton?

When you create a Portal Web Project in a portal application, the project includes predefined skeletons you can use. In most cases you can use the predefined skeletons to suit your needs. Changing the physical appearance and behavior of a portal desktop can be accomplished largely by creating new skins and shells rather than creating new skeletons.

Create a skeleton if you need to:

- Change the default behavior of an existing skeleton without modifying the existing skeleton.
- Create rendering behavior not provided in the default skeletons. For example, you can set an "Orientation" attribute on books and portlets that determine the physical location of Navigation Menus and titlebars. The default skins do not provide logic for changing orientation, so you could create a new book.jsp or titlebar.jsp skeleton to perform the desired behavior.
- Create skeletons to support specific classifications of mobile devices (for example, Palm).

Creating a Skeleton

To create a skeleton, take the following steps:

1. With your portal application open in WebLogic Workshop Platform Edition, create a copy of an existing skeleton directory. For example, right-click the <project>\framework\skeletons\default directory and choose Duplicate. When the duplicate directory appears, rename it.

   If you are creating a skeleton to support a mobile device, move the new directory to a subdirectory of the main skeleton. Give the new directory the exact name of the device's classification name in the <project>\WEB-INF\client-classifications.xml file.

2. In the new skeleton directory, open the skeleton.properties file. (If you copied the default skeleton directory, copy skeleton.properties from another skeleton into the root of your new skeleton directory and open it.)

3. In skeleton.properties, make any necessary modifications to the skeleton search order. For example:

   ```
   jsp.search.path: ., ../default
   ```

   With this entry, a skeleton file is first searched for in the current directory (.). If not found in the current directory, the ../default directory is searched. If no skeleton is found in either directory, the entire skeleton directory is searched until a skeleton is found.

4. Save skeleton.properties.

5. Modify the skeleton JSPs to perform the rendering you want. Use tags in the Portal Skeleton Rendering JSP tag library. In particular, use the <render:beginRender> and <render:endRender> tags.

   For guidance on which skeleton JSPs to modify, see the following table under How Portal Components Map to Skeletons.

   Do not rename the skeleton files. The skeleton filenames are hard-coded in their respective class files. The only exception to this is if you are creating a new rendering infrastructure that uses its own backing class files and explicitly identifies the name of the skeleton you are creating.

6. Save the skeleton JSPs.

7. Make any appropriate modifications to skeleton.properties or skeleton JSPs for any device skeletons stored in subdirectories of your skeleton.

8. To use the new skeleton, reference it in a Look and Feel file. See Creating Look and Feel Files.

Note: When working with portals in the WebLogic Workshop Portal Extensions Portal Designer, each selected portal component has a Skeleton URI property you can set in the Property Editor. This property lets you point to a specific skeleton file you want the component to use instead of the skeleton identified in the selected Look and Feel for the portal desktop.

Do not add business logic to skeletons. Skeletons are designed for physical rendering only. Add business logic to shells (headers or footers).

Creating a Skeleton Theme

A theme is represented by a single .theme file that is shared between skins and skeletons. For example, if you select a theme called "alert" for a portlet, the portal framework looks for skin and skeleton subdirectories called "alert." If a theme already exists that you want to simply create a skeleton for, start with step 5 of this procedure.

1. With your portal application open in WebLogic Workshop platform edition, duplicate an existing theme file in your Portal Web Project. For example, right-click <project>\framework\markup\theme\alert.theme and choose Duplicate.

   The new theme file appears with a number appended to the end of the name.

2. Rename the new theme file. Be sure to retain the .theme extension.

3. With the new theme file open, modify the following attributes in the <netuix:theme> element: name, title, description, markupName. The title attribute provides the name for selecting the theme in a drop-down menu; the markupName must be unique among the other themes.

4. Save the theme file.

5. In a skeleton directory, create a subdirectory with the same name as the theme.

6. Copy the appropriate skeleton JSPs from an existing skeleton directory into the new theme directory. You need skeleton JSPs for only the portal components you want the theme to affect. Portal components without corresponding theme skeleton JSPs will use the parent skeleton JSPs.

7. Modify the skeleton JSP files in the skeleton theme. Do not modify filenames.

8. Copy the skeleton theme directory as a subdirectory to other skeleton directories.

   All available themes (identified by the .theme files) are selectable for books, pages, and portlets regardless of whether or not a skeleton contains them. If the Look and Feel selected for the desktop references a skeleton that does not use the selected theme, no theme is used.

Table 1 Portal Components and Skeleton JSPs

This portal component...	uses this skeleton JSP...	to:
Desktop	desktop.jsp	Insert the HTML document declarations and insert <!-- Begin Desktop --> and <!-- End Desktop --> comments.
Shell	shell.jsp	Insert the HTML document's opening and closing <html> tag and insert <!-- Begin Shell --> and <!-- End Shell --> comments.
Shell - The <netuix:head> tag in a .shell file	head.jsp	Insert the HTML document's opening and closing <head> tag and insert <!-- Begin Head --> and <!-- End Head --> comments.

Table 1 Portal Components and Skeleton JSPs

This portal component...	uses this skeleton JSP...	to:
Shell - The <netuix:body> tag in a .shell file	body.jsp	Insert the HTML document's opening and closing <body> tag and provide presentation logic.
Shell - The <hetuix:header> tag in a .shell file	header.jsp	Render the desktop's header region.
Shell - The <netuix:footer> tag in a .shell file	footer.jsp	Render the desktop's footer region.
Book	book.jsp	Render the book framework and styles.
Navigation Menu	singlelevelmenu.jsp	Render the Single Level Menu provided by WebLogic Portal.
Navigation Menu	multilevelmenu.jsp	Render the Multi Level Menu provided by WebLogic Portal.
Book, Navigation Menu	submenu.jsp	Render Navigation Menus for books within books.
Page	page.jsp	Render a page framework and styles.
Layout - The <netuix:gridLayout> tag in a .layout file	gridlayout.jsp	Render placeholders in the layout using the Grid Layout style.
Layout - The <netuix:borderLayout> tag in a .layout file	borderlayout.jsp	Render placeholders in the layout using the Border layout style.
Layout - The <netuix:flowLayout> tag in a .layout file.	flowlayout.jsp	Render placeholders in the layout using the Flow layout style.
Layout - The <netuix:placeholder> tag in the .layout file	placeholder.jsp	Render an individual placeholder in a Layout.
Portlet titlebar	titlebar.jsp	Render a portlet titlebar.

Table 1 Portal Components and Skeleton JSPs

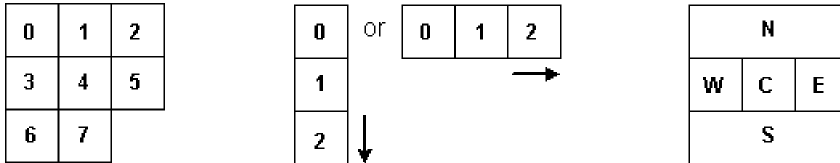
This portal component...	uses this skeleton JSP...	to:
Portlet titlebar buttons for floating windows	buttonfloat.jsp	Render a button that launches separate portlet mode windows (for example, Edit and Help).
Portlet titlebar toggle buttons	togglebutton.jsp	Render a button that toggles between portlet states (for example, Minimize/Restore and Maximize/Restore).
Portlet titlebar Delete button	togglebuttondelete.jsp	Render a button that removes a portlet from a page.
Portlet	error.jsp	Display error messages in a portlet.
Portlet	webflowportlet.jsp	Rrender a Webflow portlet created in previous versions of WebLogic Portal and running in a compatibility domain.
Book, Page, and Portlet	window.jsp	Rendering the container for the content area.
Theme	theme.jsp	Render books, pages, and portlets in the themes applied to them.

There are no skeleton files for skins. Skeletons include references to skin resources to render their components with appropriate graphics, styles, and JavaScript functionality. Though themes are related to skins, themes require a skeleton because themes are inline styles that must be inserted to override skin styles.

Layouts

Layouts provide the placeholders (table structure) for a page in which books, pages, and portlets can be placed. For example, a layout that uses three table cells provides three placeholders in which portlets can be placed on a page.

The WebLogic Workshop Portal Extensions provide the following three layout styles you can use to create your own layouts:



The **Grid Layout** automatically positions the number of placeholders you specify into the number of columns and rows you specify. This example sets columns="3" to position 8 placeholders.	The **Flow Layout** automatically positions the number of placeholders used either vertically or horizontally with no wrapping.	The **Border Layout** lets you use up to five placeholders. You can position the placeholders with the attributes "north," "south," "east," "west," and "center."

A layout involves two files:

- An XML file with a .layout extension - The actual layout that is rendered on a page.
- An HTML file with a .html.txt extension - Used simply to simulate the layout in the WebLogic Workshop Portal Extensions Portal Designer and in the WebLogic Administration Portal.

There are also skeleton JSPs that are used to render each style of layout: gridlayout.jsp, flowlayout.jsp, and borderlayout.jsp. Since these skeleton files govern the behavior of each style, you do not have to modify the skeletons.

The following topics provide instructions on creating a layout, including specific instructions for creating each type of layout.

Creating a Layout

1. With your portal application open in WebLogic Workshop Platform Edition, duplicate an existing layout file in your Portal Web Project. For example, right-click <project>\framework\markup\layout\fourcolumn.layout and choose Duplicate.

   The new layout file appears with a number appended to the end of the name.

2. Rename the new layout file. Be sure to retain the .layout extension.
3. Duplicate an existing .html.txt file to use for the new layout. Rename it for easy association with the .layout file. Be sure to retain the .html.txt extension. Structure the HTML table in the .html.txt file so that it looks like what you expect the rendered layout to look like.
4. Inside the <netuix:markup> tag, insert opening and closing <netuix:gridLayout>, </netuix:flowLayout>, or </netuix:borderLayout> tags, depending on the type of layout you want to create. (Replace the existing opening and closing <netuix:*Layout> tag.)
5. Inside the opening <netuix:*Layout> tag, add (or modify) the following attributes:

title
: Provides the name for selecting the layout in a drop-down menu.

description
: Provides a description for the selected layout.

Grid Layout attributes	**columns** Determines the number of columns in the layout. The number of rows are determined automatically. Do not use the "rows" attribute if you use the "columns" attribute. **rows** Determines the number of rows in the layout. The number of columns is determined automatically. Do not use the "columns" attribute if you use the "rows" attribute.
Flow Layout attributes	**orientation** Enter "vertical" or "horizontal" to determine the direction in which the placeholders are positioned.
Border Layout attributes	**layoutStrategy** Enter "order" or "title". • If you enter "order," the placeholders are ordered according to the value you put in the <netuix:placeholder> tag (covered in the following steps). For example: <netuix:placeholder>North</netuix:placeholder> makes the placeholder the north placeholder. • If you enter "title," the placeholders are ordered according to the <netuix:placeholder> "title" attribute value. For example: <netuix:placeholder title="south" ...></netuix:placeholder> makes the placeholder the south placeholder.

htmlLayoutUri

Provides the path (relative to the project) to the .html.txt file you created.

For example, "/framework/markup/layout/yourNewLayout.html.txt"

iconUri

For compatibility domain administration only. When administering a portal running in a compatibility domain, this provides a path (relative to the project) to an icon that graphically represents the layout.

For example, "/framework/markup/layout/yourNewLayout.gif"

markupName

The markupName must be unique among the other layouts.

6. Inside the <netuix:*Layout> tag, add opening <netuix:placeholder> and closing </netuix:placeholder> tags for each placeholder you want in the layout.

   If you are creating a border layout, use no more than five placeholders.

7. In the opening <netuix:placeholder> tag of each placeholder, add the following attributes:

title

Enter a title for the placeholder. If you are using a **border layout** with the layoutStrategy attribute set to "title," enter "north," "south," "east," "west," or "center" for the title to determine which position of the placeholder in the border layout.

description

Enter a description for the placeholder.

flow

Optional. If you want to control the positioning of books and portlets in the placeholder, enter "true."

usingFlow

Optional. If you set the "flow" attribute to "true," enter "vertical" or "horizontal" for this attribute value. This value determines whether books and portlets are positioned on top of each other in the placeholder (vertical) or side by side (horizontal).

width

Optional. Set a width for the placeholder.

markupType

Required. Enter "Placeholder".

markupName

Required. Used as an ID for the placeholder. Each placeholder must have a unique markupName across all layouts.

8. If you are creating a border layout and the layoutStrategy attribute is set to "order," enter "North," "South," "East," "West," or "Center" as the content in each <netuix:placeholder> tag to determine each placeholder's position in the layout. For example, <netuix:placeholder>North</netuix:placeholder> makes a placeholder the north placeholder.

9. Save the layout file.

10. Modify the layout's *.html.txt file to create an HTML table that simulates the layout of the .layout file.

11. To use the layout, in the Portal Designer select a page in the Document Structure window. In the Property Editor window, select the new layout in the Layout Type field. (The server must be running for the new layout to appear in the Layout Type drop-down list.)

Navigation Menus

Navigation Menus provide a way to select different pages in a portal desktop. WebLogic Portal provides a default set of Navigation Menus:

Single Level Menu

Provides visible layering of book and page links. Any sub-books and pages appear in rows below the main book navigation.

Multi Level Menu

Provides a single row of tab-like links for the books and pages at the level you apply the Multi Level Menu. Any sub-books and pages appear in a drop-down list for selection. The Multi Level Menu implements JavaScript functionality contained in the skins.

If you want navigation menu behavior other than what is provided with the default menus, you can modify either of the default menus to suit your needs. The default navigation menus can be modified using either of the following procedures: Modifying the Navigation Menu File and Modifying The Skeleton JSP File.

Modifying the Navigation Menu File

To modify the navigation menu file, take the following steps:

1. Open either .menu file in <project>\framework\markup\menu\.
2. Inside the <netuix:markup> tag, modify the following attributes in the <netuix:singleLevelMenu ... /> or the <netuix:multiLevelMenu ... /> tag. Do not change the name of the tag.
    - title - Change the title of the menu. The title appears in drop-down lists. (Your development server must be running to see the title change in the Property Editor window.)
    - description - Change the description of the modified menu.
    - markupName - Change the markupName to better reflect the name of the menu.

Note: The align attribute is optional:

- align - Set to "left," "center," or "right." This is a rendering hint to align the menu. Neither default menu skeleton supports this functionality by default. You must modify the skeleton to support this.

Be sure to leave the schema references in the file header.

3. Save the file.

Modifying The Skeleton JSP File

To modify the skeleton JSP file, take the following steps:

1. Back up the original skeleton JSP in case you want to revert back to it later. The skeleton files, multilevelmenu.jsp and singlelevelmenu.jsp, are located in the following directory: <project>\framework\skeletons\default.

   These files are also located in other skeleton directories. You will replace those files with the modified file when you are finished.

2. Modify the skeleton JSP. Do not rename the file.

   The multilevelmenu.jsp skeleton file uses JavaScript functions contained in the menu.js, located in different skin directories under <project>\framework\skins.

3. After you have finished modifying the JSP, copy it to the other relevant skeleton directories, replacing the existing versions of that file.

To use the modified navigation menu, in the Portal Designer select a book in the Document Structure window. In the Property Editor window, select the new navigation menu in the Navigation field. (The server must be running for the new navigation menu to appear in the Navigation drop-down list.)

Shells

A shell represents the rendered area surrounding a portal desktop's main content area (books, pages, and portlets). Most importantly, a shell controls the content that appears in a desktop's header and footer regions.

You can configure a shell to use specific JSPs or HTML files to display content-especially personalized content-in a header or footer. For each set of different header/footer combinations, create a new shell.

Creating a Shell

To create a shell, take the following steps:

1. Make sure the server is running on your development machine. If it is not, open the portal application and choose Tools-->WebLogic Server-->Start WebLogic Server.
2. In WebLogic Workshop Platform Edition, open an existing shell (in the <project>\framework\markup\shell directory).
3. In the <netuix:shell> element, modify the title, description, and markupName attributes. The title attribute provides the name for selecting the shell in a drop-down menu; the markupName must be unique among the other shells.
4. Use the <netuix:header> or <netuix:footer> elements to point to the JSPs or HTML file you want to use for the header or footer using the following steps:
   a. Change the element from an empty element to one with opening and closing tags.

      For example:

      Change

      ```
      <netuix:header/>
      ```

      to

      ```
      <netuix:header>
      </netuix:header>
      ```

b. Add the <netuix:jspContent conentUri=" "> tag to the <netuix:header> or <netuix:footer> tag, using the contentUri attribute to point to the JSP or HTML file you want to appear in the header or footer. The path to this file is relative to the project.

For example, if you want your header to use the file <project>\my_jsps\campaign_header.jsp, set up your shell header as follows:

```
<netuix:header>

<nexuix:jspContent contentUri="/my_jsps/campaign_header.jsp"/>

</netuix:header>
```

Make sure the JSP or HTML file does not contain <html>, <head>, <title>, or <body> HTML tags, because the file will be inserted into a single HTML file that already has these tags. You can format the file simply with <div>, <table>, <p>, or any other nested HTML tags.

c. In the <netuix:jspContent> tag you can also point to an error JSP and a backing file to use for the header or footer JSP, using the following attributes:

– errorUri - Enter the path (relative to the project) to an error JSP to be used if there are problems with the contentUri JSP.

– backingFile - If you want to class for any preprocessing prior to the rendering of the header or footer JSP (for example, authentication), enter the fully qualified name of that class. That class should implement the interface com.bea.netuix.servlets.controls.content.backing.JspBacking or extend com.bea.netuix.servlets.controls.content.backing.AbstractJspBacking.

The following example shows a shell header that uses a JSP, an error JSP, and a backing file:

Listing 0-1 Shell Header

```
<netuix:header>

<nexuix:jspContent contentUri="/my_jsps/campaign_header.jsp"
errorUri="/my_jsps/my_error.jsp"

backingFile="custom.portal.backing.campaignBacking" />

</netuix:header>
```

d. Save the shell file.

e. To select the shell for a desktop, select Desktop in the Document Structure window and select the shell in the Property Editor Shell field. (The name of the new shell will not appear if the server is not running.)

Selecting a Shell for a desktop in the Portal Designer simply gives the portal a default Shell setting. Portal administrators can use the WebLogic Administration Portal to change the Shell used for a desktop.

Configuration and Administration

Configuring WebLogic Portal 8.1

This section contains configuration suggestions for development as well as production, and includes information on the following topics:

- Split Configuration
- Deploying an EAR Versus an Exploded Application
- How to Use Active Directory
- Single Signon Between Two Portal Web Applications

Split Configuration

Note on Split Configuration - WebLogic Portal does not support a split configuration, where servlet and EJB containers run on separate server instances.

Deploying an EAR Versus an Exploded Application

Once the development of a Portal Web application has been completed, what remains is to deploy it to the production environment. A number of choices need to be made at this point. For example, the application can be deployed as an EAR if nothing inside it needs to be changed. For details on the advantage to this approach, along with instructions, consult the "Packaging Enterprise Applications" section of the WebLogic Platform documentation. Applications can also be deployed without being archived into an EAR file. This enables some kinds of files within the

application to be changed without having to un-deploy and re-deploy the entire application. For instance, in an application that uses e-mail campaigns, certain JSPs may need to be altered during production.

How to Use Active Directory

If you have used a third party single signon solution and they want to integrate to WebLogic Portal 8.1, the single signon solution is to authenticate users in Active directory but store all policy and group information in the RDBMS.

If the single signon solution is Netegrity Siteminder, for instance, Siteminder offers both the Web Server Agent (authentication by Siteminder, Authorization by WebLogic Portal) and App Server Agent integrations (both authentication and authorization by Siteminder). Currently, only the Web Server Agent integration works with WebLogic Portal. Siteminder will provide the Application Server Agent integration with WLP 8.1.

Single Signon Between Two Portal Web Applications

To implement single signon within two portal Web applications, set the cookie to be the same in the WebLogic Server Console. That is, set the cookie name to be the same, with different cookie paths, one for each application.

Performance

Performance Tuning

This section includes information on caching as well as portlet threading. The following topics are covered in this section:

- Managing Caches
- How to Use Caching Tables
- Notes on Threads

Managing Caches

Table 0-1 lists caches that might be used by your portal application. Use the list to assist in your tuning. Keep in mind the memory that is available to your system. When modifying the maximum cache sizes also monitor the system memory to determine the effects.

Statically defined caches are established in the **application-config.xml** file for the application. These caches have dynamically, not statically, configurable attributes, which can be configured in the Weblogic 8.1 Administration Portal by navigating to: **Applications --> [Portal App Name] --> Service Configuration --> Caches -> service configuration**

> Some caches listed below may not show up in the Weblogic 8.1 Administration Portal. To configure such caches, you can create an entry in the Weblogic 8.1 Administration Portal and redeploy the application. These caches will then be configurable from the Weblogic 8.1 Administration Portal.

How to Use Caching Tables

Table 0-1 and Table 0-2 list caches used to tune performance in WebLogic Portal 8.1 domain, and Portal Compatibility Domain, respectively. Find a cache you need to adjust, then use the WebLogic Administration Portal to adjust the cache settings.

Table 0-1 Configurable Caches in WebLogic Portal 8.1

Cache	actionNameCache
Use	Used to store the action classes for campaigns
Key	The class name (java.lang.String)
Value	The actual Class for the action (java.lang.Class)
Notes	Not configured in application-config.xml but rather in cache-actionNameCache.properties. Default values should be okay.
Cache	adServiceCache
Use	Used by the AdHelper to increase the speed of ad queries.
Key	Used by the AdHelper to increase the speed of ad queries.
Value	The ad query (java.lang.String)
Notes	Stores a Content[] of ads keyed off an ad query. If the ads returned from a particular query do not change, consider increasing the TTL. Consider basing the maximum size on the total number of ad queries.

Cache	CategoryCache
Use	Stores the root com.beasys.commerce.ebusiness.catalog.Category, the total number of categories in the product catalog (java.lang.Integer) and the CategoryInfo for each category.
Key	The key for the root Category is a static final String variable in the CategoryManagerImpl class. The key for the total number of categories is also a static final String variable in the CategoryManagerImpl class. The key for a given CategoryInfo object is a com.beasys.commerce.ebusiness.catalog.CategoryKey.
Value	The value for the root Category is com.beasys.commerce.ebusiness.catalog.Category. The value for the total number of categories is a java.lang.Integer. The value for the category info objects is a com.beasys.commerce.ebusiness.catalog.service.category.CategoryInfo.
Notes	CategoryManagerImpl gets the cache name from the ejb-jar.xml in commerce.jar The root Category and the total number of categories occupy two slots in the cache and the remaining slots are occupied by the CategoryInfo objects, so consider the total number of categories in the product catalog plus 2 when setting the maximum cache size. Consider how often these categories will change when setting TTL.
Cache	configurableEntityMethodCache
Use	Stores method information for setter methods for explicit properties
Key	Based on the class name and the method name (java.lang.String).
Value	A com.bea.p13n.property.internal.CacheMethod object.
Notes	When setting the maximum cache size, consider the number of explicit properties in your application. Consider how often explicit properties will change in your system when setting TTL.

Cache	discountCache
Use	Stores com.bea.commerce.ebusiness.discount.mgmt.QualificationDiscountDef objects. These discounts are applicable to particular customers or customer segments.
Key	A com.bea.commerce.ebusiness.discount.mgmt.QualificationDiscountId. Essentially this is a wrapper around a java.lang.Integer that represents the id of the discount.
Value	A com.bea.commerce.ebusiness.discount.mgmt.QualificationDiscountDef
Notes	Set the maximum cache size based on the number of possible discounts. Consider how often discounts change when setting the TTL.

Cache	documentContentCache
Use	Used by the DocumentManager to cache the actual bytes of content
Key	Based on id, start, and length of content (an inner class called BytesKey).
Value	byte[]
Notes	This cache is only used with the WebLogic Portal 7.0 SP2 content repository. By default, the maximum number of bytes that can be stored as a cache entry is 32K. This is adjustable and is set via the DocumentManagerMBean. Applications with a large amount of content could benefit with a larger max cache size. Keep memory consumption in mind here. Applications with substantial static content might benefit from increasing the TTL.

Cache	entityIdCache
Use	Caches the id for an entity (user or group id, ENTITY.ENTITY_ID)
Key	Object key is a com.bea.p13n.property.PropertyLocator. PropertyLocator is based on a user or group name (ENTITY.ENTITY_NAME) and entity type (ENTITY.ENTITY_TYPE).
Value	The entity id (java.lang.Long).
Notes	As in the entityPropertyCache uses the ENTITY table as a guide for the maximum size. The object being stored is a Long, which is fairly small. Therefore, it might be possible to set this cache's maximum size to the number of entries in the ENTITY table. Consider how often the ENTITY table might change when setting the TTL.

Cache	entityPropertyCache
Use	Caches property values for users and groups
Key	Object key is a com.bea.p13n.property.PropertyLocator. PropertyLocator is based on the user or group name (ENTITY.ENTITY_NAME), entity type (ENTITY.ENTITY_TYPE, user or group) and property set type (PROPERTY_KEY.PROPERTY_SET_TYPE, usually USER).
Value	A com.bea.p13n.property.EntityPropertyCache object. This object contains a Map that stores property values keyed off the property set name and property name.
Notes	The larger you can afford to make this cache, the better. Use the ENTITY table as a guide for maximum size. The number of entries in this table should be the maximum number of cache entries that would ever be created. In most cases, there will be more entries here than you would want for a maximum cache size. So consider the average number of users you expect to be using your application at the same time. Consider a TTL based on how often new properties will be added to the property sets. If they are not being modified often, then a higher TTL might be appropriate.

Cache	globalDiscountCache
Use	Stores a java.util.Set of QualificationDiscountDef objects. This is the set of global discounts that is applicable to all users.
Key	The globalDiscountSet name (java.lang.String)
Value	The java.util.Set of global discounts
Notes	Default maximum size of 10 should be fine. Frequency of changes to the global discounts should determine TTL.
Cache	ldapPropertyCache
Use	Caches property values for users and groups when using LDAP for storing properties.
Key	Object key is a com.bea.p13n.property.PropertyLocator. PropertyLocator is based on the user or group name, entity type and property set type.
Value	A com.bea.p13n.property.EntityPropertyCache object. This object contains a Map that stores property values. The Map is keyed off the property set name and property name.
Notes	Serves the same purpose as the entityPropertyCache. See Notes on entityPropertyCache.
Cache	ProductItemCache
Use	Stores the total number of product items in the catalog as well as the product items
Key	The key for the total number of product items is a static final String variable in ProductItemManagerImpl. The key for the product items is a com.beasys.commerce.ebusiness.catalog.ProductItemKey.
Value	The value for the total number of product items is a java.lang.Integer. The value for the product item is a com.beasys.commerce.ebusiness.catalog.ProductItem.
Notes	ProductItemManagerImpl gets the cache name from the ejb-jar.xml in commerce.jar. Consider the total number of product items when setting the maximum cache size. Consider how often these product items will change when setting the TTL.

Cache	profileTypeCache
Use	Stores the profile type for a user. This profile type is used to lookup the correct user profile manager when fetching the user's Profile.
Key	A username (java.lang.String)
Value	The profile type (java.lang.String)
Notes	Consider the number of users you expect to be logged into your system at a given time when setting the maximum cache size. Unless you plan on changing a user's profile manager during runtime, a high TTL or one that never expires might be appropriate
Cache	propertyKeyIdCache
Use	Caches the unique id associated with a property set type, property set and property name combination (primary key in the PROPERTY_KEY database table).
Key	Based on a property set type, property set, and property name combination (inner class called PropertyKeyLocator).
Value	The id (java.lang.Long)
Notes	Maximum size should be set with an eye towards the maximum number of properties in the application (use the PROPERTY_KEY table as an indicator). Consider a TTL based on how often these unique id combinations are likely to change.
Cache	portalControlTreeCache
Use	Used to store portal control trees
Key	The combination of webapp, portal, desktop, locale and optional user name.
Value	A portal control tree.
Notes	Configured in application-config.xml Default TTL value should be okay, Max Entries could be set to a number based on number of users and available memory. If there are any changes to portal this cache will be flushed.

Cache	portletControlTreeCache
Use	Used to store portlet control trees.
Key	The combination portletInstanceId and locale.
Value	A portlet control tree.
Notes	Configured in application-config.xml Default TTL value should be okay, Max Entries could be set to a number based on number of floatable portlet instances in a portal (Including user customized portlets)and number of supported locales.
Cache	portalContentUriCache
Use	Used to store portal content URI for portal path.
Key	The combination portal path and webAppName.
Value	A portal content URI.
Notes	Configured in application-config.xml Default TTL value should be okay, Max Entries could be set to a number based on number of portals that have associated content URIs.
Cache	portalLocalizationResourceCache
Use	Used to store localization resources.
Key	The localizationIntersection.
Value	A LocalizationResource.
Notes	Configured in application-config.xml Default TTL and Max Entries values could be set to a value based on total number of localization resources in the system which is a combination of non-customized and customized localization resources and amount of available memory.

Cache	portalLocalizationlocaleCache
Use	Used to store collection of LocalizationLocales.
Key	The key is private static final string called "portalLocalizationLocaleCachekey"
Value	A set of LocalizationLocales.
Notes	Configured in application-config.xml Default TTL value should be okay, Max Entries could be set to a number based on number of rows in L10N_LOCALE table i.e. number of supported locales.
Cache	portalMarkupdefinitionCache
Use	Used to store MarkupDefinitions.
Key	The MarkDefinitionId.
Value	A MarkupDefinition.
Notes	Configured in application-config.xml Default TTL value should be okay, Max Entries could be set to a number based on total number of rows in PF_MARKUP_DEFINITION table.
Cache	portletPreferencesCache
Use	Used to store portlet preferences.
Key	An instance of PortletPreferenceId.
Value	A map of preferences.
Notes	Configured in application-config.xml Default TTL and Max Entries values could be set to a value depending on amount of available memory and total number of preferences (at application level).

Cache	nodeCache.<Repository Name>
Use	Caches Nodes for a Repository based on ID.
Key	The Node ID as a String.
Value	The actual Node instance (com.bea.content.Node).
Notes	This cache can be configured using the WebLogic Administration Portal, or programmatically through the P13N CacheManager. Each Repository has its own separate Node cache. This cache is used by NodeOps when retrieving and updating Nodes. Please use in conjunction with the Repository vendor's cache architecture. Default values: • Maximum Entries: 50 • Time To Live(seconds): 6000 • Is Enabled: True
Cache	binaryCache.<Repository Name>
Use	Caches Binary properties for a Node.
Key	The Node ID as a String + the unique Property ID as a String.
Value	The byte array associated with the Binary property.
Notes	This cache can be configured using the Portal Administration Tool or programmatically through the P13N CacheManager. Each Repository has its own separate Binary cache. Keep memory consumption in mind when setting the Maximum Entries and Cache Size values as they may affect your server's performance. Please use in conjunction with the Repository vendor's cache architecture. Default values: • Maximum Entries: 10 • Time To Live(seconds): 6000 • Cache Size/Item(bytes): 1024 • Is Enabled: True

Cache	nodePathCache.<Repository Name>
Use	Caches Nodes for a Repository based on path.
Key	The Node path as a String.
Value	The actual Node instance (com.bea.content.Node).
Notes	This cache can be configured using the Portal Administration Tool or programmatically through the P13N CacheManager. Each Repository has its own separate Node Path cache. This cache is used by NodeOps when retrieving and updating Nodes based on path. Please use in conjunction with the Repository vendor's cache architecture. • Default values- • Maximum Entries: 50 • Time To Live(seconds): 6000 • Is Enabled: True

Cache	searchCache
Use	Caches the results of content searches for the Virtual Content Repository.
Key	The Search object containing the parameters for a query (com.bea.content.expression.Search).
Value	An ID array reflecting the Nodes that satisfy the query.
Notes	This cache can be configured using the Portal Administration Tool under Service Administration or programmatically through the P13N CacheManager. There is only one search cache for all Repositories. This cache is used by SearchOps when handling Search queries and results. Flush, enable, or disable this cache based on your needs for real-time data and other system requirements. Default values: • Maximum Entries: 20 • Time To Live(seconds): 6000 • Is Enabled: True

Table 0-2 Portal Compatibility Domain Caches

Cache	actionNameCache
Use	Used to store the action classes for campaigns
Key	The class name (java.lang.String)
Value	The actual Class for the action (java.lang.Class)
Notes	Not configured in application-config.xml but rather in cache-actionNameCache.properties. Default values should be okay.
Cache	https_cache
Use	This cache is implimented as part of WebFlow, is not used in WebLogic Portal 8.1 except in support of applications developed in WebLogic Portal 7.0 SP2 re-hosted to run in a Portal Compatibility Domain.
Key	java.lang.String composed of the application name, namespace name, origin name and event name
Value	Java.lang.Boolean indicating if HTTPS is required.
Notes	Caches a Boolean indicating if https should be used for a particular webflow request. A value for each application, namespace, origin and event combination may be cached. This check occurs only if SSL is enabled in config.xml and the HTTPSIND_DEFAULT_VALUE in web.xml is set to CALCULATE, or the “httpsInd” attribute in a calculate weblflow URL tag is set to “CALCULATE.” Because a Boolean is a small and the HTTPS calculation can be expensive, it is important to set this cache’s size large enough to avoid excess calculation. Provided that you are not modifying the context param HTTPS_URL_PATTERNS in web.xml and redeploying or you are not modifying the webflows in a running server and re-synching then the TTL for this cache could be set to never expire. Only available in Portal Compatibility Domain.

Cache	interface-cache
Use	Used by the PipelineExecutorImpl to improve performance when executing a pipeline.
Key	The pipeline component class name (java.lang.String)
Value	An instance of the pipeline component
Notes	Consider the total number of pipeline components in your application when setting the maximum cache size. If you do not plan on modifying pipeline components to a running server consider setting a higher TTL or one that never expires. Only available in Portal Compatibility Domain.

Cache	MainPersistenceManager.GroupPortalP13nCache
Use	Stores a java.util.List of group PortalPersonalizations for each portal
Key	com.bea.portal.model.PortalIdentifier
Value	java.util.List with each list entry containing personalizations for a particular group portal.
Notes	Consider the number of portals when setting the maximum cache size. Generally, this can be set to a low number(e.g. If there is only one portal, this size can be set to one). Only available in Portal Compatibility Domain.

Cache	MainPersistenceManager.PortalP13nCache
Use	Stores the PortalPersonalization objects for groups and users
Key	com.bea.portal.model.PortalPersonalizationIdentifier
Value	com.bea.portal.model.PortalPersonalization
Notes	Consider average number of concurrent users as well as the total number of groups when setting the maximum cache size. Only available in Portal Compatibility Domain.

Notes on Threads

This section discusses portlet threading issues such as what happens when a portlet is spawned in its own thread and then the calling system fails during execution.

About the 8.1 Forkable Portlet Feature

If the system calling a threaded portlet fails:

Although all of the forkable portlets for a page are processed in parallel, it is in the end still a blocking operation. If one of the portlets gets hung up in its processing, it will block the entire page render. Forked threads do not have a timeout setting, meaning that the effect of a failed thread would depend on what the portlet is doing. If it fails gracefully (and quickly), there shouldn't be a problem.

If the user leaves the site, or moves to another page:

In this case, processing will not stop by itself. The portal framework will continue to build that response, even if the client has issued another request.