BEA WebLogic Workshop™ Help

Version 8.1 SP4
December 2004

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 Enterprise Security, 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.

Table of Contents

Developing Portal User Interfaces....1

The Portal User Interface Framework....2

How Look & Feel Determines Rendering....5

How the Shell Determines Header and Footer Content....17

How Portal Components Are Rendered....26

Portal User Interface Framework Reference....40

Designing a Portal User Interface....45

Building User Interfaces to Address Accessibility Guidelines....46

Creating a Portal Application and Portal Web Project....50

Creating a Portal File....53

Creating Look & Feels....55

Look & Feel Architecture....56

Creating Look & Feel Files....60

Creating Skins and Skin Themes....63

Creating Skeletons and Skeleton Themes....67

Creating Layouts....72

Modifying Navigation Menus....76

Creating Shells....78

Left Navigation Shell....80

Building Portlets....83

Enabling Desktop Selection....84

Adding Visitor Tools to Portals....86

Properties for Portal Components....89

Table of Contents

Properties for All Portal Components **90**
..... **92**

Desktop Properties **93**
..... **94**

Header and Footer Properties **95**
..... **96**

Book and Page Properties **97**
..... **101**

Placeholder Properties **102**
..... **103**

Portlet Properties **104**

Developing Portal User Interfaces

WebLogic Portal provides a flexible, highly extensible framework for building portal user interfaces. This section guides you through design considerations and steps necessary to create portal user interfaces.

The Portal User Interface Framework

Describes the portal user interface framework and illustrates how portal components developed in the Portal Designer are converted to rendered HTML.

Designing a Portal User Interface

Discusses design considerations for putting a traditional Web application into a portal interface. This section also contains guidelines for building user interfaces that address accessibility guidelines.

Creating a Portal Application and Portal Web Project

Shows you how to lay the foundation of portal development by creating a portal application and portal Web project or installing Portal in existing applications and projects.

Creating a Portal File

Shows you how to create a portal file so you can begin creating portal desktops in WebLogic Workshop and the WebLogic Administration Portal.

Creating Look & Feels

Provides instructions on creating the resources that determine a portal desktop's look & feel.

Building Portlets

Provides instructions on creating and customizing portlets, adding portlets from Sample Portal to your portal application, and establishing inter−portlet communication.

Enabling Desktop Selection

Shows you how to let users access any of the portal desktops to which they are entitled.

Adding Visitor Tools to Portals

Shows you how to let users customize their portal desktops.

Related Topics

Portal Key Concepts and Architecture

The Portal User Interface Framework

This section details how the portal framework turns a portal you develop in WebLogic Workshop into the portal desktop visitors see in a browser (see the figures at the bottom of this topic). The goal of describing the portal framework is to help you develop and troubleshoot your portals. These topics should enable you to look at a rendered portal in a browser and understand which pieces of the underlying framework you need to modify to get the results you want.

Topics Included in This Section

The following topics describe key portal framework components and walk you through the portal rendering process:

How Look & Feel Determines Rendering

Describes how look & feel determines how a portal desktop is rendered and what it looks like.

How the Shell Determines Header and Footer Content

Describes how shells determine the content of a desktop header and footer.

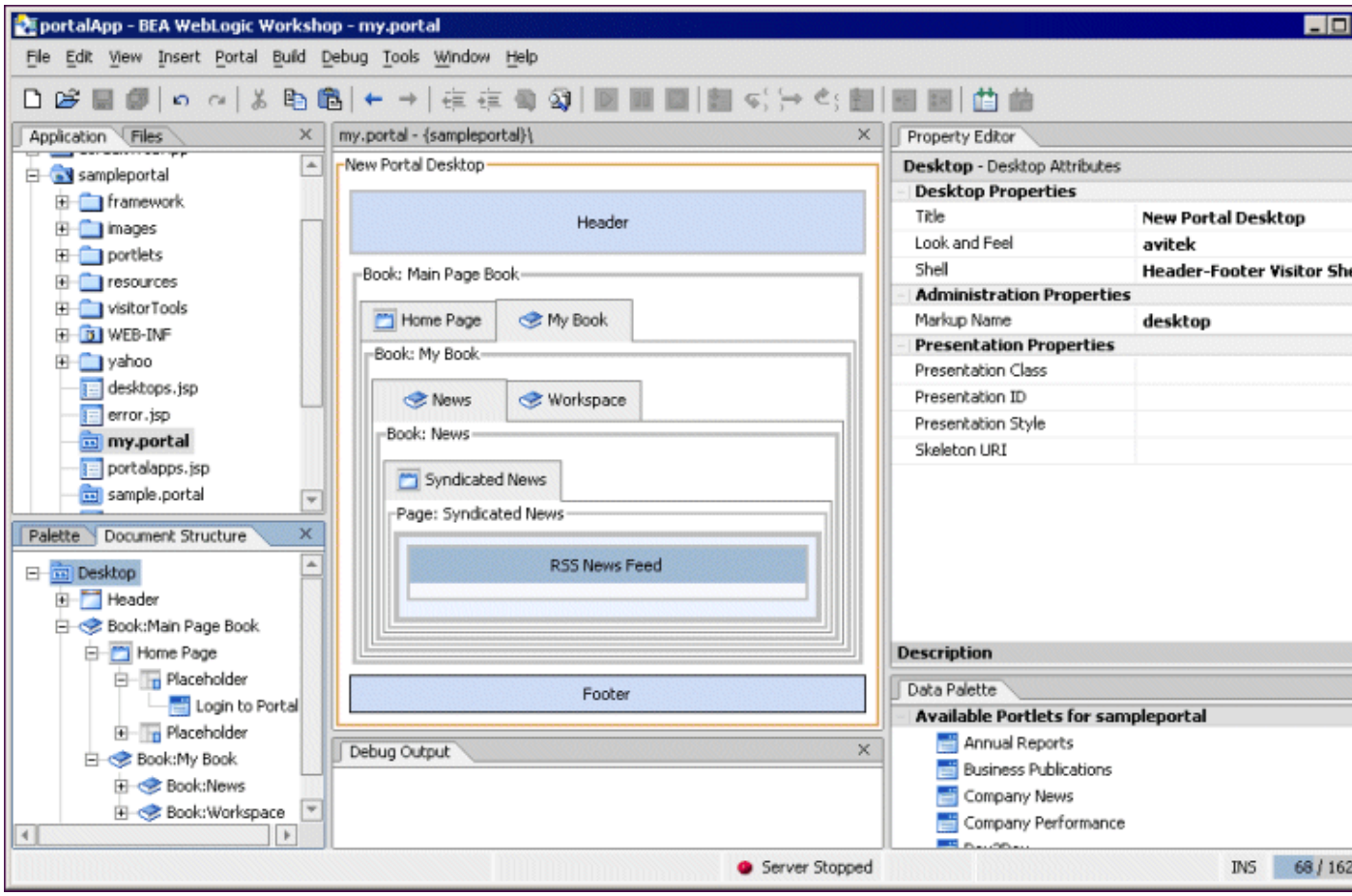
How Portal Components Are Rendered

Illustrates the rendering process, showing how a portal component is converted to HTML.

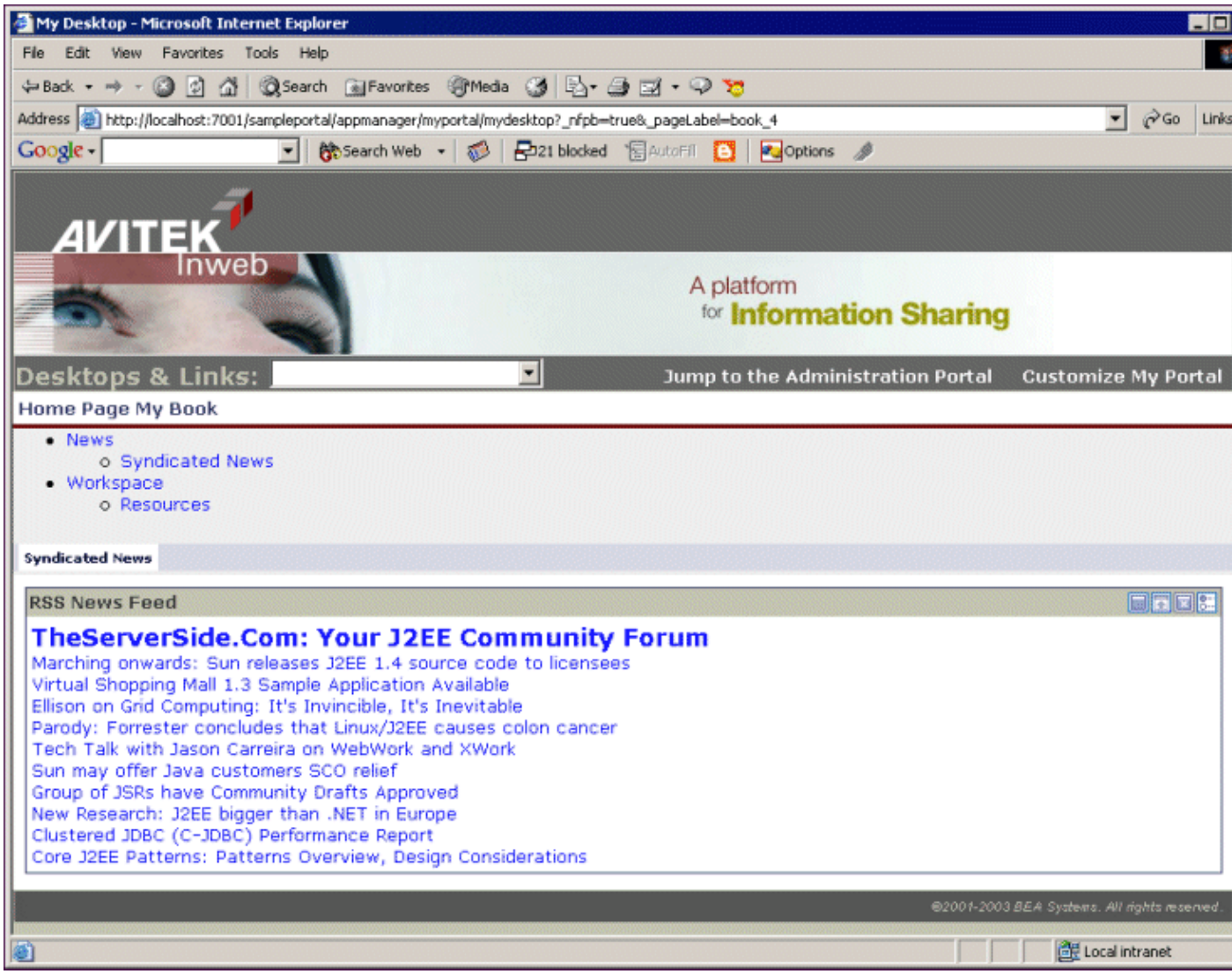
Portal User Interface Framework Reference

Shows some of the main cascading style sheet classes and the skeleton JSPs that affect portal rendering.

The topics in this section show you the underlying processing that converts an XML−based .portal file created in WebLogic Workshop (below)...



...to fully rendered HTML (below).


My Desktop - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:7001/sampleportal/appmanager/myportal/mydesktop?_nfpb=true&_pageLabel=book_4
Go
Google Search Web 21 blocked AutoFill Options
AVITEK
Inweb
A platform
for Information Sharing
Desktops & Links:
Jump to the Administration Portal
Customize My Portal
Home Page My Book
News
Syndicated News
Workspace
Resources
Syndicated News
RSS News Feed
TheServerSide.Com: Your J2EE Community Forum
Marching onwards: Sun releases J2EE 1.4 source code to licensees
Virtual Shopping Mall 1.3 Sample Application Available
Ellison on Grid Computing: It's Invincible, It's Inevitable
Parody: Forrester concludes that Linux/J2EE causes colon cancer
Tech Talk with Jason Carreira on WebWork and XWork
Sun may offer Java customers SCO relief
Group of JSRs have Community Drafts Approved
New Research: J2EE bigger than .NET in Europe
Clustered JDBC (C-JDBC) Performance Report
Core J2EE Patterns: Patterns Overview, Design Considerations
©2001-2003 BEA Systems. All rights reserved.
Local intranet

How Look & Feel Determines Rendering

When you build a portal in WebLogic Workshop, the look & feels you use are the key to how your portal is rendered and what it looks like when it is rendered. This topic shows you how the different pieces of the look & feel framework are combined and configured to provide what the portal framework needs to render the look & feel in HTML.

This topic contains the following sections:

Overview

The Look & Feel File

The Portal File

Location of the Look & Feel Resources

The skin.properties File

Look & Feel Overrides

Summary

Overview

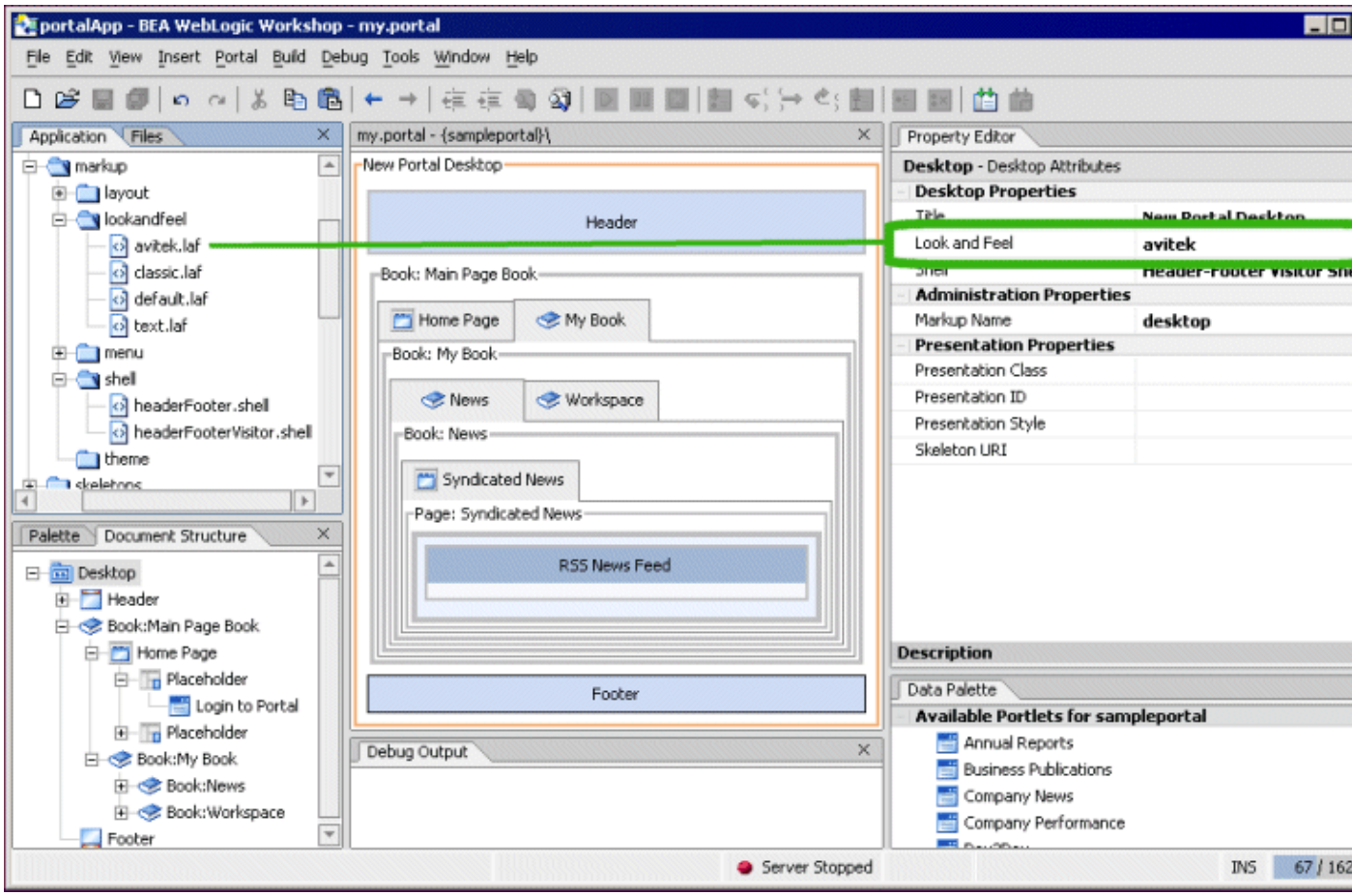
The look & feel determines the following:

- ***Skin*** − A skin is a group of Cascading Style Sheet (CSS) files, framework images (mainly for portlet titlebar icons), and JavaScript functionality that is used in the portal desktop when it is rendered in HTML. A portal Web project can have multiple skins. When you select a look & feel for a desktop, a specific skin is used. Following are example skin elements:

Image	***CSS Style***	***JavaScript Function***
	`.bea-portal-button-float { } .bea-portal-button-float img { vertical-align: top; margin: 1px; border-style: solid; border-width: 1px; border-color: #666699; }`	`function initPortletFloatButtons() { var links = document.getElementsByTagName("a"); for (var i = 0; i < links.length; i++) { if (links[i].className && links[i].classNa { initPortletFloatButton(links[i]); } } }`

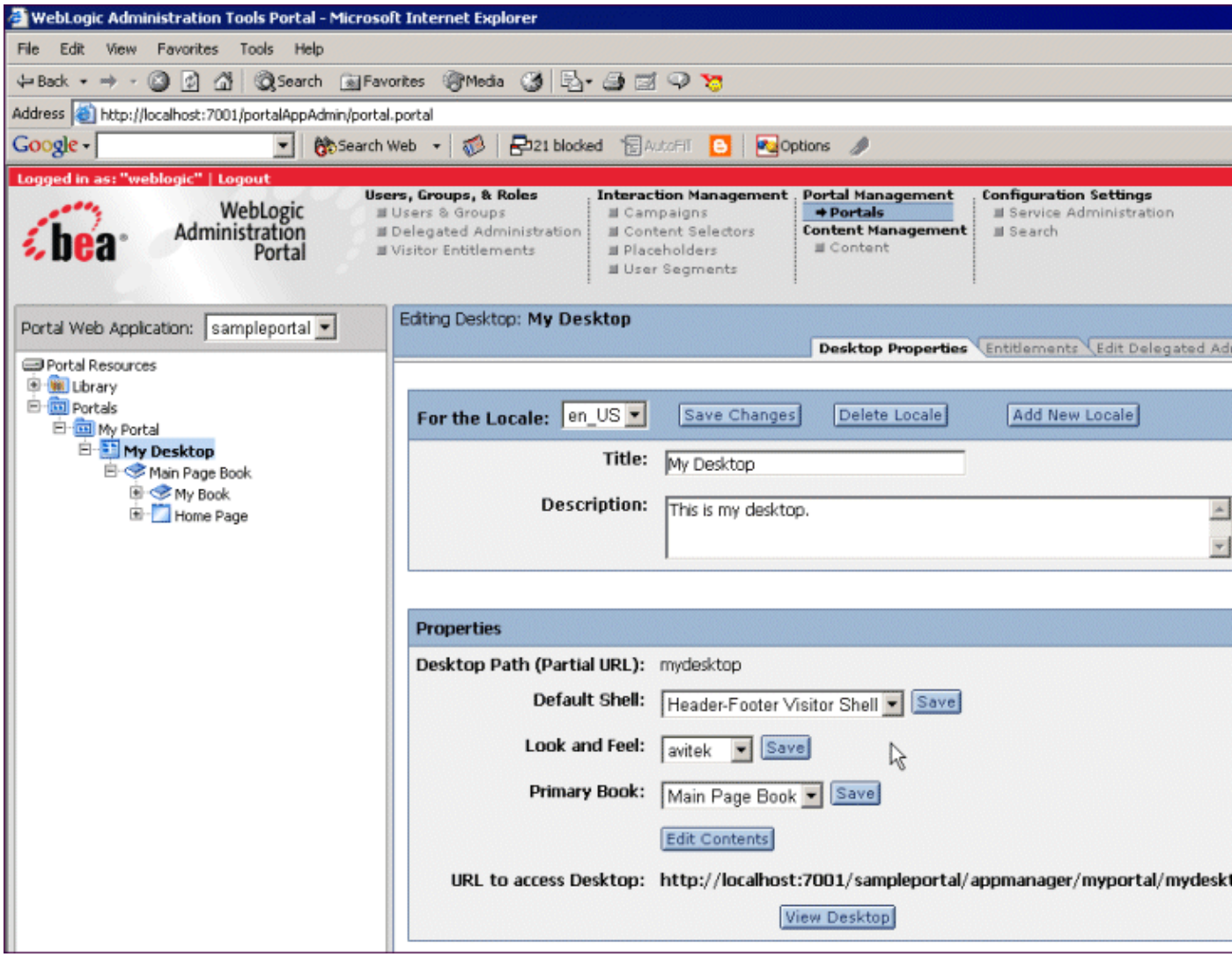
- ***Skeleton JSPs*** − A skeleton is a group of JSPs that are used to render each component of the portal desktop as HTML, from the desktop to books and pages to portlet titlebars. The skeleton provides the physical boundaries of the portal components and provides references to the images, CSS classes, and JavaScript functions from the skin needed to render the portal. A portal Web project can have multiple skeletons. When you select a look & feel for a desktop, a specific skin and skeleton is used.

A look & feel is represented by an XML file (with .laf extension), as shown in the following figure.

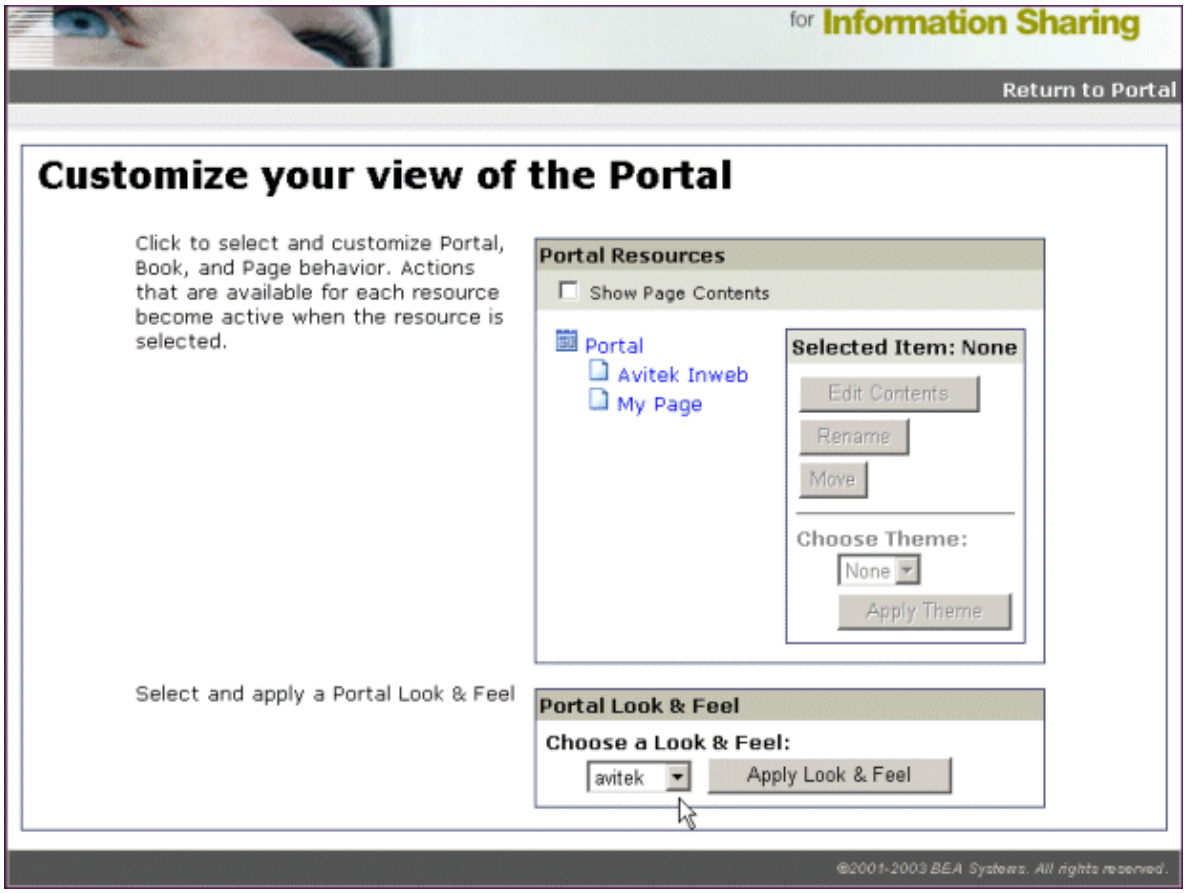


Developers building portals with WebLogic Workshop are not the only users who can select the look & feel used by a portal desktop. While developers create look & feels and select the default look & feel used by a portal, portal administrators and visitors may ultimately determine the desktop look & feel. The following figures show how portal administrators and users can change the look & feel used by the desktop.

After a portal administrator creates a desktop in the WebLogic Administration Portal, the administrator can change the desktop look & feel on the Dekstop Properties page.



If visitor tools are enabled for the desktop, visitors can click the "Customize My Portal" link and change the desktop look and feel.



The following section shows the look & feel XML file and describes how it is used as the basis of portal desktop rendering.

The Look & Feel File

Look & feel files point to the specific skin and skeleton to be used for the overall desktop look & feel.

Look & feel files are stored in the following location: <portal_Web_project>/framework/markup/lookandfeel/. Following is the avitek.laf provided by BEA. The key attributes are highlighted.

```
<?xml version="1.0" encoding="UTF-8"?>
<netuix:markupDefinition xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0 markup-netuix-1
    <netuix:locale language="en"/>
    <netuix:markup>
        <netuix:lookAndFeel
            definitionLabel="avitek" title="avitek"
            description="The avitek look and feel"
            skin="avitek" skinPath="/framework/skins/"
            skeleton="default" skeletonPath="/framework/skeletons/"
            markupType="LookAndFeel" markupName="avitek"/>
    </netuix:markup>
</netuix:markupDefinition>
```

The following table describes the look & feel attributes.

Attribute	*Description*
definitionLabel	Requried. The unique label used to identify the look & feel for setting entitlements. Each look & feel in the portal Web project must have a unique definitionLabel. For best practices, use the same name as the markupName.
title	Required. The string used to display the name in the Look & Feel drop−down fields in WebLogic Workshop, the WebLogic Administration Portal, and on the visitor tools page.
description	Optional. Description of the look & feel. The description is used in the WebLogic Administration Portal; when you select a look & feel in the portal Library, the description appears on the Look & Feel Properties page.
skin	Optional. The name of the directory containing the skin you want to use. If you do not set this attribute, the /framework/skins/default skin is used.
skinPath	Optional. The path, relative to the portal Web project, to the parent directory of the skin directory. • If you do not set this attribute, the /framework/skins/<skin_attribute_name> skin is used. • If no skin attribute is set, the /framework/skins/default skin is used.
skeleton	Optional. The name of the directory containing the skeleton JSPs you want to use. • If you do not set this attribute, the framework uses the default.skeleton.id path in the skin.properties file of the skin used. • If you do not set this attribute and no default.skeleton.id path is set in skin.properties, the /framework/skeletons/default skeleton is used.
skeletonPath	Optional. The path, relative to the portal Web project, to the parent directory of the skeleton directory. • If you do not set this attribute, the framework uses the default.skeleton.path in the skin.properties file of the skin is used. • If you do not set this attribute and no default.skeleton.path is set in skin.properties, the /framework/skeletons/<skeleton_attribute_name> skeleton is used. • If you do not set this attribute and no skeleton attribute is set, the /framework/skeletons/<default.skeleton.id> skeleton is used. • If you do not set this attribute and no skeleton attribute is set, and skin.properties contains no default.skeleton.id, the /framework/skeletons/default skeleton is used.

markupType	Required. The name of the type of component. Must always be "LookAndFeel".
markupName	Required. The name for the look & feel. Each look & feel in the portal Web project must have a unique markupName. For best practices, use the same name as the definitionLabel.

When you select a Look and Feel in the WebLogic Workshop Property Editor for a selected desktop, the look & feel XML is automatically added to the underlying XML in the .portal file, as shown in the following section.

The Portal File

Following is an example portal file, created with the Portal Designer, showing the inserted look & feel XML from the .laf file. The look & feel XML was inserted when the Look and Feel property was set for the selected desktop in the Property Editor (see the first figure in this topic). When the .laf file is inserted into the .portal file, its job is finished in the rendering process and the .portal file is used to set look & feel.

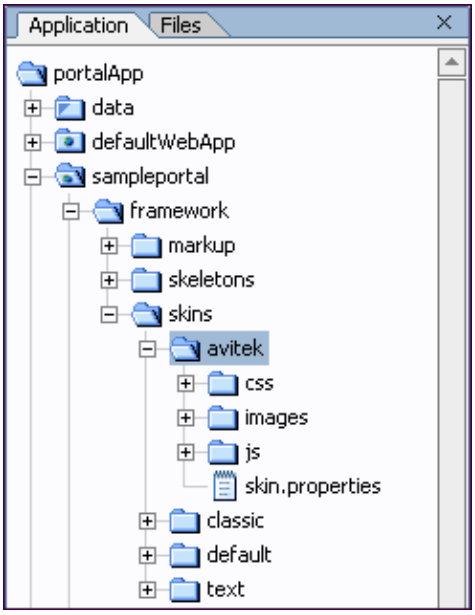
```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root xmlns:html="http://www.w3.org/1999/xhtml-netuix-modified/1.0.0"
    xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0"
    xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0 portal-suppo
    <portal:directive.page contentType="text/html;charset=UTF-8"/>
    <netuix:desktop definitionLabel="defaultDesktopLabel" markupName="desktop" markupType="Desk
        <netuix:lookAndFeel definitionLabel="avitek" description="The avitek look and feel"
            markupName="avitek" markupType="LookAndFeel" skeleton="default"
            skeletonPath="/framework/skeletons/" skin="avitek" skinPath="/framework/skins/" tit
        <netuix:shell description="A header with a link and footer is included in this shell."
            markupName="headerFooterVisitor" markupType="Shell" title="Header-Footer Visitor Sh
            <netuix:head/>
            <netuix:body>
                <netuix:header>
                    <netuix:jspContent contentUri="/portlets/header/header.jsp"/>
                </netuix:header>

                    [XML for books, pages, and portlets...]

                <netuix:footer>
                    <netuix:jspContent contentUri="/portlets/footer/footer.jsp"/>
                </netuix:footer>
```

The portal file is a template with which multiple desktops can be created in the WebLogic Administration Portal. When used as a template, the portal file determines the default look & feel of any desktop created from it.

Location of the Look & Feel Resources

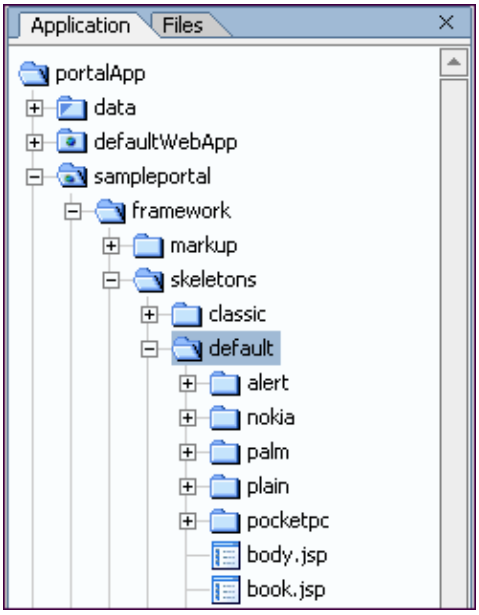
The look & feel attributes in the portal file tell the portal which skin and skeleton to use to render the portal in HTML. The portal in the previous example will use the following skin and skeleton resources:

Skin ***Skeleton***



The /css, /images, and /js directories contain the CSS files, framework images (mainly portlet titlebar icons), and JavaScript files that will be used in the skin. The skin.properties file (discussed in the next section) contains references to these resources, and at rendering time those resource references are inserted into the HTML <head> region. You can name your skin resource directories anything you like as long as you reference them correctly in skin.properties (or skin_custom.properties).

Skins can also contain subdirectories for sub−skins, or themes (discussed in Look & Feel Overrides).



The skeleton is made up of JSPs that map to and convert each portal component to HTML. The XML elements for the portal components in the .portal file determine the order in which the skeleton JSPs are called and rendered as HTML.

This figure shows a clipped view of the skeleton contents. The subdirectories shown are skeleton themes and skeletons used for mobile devices. The JSPs in the root /default directory make up the "default" skeleton.

Themes are discussed in Look & Feel Overrides.

Note About Portlet Titlebar Icons

The icons used in portlet titlebars are stored in the skin's /images directory. The portal framework reads the portal Web project's WEB−INF/netuix−config.xml file to determine which of these graphics to use for the portlet's different states and modes (minimize, maximize, help, edit, and so on).

The skin.properties File

Each skin has a skin.properties file, which is used by the portal framwork to populate the <head> section of the rendered HTML, among other things. Included in skin.properties are references to the images directory, the CSS files containing the styles to be used in the HTML, and the JavaScript files containing the functions that will be used in the HTML.

Note: You can also create a file called skin_custom.properties in the same directory as skin.properties. Any entries you include in skin_custom.properties are also added to the HTML <head> region. This feature lets you customize the properties without having them overwritten by product updates.

The following will not be rendered:

- Style Sheet styles that do not exist in one of the .css files listed in the <head>.
- JavaScript functions that do not exist in one of the .js files listed in the <head>.

That is why it is important to add references to all skin resources in skin.properties or skin_custom.properties.

The skin.properties or skin_custom.properties files can also contain skeleton path information that is used if skeleton attributes are omitted from the look & feel (.laf) file, as described in The Look & Feel File.

The following table shows an example of how entries in skin.properties for the active skin are converted to HTML <head> entries. Different skins may have different entries.

skin.properties Entries	***Rendered HTML <head> Entries***
images.path: images	content="/framework/skins/avitek/images"/>
link.body.href: css/body.css link.body.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/body.css" rel="stylesheet"/>
link.book.href: css/book.css link.book.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/book.css" rel="stylesheet"/>
link.button.href: css/button.css link.button.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/button.css" rel="stylesheet"/>
link.fix.href: css/fix.css link.fix.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/fix.css" rel="stylesheet"/>
link.form.href: css/form.css link.form.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/form.css" rel="stylesheet"/>
link.layout.href: css/layout.css link.layout.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/layout.css" rel="stylesheet"/>
link.portlet.href: css/portlet.css link.portlet.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/portlet.css" rel="stylesheet"/>
link.window.href: css/window.css link.window.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/window.css" rel="stylesheet"/>
link.window−plain.href: css/plain/window.css link.window−plain.rel: stylesheet	<link href="/sampleportal/framework/skins/avitek/css/plain/window.css" rel="stylesheet"/>

script.skin.src: skin.js script.skin.type: text/javascript	<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/skin.js"></script>
script.menu.src: menu.js script.menu.type: text/javascript	<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/menu.js"></script>
script.float.src: float.js script.float.type: text/javascript	<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/float.js"></script>
script.menufx.src: menufx.js script.menufx.type: text/javascript	<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/menufx.js"></script>
script.util.src: util.js script.util.type: text/javascript	<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/util.js"></script>
script.delete.src: delete.js script.delete.type: text/javascript	<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/delete.js"></script>
script.search.path: js	Provides the directory for the location of the JavaScript files.

You can control the order in which the CSS and JavaScript entries are inserted into the HTML <head> section by adding link.input.index: 1 to a CSS entry and script.util.index: 1 to a script entry, where the number is the order in which the entry should be inserted. All CSS entries are inserted first, followed by all script entries.

Look & Feel Overrides

You can override the skin elements and skeletons on individual portal components so that those components have a different look & feel than the other portal components. For example, you can override the look & feel of a portlet so that it looks different than the other portlets on a page.

Overriding Look & Feel with Themes

As part of each skin or skeleton, you can create sub−skins and sub−skeletons called "themes." Themes contain all or part of the resources contained in a skin or skeleton. For example, a skin theme can contain a /css subdirectory with a single CSS file, and a skeleton theme can contain a single JSP to render a portlet titlebar. Themes can be used on books, pages, and portlets.

Each theme requires a .theme file located in <portal_Web_project>/framework/markup/theme/. Following is a sample theme file:

```
<?xml version="1.0" encoding="UTF−8"?>
<netuix:markupDefinition xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.
xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"
xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0 markup−netuix−1
    <netuix:locale language="en"/>
    <netuix:markup>
        <netuix:theme
            name="alert"
            title="Alert Theme" description="A simple alert theme."
            markupType="Theme" markupName="alert"/>
    </netuix:markup>
</netuix:markupDefinition>
```

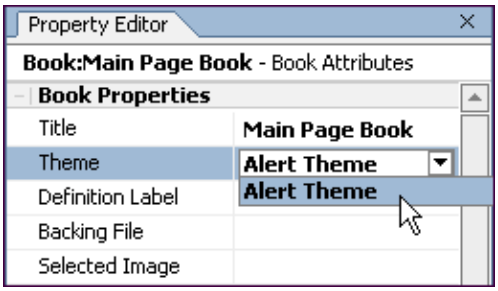
The theme file contains two key attributes:

- ***title*** − The title attribute value is used to populate the Theme drop−down list where it appears in the book, page, and portlet properties in the Portal Designer and in the WebLogic Administration Portal.
- ***name*** − The name attribute value tells the portal framework the name of the theme directory to look in to apply theme resources to the book, page, or portlet.

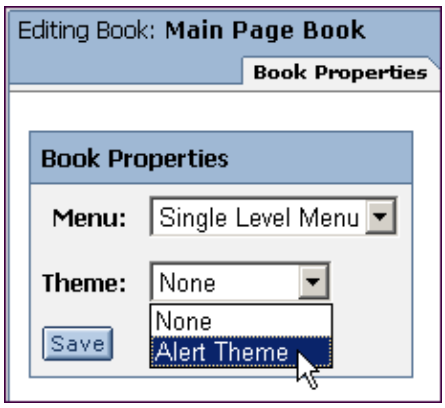
The theme XML is inserted in the .portal around the XML for the book, page, or portlet to which the theme applies.

The following figures show where you set a theme in WebLogic Workshop and in the WebLogic Administration Portal.

WebLogic Workshop



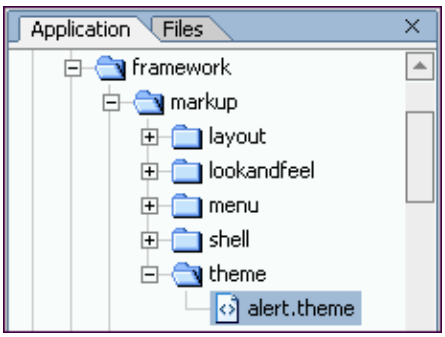
WebLogic Administration Portal



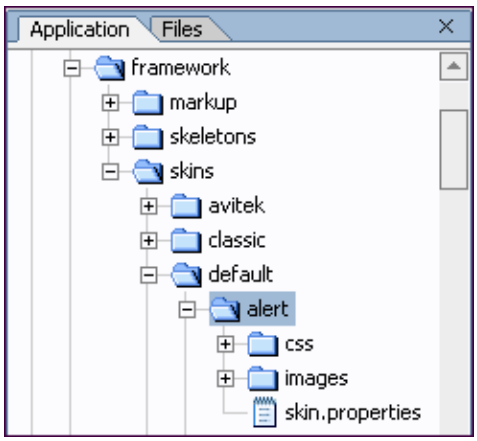
Theme selection for a book, page, or portlet does not depend on the look & feel selected for a desktop. All themes are available for selection for all look & feels, whether or not the skins and skeletons for the look & feels contain the selected theme. If a skin or skeleton does not contain the selected theme, the theme is ignored. If both a skin and a skeleton theme exists for the selected look & feel, both are used.

The following figures show an example theme directory structure for the theme file, a skin theme, and a skeleton theme:

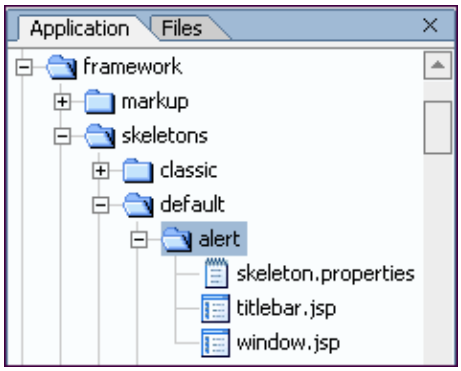
Theme File



Skin Theme



Skeleton Theme



If skin or skeleton resources are not explicity contained in a theme, the parent skin or skeleton resources are

used. For example, if a skeleton theme uses only a JSP to render a portlet titlebar, the parent skeleton JSPs are used to render the rest of the portlet.

For skeletons, the ability to use parent resources is dependent on a file in the skeleton theme directory called skeleton.properties,which contains a single entry:

jsp.search.path: ., ..

where ., .. is a relative path to the theme's own skeleton JSPs and to the parent skeleton's JSPs.

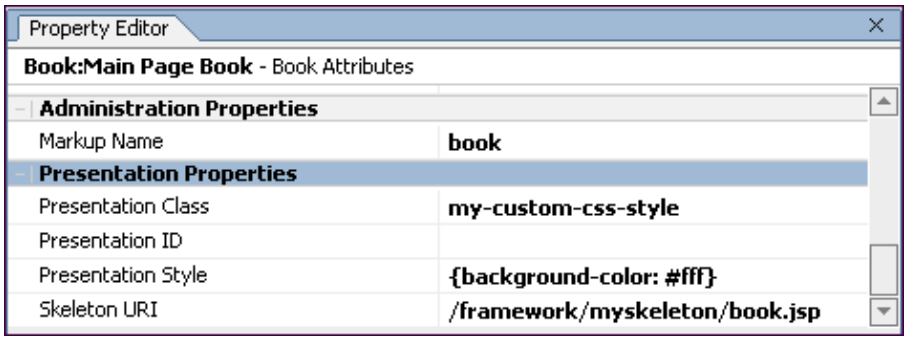
In the parent skin, the skin.properties must contain path information to its skin themes in the following format:

theme.***alert***.search.path: alert/images, images

The name of the theme directory is the second entry in the property. The path to the theme images is set (alert/images), along with the path to the parent skin's images directory (images) in case the theme images are an incomplete subset of the necessary images.

Overriding Look & Feel with Properties

For any selected component in the Portal Designer, you can override CSS properties and the skeleton JSP used to render the component. With the portal component selected in the Portal Designer, set the property overrides you want in the Property Editor under Presentation Properties, as shown in the following figure.



When the portal desktop is rendered as HTML, the skeleton JSP you selected is used to render the component, and the style overrides you entered are automatically inserted into the XML of the .portal file.

Summary

The look & feel selected for a portal desktop serves as the basis for how the desktop is rendered in HTML. The look & feel XML file (.laf) points to a specific skin and a specific skeleton on the file system to use for rendering.

Skins are made up of framework images (like portlet titlebar icons), CSS files, and script files, such as JavaScript. Skeletons are JSPs that convert XML−based portal components to HTML.

Once a look & feel is selected, its XML is inserted into the .portal XML file, which is the primary XML file used to control desktop rendering (.portlet XML files are used to render portlets). The look & feel settings point to the file−based skin and skeleton resources that are used to generate and used in the rendered HTML.

The skin used in a look & feel contains a skin.properties and an optional skin_custom.properties file that contains references to all images, CSS files, and script files that are used by the skin. The entries in skin.properties and skin_custom.properties are converted to HTML <head> entries so that any framework images, CSS styles, and script functions used in the HTML are recognized.

You can override the look & feel for any book, page, or portlet by using themes; and using the Portal Designer and Portlet Designer Property Editor you can override CSS styles, attributes, and the skeleton JSP used to render desktops, books, pages, and portlet titlebars and windows.

Related Topics

The Portal User Interface Framework

How the Shell Determines Header and Footer Content

How Portal Components are Rendered

Portal User Interface Framework Reference

Creating Look & Feel Files

Creating Skins and Skin Themes

Creating Skeletons and Skeleton Themes

Creating a Portal File

Properties for All Portal Components (property overrides)

Adding Visitor Tools to Portals

Tutorial: Changing a Portal's Look & Feel and Navigation

How the Shell Determines Header and Footer Content

When you build a portal in WebLogic Workshop, the shell that you select determines the header and footer content of the portal desktop. The shell can point to JSP or HTML files that contain the content, personalization, or other behavior you want to include in your headers and footers.

This topic contains the following sections:

Overview

The Shell File

The Portal File

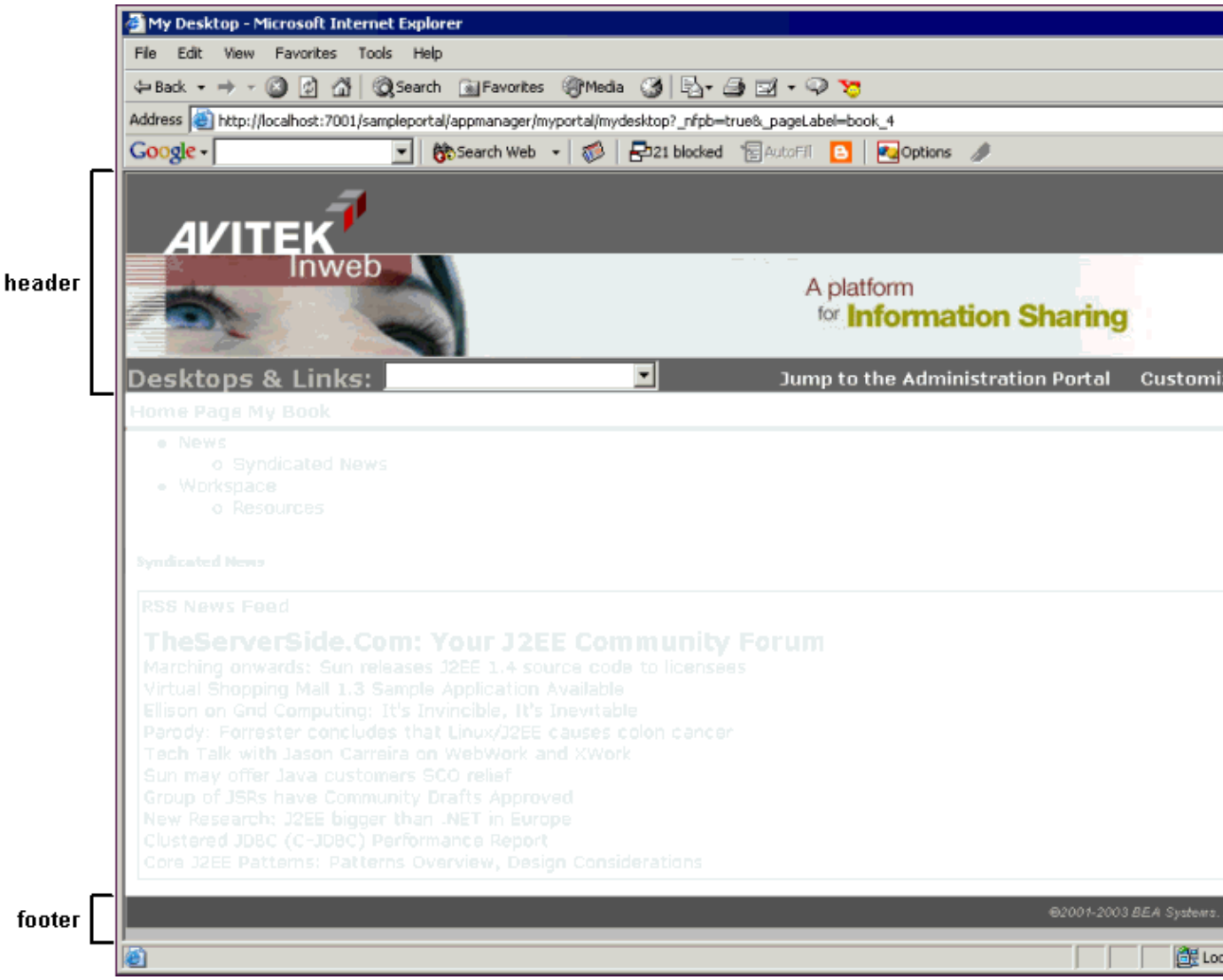
Location of the Shell Resources

How the Shell Relates to Look & Feel

Summary

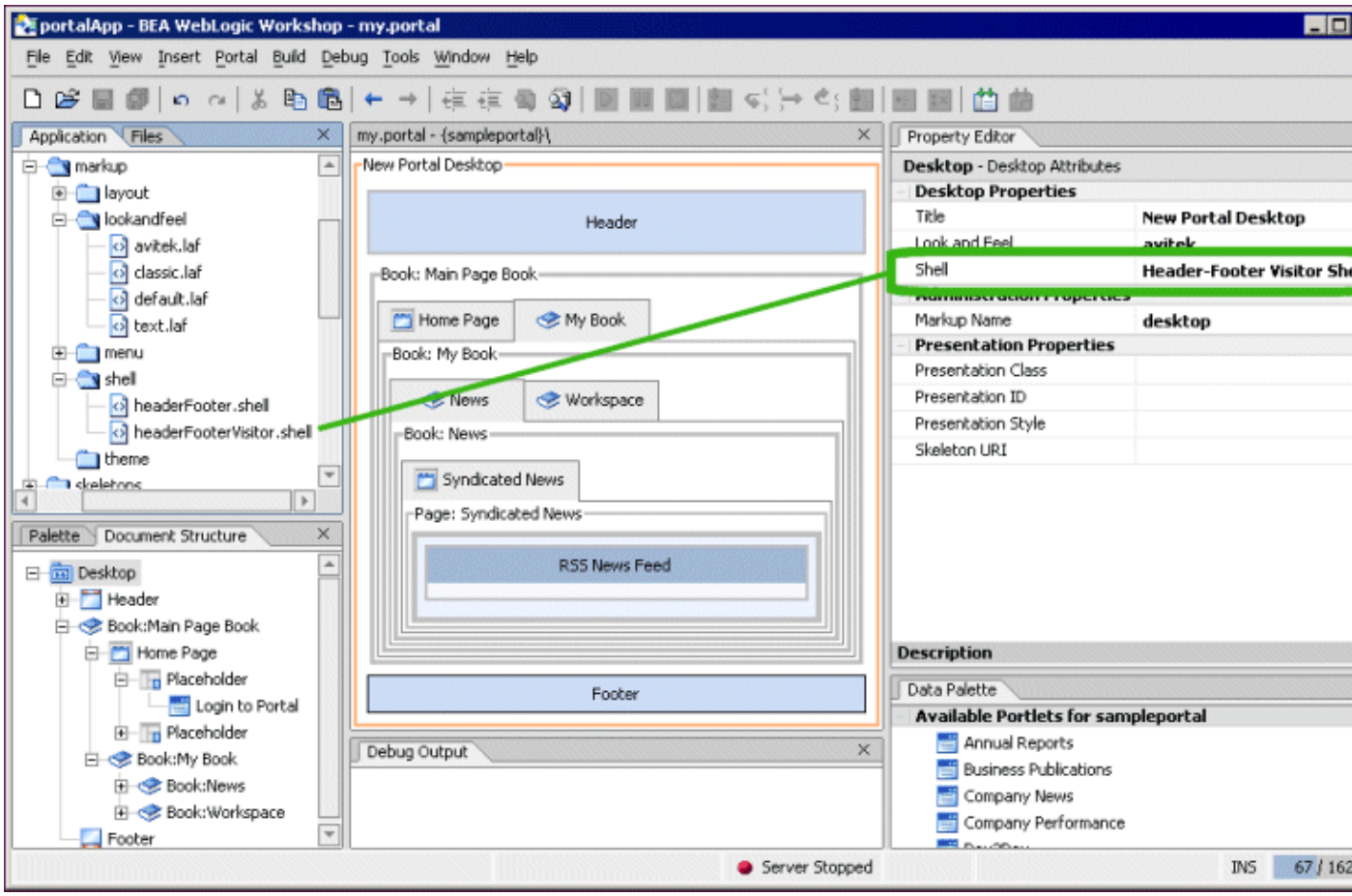
Overview

The following figure shows the area of a portal desktop controlled by the shell:



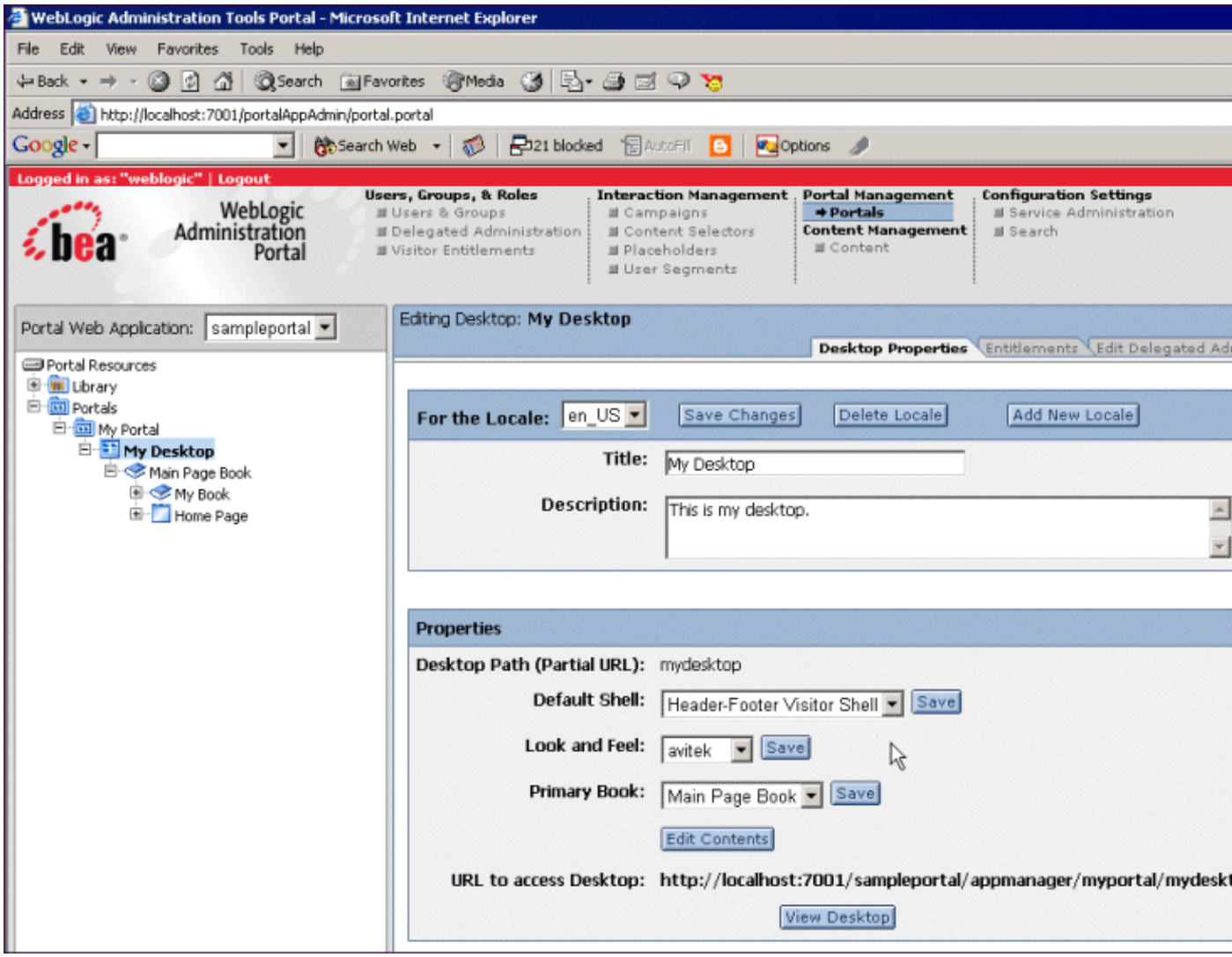
The shell could also be set up to include a left navigation region, as illustrated in the left navigation sample in the Sample Portal. So the shell really controls everything outside the main page book in a portal.

A shell is represented by an XML file (with .shell extension), as shown in the following figure.



Developers building portals with WebLogic Workshop are not the only users who can determine the shell used by a portal desktop. While developers create shells and select the default shell used by a portal, portal administrators ultimately determine the desktop shell.

After a portal administrator creates a desktop in the WebLogic Administration Portal, the administrator can change the desktop shell on the Dekstop Properties page.



The following section shows the shell XML file and describes how it is used to provide header and footer content.

The Shell File

The shell provides paths to the JSP or HTML files to be used in the desktop header and footer.

Shell files are stored in the following location: <portal_Web_project>/framework/markup/shell/. Following is the headerFooterVisitor.shell provided by BEA with the key attributes highlighted.

```
<?xml version="1.0" encoding="UTF-8"?>
<netuix:markupDefinition xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0 markup-netuix-1
    <netuix:locale language="en"/>
    <netuix:markup>
        <netuix:shell
            title="Header-Footer Visitor Shell"
            description="A header with a link and footer is included in this shell."
            markupType="Shell" markupName="headerFooterVisitor">
            <netuix:head/>
```

```
            <netuix:body>
                <netuix:header>
                    <netuix:jspContent contentUri="/portlets/header/header.jsp"/>
                </netuix:header>
                    <netuix:break/>
                <netuix:footer>
                    <netuix:jspContent contentUri="/portlets/footer/footer.jsp"/>
                </netuix:footer>
            </netuix:body>
        </netuix:shell>
    </netuix:markup>
</netuix:markupDefinition>
```

The following table describes the shell attributes and shows how they are used to put content in the desktop header and footer:

This <element> or attribute...	***does this***
title	Required. The string used to display the name in the shell drop−down fields in WebLogic Workshop and the WebLogic Administration Portal.
description	Optional. Description of the shell. The description is used in the WebLogic Administration Portal; when you select a shell in the portal Library, the description appears on the Shell Properties page.
markupType	Required. The name of the type of component. Must always be "Shell".
markupName	Required. The name for the shell. Each shell in the portal Web project must have a unique markupName.
<netuix:head/>	Required. This element maps to the head.jsp skeleton file that renders the boundaries of the HTML <head> region.
<netuix:body>	Required. This element maps to the body.jsp skeleton file that renders the boundaries of the HTML <body> region.
<netuix:header>	Required. This element maps to the header.jsp skeleton file that renders the boundaries of the header region in HTML.
<netuix:footer>	Required. This element maps to the footer.jsp skeleton file that renders the boundaries of the footer region in HTML.
<netuix:jspContent>	Optional. Use this element to reference the JSPs or HTML files you want to use for content in the header and/or footer (by way of the contentUri attribute). To use this element, make sure the <netuix:header> and <netuix:footer> tags have opening and closing elements inside which this tag is inserted. Use the contentUri attribute to reference the JSP or HTML file relative to the portal Web project.

The shell XML is automatically added to the underlying XML in the .portal file.

The Portal File

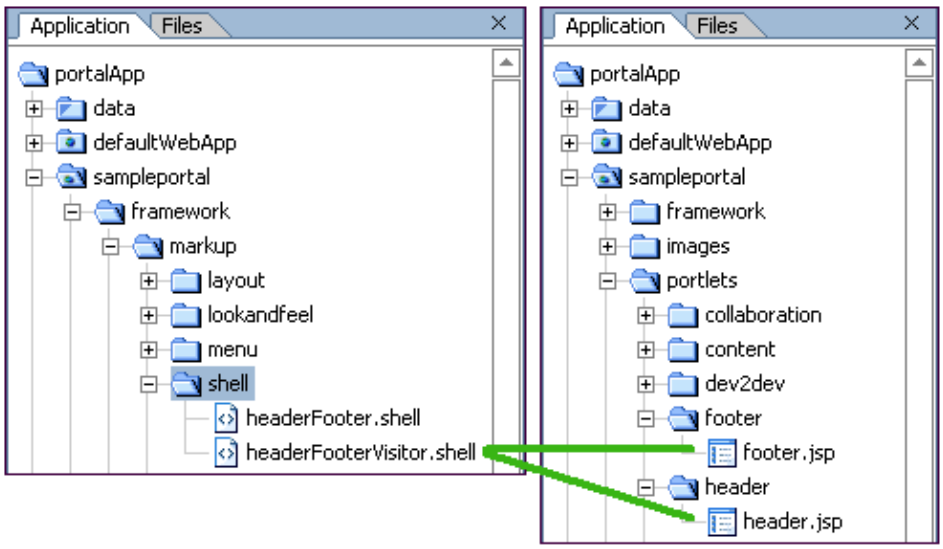
Following is an example portal file, created with the Portal Designer, showing the inserted shell XML from the .shell file. The shell XML was inserted when the Shell property was set for the selected desktop in the Property Editor. When the .shell file is inserted into the .portal file, its job is finished in the rendering process and the .portal file is used to set the header and footer content.

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root xmlns:html="http://www.w3.org/1999/xhtml-netuix-modified/1.0.0"
    xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0"
    xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0 portal-suppo
    <portal:directive.page contentType="text/html;charset=UTF-8"/>
    <netuix:desktop definitionLabel="defaultDesktopLabel" markupName="desktop" markupType="Desk
        <netuix:lookAndFeel definitionLabel="avitek" description="The avitek look and feel"
            markupName="avitek" markupType="LookAndFeel" skeleton="default"
            skeletonPath="/framework/skeletons/" skin="avitek" skinPath="/framework/skins/" tit
        <netuix:shell description="A header with a link and footer is included in this shell."
            markupName="headerFooterVisitor" markupType="Shell" title="Header-Footer Visitor Sh
            <netuix:head/>
            <netuix:body>
                <netuix:header>
                    <netuix:jspContent contentUri="/portlets/header/header.jsp"/>
                </netuix:header>

                    [XML for books, pages, and portlets...]

<netuix:footer>
                    <netuix:jspContent contentUri="/portlets/footer/footer.jsp"/>
                </netuix:footer>
```

The portal file is a template with which multiple desktops can be created in the WebLogic Administration Portal. When used as a template, the portal file determines the default shell of any desktop created from it.

Location of the Shell Resources

The shell attributes in the portal file tell the portal which content to use for the portal header and footer when the portal is rendered in HTML. The portal in the previous example will use the following shell resources:



The headerFooterVisitor.shell example file contains two tags that point to the content to use in the header and footer, one inside the <header> element and one inside the <footer> element:

- <netuix:jspContent contentUri="/portlets/header/header.jsp"/>
- <netuix:jspContent contentUri="/portlets/footer/footer.jsp"/>

The locations of those files are highlighted in the previous figure. When the portal is rendered, those JSPs are converted to HTML and inserted into the header and footer regions of the portal. The JSP files can contain any content or functionality allowed in a JSP, including personalization.

The JSPs referenced by this example shell do not have to be called header.jsp and footer.jsp. They could be any JSPs in the portal Web project. However, the skeleton JSPs used to render the boundaries of the header and footer regions are always called header.jsp and footer.jsp in the skeleton framework. The skeleton JSPs are different than the JSPs referenced by the shell. The following section explains the difference in more detail.

How the Shell Relates to Look & Feel

While the shell controls the content of the area surrounding the main book of a portal, the look & feel determines which skeleton header.jsp and footer.jsp are used to render the boundaries and styles of the header and footer areas.

In the following examples, do not be confused by the identically named header.jsp for both the look & feel and the shell header. They are different files with different uses. The fact that both have the same name is coincidence.

Look & Feel (header.jsp skeleton)

The "avitek" look & feel in the portal file uses the "default" skeleton located in /sampleportal/framework/skeletons/default/. Included in the default skeleton is a file called header.jsp that is used to render the <header> element in the portal file.

```
<%@ page import="com.bea.netuix.servlets.controls.application.HeaderPresentationContext" %>
<%@ page session="false"%>
<%@ taglib uri="render.tld" prefix="render" %>

<%
    HeaderPresentationContext header = HeaderPresentationContext.getHeaderPresentationContext(r
%>

<render:beginRender>
    <%-- Begin Body Header --%>
    <div
        <render:writeAttribute name="id" value="<%= header.getPresentationId() %>"/>
        <render:writeAttribute name="class" value="<%= header.getPresentationClass() %>" defaul
        <render:writeAttribute name="style" value="<%= header.getPresentationStyle() %>"/>
    >
</render:beginRender>

[The JSP referenced in the shell <header> element is inserted here at rendering.]

<render:endRender>
    </div>
    <%-- End Body Header --%>
```

```
</render:endRender>
```

This is a simple skeleton file that, when rendered, produces the following HTML:

```
<!-- Begin Body Header -->

<div
    class="bea-portal-body-header"
>

</div>

<!-- End Body Header -->
```

The opening tag uses a CSS class called "bea−portal−body−header" and then closes itself. The ending tag at the end of rendering closes the section. The JSP referenced in the shell header is inserted between the opening and closing tags where its content is rendered as shown in the following example:

```
<!-- Begin Body Header -->

<div
    class="bea-portal-body-header"
>

[The JSP referenced in the shell <header> element is inserted here at rendering.]

</div>

<!-- End Body Header -->
```

The shell's header.jsp inserted into the tag of the header region controls the content, styles, and behavior of the header content. The only elements provided by the look & feel are the tag and the bea−portal−body−header style class.

For troubleshooting purposes, you could view the rendered portal and view the bea−portal−body−header class (contained in the avitek skin's body.css) to find out which style elements for which the look & feel is responsible. Following is the definition of bea−portal−body−header:

```
.bea-portal-body-header, .bea-portal-body-footer
{
    margin: 0px;
    padding: 1px;
    color: #C3C6B1;
}
.bea-portal-body-header
{
    font-size: large;
    font-weight: bold;
}
```

Summary

The shell selected for a portal desktop determines the content of the area surrounding the portal's main book. The shell XML file (.shell) includes references to the HTML or JSP files you want to appear in the desktop

header and footer.

HTML and JSP files used in a header or footer can contain any content or functionality allowed in those types of files, including personalization in JSP files.

Once a shell is selected, its XML is inserted into the .portal XML file, which is the primary XML file used to control desktop rendering (.portlet XML files are used to render portlets).

While look & feel determines the physical boundaries of the header and footer and can include CSS styles and other skin elements generated by the skeleton header.jsp or footer.jsp files, the HTML or JSP files inserted in the header or footer by the shell control the content, styles, style overrides, and behavior of the header and footer.

Related Topics

The Portal User Interface Framework

How Look & Feel Determines Rendering

How Portal Components are Rendered

Creating Shells

Creating a Portal File

How Do I: Personalize a Desktop Header or Footer?

How Portal Components Are Rendered

With the look & feel and shell selected for a portal desktop, the rendering service has the basic information it needs to convert a .portal XML file into a final HTML file.

This topic shows the rendering lifecycle, step by step, for a single portal component. The same rendering principles apply for all other portal components.

This topic includes the following sections:

Overview

Single File vs. Streamed Rendering

Rendering Lifecycle of a Book

Summary

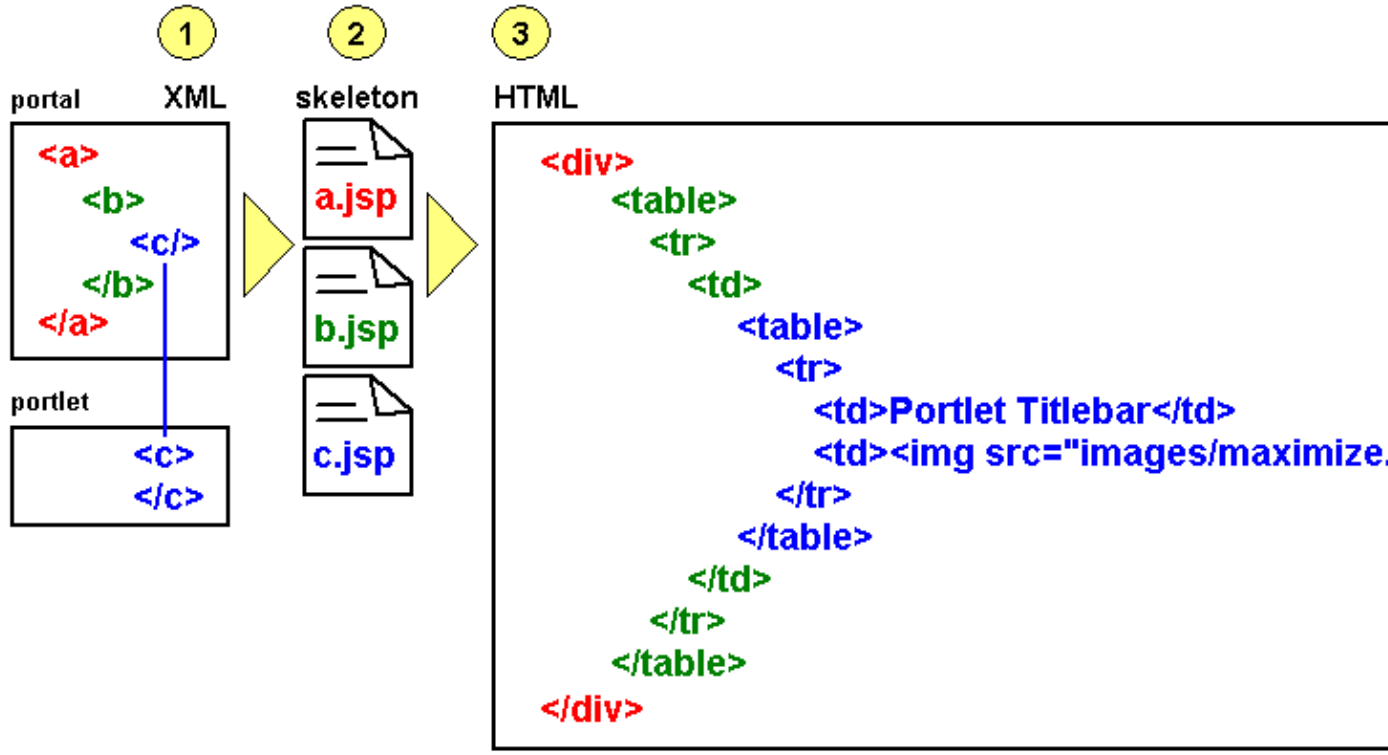
Overview

There are three basic stages in the portal rendering process a process that ultimately results in a portal desktop being displayed in a browser:

1. ***Building a portal in XML*** − In the portal development process, you use the Portal and Portlet designers in WebLogic Workshop to build .portal and .portlet files. Both types are XML files. As you build portals and portlets in WebLogic Workshop, the XML elements and attributes are automatically built under the surface.

   The previous topics, How Look & Feel Determines Rendering and How the Shell Determines Header and Footer Content, described part of the XML−building process: how the look & feel and shell XML files are added to the portal XML file to provide rendering instructions.
2. ***Portal XML elements mapped to JSP skeleton files*** − The portal framework maps specific XML elements to specific JSP skeleton files. They are called skeleton files because they are used to render the physical boundaries and structure the skeleton of their portal components. For example, a portlet titlebar in a portlet XML file uses an element called <netuix:titlebar>. The portal framework knows to use the titlebar.jsp skeleton file to render the portlet titlebar.
3. ***JSP skeleton files and skin.properties are rendered as HTML*** − Each skeleton JSP file performs its own processing, such as retrieving property values you set in the WebLogic Workshop Property Editor (and were automatically added to the portal XML file), and generates the appropriate HTML for the portal component. The skin.properties and optional skin_custom.properties files for the selected look & feel are converted to image path entries, CSS file entries, and script file entries in the HTML <head> area.

The following figure is a simplified illustration of the rendering process.



This topic will expand on these three stages using the rendering lifecycle of a single portal component as an example.

Before going into greater detail on the rendering process, it is important to understand the difference between viewing a portal in the development environment (WebLogic Workshop) and viewing it in the administration/end user environment (WebLogic Administration Portal/browser). The three−stage rendering process occurs in slightly different ways in the two different environments. The following section describes the basic principles of each.

Single File vs. Streamed Rendering

The .portal file you create in WebLogic Workshop is a template. In this template you create books, pages and portlets and define defaults for them. When you view the .portal file with your browser the portal is rendered in "single file mode," meaning that you are viewing the portal from your file system as opposed to a database. The .portal file's XML is parsed and the rendered portal is returned to the browser. The creation and use of a .portal is intended for development purposes, but you can access a .portal file in production. Because there is no database involved you cannot take advantage of features such as user customization or entitlements.

Once you have created a .portal file you can use it to create desktops for a production environment.

A desktop is a particular view of a portal that visitors access. A portal can be made up of multiple desktops, making the portal a container for desktops. A desktop contains all the portlets, content, shells, layouts, and look and feel elements necessary to create individual user views of a portal.

When you create a desktop based on the .portal file in the WebLogic Administration Portal, the .portal and its resources are placed into the database. The settings in the .portal file, such as the look & feel, serve as defaults

to the desktop. Once a new desktop is created from a .portal template, the desktop is decoupled from the template, and modifications to either the .portal file do not affect the desktop, and vice versa. For example, when you change a desktop's look & feel in the WebLogic Administration Portal, the change is made only to the desktop, not to the original .portal file. When you view a desktop with a browser it is rendered in "streaming mode" (from the database). Now that a database is involved, desktop customizations can be saved and delegated administration and entitlements can be set on portal resources.

Rendering Lifecycle of a Book

This section illustrates the rendering lifecycle of a book, which will help you understand the rendering lifecycle of other portal components, such as pages and portlets.

This section contains the following topics:

1. Building a portal in XML

2. Portal XML elements mapped to JSP skeleton files

3. JSP skeleton files and skin.properties are rendered as HTML

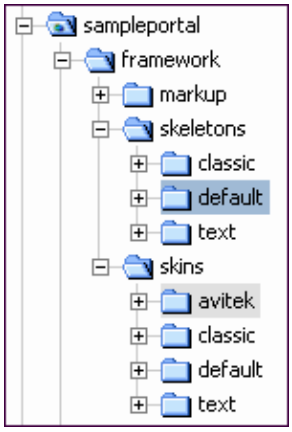
1. Building a portal in XML

This section describes steps that populate and configure the .portal XML file.

Selecting Look & Feel

When you select the look & feel for a desktop, the look & feel file determines which skin and skeleton is used to render all desktop components. In the following example, the "avitek" look & feel has been selected, which points uses the "avitek" skin and the "default" skeleton. The look & feel XML is added to the .portal XML file.

```
<netuix:markup>
    <netuix:lookAndFeel
        definitionLabel="avitek" title="avitek"
        description="The avitek look and feel"
        skin="avitek" skinPath="/framework/skins/"
        skeleton="default" skeletonPath="/framework/skeletons/"
        markupType="LookAndFeel" markupName="avitek"/>
</netuix:markup>
```

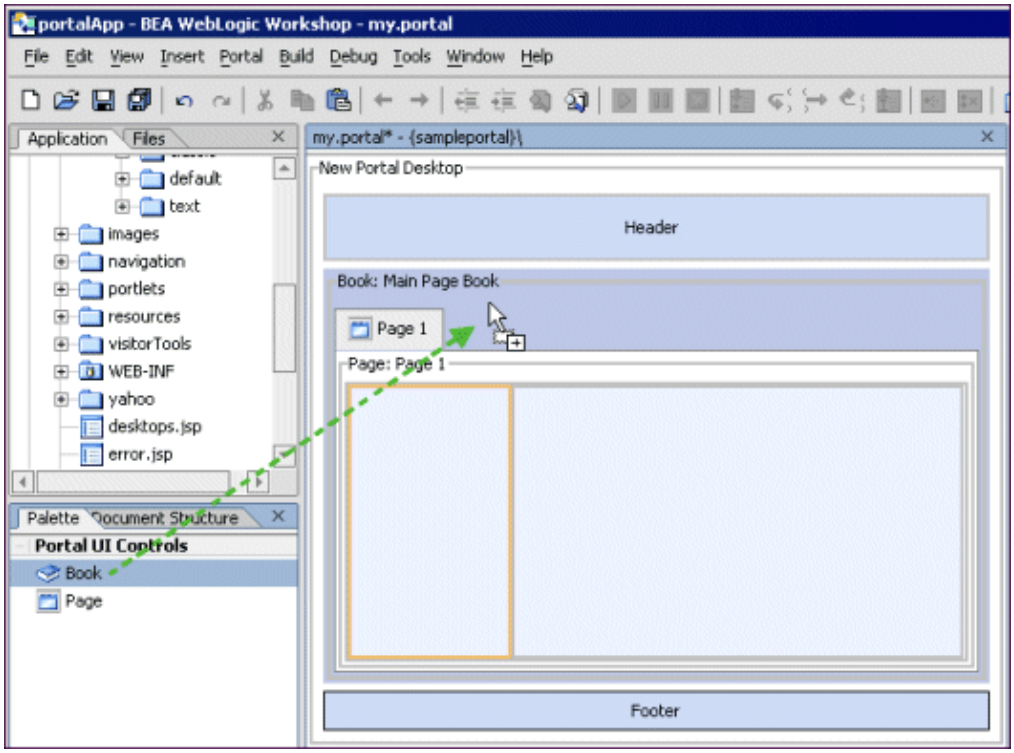


The skin and skeleton come into play later in the rendering process, when the desktop is viewed with a browser. Before that happens, the book that will be used to illustrate the rendering process will be added to the portal.

Adding a Book to a Portal

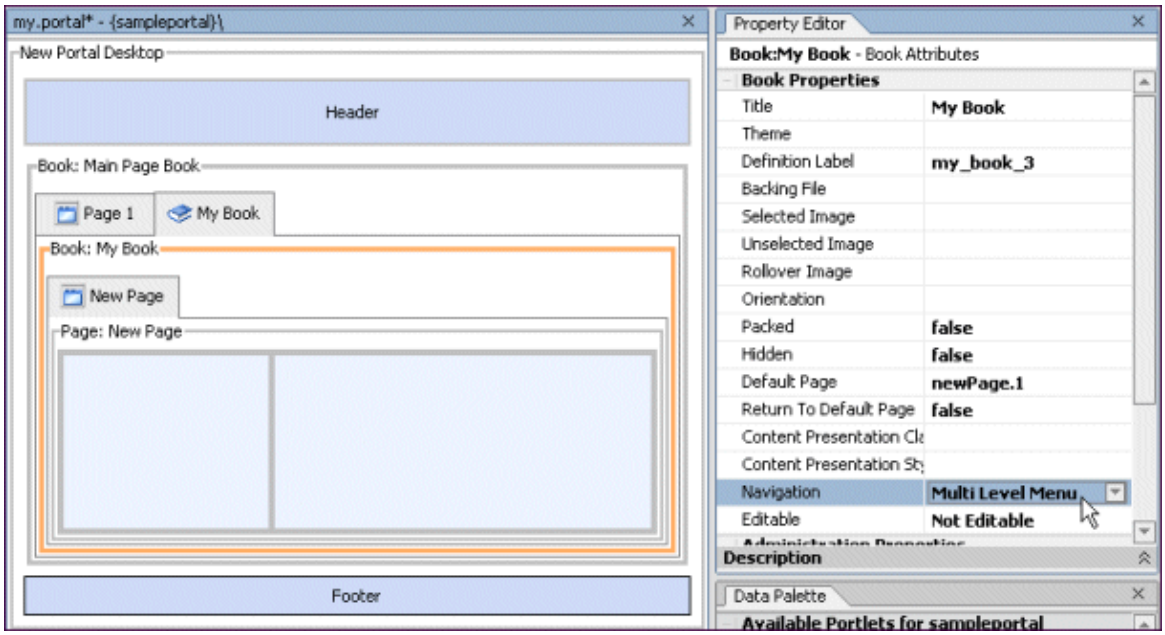
In this section a book is added to the desktop in the .portal file and configured. Books can also be added by portal administrators in the WebLogic Administration Portal, which adds the book directly to the database.

The following figure shows a book control being dragged onto the desktop.



After the book is added to the desktop, the book title is changed from "New Book" to "My Book," and the navigation style is set to Multi Level Menu, as shown in the following figure.

Navigation controls the way a book's sub−books and pages are accessed. The single−level menu provides text links/tabs to sub−books and pages, and the multi−level menu provides a drop−down menu to access sub−books and pages. (Books must be added to books rather than to pages inside books for drop−down navigation to work. So in the following example, for the multi−level menu to produce a drop−down menu, you would need to drag a new book control into Main Page Book, right next to Page 1, as shown in the figure.)



After the Navigation style is set on My Book, the following is what the book looks like in XML in the .portal file. If you add a book in the WebLogic Administration Portal, the XML is added to the database. This XML is used as the basis for the rendering of the book.

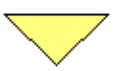
```
<netuix:book defaultPage="newPage.1" definitionLabel="my_book_3"
    markupName="book" markupType="Book" title="My Book">
    <netuix:multiLevelMenu
        description="This menu can navigate across may nested books."
        markupName="multiLevelMenu" markupType="Menu" title="Multi Level Menu"/>
            <!-- in this example, the nested page content has been removed -->
</netuix:book>
```

2. Portal XML elements mapped to JSP skeleton files

When the desktop is viewed in a browser, the portal framework reads the XML elements and uses the skeleton path to map the desktop's XML elements to skeleton JSPs. The following examples show which elements in the book XML are mapped to skeleton JSPs and which skeleton JSPs are used to render the elements.

book XML − The highlighted elements are mapped to skeleton JSPs.

```
<netuix:book defaultPage="newPage.1" definitionLabel="my_book_3"
    markupName="book" markupType="Book" title="My Book">
    <netuix:multiLevelMenu
        description="This menu can navigate across may nested books."
        markupName="multiLevelMenu" markupType="Menu" title="Multi Level Menu"/>
            <!-- in this example, the nested page content has been removed -->
</netuix:book>
```



skeleton − Referenced in the look & feel

/framework/skeletons/default/
book.jsp
multilevelmenu.jsp
submenu.jsp (referenced in multilevelmenu.jsp)

Once rendering has been handed off to the JSPs, the JSPs perform the tasks necessary for conversion to HTML. Following are the book.jsp, multilevelmenu.jsp, and submenu.jsp used in this example. Comments are added to describe what the JSPs are doing.

book.jsp

The book.jsp serves as a high−level container for the book's menu and the book's child books and pages.

```
<%@ page import="com.bea.netuix.servlets.controls.page.BookPresentationContext,
                 com.bea.netuix.servlets.controls.page.MenuPresentationContext"
%>
<%@ page session="false"%>
<%@ taglib uri="render.tld" prefix="render" %>

<render:beginRender>

<%−− The content inside the <render:beginRender> tag is processed first
     and ultimately renders whatever is inside it. In most cases, the
     skeletons produce an opening <div> HTML tag with specific attributes
     such as CSS classes.

     The following block determines where the book falls in the desktop
     hierarchy (whether it is the top−level book or a nested book). It
     also sets the base name of the CSS class to use (bea−portal−book)
     and appends different endings to the base class to apply a
     different CSS class for each book context. Only processing,
     not HTML rendering, occurs in this block.
−−%>


<%
    BookPresentationContext book = BookPresentationContext.getBookPresentationContext(request);
    MenuPresentationContext menu = (MenuPresentationContext) book.getFirstChild("page:menu");
    String bookClass = "bea−portal−book";
    String useBookClass = bookClass;

    if (book.isDesktopBook())
    {
        bookClass += "−primary";
        useBookClass = bookClass;
    }
    else if (book.isLikePage())
    {
        useBookClass += "−invisible";
    }

    String bookContentClass = bookClass + "−content";
%>

<%−− The next block begins the actual HTML rendering, beginning with
     the comment "Begin Book" followed by an opening <div> HTML
     tag. Notice the JSP tags used before the closing bracket of the
     <div> tag. These populate the div tag with style attributes.
     The methods retrieve any presentation property override values
```

```
     you entered in the WebLogic Workshop Property Editor for the book.
     For the "class" attribute, the default value is useBookClass,
     which earlier is set to "bea-portal-book". (If through getting
     the context the book was found to be the top-level book, the value
     of useBookClass would be "bea-portal-book-primary".)
     With no overrides, the useBookClass variable will produce the
     following HTML, because the book is acting like a page:

     <div
         class="bea-portal-book-invisible"
     >

     The style sheet class is provided by the skin, and the CSS file
     containing the class is referenced in the skin's skin.properties
     file and added to the HTML <head> region.
--%>

    <%-- Begin Book --%>

    <div
        <render:writeAttribute name="id" value="<%= book.getPresentationId() %>"/>
        <render:writeAttribute name="class" value="<%= book.getPresentationClass()%>" defaultVa
        <render:writeAttribute name="style" value="<%= book.getPresentationStyle() %>"/>
    >

<%-- The following JSP tag gets the names of the pages and books it will
     display in its navigation menu, and based on the navigation element
     used in the portal XML file (in this case netuix:multiLevelMenu),
     uses the corresponding menu JSP (multilevelmenu.jsp) to render the menu
     in this position of the HTML.
--%>

        <render:renderChild presentationContext="<%= menu %>"/>

<%-- The following block provides a <div> HTML container for the
     book's content area--the child books and pages. Again, it uses
     a JSP tag to set the style sheet "class" attribute.
--%>

        <%-- Begin Book Content --%>
        <div
            <render:writeAttribute name="class" value="<%= book.getContentPresentationClass()%>
            <render:writeAttribute name="style" value="<%= book.getContentPresentationStyle() %
        >
</render:beginRender>

<%-- The closing </render:beginRender> tag signals the portal framework
     to stop rendering the book. After the book's children and their
     children are rendered, the portal framework uses the following
     <render:endRender> tag to close the book's parent HTML tags.
--%>

<render:endRender>
        </div>
        <%-- End Book Content --%>
    </div>
    <%-- End Book --%>
</render:endRender>
```

Following is a description of the multilevelmenu.jsp, which is used by the book to render the navigation menu

for the book's child books and pages.

multilevelmenu.jsp

The multilevelmenu.jsp is rendered inside the book container and provides the boundaries for multi−level menus on books. This JSP also uses submenu.jsp to perform the actual rendering of the menu links.

```
<%@ page import="com.bea.netuix.servlets.controls.window.WindowPresentationContext,
                 com.bea.netuix.servlets.controls.page.BookPresentationContext,
                 com.bea.netuix.servlets.controls.page.MenuPresentationContext,
                 java.util.List,
                 java.util.Iterator,
                 com.bea.netuix.servlets.controls.page.PagePresentationContext,
                 com.bea.netuix.servlets.controls.window.WindowCapabilities" %>
<%@ page session="false"%>
<%@ taglib uri="render.tld" prefix="render" %>

<%−− The following block determines where the book falls in the desktop
     hierarchy (whether it is the top−level book or a nested book). It
     also sets the base name of the CSS class to use (bea−portal−book)
     and defines different menu style classes by appending different
     endings to the base class. Only processing, not HTML rendering,
     occurs in this block.
−−%>

<%
    BookPresentationContext book = BookPresentationContext.getBookPresentationContext(request);
    MenuPresentationContext menu = MenuPresentationContext.getMenuPresentationContext(request);
    String bookClass = "bea−portal−book";

    if (book.isDesktopBook())
    {
        bookClass += "−primary";
    }

    final String menuClass = bookClass + "−menu";
    final String menuContainerClass = menuClass + "−container";
    final String menuItemClass = menuClass + "−item";
    final String menuItemActiveClass = menuItemClass + "−active";
    final String menuItemLinkClass = menuItemClass + "−link";
    final String menuHookClass = menuClass + "−hook";
    final String menuButtonsClass = menuItemClass + "−buttons";
    List menuChildren = menu.getChildren();
%>

<render:beginRender>

<%−− The content inside the <render:beginRender> tag is processed first
     and ultimately renders whatever is inside it, such as opening <div>
     HTML tags with specific attributes and tables.

     The following block creates a table cell, sets CSS styles on the <td>
     tag (based on the members defined in the previous block).
−−%>


<%−− Begin Multi Level Menu −−%>
    <div class="bea−portal−ie−table−buffer−div">
        <table border="0" cellpadding="0" cellspacing="0" width="100%">
            <tr>
```

```
                <td class="<%= menuContainerClass %>" align="left" nowrap="nowrap">

<%-- The following block builds the menu in the table cell. It first adds an
     unordered list <ul> to the cell and sets its style class. Then, an IF
     statement checks to see if the book is in VIEW mode. If true, CSS styles
     are put in the request as attributes to be used by the menu.

     After the attributes are added to the request, the skeleton's submenu.jsp
     is inserted, which does the following:

        * Gets the CSS styles from the request.
        * Gets the book's child pages and books.
        * Creates list items <li> of the children and creates links out of them.
          The menuHookClass at the end of the block is used by the skin's menu.js file
          to insert the rendered menu. The <ul> that is generated is a menu
          structure description that is read and rewritten by menu.js.
        * Adds CSS styles to the request and includes submenu.jsp to handle
          the menus of nested books.

     After the menu is built, the CSS styles are removed from the request.
--%>

                    <ul
                        <render:writeAttribute name="id" value="<%= menu.getPresentationId() %>
                        <render:writeAttribute name="class" value="<%= menu.getPresentationClas
                        <render:writeAttribute name="style" value="<%= menu.getPresentationStyl
                    ><%
                        if (book.getWindowMode().equals(WindowCapabilities.VIEW))
                        {
                            request.setAttribute(BookPresentationContext.class.getName() + ".ro
                            request.setAttribute(BookPresentationContext.class.getName() + ".me
                            request.setAttribute(BookPresentationContext.class.getName() + ".me
                            request.setAttribute(BookPresentationContext.class.getName() + ".me
                            request.setAttribute(BookPresentationContext.class.getName() + ".me
                            request.setAttribute(BookPresentationContext.class.getName() + ".me
                            %><jsp:include page="submenu.jsp"/><%
                            request.removeAttribute(BookPresentationContext.class.getName() + "
                            request.removeAttribute(BookPresentationContext.class.getName() + "
                            request.removeAttribute(BookPresentationContext.class.getName() + "
                            request.removeAttribute(BookPresentationContext.class.getName() + "
                            request.removeAttribute(BookPresentationContext.class.getName() + "
                            request.removeAttribute(BookPresentationContext.class.getName() + "
                        }
                    %></ul>

                    <div class="<%= menuHookClass %>"></div>
                </td>

<%-- The following block adds a table cell next to the menu table cell
     if a menu is present. The <render:endRender> contents are inserted
     in the HTML after all menu children are inserted, which closes
     the menu table.
--%>

<%
    if (menuChildren != null && menuChildren.size() > 0)
    {
%>
                <td class="<%= menuButtonsClass %>" align="right" nowrap="nowrap">
<%
    }
```

```
%>
</render:beginRender>
<render:endRender>
<%
    if (menuChildren != null && menuChildren.size() > 0)
    {
%>
                </td>
<%
    }
%>
            </tr>
        </table>
    </div>
    <%-- End Multi Level Menu --%>
</render:endRender>
```

submenu.jsp

The submenu.jsp is inserted inside the multilevelmenu.jsp. It retrieves a book's child books and pages and builds the navigation links to those children.

```
<%@ page import="java.util.Iterator,
                 java.util.List,
                 com.bea.netuix.servlets.controls.page.BookPresentationContext,
                 com.bea.portlet.PageURL,
                 com.bea.netuix.servlets.controls.page.PagePresentationContext"%>
<%@ page session="false"%>

<%-- The following block gets the CSS styles placed in the request by
     multilevelmenu.jsp.
--%>

<%
    Boolean isRoot
        = (Boolean) request.getAttribute(BookPresentationContext.class.getName() + ".root-flag"
    BookPresentationContext bookCtx
        = (BookPresentationContext) request.getAttribute(BookPresentationContext.class.getName(
    String menuClass
        = (String) request.getAttribute(BookPresentationContext.class.getName() + ".menu-class"
    String menuItemClass
        = (String) request.getAttribute(BookPresentationContext.class.getName() + ".menu-item-c
    String menuItemActiveClass
        = (String) request.getAttribute(BookPresentationContext.class.getName() + ".menu-item-a
    String menuItemLinkClass
        = (String) request.getAttribute(BookPresentationContext.class.getName() + ".menu-item-l
%>

<%-- The following block checks to see if the book and its children
     are visible. If true, the labels of the children are retrieved,
     iterated over, and inserted as hyperlinked list items <li>
     inside the unordered list <ul> inserted by multilevelmenu.jsp.
     Notice the nested <ul> at the end of the block, which provides
     for submenu nesting.
--%>

<%
    if (!bookCtx.isHidden() && bookCtx.isVisible())
    {
        if (bookCtx instanceof BookPresentationContext)
```

```
        {
            List bookChildren = bookCtx.getPagePresentationContexts();
            Iterator it = bookChildren.iterator();

            while (it.hasNext())
            {
                PagePresentationContext childPageCtx = (PagePresentationContext) it.next();

                if (!childPageCtx.isHidden() && childPageCtx.isVisible())
                {
                    %><li class="<%= isRoot.booleanValue() && childPageCtx.isActive() ? menuIte
                    %><a class="<%= menuItemLinkClass %>" href="<%= PageURL.createPageURL(reque

                    if (childPageCtx instanceof BookPresentationContext)
                    {
                        %><ul class="<%= menuClass %>"><%
                        request.setAttribute(BookPresentationContext.class.getName() + ".root-f
                        request.setAttribute(BookPresentationContext.class.getName() + ".menu-i
                        %><jsp:include page="submenu.jsp"/><%
                        request.removeAttribute(BookPresentationContext.class.getName() + ".roo
                        request.removeAttribute(BookPresentationContext.class.getName() + ".men
                        %></ul><%
                    }

                    %></li><%
                }
            }
        }
    }
%>
```

JavaScript in Menus

The menus in a desktop use JavaScript functions for such functionality as drop−down menus and rollovers. These JavaScript functions are called from the skeleton's body.jsp, which contains the following entry:

```
<render:writeAttribute name="onload" value="<%= body.getOnloadScript() %>"/>
```

The onload value is retrieved from the following property in the skin's skin.properties file:

```
document.body.onload: initSkin()
```

Following is the HTML written by the body.jsp:

```
<body

    class="bea-portal-body"

    onload="initSkin();"

>
```

The initSkin() JavaScript function is the base function that calls menu−rendering functions in other JavaScript files. The initSkin() function is contained in the skin.js file. Other menu functions are contained in the menu.js and menufx.js files. Since all of those JavaScript files are listed in the skin's skin.properties file, they are automatically added to the HTML <head> region at rendering, and the functions they contain are recognized.

The next section describes the final process of the skeleton JSPs and skin.properties being converted to HTML.

3. JSP skeleton files and skin.properties are rendered as HTML

The previous section described the skeleton JSPs that are used to convert a book with a multi−level menu to HTML. The descriptions in that section described briefly some of the HTML generated by the JSPs.

This section shows the final HTML that is generated for a book, describes where it came from, and shows where some of the CSS styles used are defined.

Not all HTML for the desktop is shown in the following table. Only the sections that relate to the look & feel and the example book are shown.

skin.properties and ***skin_custom.properties*** − The paths to skeletons, skins, images, style sheets, and JavaScript files i HTML <head> region are inserted from the skin's skin.properties and skin_custom.properties files. To see the original skin.properties entries, see The skin.properties File in "How Look & Feel Determines Rendering." The <head> tag is i by the head.jsp file used for the shell. The <title> is inserted from the desktop title in the .portal file.

The first three <meta> tags are for testing and debugging purposes. These can be removed from skin.properties by sett enable.meta.info property to false.

```
<head>

<title>New Portal Desktop</title>
<meta name="bea-portal-meta-skeleton" content="/framework/skeletons/default"/>
<meta name="bea-portal-meta-skin" content="/framework/skins/avitek"/>
<meta name="bea-portal-meta-skin-images" content="/framework/skins/avitek/images"/>
<link href="/sampleportal/framework/skins/avitek/css/body.css" rel="stylesheet"/>
<link href="/sampleportal/framework/skins/avitek/css/button.css" rel="stylesheet"/>
<link href="/sampleportal/framework/skins/avitek/css/window.css" rel="stylesheet"/>
<link href="/sampleportal/framework/skins/avitek/css/plain/window.css" rel="stylesheet"/>
<link href="/sampleportal/framework/skins/avitek/css/portlet.css" rel="stylesheet"/>
<link href="/sampleportal/framework/skins/avitek/css/book.css" rel="stylesheet"/>
<link href="/sampleportal/framework/skins/avitek/css/fix.css" rel="stylesheet"/>
<link href="/sampleportal/framework/skins/avitek/css/layout.css" rel="stylesheet"/>
<link href="/sampleportal/framework/skins/avitek/css/form.css" rel="stylesheet"/>
<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/menu.js"></script>
<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/util.js"></script>
<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/delete.js"></script
<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/float.js"></script>
<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/menufx.js"></script
<script type="text/javascript" src="/sampleportal/framework/skins/avitek/js/skin.js"></script>

</head>
```

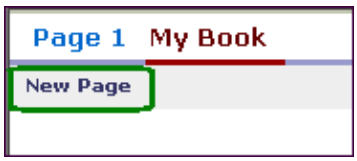
The following section shows the HTML that is produced by each skeleton JSP.

book.jsp

```
<div
    class="bea-portal-book-invisible"
>
```

```
<div class="bea-portal-ie-table-buffer-div">
        <table border="0" cellpadding="0" cellspacing="0" width="100%">
            <tr>
                <td class="bea-portal-book-menu-container" align="left" nowrap="nowrap">
                    <ul
                        class="bea-portal-book-menu"
                    >


<li class="bea-portal-book-menu-item-active"><a class="bea-portal-book-menu-item-link"
href="http://localhost:7001/sampleportal/my.portal?_nfpb=true&_pageLabel=my_page_6">New Page</

                    </ul>
                    <div class="bea-portal-book-menu-hook"></div>
                </td>

</tr>
        </table>
    </div>
        <div
            class="bea-portal-book-content"
        >

        <%-- The book content (sub-books and pages) is inserted here. -->

        </div>
</div>
```

When the desktop for this example is rendered, the following appears in the browser:



The circled area in this figure is the only content rendered for the book. The book contains only one page, so there is only one menu item for the book. The "Page 1" and "My Book" tabs are menu items rendered by the parent Main Page Book. That is why you do not see the "My Book" in the previous HTML block: because the book is responsible for rendering only a menu of its child books and pages.

If "New Page" contained a portlet, the portlet would appear in the browser. However, the rendering of the page and portlet is handled by different skeleton JSPs: one to provide a container for the page content, one to render the layout of the page (table cells that contain portlets and sub−books), and a few to handle the rendering of the portlet.

The book is responsible for rendering only two things:

- The menu of sub−books and pages it contains.
- Opening and closing tags to serve as the container for sub−books, pages, portlets, and other sub−components contained in the book.

CSS Styles in the Example

As you can see from the previous example of rendered HTML code, the skeleton JSPs insert many CSS styles. For example, the multilevelmenu.jsp inserts <td class="bea−portal−book−menu−container" ...>. The style classes inserted by multilevelmenu.jsp are rewritten by the skin's menu.js file.

Also, some of the style classes inserted by the skeleton JSPs are not defined in any of the CSS files provided by BEA. You can add these style classes to your custom CSS files to control those styles in your portal desktops.

To determine which styles you want to modify, see Portal User Interface Framework Reference.

Changing Look & Feel

If the look & feel is changed, a different skin and skeleton is referenced by the look & feel file, and rendering is subject to that skin and skeleton. With a different skin and skeleton, CSS files and script code can change completely.

Summary

There are three basic stages in the portal rendering process: building a portal in XML, portal XML components being mapped to skeleton JSPs, and skeleton JSPs rendering the portal desktop in HTML. The latter two stages are handled automatically by the portal framework.

There is a rendering difference between viewing a portal desktop in development mode and in administration/end user mode. In development mode, when you view the .portal file in a browser you see it in "single file" mode, meaning the desktop is being rendered from the file system. In administration/end user mode, you view a portal desktop in a browser in "streamed" mode, meaning the desktop components are being streamed from a database. When you create a portal desktop in the WebLogic Administration Portal using a .portal file as a template for the desktop, the portal components are added to the database and are decoupled from the original .portal file.

Related Topics

Portal Key Concepts and Architecture

The Portal User Interface Framework

How Look & Feel Determines Rendering

How the Shell Determines Header and Footer Content

Portal User Interface Framework Reference

Portal User Interface Framework Reference

Many styles and skeletons are used to render a portal desktop. This topic shows some of the main CSS style classes used in desktop look & feels that are shipped with WebLogic Portal and describes the skeleton JSPs used to render portal components in HTML. Use this topic to help you troubleshoot your portal desktops.

This topic contains the following sections:

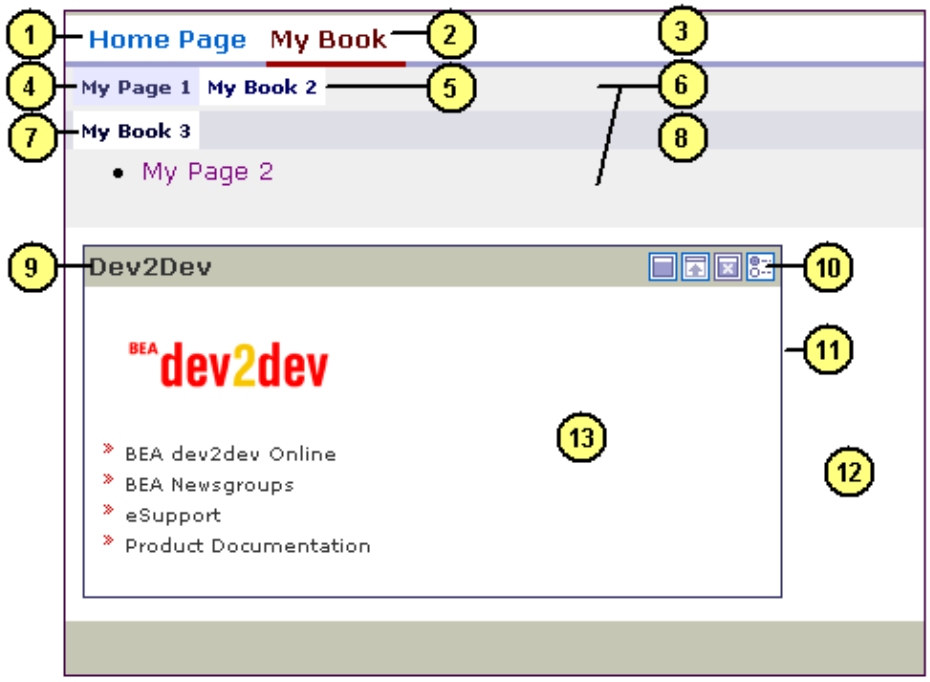
Style Sheet Class Reference

Skeleton Reference

Style Sheet Class Reference

The following figure points to some key pieces of a desktop that are controlled by CSS classes. The tables that follows identify the style classes used. For each piece, the style class controls all aspects, including font color, background color, border, padding, and so on.

The style classes listed occur in multiple places in the CSS file. For example, .bea−portal−book−primary−menu−single, .bea−portal−book−primary−menu−single a, and .bea−portal−book−primary−menu−single span are grouped to share a specific set of attributes, yet a standalone entry for .bea−portal−book−primary−menu−single a exists to let you apply style attributes to that class only. When you modify one of the classes listed, determine whether you want the attributes to be shared among multiple classes or used by only that one class. Then modify or add the attribute(s) you want in the appropriate location in the CSS file.



The style classes used in the illustration change depending upon the navigation type of the book at each level. The figure shows a mix of books with single−level menus and books with multi−level menus.

If the book is set to Single−Level Menu

	Description	*Style(s)*	*Style sheet*
1	Top−level unselected menu item	.bea−portal−book−primary−menu−single a	book.css
	Rollover on top−level unselected menu item	.bea−portal−book−primary−menu−single a:hover	book.css
2	Top−level selected menu item	.bea−portal−book−primary−menu−single span	book.css
3	Top−level menu background	.bea−portal−book−primary−menu−single	book.css
4	First−level unselected menu item	.bea−portal−book−menu−single a	book.css
	Rollover on first−level unselected menu item	.bea−portal−book−menu−single a:hover	book.css
5	First−level selected menu item	.bea−portal−book−menu−single span	book.css
6	First−level menu background. Also applies to the background that displays sub−books and pages.	.bea−portal−book−single	book.css
7/8	Second−and−higher−level selected menus use the same style classes as the first−level menus.		book.css
9	Portlet titlebar	background: .bea−portal−window−titlebar title: .bea−portal−window−titlebar−title	window.css
10	Portlet titlebar icons	These are not determined by style classes. They come from the skin's /images directory. This information is presented for convenience.	N/A
11	Portlet border	.bea−portal−window	window.css
12	Page background	.bea−portal−book−page	book.css
13	Portlet content (mainly for padding) The content in the portlet itself (JSP, HTML, or JPF) are not skin−related and controls the styles used within itself.	.bea−portal−window−content	window.css

If the book is set to Multi−Level Menu

	Description	*Style(s)*	*Style sheet*
1	Top−level unselected menu item	.bea−portal−book−primary−menu−root a	book.css
	Rollover on top−level unselected menu item	.bea−portal−book−primary−menu−root a:hover	book.css

2	Top−level selected menu item	.bea−portal−book−primary−menu−root a	book.css
3	Top−level menu background	.bea−portal−book−primary−menu−root	book.css
4	Sub−level menu items in a drop−down list from the top−level menu (not shown in the figure)	.bea−portal−book−primary−menu−nested−item a	book.css

Different combinations of menus result in different styles used that are not listed. For example, if a book with a multi−level menu is nested in a book with a single−level menu, the drop−down list for the nested books and pages in the multi−level menu is controlled by .bea−portal−book−menu−root a:hover; and if the same book contains only pages, resulting in no drop−down list, the styles for the menu items are controlled by .bea−portal−book−menu−root a.

Troubleshooting the Styles

If you do not find the styles you need to modify in the previous tables, use the following guidelines.

- Style classes ending with space−a (" a") control clickable menu items.
- Style classes ending with a:hover control rollover styles.
- Style classes ending with space−span (" span") usually control selected menu items.
- Style classes ending with space−ul or space−li (" ul" or " li") control unordered lists and list items, particularly their inline horizontal placement rather than vertical placement.
- Base style classes usually control the menu background.
- Style classes inserted by the skeleton JSPs that do not have corresponding entries in a CSS file are ignored.

Another way to pinpoint the styles you need to modify is to view the HTML source and look at the styles used, as shown in the following example:

```
<div class="bea−portal−ie−table−buffer−div">
    <table border="0" cellpadding="0" cellspacing="0" width="100%">
        <tr>
            <td class="bea−portal−book−primary−menu−single−container" align="left" nowrap="
<ul

    class="bea−portal−book−primary−menu−single"

><li class="bea−portal−book−primary−menu−single−item"><a href="http://localhost:7001/sample
            </td>
```

In this example, the unordered list <ul>, which corresponds to a list of menu items, controls the menu styles. The style class used is bea−portal−book−primary−menu−single. However, the context of the menu's sub−elements determines the more fine−grained styles used. For example, if a menu item is linked (using an <a> tag), the style class you would modify to control the linked menu items is bea−portal−book−primary−menu−single a. If an active menu item was wrapped in a tag, the style class you would modify is bea−portal−book−primary−menu−single span.

Skeleton Reference

The following table lists portal components and shows which skeletons are used to render each.

This portal component...		***to:***

	uses this skeleton JSP...	
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.
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.
Navigation Menu	submenu.jsp	Used by multilevelmenu.jsp to create a book's navigation links. Also provides rendering for nested books and pages.
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	placeholder.jsp	Render an individual placeholder in a Layout.

in the .layout file		
Portlet titlebar	titlebar.jsp	Render a portlet titlebar.
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	buttondelete.jsp	Render a button that removes a portlet from a page.
Portlet	error.jsp	Display error messages in a portlet.
Portlet	webflowportlet.jsp	Render 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.

Related Topics

How the Shell Determines Header and Footer Content

How Portal Components are Rendered

Creating Look & Feel Files

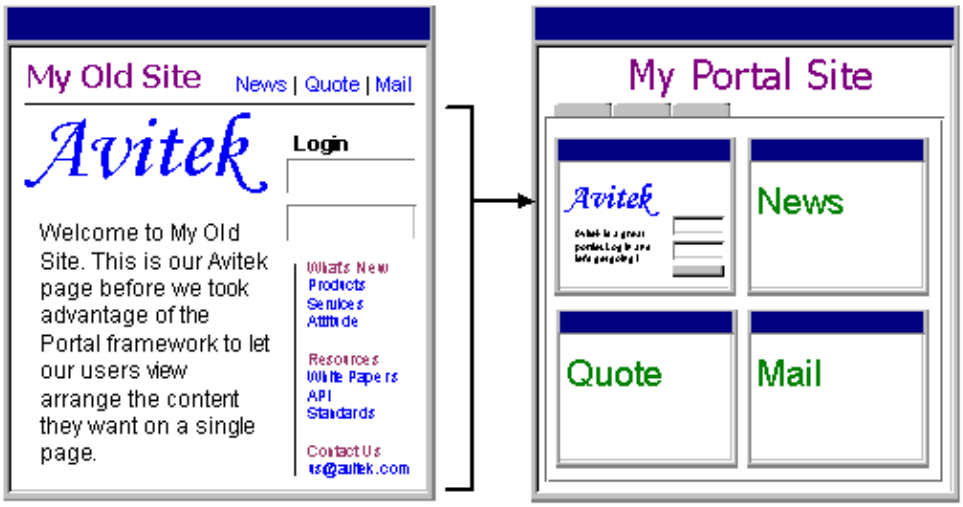
Creating Skins and Skin Themes

Creating Skeletons and Skeleton Themes

Designing a Portal User Interface

The WebLogic Workshop Portal Extensions let you surface your applications in portlets and arrange those portlets on pages (that use tabs or other navigation functionality) within a portal. Portals can use different looks and feels, and you can let users customize their portal desktops by adding their own pages, adding and arranging portlets, and changing the look and feel of the desktop.

The following illustration shows a JSP application before and after it is surfaced in a portal interface. The JSP application that took up an entire browser window can be displayed in a portlet, along with other applications.



Displaying applications in a portal environment is flexible and powerful, but it also potentially reduces the amount of screen real estate available for your applications (moving from an entire browser window to a Portlet window), especially when mobile devices access your portals.

Portlet real estate depends on the page layout used and the number of portlets on a page. For example, a page layout can use a single table cell to render a single large portlet, or a layout can contain a number of rows and columns and render multiple, smaller portlets. So JSP and HTML design must account for that reduced real estate. For example, the content or functionality of a single JSP or HTML page may need to be divided between two or more portlets, which requires the creation of more JSPs or HTML pages.

In addition to basic portal design considerations, you may also want to consider accessibility design issues. For more information, see Building User Interfaces to Address Accessibility Guidelines.

To learn more about building portal interfaces, see Developing Portal User Interfaces.

Building User Interfaces to Address Accessibility Guidelines

This topic contains information on the following subjects:

- Accessibility Standards and the Internet
- Accessibility Checkpoints
- Industry Guidelines
- Government Regulations and Standards
- Accessibility Evaluation and Testing Tools

Accessibility Standards and the Internet

The Internet provides you with the ability to communicate with a diverse audience from a single point via a portal or web site. Many organizations are required to provide web sites that meet industry or government standards for supporting people with special needs. And even if you do not have specific requirements, it is just good business to design your site to serve the needs of a diverse audience.

WebLogic Portal provides a flexible architecture that supports the design, development, and management of accessible portals and applications, for example, the ability to target specific user interfaces based on user preferences or browser and request attributes.

To learn more about building user interfaces with WebLogic Portal see Creating Look and Feels.

Accessibility Checkpoints

When you develop web sites, you can use the following general guidelines to facilitate accessibility. For a complete list, refer to the industry or government regulation relevant for your implementation.

Text Tags
Provide a text equivalent for every non−text element (for example, via "alt", "longdesc", or in element content).

Multimedia Presentations
Synchronize equivalent alternatives for any multimedia presentation.

Color
Design web pages so that all information conveyed with color is also available without color. (for example, from context or markup.)

Readability (style sheets)
Organize documents so they are readable without requiring an associated style sheet.

Server−Side Image Maps
Provide redundant text links for each active region of a server−side image map.

Client−Side Image Maps
Provide client−side image maps instead of server−side image maps except where the regions cannot be

defined with an available geometric shape.

Data Table (simple row and column headers)
Identify row and column headers for data tables

Data Tables (multiple levels of row and column headers)
Use markup associate data cells and header cells for data tables that have two or more logical levels of row or column headers.

Frames
Entitle frames with text that facilitates frame identification and navigation.

Flicker Rate
Design pages to avoid causing the screen to flicker with a frequency greater than 2 Hz and lower than 55 Hz.

Text−Only Alternative
Provide atext−only page, with equivalent information or functionality, to make a web site comply with the provisions of this section when compliance cannot be accomplished in any other way. You must remember to update the content of the text−only page whenever the primary page changes.

Scripts
When pages use scripting languages to display content, or to create interface elements, you must identify the information provided by the script with functional text that can be read by assistive technology.

Applets and Plug−ins
When a web page requires that an applet, plug−in or other application be present on the client system to interpret page content, you must provide a link in the page to a plug−in or applet that complies with Section 508 §1194.21(a) through (l).

Electronic Forms
When electronic forms are designed to be completed on−line, you must create the form to allow people using assistive technology to access the information, field elements, and functionality required for completion and submission of the form, including all directions and cues.

Navigation Links
Provide a method that permits users to skip repetitive navigation links.

Time Delays
When a timed response is required, you must alert users and give them sufficient time to indicate more time is required.

Industry Guidelines

W3C Web Content Accessibility Guidelines

- http://www.w3.org/WAI/w3c.htm − Checklist of Checkpoints for Web Content Accessibility Guidelines 1.0
- http://www.w3.org/TR/WCAG10/full−checklist.html

Government Regulations and Standards

UNITED STATES

Section 508
http://www.section508.gov

CLF Accessibility best practices
http://www.cio−dpi.gc.ca/clf−upe/1/1d_e.asp

CANADA

Adaptive Computer Technology Training Center (Canada)
http://www.cio−dpi.gc.ca/clf−upe/standards/1−1/references/references_e.asp

UNITED KINGDON

Guidelines for U.K. Government Web Sites
http://www.web−access.org.uk/

Accessibility Evaluation and Testing Tools

These tools allow you to validate web page code. They do not repair your code, but they do provide reports on what does and doesn't need to be fixed, as relating to HTML 4.0, W3C, Section 508 and general Accessibility issues.

W3C Web Accessibility Initiative

- W3C Web Accessibility Initiative's Web Tools Page − Evaluation, Repair, and Transformation Tools for Web Content Accessibility
  http://www.w3.org/WAI/ER/existingtools.html
- W3C HTML Validator − The W3C HTML Validation Service checks HTML documents for conformance to W3C HTML and XHTML recommendations and other HTML standards.
  http://validator.w3.org/
- CSS Validator − If you are using Cascading Style Sheets (CSS), then you should use the CSS Validator. http://jigsaw.w3.org/css−validator/

For more information, visit the W3C's Evaluation & Repairs Tools page.
http://www.w3.org/TR/2000/WD−AERT−20000426

Bobby

The Bobby tool identifies W3C accessibility issues by priority level and Section 508 issues.
http://bobby.watchfire.com/bobby/html/en/index.jsp

Note: The free version of Bobby allows you to test one page at a time.

Lynx Viewer

The Lynx Viewer generates an HTML page that indicates how much of the content of your page would be available to Lynx, which is a text−only browser. In addition to showing how useful a site would be for a visually−impaired person, it is also a good indicator for anyone with older technology.
http://www.delorie.com/web/lynxview.html

Creating a Portal Application and Portal Web Project

To create the necessary resources for portal development, you must do one of two things:

- Option 1: Create a new portal application and add a Portal Web Project to it

  or
- Option 2: Install Portal into an existing application and add a Portal Web Project to it

Following are the procedures for each option.

Option 1: To create new portal application and add a Portal Web project to it

Use this procedure to create a new portal application.

You do not need to perform these steps if you are developing on a shared domain and the portal application has already been created and stored in a version−control system. Simply synchronize to the current version of the domain to put the portal application on your machine.

1. If you have not yet created a portal domain on your development machine, create one with the Configuration Wizard. For instructions, see the Overview of Platform Configuration on the BEA's e−docs Web site.

   Performing this step ensures you have a server (config.xml) for your portal application to use, as described later in this procedure.
2. Create a new portal application. In WebLogic Workshop Platform Edition, choose ***File −−>New −−>Application***.
3. In the New Application window, select ***Portal Application*** in the right pane.
4. In the ***Directory*** field, click ***Browse*** to set the location of the new application. The application will be created in a subdirectory of the directory you select.
5. Make sure the ***Name*** field contains the name of the application. This name will be the application directory.
6. In the ***Server*** field, click ***Browse*** and select the config.xml file for the server (domain) you want to use.

   The config.xml file is in the portal domain directory you created.
7. Click ***Create***. The application directory appears in the Application window. The application contains the WebLogic Administration Portal (contained in Modules/adminPortal.war), a datasync directory (data) for interaction management development, and application−level EJBs and APIs.
8. Create a portal Web project for your application. Right−click the <***app_name***> directory in the Application window, and choose ***New −−>Project***.
9. In the New Project window, select ***Portal Web Project*** in the right pane.
10. In the ***Project name*** field, enter the name for the portal Web project. This will be the name of a Web application directory.
11. Click ***Create***. The project folder appears in the Application window. The portal Web project contains WebLogic Portal JSP tags, Web−application−level APIs, and default portal framework files.
12. If you have any external projects or files you want to include in your portal application, perform any of the following steps:
    - To import a project, right−click the <***app−name***> directory in the Application window and choose ***Import Project***. In the Import Project window, select the type of project to import,

browse to select the project folder, and click ***Import***.

- To import files, such as existing datasync files (User Segments, Campaigns, Placeholders, and so on) or the Workshop Portal Extensions sample portlets to use in your portals , right−click the appropriate directory in the Application window and choose ***Import***. In the Import Files window, select the directory or files you want to import, and click ***Import***.

  The sample portlets are located in `<BEA_HOME>\<WEBLOGIC_HOME>\samples\portal\portalApp\sampleportal\port`
  There are other useful sample files throughout the `<BEA_HOME>\<WEBLOGIC_HOME>\samples` directory. See the instructions in Portal Samples for more information.

You now have the resources and directories for developing personalized applications and creating portals to surface applications.

13. Start your development server. In WebLogic Workshop, choose ***Tools−−>WebLogic Server−−>Start WebLogic Server***. The server you assigned to your application in the previous steps starts. All your work is deployed automatically on your machine as you develop.

Option 2: To install portal in an existing application and add a Portal Web project to it

Use this procedure to add portal services to an existing application.

You do not need to perform these steps if you are developing on a shared domain and the portal−enabled application has already been created and stored in a version−control system. Simply synchronize to the current version of the domain to put the portal application on your machine.

1. In WebLogic Workshop Platform Edition, open the application in which you want to install portal.
2. In the Application window, right−click the <***app_name***> directory and choose ***Install−−>Portal***. WebLogic Workshop adds the WebLogic Administration Portal (contained in Modules/adminPortal.war), a datasync directory (data) for interaction management development, and application−level EJBs and APIs.
3. Create a portal Web project for your application. Right−click the <***app_name***> directory in the Application window, and choose ***New−−>Project***.
4. In the New Project window, select ***Portal Web Project*** in the right pane.
5. In the ***Project name*** field, enter the name for the portal Web project. This will be the name of a Web application directory.
6. Click ***Create***. The project folder appears in the Application window. The portal Web project contains WebLogic Portal JSP tags, Web−application−level APIs, and default portal framework files.
7. If you have any external projects or files you want to include in your application, perform any of the following steps:
   - To import a project, right−click the <***app−name***> directory in the Application window and choose ***Import Project***. In the Import Project window, select the type of project to import, browse to select the project folder, and click ***Import***.
   - To import files, such as existing datasync files (User Segments, Campaigns, Placeholders, and so on) or the Workshop Portal Extensions sample portlets to use in your portals , right−click the appropriate directory in the Application window and choose ***Import***. In the Import Files window, select the directory or files you want to import, and click ***Import***.

     The sample portlets are located in `<BEA_HOME>\<WEBLOGIC_HOME>\samples\portal\portalApp\sampleportal\port`
     There are other useful sample files throughout the `<BEA_HOME>\<WEBLOGIC_HOME>\samples` directory. See the instructions in Portal

Samples for more information.

You now have the resources and directories for developing personalized applications and creating portals to surface applications.

8. Start your development server if it is not already running. In WebLogic Workshop, choose ***Tools−−>WebLogic Server−−>Start WebLogic Server***. All your work is deployed automatically on your machine as you develop.

Related Topics

Portal Samples

Developing Portal Applications

How Do I: Create a New Application?

The WebLogic Workshop Development Environment

Creating a Portal File

The first step in creating a Portal is creating a Portal file in the Portal Designer. The file, which represents a portal desktop, appears in the Portal Designer as a graphic representation of your desktop, letting you drag and drop books, pages, and portlets and set properties to construct your portal. The .portal file that is created, along with any .portlet files for corresponding portlets added to the portal, supply all the configuration information the portal framework needs to perform the rendering process.

The .portal files you create serve as templates that portal administrators use to create new desktops in the WebLogic Administration Portal.

1. Create a portal application.
2. With your portal application open in WebLogic Workshop Platform Edition, right−click a portal Web project in the Application window and choose ***New −−> Portal***.
3. In the New File window, enter the name of the portal file in the ***File name*** field. Be sure to keep the .portal extension. Click ***Create***.

   The new portal desktop appears in the Portal Designer with a default header, body, and footer. The main book contains a page.
4. Add books and pages to the Portal Desktop.
5. Apply layouts to pages.
6. Set up navigation for the Portal Desktop.
7. Select the default header and footer you want the desktop to use.
8. Add portlets to your pages.
9. Change the Look & Feel of your portal desktop.
10. In the Document Structure window, select each component and set its properties in the Property Editor window. See Properties for Portal Components for guidance.
11. Save the portal file.
12. To view your Portal in a browser, choose ***Portal−−>Open current portal***.

Samples

See the Portal Samples for instructions on viewing a sample portal file.

Related Topics

The Portal User Interface Framework

Tutorial: Building Your First Portal

Adding Books and Pages to a Portal

Setting up Navigation

Adding Portlets to Pages

Changing Look & Feel

Deploying Portal Applications

Creating Look & Feels

Look & Feels are a combination of interrelated parts that control the physical appearance of your portals. The following topics describe the Look & Feel architecture and show you how to create and use Look & Feels in your portals.

Look & Feel Architecture

Describes the architecture behind Look & Feel components that determine the appearance of portal desktops.

Creating Look & Feel Files

Provides instructions on creating Look & Feel files that reference different skins and skeletons.

Creating Skins and Skin Themes

Provides instructions for creating the styles, graphics, and and JavaScript behavior of skins and themes for use in desktop Look & Feels.

Creating Skeletons and Skeleton Themes

Provides instructions for creating skeletons used as part of Look & Feels to render portal desktops

Creating Layouts

Provides instructions for creating page layouts for placing books and portlets in portal desktops.

Creating Shells

Provides instructions for creating shells and using them to add content to portal desktop headers and footers.

Modifying Navigation Menus

Provides instructions for creating custom navigation menus for books and pages.

Related Topics

The Portal User Interface Framework

Developing Portal Applications

WebLogic Portal Overview

Look & Feel Architecture

WebLogic Portal's Look & Feel architecture provides flexible, powerful framework for determining the appearance of your portals. The following sections describe in general terms the different resources of a portal Look & Feel and the relationships among them.

The Look & Feel architecture involves the following resources:

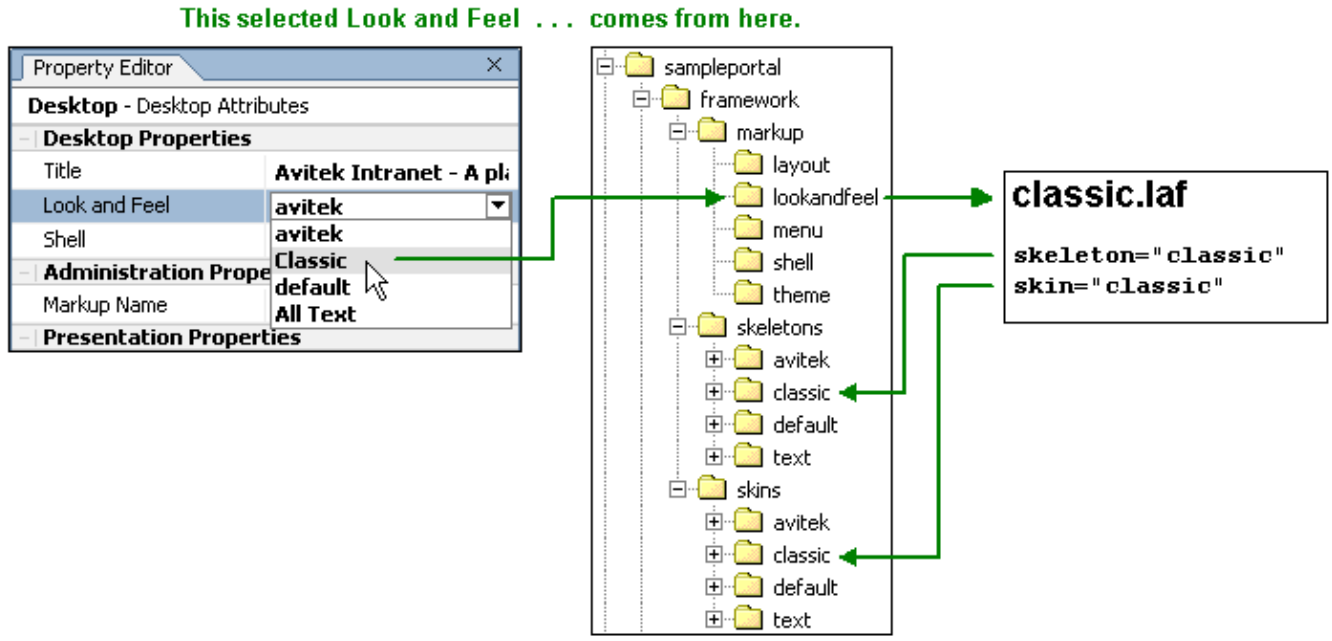
- Look & Feel Files, Skins, Skeletons, and Themes
- Layouts
- Navigation Menus
- Shells

Look & Feel Files, Skins, Skeletons, and Themes

A portal Look & Feel determines the physical appearance of a portal. It includes two main elements:

- ***Look (skins)*** − The graphics, cascading style sheets (CSS), and JavaScript files used in a skin.
- ***Feel (skeletons)*** − The skeleton JSP files that determine the physical boundaries of portal components such as header, footer, book, page, portlet, and portlet title bar.

A portal Look & Feel stems from a single XML file (with a .laf extension). The .laf file determines the name of the Look & Feel and points to the skeleton and skin that the Look & Feel will use, as shown in the following illustration. This simple, extensible framework makes it easy for portal administrators and end users to completely change the appearance of a portal by selecting a different Look & Feel.



You can apply subsets of skins and skeletons to books, pages, and portlets. These subsets are called ***themes***. Themes let you give individual books, pages, and portlets a different Look & Feel than the rest of the portal desktop. When you select a theme for a book, page, or portlet, the portal framework looks for theme subdirectories under the main skin and skeleton directories used by the selected Look & Feel.

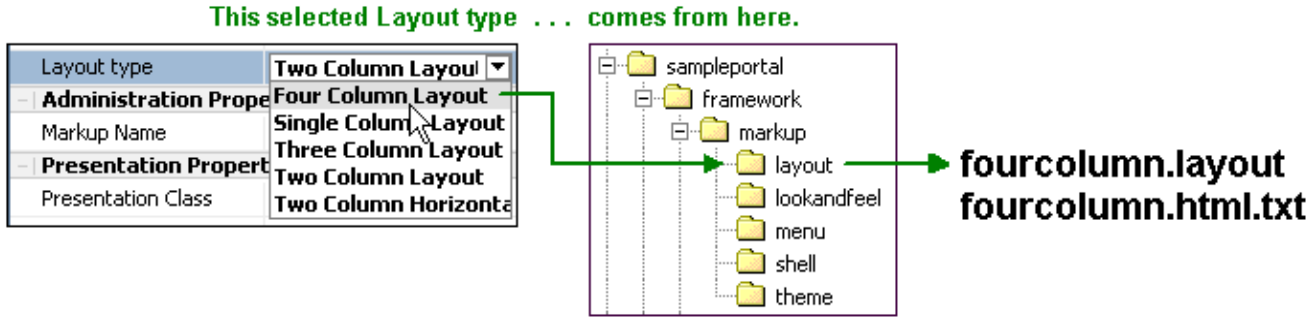
For example, if a desktop Look & Feel uses the /framework/skeletons/default skeleton and you select a theme called "modern" for a portlet, the portal framework looks in the /framework/skeletons/default/modern directory for skeleton JSPs to render just that portlet. Other non−themed portlets are rendered with skeleton JSP files in the /framework/skeletons/default directory.

Layouts

Layouts define the placeholders (rows and columns) of a portal page in which portlets can be placed. Layouts also determine whether books and portlets are placed on top of each other (vertical) or beside each other (horizontal) in a placeholder.

A Layout is made up of two files: an XML file (with a .layout extension) and an HTML file (with a .html.txt extension), as shown in the following illustration.

The .layout file is the actual layout structure rendered. The .html.txt file is used to simulate the layout in the WebLogic Workshop Portal Extensions Portal Designer and in the WebLogic Administration Portal.



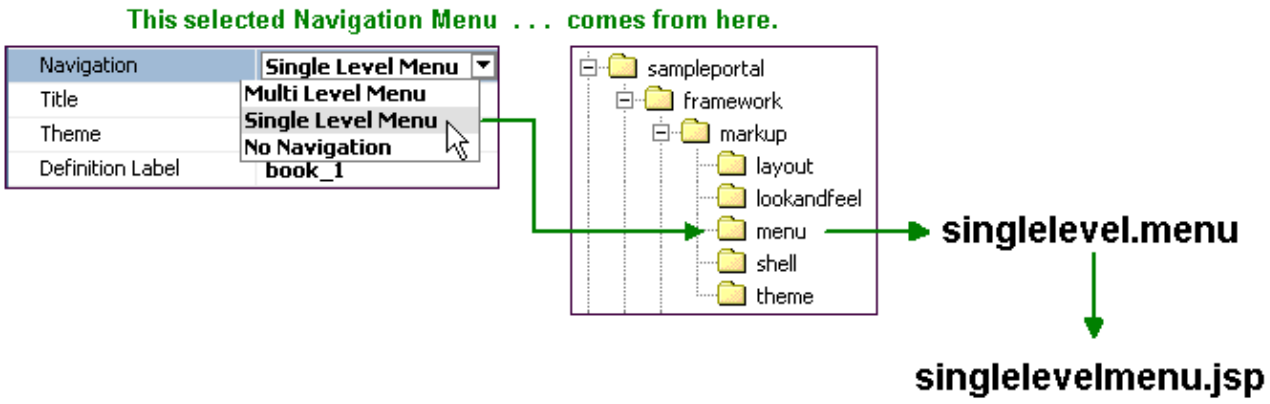
Layouts also have corresponding skeleton JSPs that render them. The desktop's selected Look & Feel determines which skeletons are used. The Creating Skeletons and Skeleton Themes topic contains a table that shows which skeleton JSPs are used.

Navigation Menus

Navigation Menus are used to navigate among books and pages in a portal. WebLogic Portal supplies the following default Navigation Menu styles:

- ***Single Level Menu*** − Provides visible layering of 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 page links for the top−level pages. Any sub−books and pages appear in a drop−down list for selection. The Multi Level Menu implements JavaScript functionality contained in the skins.
- ***No Navigation*** − No navigation is provided.

A Navigation Menu is made up of an XML file (with a .menu extension), as shown in the following illustration. The menu file references a menu class that determines the menu's behavior, and the menu class references the skeleton JSP file that renders the menu.

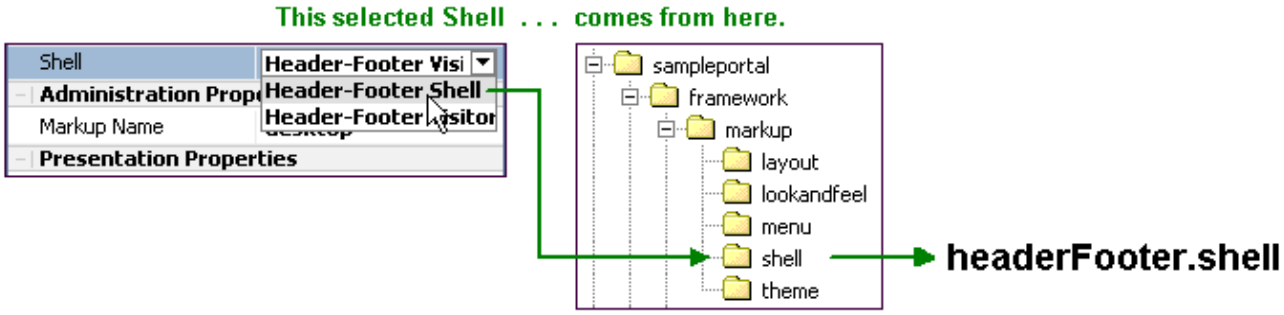


Shells

A shell defines the area around the books and pages of a portal. When a portal is rendered, the shell creates the HTML opening and closing <html>, <head>, and <body> tags and creates the header and footer regions above and below a portal's books and pages.

In the header and footer elements of the shell, you can reference JSPs that display content in the header or footer.

A Shell is made up of an XML file (with a .shell extension), as shown in the following illustration.



Shells also have corresponding skeleton JSPs that render them. The Creating Skeletons and Skeleton Themes topic contains a table that shows which skeleton JSPs are used.

Samples

The Portal Samples contain portals that use a default set of Look & Feel resources. See the section in that topic on "Viewing the Samples" for instructions on opening the portals in WebLogic Workshop Platform Edition to see how the Look & Feel resources are used. Run the Tutorial Portal in a browser to view different Look & Feel implementations.

Also, when you add a Portal Web Project to an application in WebLogic Workshop Platform Edition, default Look & Feel resources are included with the project in the <project>\framework directory.

Related Topics

The Portal User Interface Framework

Creating Look & Feels

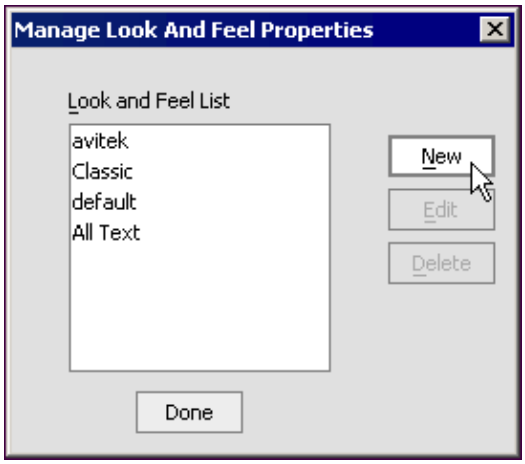
Portal Overview

Creating Look & Feel Files

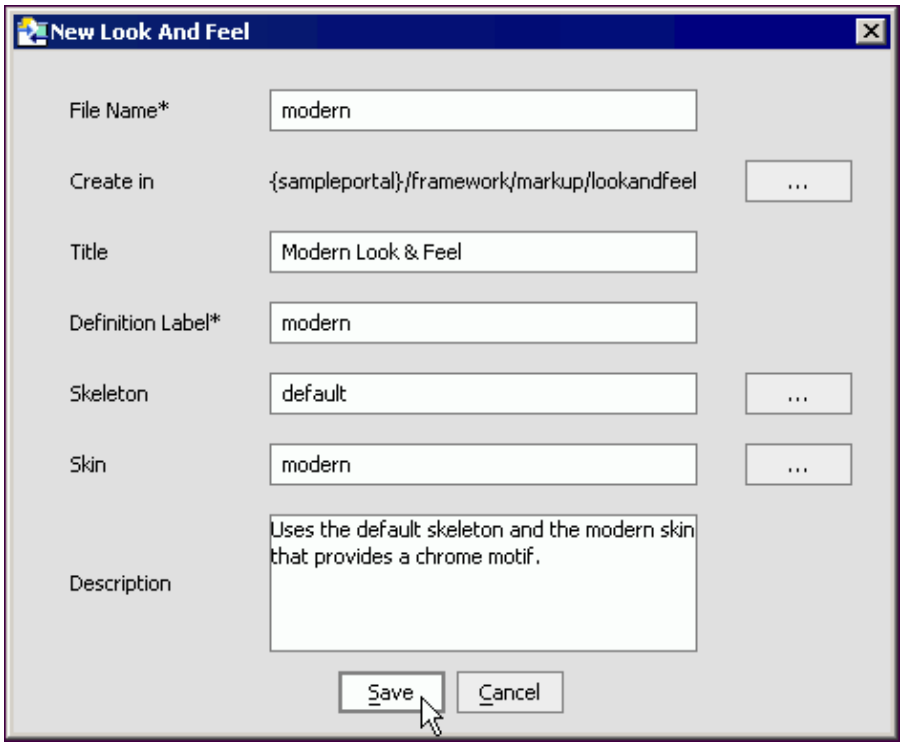
A Look & Feel file is a simple XML file that determines the skin and skeleton used for the desktop Look & Feel. When you create a Look & Feel file, you can select the new Look & Feel in the Portal Designer for your portal desktops.

To create a Look & Feel file

1. In WebLogic Workshop, with your portal application open, open any .portal file in the Portal Designer.
2. In the WebLogic Workshop menu, choose ***Portal*** **>** ***Manage Look and Feel Properties***. The Manage Look and Feel Properties window appears, showing a list of existing look & feels.



3. Click ***New***. The New Look and Feel window appears.



4. Enter values for the fields using the following table for guidance:

File Name	Enter a filename for the look and feel. A .laf extension is added automatically.
Create in	Leave the default value. The only time you should change this value is if you want to create the look & feel in a different portal Web project than the one shown. The path must always be <project>/framework/markup/lookandfeel.
Title	Enter a title that will appear in the look & feel drop−down list.
Definition Label	Enter a unique label name. Each look & feel in a portal Web project must have a unique definition label. Definition labels are used for setting delegated administration.
Skeleton	Click the ellipsis icon [...] and select the skeleton directory you want the look & feel to use. In many cases, you may just want to use the "default" skeleton stored in <project>/framework/skeletons/default. The directory you select must be located in <project>/framework/skeletons/. If you do not provide a skeleton, the skeleton identified in the skin's skin.properties file is used. See Creating Skins and Skin Themes.
Skin	Click the ellipsis icon [...] and select the skin directory you want the look & feel to use. The directory you select must be located in <project>/framework/skins/.
Description	Enter an optional description of the look & feel.

5. Click ***Save***. In the Manage Look and Feel Properties window, click ***Done***.

   Note: You can also set the default icon to be used in portlet titlebars by opening the .laf file in WebLogic Workshop and adding ***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/"
6. To use the Look & 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 & 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 key to Look & 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 skin resources (such as images, CSS files, and JavaScript files) must be referenced correctly in your JSP skeleton files. For example, the icons for the portlet title bars must use the correct graphics names.

Samples

The WebLogic Workshop Portal Extensions include a set of predefined Look & Feel files. You can find these predefined files in the sampleportal project at <BEA_HOME>/weblogic81/samples/portal/portalApp/sampleportal/framework/markup/lookandfeel. The predefined Look & Feels are also included with any Portal Web Project you add to a portal application. The files are located in <project>/framework/markup/lookandfeel.

The Tutorial Portal contains examples of Look & Feel implementations that let end users change a portal's Look & Feel. For instructions on viewing the Tutorial Portal, see Portal Samples.

Related Topics

How Look & Feel Determines Rendering

Creating Look & Feels

Creating Skins and Skin Themes

Creating Skeletons and Skeleton Themes

Changing Look & Feel

What is a Portal?

Creating a Portal File

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 & Feel. When you select a Look & Feel for a portal desktop, the Look & 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.

This topic contains the following procedures:

- To create a skin
- To create a skin theme

To create 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 modify filenames without also modifying the names where they appear in your JSPs or skeleton JSPs.
5. In the ***css*** subdirectories, add any new CSS files you have created. Do not rename any of the default styles or CSS files provided by BEA. The styles 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, style sheet 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.

   Note: You can also create a file called skin_custom.properties in the same directory as skin.properties. Use skin_custom.properties to list custom CSS and JavaScript files you create or to override the

settings in skin.properties. Creating a skin_custom.properties file is useful for custom entries, because you can replace skin.properties with new product releases without overwriting your customizations.
8. Open or create a Look & Feel file and associate the skin with the Look & Feel. See Creating Look & 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 & Feel in the Property Editor ***Look and Feel*** field.

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

To create 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."

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	Required. Tells the portal framework the name of the theme directory to look in to apply theme resources to the book, page, or portlet.
title	Required. Used to populate the Theme drop−down list where it appears in the book, page, and portlet properties in the Portal Designer and in the WebLogic Administration Portal.
description	Optional description of the theme.
markupName	Required. The markupName must be unique among the other themes in the portal Web project. For best practices, markupName should be the same as the name.

   Note: The markupType attribute for themes must always be "Theme".
4. Save the theme file.

Now create the theme resources.

5. In a skin directory, create a subdirectory with the same name as the theme.
6. Determine which (if any) files from the skin you want to modify for the theme, and copy them into the new theme directory, retaining any subdirectory structure.
7. Make the necessary modifications to the theme resources. (Do not modify filenames without also modifying the names where they appear in either skin.properties or in your JSPs or skeleton JSPs.)
8. In the skin.properties file for all skins that will use the theme, reference the theme in the "THEME IMAGES DIRECTORIES" section of the file with the following type of entry:

   theme.***alert***.search.path: alert/images, images

Note that the name of the theme is the second item in the property name.
9. Save all modified files.
10. If you want other skins to be able to use the theme, copy the theme directory into those skin directories and add appropriate entries in each skin.properties "THEME IMAGES DIRECTORIES" section.

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 & 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.

Creating Your Own Style Sheet Classes

If you want to add your own CSS styles to a skin, do the following:

1. Instead of adding your own styles to the BEA−provided CSS files, where the files are subject to possibly being overwritten by product updates, create your own CSS files, even if you want to add only a few classes.
2. Add your CSS paths and filenames to the "Link Entries" section of the skin_custom.properties files of any skins in which you want to use your styles. You may need to create the skin_custom.properties file, in the same directory as skin.properties. Doing this lets you use CSS styles in any JSP or HTML that will be used in the portal, because the CSS reference is automatically inserted into the rendered HTML from the skin.properties file.

   Note: You could also add your custom CSS entries to skin.properties, but future product releases could overwrite the skin.properties files.
3. If you want to use your styles in any skeleton JSP files to render portal components, be sure to add those styles inline in the JSPs where appropriate. Open an existing skeleton JSP for guidance to see how it implements styles.

About Portlet Titlebar Icons

The icons used in portlet titlebars are stored in a skin or theme /images directory. The portal framework reads the portal Web project's WEB−INF/netuix−config.xml file to determine which of these graphics to use for the portlet's different states and modes (minimize, maximize, help, edit). If you want to change the name of the graphics used for the portlet titlebar icons, change the filenames and the corresponding entries for those graphics in netuix−config.xml.

Samples

The WebLogic Workshop Portal Extensions have sample portals that contain skin and theme files you can view in WebLogic Workshop Platform Edition. For instructions on viewing the sample portals, see Portal Samples.

Also, when you create a Portal Web Project, a predefined set of skins and themes is added to your project in <project>\framework\skins and <project>\framework\markup\theme.

Related Topics

How Look & Feel Determines Rendering

Look & Feel Architecture

Creating Look & Feel Files

Creating Skeletons and Skeleton Themes

Creating Portals for Mobile Devices

Creating 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 & Feel. When you select a Look & Feel for a portal desktop, the Look & Feel points to the skeletons and skins to use.

This topic contains the following procedures and information:

- When should you create a skeleton?
- To create a skeleton
- To create a skeleton theme
- How portal components map to skeletons

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).

To create a skeleton

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 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 & Feel file. See Creating Look & Feel Files.

   Also, 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 & 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). See Creating Shells and How Do I: Personalize a Desktop Header or Footer?

To create 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 & Feel selected for the desktop references a skeleton that does not use the selected theme, no theme is used.

How portal components map to skeletons

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.
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.
Navigation Menu	submenu.jsp	Used by multilevelmenu.jsp to create a book's navigation links. Also provides rendering for nested books and pages.
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.
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.

Samples

See Portal Samples for instructions on viewing a sample portal file.

Related Topics

How Look & Feel Determines Rendering

Look & Feel Architecture

Creating Look & Feel Files

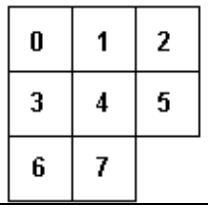
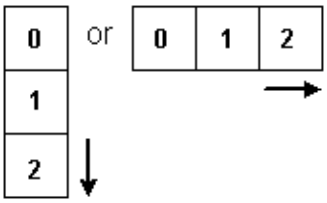
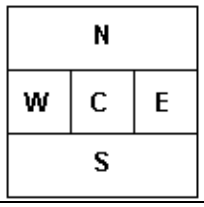
Creating Skins and Skin Themes

Creating Portals for Mobile Devices

Creating 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:

		
Grid Layout 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.	***Flow Layout*** The flow layout automatically positions the number of placeholders used either vertically or horizontally with no wrapping.	***Border Layout*** The border layout lets you use up to five placeholders. You can position the placeholders with the attributes "north," "south," "east," "west," and "center."

For the purposes of creating layouts, a layout includes 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.

To create a standard 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.

For example, if you are modifying a copy of the fourcolumn.layout to create a border layout, replace the columns attribute with the layoutStrategy attribute and change its value.

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 between the opening and closing <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. This file is used to display the layout in the Portal Designer.
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.)

Samples

The WebLogic Workshop Portal Extensions have sample portals that contain layouts you can view in WebLogic Workshop Platform Edition. For instructions on viewing the sample portals, see Portal Samples. In particular, the Tutorial Portal provides sample implementations of each layout style.

Also, when you create a Portal Web Project, a predefined set of layouts is added to your project in <project>\framework\markup\layout.

Related Topics

The Portal User Interface Framework

Look & Feel Architecture

Creating a Portal File

Adding Books and Pages to a Portal

Changing a Page Layout

Setting up Navigation

Building Portlets

Changing the Header and Footer

Modifying 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.

To modify one of the default navigation menus, do the following:

- Modify the navigation menu file
- Modify the skeleton JSP file

To modify the navigation menu file

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
   <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.
   - Other attributes:

     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.

To modify the skeleton JSP file

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.)

Samples

See Portal Samples for instructions on viewing a sample portal file.

Related Topics

Creating Skeletons and Skeleton Themes

The Portal User Interface Framework

Look & Feel Architecture

Setting up Navigation

Creating 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.

To create a shell

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>
    <netuix: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 , <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:

```
<netuix:header>
<netuix:jspContent contentUri="/my_jsps/campaign_header.jsp"
errorUri="/my_jsps/my_error.jsp"
backingFile="custom.portal.backing.campaignBacking" />
</netuix:header>
```

5. Save the shell file.
6. 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.

Samples

See Portal Samples for instructions on viewing a sample portal file. The Sample Portal also includes an example of a shell that creates a left navigation frame.

Related Topics

The Portal User Interface Framework

Look & Feel Architecture

Creating a Portal File

Adding Books and Pages to a Portal

Changing a Page Layout

Setting up Navigation

Building Portlets

Changing the Header and Footer

Left Navigation Shell

The left navigation shell demonstrates using a navigation tree in the left frame of a portal desktop.

Concepts Demonstrated by this Sample

This sample uses a shell to create a left table cell that contains a navigation tree of links for the selected book. The main page book is rendered in the right table cell.

This portlet uses functionality similar to the Targeted Menu Portlet. The difference between the two samples is scope. The left navigation shell provides navigation an entire desktop, and the Targeted Menu portlet provides navigation for a single book.

Following is the leftNav.shell file, which shows how the header, footer, and left navigation JSPs are included to provide a left navigation area in the shell:

```
<?xml version="1.0" encoding="UTF-8"?>

<netuix:markupDefinition xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0 markup-netuix-1
    <netuix:locale language="en"/>
    <netuix:markup>
        <netuix:shell
            title="Left Nav Shell" description="A shell with a left nav section." markupType="S
            <netuix:head/>
            <netuix:body>
                <netuix:header>
                    <netuix:jspContent contentUri="/portlets/header/header.jsp"/>
                </netuix:header>
                <netuix:jspContent contentUri="/navigation/shell/beginBody.jsp"/>
                <netuix:break/>
                <netuix:jspContent contentUri="/navigation/shell/endBody.jsp"/>
                <netuix:footer/>
            </netuix:body>
        </netuix:shell>
    </netuix:markup>
</netuix:markupDefinition>
```

Following are descriptions of the JSPs used in the shell:

- header.jsp is included to provide a header only. It is not required for left navigation functionality.
- beginBody.jsp file provides two table cells: one for the left navigation area and the other to contain the main page book of the desktop, represented by the <netuix:break/> tag. The left cell includes a <jsp:include page="/navigation/shell/leftNav.jsp"/> tag to insert the contents of leftNav.jsp in the left navigation cell. leftNav.jsp includes all the navigation functionality.
- endBody.jsp supplies the closing HTML tags for the right table cell, the table row, and the table, wrapping the main page book area within the shell (which is why the JSP is referenced after the <netuix:break/> tag).

Location of Sample Files

This sample is located in the `<BEA_HOME>/<WEBLOGIC_HOME>/samples/portal/portalApp/portalApp.work` application.

How to Run the Sample

See Viewing the Samples in Portal Samples.

How to Use the Sample in Your Portals

1. Create a portal application. Make sure you create a Portal Application and add a Portal Web Project to it.
2. Import or copy the following directories and files into your portal application and portal Web project. You may need to create the appropriate directories in your application:

Import or copy this	***to this directory*** (crea
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/framework/markup/shell/leftNav.shell`	<PORTAL_APP>/<p

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/navigation/shell/`
`beginBody.jsp`
`endBody.jsp`
`leftNav.jsp`
	<PORTAL_APP>/<p

`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/framework/markup/skins/<skin>/css/treenav.css`	<PORTAL_APP>/<p
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/framework/markup/skins/<skin>/images/treenav*.gif`	<PORTAL_APP>/<p `<skin>/images/`
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/framework/markup/skins/<skin>/skin.properties`	If you already have a that you do not want entry for treenav.css into your skin.proper entire sample skin.pr <PORTAL_APP>/<p

3. Open your portal file and navigate the page where you want the portlet to appear.
4. In the ***Data Palette*** window, drag the ***Login to Portal*** portlet onto a placeholder on the page.
5. In the ***Property Editor*** window, set any relevant properties.
6. Save the portal file.
7. View your portal with the WebLogic Test Browser or with your default browser.
   - ***WebLogic Test Browser*** − In the WebLogic Workshop toolbar, click the ***Start*** button (or press ***Ctrl+F5***).
   - ***Default Browser*** − In the WebLogic Workshop menu, choose ***Portal−−>Open Current Portal***.

Related Topics

How the Shell Determines Header and Footer Content

Targeted Menu Portlet

Creating Shells

Creating Skins and Skin Themes

Portal Samples

Building Portlets

Portlets are the basic building blocks of portal Web applications, and enable the presentation behavior of a subset of an application to be managed as a single unit. A portlet exists as a set of associated files, mostly XML and JPSs. In the WebLogic Workshop IDE, portlets can be edited visually in Design View, and the JSPs can be edited in Design View and Source View.

You can think of portlets as the windows that surface your applications, information, and business processes. Portlets can communicate with each other, they can work with Java controls, and take part in Java Page Flows that determine a user's path through an application. You can have multiple portlets on a page. You can also have multiple instances of a single portlet.

Architecturally, a portlet is a collection of objects described by an XML file with the .portlet extension. The framework uses the elements described in that file to render the Portlet at runtime, applying any permissions and customization at specific points in the assembly of the HTML that is eventually generated. The Portlet itself can use a JSP, a Page Flow, or an optional backing file, and can be built to conform to the JSR 168 standard for portlet compatibility. Portlets can consume existing Web applications and content (ASP, JSP, HTML, XML, and so on).

The following topics guide you through the portlet creation process:

Using Portlets from the Portlet Library

To save time and get a head start on your development work, you can use the sample portlets in the Portlet Library.

Creating Portlets

If the portlets in the Portlet Library do not match your current needs, you can develop new portlets in a variety of ways, including through the use of the Portlet Wizard.

Customizing Portlets

After creating your portlets, you can customize them to suit the requirements of your portal application.

Related Topics

Developing Portal Applications

Developing Personalized Applications

Portal Reference

Portal Samples

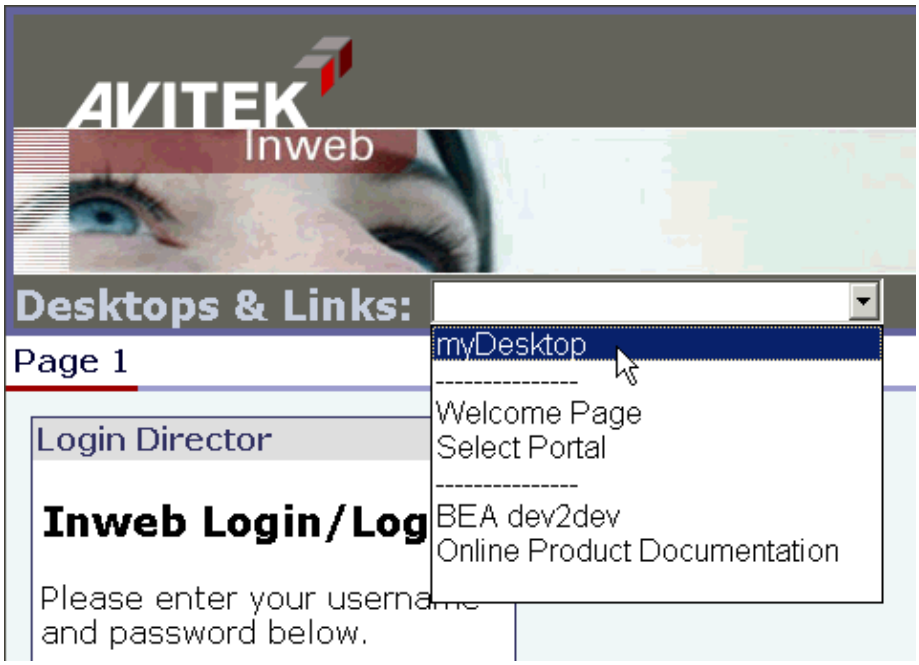
Portal Tutorials

Enabling Desktop Selection

Oftentimes users are entitled to view multiple desktops in your portals. This topic shows you how to let users select from a list of the specific desktops to which they are entitled.

The desktop selection feature is a JSP used by the shell that provides a drop−down list of desktops and links to other resources. Because the desktop selector lets users switch between multiple desktops, it must run in streaming mode where multiple desktops exist. When viewing the feature in single file mode (development), only one desktop is ever available at a time.

The following figure shows the desktop selector in action.



To add the desktop selector to your desktops

The following procedure for adding Visitor Tools assumes you are adding them to a custom portal application (not the portalApp sample). If your application an/or project is not portal−enabled, install portal in both. See Creating a Portal Application and Portal Web Project.

1. Set up some form of authentication for your portal desktop. Authentication allows visitor entitlements to take effect. See Login Portlet, Login Director, or Implementing Authentication for information on adding authentication to your desktops.
2. In WebLogic Workshop create a new portal file.
3. Import the following files from Sample Portal into your application:

Import or copy this	***to this directory*** (create if necessary)
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/portlets/header/header.jsp`	<PORTAL_APP>/<project>/portlets/header/
`<WEBLOGIC_HOME>/samples/portal/portalApp/sampleportal/images/`	<PORTAL_APP>/<project>/images/

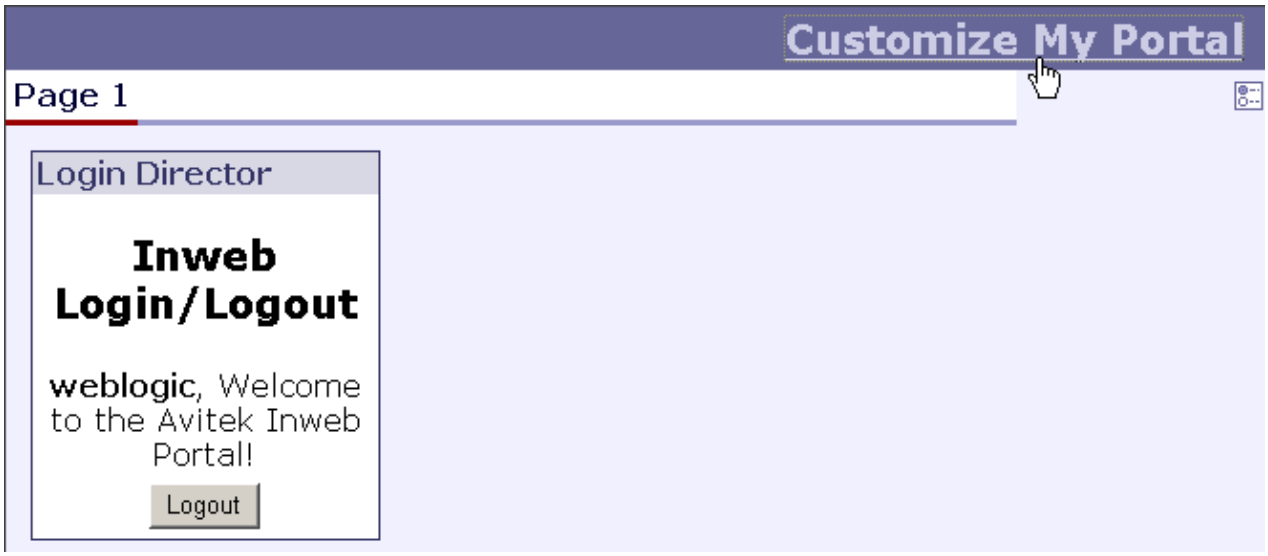
4. Open <PORTAL_APP>/<project>/portlets/header/header.jsp in WebLogic Workshop and replace the string sampleportal with the name of your project.
5. Create a shell and make <PORTAL_APP>/<project>/portlets/header/header.jsp the header content.
6. In a .portal file open in the Portal Designer, select the new shell for the desktop.
7. Save the portal file.

When portal administrators create desktops in the WebLogic Administration Portal and select that shell for the desktop, the desktop selector appears in the rendered desktops.

Adding Visitor Tools to Portals

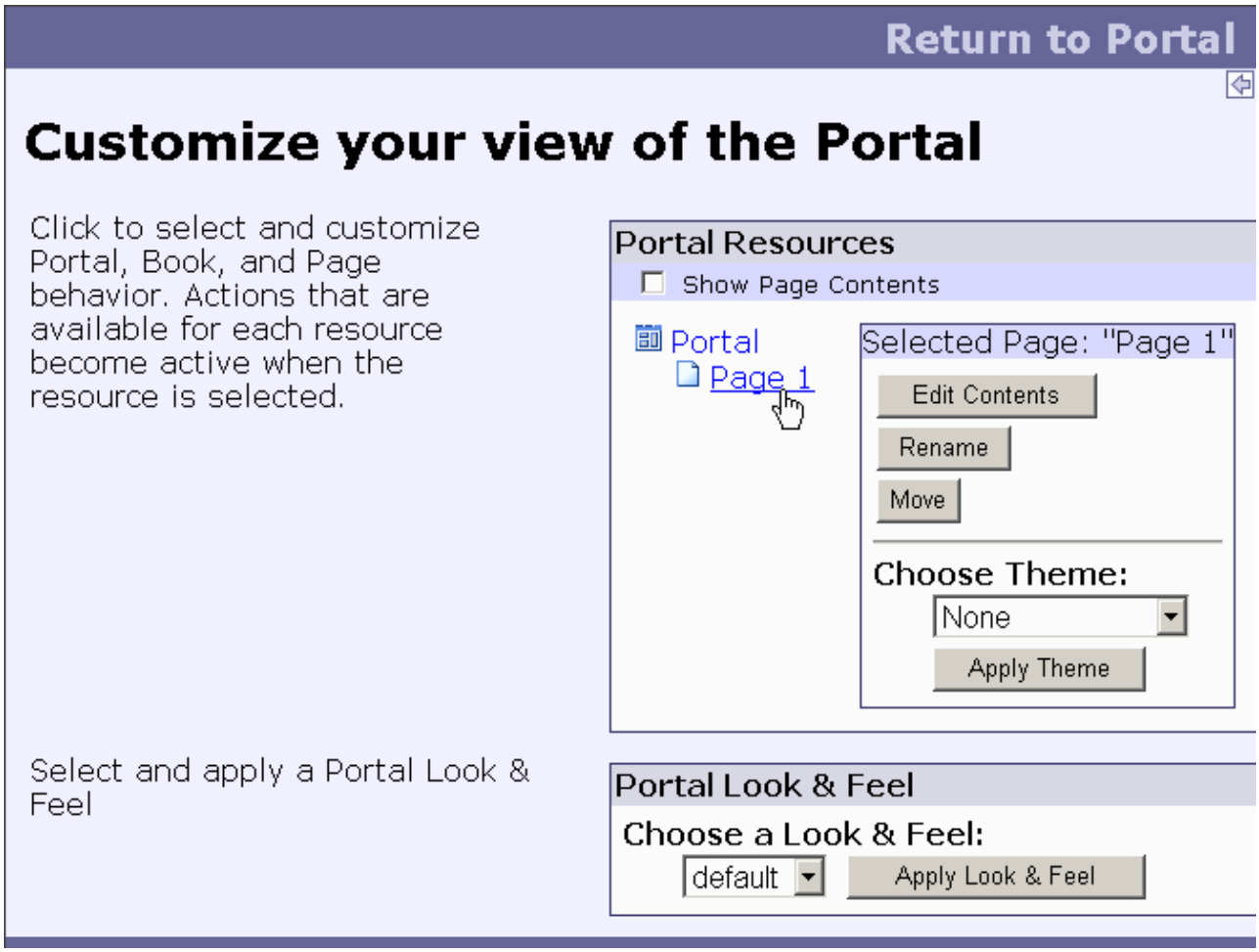
You can add functionality to your portal desktops that lets visitors modify their desktops, books, and pages. In order to use these Visitor Tools, visitors must be logged in to a desktop that is running in streaming mode.

Visitors access the visitor tools by clicking a text link or an icon in the desktop menu bar, as shown in the following illustration.



In this example, visitors can click on either the ***Customize My Portal*** link or the icon below it to access the Visitor Tools. The Customize My Portal link is supplied by the JSP used in the shell (described later in this topic), and the Edit icon is inserted by the menu skeleton JSP. Notice that the visitor must be logged in to access the Visitor Tools.

The following figure shows the Visitor Tools.



To add the Visitor Tools to Your Portals

The following procedure for adding Visitor Tools assumes you are adding them to a custom portal application (not the portalApp sample). If your application an/or project is not portal−enabled, install portal in both. See Creating a Portal Application and Portal Web Project.

1. Set up some form of authentication for your portal desktop. See Login Portlet, Login Director, or Implementing Authentication for information on adding authentication to your desktops.
2. In WebLogic Workshop create a new portal file.
3. In the Portal Designer, select the Main Page Book.
4. In the Property Editor window, set either of the following combinations of property values:

   Navigation: Single Level Menu or Multi Level Menu
   Editable: Edit in Menu

   or

   Navigation: No Navigation
   Editable: Edit in Titlebar
5. In the Mode Properties that appear, click the ellipsis icon [...] in the ***Content URI*** field, select <project>/visitorTools/visitorTools.portion, and click ***Open***.
6. Set the ***Visible*** property to false.

7. In the Portal Designer, select the Desktop.
8. In the Property Editor window, set the ***Shell*** to "Visitor Tools Shell."

   Now you must create a streaming desktop using the .portal file as a template to use the Visitor Tools.
9. If the server is not running, start it. Choose ***Tools −−> WebLogic Server −−> Start WebLogic Server***.
10. When the server is running, choose ***Portal −−> Portal Administration*** to start the WebLogic Administration Portal.
11. Log in to the WebLogic Administration Portal (the default username and password is ***weblogic***/***weblogic***).
12. Create a new desktop using your .portal file as a template. See Create a New Portal and Create a Desktop in the WebLogic Administration Portal online help posted on e−docs.
13. Select the new desktop in the Portal Resources tree, and go to the Desktop Properties page. At the bottom of the page, click ***View Desktop***.
14. When the desktop appears, log in and access the Visitor Tools.

You will notice that you can access the Visitor Tools by clicking the Customize My Portal link or the Edit icon. You do not have to use both ways to access the Visitor Tools.

- To use the Customize My Portal link only, use the Visitor Tools Shell and set the Main Page Book's ***Editable*** property to "Not Editable."
- To use the Edit icon, leave the ***Editable*** and ***Content URI*** property values in place and choose a shell other than Visitor Tools Shell.

The main page book in your .portal file can be used as the main page book when creating a desktop in the WebLogic Administration Portal to enable Visitor Tools. This will provide the desktop with Visitor Tools.

Note: You can also use the default New Blank Desktop template in the WebLogic Administration Portal to create a desktop that has Visitor Tools enabled.

Related Topics

Creating Shells

Properties for Portal Components

As you modify a portal file in WebLogic Workshop Platform Edition, you can use the Property Editor window to set component properties. The following topics describe the properties you can view and set for the following components:

Properties for All Portal Components

Desktop Properties

Header and Footer Properties

Book and Page Properties

Placeholder Properties

Portlet Properties

Samples

The WebLogic Workshop Portal Extensions contain sample portals you can open in WebLogic Workshop Platform Edition to view portal properties in the Property Editor window. See Portal Samples.

Properties for All Portal Components

As you modify a .portal or .portlet file in WebLogic Workshop Platform Edition, you can use the Property Editor window to set portal and portlet properties. The following table describes the common properties you can set for all portal components.

Administration Properties	
Markup Name	Read−only. The unique name of a component markup type. For example, three shell files (markup type "shell") must each use a unique Markup Name inside their .shell files. (Because desktops, books, and pages do not have associated .desktop, .book, and .page files, the exact same Markup Name is used for each.)
Presentation Properties	
Presentation Class	Optional. Typically a style sheet style, or class. Overrides the class attribute that would otherwise be used by the component's skeleton. For proper rendering, the class must exist in a cascading style sheet (CSS) in the desktop's selected skin, and the skin's skin.properties file must reference the CSS file. Sample − If you enter my−custom−class, the rendered HTML from the default skeletons will look like this: The properties you enter are added to the component's parent tag. On books, pages, and portlets, use the Content Presentation Class property to set properties on the component's content/child tag, especially for setting a style class that enables content scrolling and height−setting.
Presentation ID	Optional. A unique ID inserted in the rendered HTML tag for the component. The value you enter (which must be unique among all presentation IDs in the portal) overrides the ID that might otherwise be inserted by the component's skeleton. An example use would be inserting a unique ID that JavaScript could operate on. Sample − If you enter 12345, the rendered HTML from the default skeletons will look like this:

Presentation Style	Optional. HTML style attribute to insert for the portal component. This attribute is equivalent to a style sheet class attribute and overrides any attributes in the style sheet class. Separate multiple entries with a semicolon. Sample − If you enter {background−color: #fff} for a portlet titlebar, the rendered HTML from the default skeletons will look like this: and the portlet titlebar will have a white background. The background−color attribute you entered overrides the background−color attribute in the bea−portal−window−titlebar class. The properties you enter are added to the component's parent tag. On books, pages, and portlets, use the Content Presentation Style property to set properties on the component's content/child tag, especially for setting content scrolling and height.
Skeleton URI	Optional. The path (relative to the project) to a skeleton JSP that will be used to render the portal component. This JSP overrides the skeleton JSP that would otherwise be used by the selected Look & Feel for the desktop. For example, enter /frameworks/myskeletons/mytitlebar.jsp.

Related Topics

Desktop Properties

Header and Footer Properties

Book and Page Properties

Placeholder Properties

Portlet Properties

Desktop Properties

When you select the desktop in the Portal Designer, the following properties appear in the Property Editor window:

Title	Required. Enter a title for the desktop. The title is used only for labeling in the Portal Designer.
Look and Feel	Required. Select the Look & Feel to determine the default desktop appearance (combination of skins and skeletons).
Shell	Required. Select the default shell for the area outside of the books, pages, and portlets. Shells determine the content for the desktop header and footer.

Related Topics

Properties for All Portal Components

Header and Footer Properties

Book and Page Properties

Placeholder Properties

Portlet Properties

Header and Footer Properties

When you select the "Content" node under the Header or Footer node in the Portal Designer's Document Structure window, the following properties appear in the Property Editor window:

Content URI	Read−only. The JSP file referenced in a shell that is used to display content in the desktop header or footer. This value is read from the <netuix:jspContent> "contentUri" attribute in the .shell file <header> or <footer> tag. If you select a different shell for the desktop, you may see a different value here.
Error Page URI	Read−only. The file referenced in a shell that is used to display an error message in the desktop header or footer if the contentUri JSP encounters errors. This value is read from the <netuix:jspContent> "errorUri" attribute in the .shell file <header> or <footer> tag. If you select a different shell for the desktop, you may see a different value here.
Backing File	Read−only. The class referenced in a shell that is used for preprocessing prior to rendering a shell's header or footer JSP. This value is read from the <netuix:jspContent> "backingFile" attribute in the .shell file <header> or <footer> tag. If you select a different shell for the desktop, you may see a different value here.

Related Topics

Properties for All Portal Components

Desktop Properties

Book and Page Properties

Placeholder Properties

Portlet Properties

Book and Page Properties

When you select a book or page in the Portal Designer, the following properties appear in the Property Editor window.

The term "hint" in the descriptions means available capabilities that are not supported in the default skeletons provided with the WebLogic Workshop Portal Extensions.

Title	Required. Enter a title for the book or page. Page titles are used for the page tabs/navigation menus.
Theme	Optional. Select a theme to give the book or page a different look and feel than the rest of the desktop.
Definition Label	Required. Unique identifier for each book or page. A default value is entered automatically, but you can change the value. Each book or page must have a unique Definition Label. Definition Labels can be used to navigate to books or pages. Also, components must have Definition Labels for entitlements and delegated administration.
Backing File	Optional. If you want to use a class for preprocessing (for example, authentication) prior to rendering the book or page, 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. See the Tutorial Portal in the Portal Samples for examples of backing files.
Unselected Image	Optional. Select an image to override the icon that appears next to the book or page title. This image appears on the tab of unselected pages.
Selected Image	Optional. Select an image to override the icon that appears next to the book or page title. This image appears on the tab of selected pages.
Rollover Image	Optional. Select an rollover image for the icon that appears next to the book or page title. This image appears when the mouse rolls over the tabs of unselected pages.
Orientation	Optional. Hint to the skeleton to position the navigation menu on the top, bottom, left, or right side of the book. You must build your own skeleton to support this property. Following are the numbers used in the .portal file for each orientation value: top=0, left=1, right=2, bottom=3.
Packed	Optional. Rendering hint that can be used by the skeleton to render the book or page in either expanded or packed mode. You must build your own skeleton to support the property. When packed="false" (the default), the book or page takes up as much horizontal space as it can. When packed="true," the book or page takes up as little horizontal space as possible.

	From an HTML perspective, this property is most useful when the window is rendered using a table. When packed="false," the table's relative width would likely be set to "100%." When packed="true," the table width would likely remain unset.
Hidden	Optional. Hides the navigation tab for the book or page to prevent direct access. You can access the page or book with a link (to the definition label) or by using a backing file.
Default Page (Book only)	Required. Select the page that appears by default when the desktop is accessed. The list is populated with Definition Labels of all pages in the portal.
Return to Default Page (Book only)	Optional. Determines the page displayed when a book is selected. When Return to Default Page="false" (the default), the last page that was active in a book is displayed when the book is selected. When Return to Default Page="true," the page selected in the Default Page property is always displayed when a book is selected.
Content Presentation Class	Optional. The primary uses are to allow content scrolling and content height−setting. For scrolling, enter a stylesheet class that uses one of the following attributes: • overflow−y:auto − Enables vertical (y−axis) scrolling • overflow−x:auto − Enables horizontal (x−axis) scrolling • overflow:auto − Enables vertical and horizontal scrolling For setting height, enter a stylesheet class that uses the following attribute: height:200px where 200px is any valid HTML height setting. You can also set other style properties for the content as you would using the Presentation Class property. The properties are applied to the component's content/child tag.
Content Presentation Style	Optional. The primary uses are to allow content scrolling and content height−setting. For scrolling, enter one of the following attributes: • overflow−y:auto − Enables vertical (y−axis) scrolling • overflow−x:auto − Enables horizontal (x−axis) scrolling • overflow:auto − Enables vertical and horizontal scrolling For setting height, enter the following attribute: height:200px

	where 200px is any valid HTML height setting. You can also set other style properties for the content as you would using the Presentation Style property. The properties are applied to the component's content/child tag.
Layout Type (Pages only)	Required. Select the page layout style for positioning books and portlets in placeholders on a page.
Navigation (Book only)	Required. Select the default type of menu to use for navigation among books and pages.
Editable (Book only)	Optional. If you have visitor tools enabled so that users can modify book properties, setting Editable to "Edit in Titlebar" or "Edit in Menu" puts a visitor tool link in that location. "Edit in Menu" is only available if you select a menu type for the Navigation property. When you select "Edit in Titlebar" or "Edit in Menu," a group of Mode Properties appears in the Property Editor.
Mode Properties (edit mode properties available on books when you select "Edit in Titlebar" or "Edit in Menu" for the Editable property)	
Content URI	Required. The path (relative to the project) to the JSP or HTML file to be used for book's edit page. For example, if the content is stored in <project>/edit/editBook.jsp, the Content URI is /edit/editBook.jsp.
Backing File	Optional. If you want to use a class for preprocessing (for example, authentication) prior to rendering the book's edit page, 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. See the Tutorial Portal in the Portal Samples for examples of backing files.
Error URI	Optional. The path (relative to the project) to the JSP or HTML file to be used for the error message if the book's edit page cannot be rendered. For example, if the error page is stored in <project>/exception/errorBookEdit.jsp, the Content URI is /exception/errorBookEdit.jsp.
Name	Enter the name of the mode, such as "Edit".
Visible	Optional. Makes the edit icon in the titlebar or menu invisible (false) or visible (true). Set Visible to "false" when, for example, you want to provide an edit URL in a desktop header.

Related Topics

Properties for All Portal Components

Desktop Properties

Header and Footer Properties

Placeholder Properties

Portlet Properties

Placeholder Properties

When you select a placeholder in the Portal Designer, the following read−only properties appear in the Property Editor window. The property values are read in from the .layout file that corresponds to the selected Layout Type for the page.

Title	Read−only. The name of the placeholder. This value is read from the .layout file for the page's selected Layout Type.
Flow	Read−only. If the "Using Flow" property is set to true, this value can be "vertical" or "horizontal." Flow determines whether books or portlets put in the placeholder are positioned on top of each other (vertical) or beside each other (horizontal). This value is read from the .layout file for the page's selected Layout Type.
Using Flow	Read−only. If this value is set to "true," books and portlets put in the placeholder are positioned according to the value of the "Flow" property. If this value is set to "false," the default flow is used (vertical). This value is read from the .layout file for the page's selected Layout Type.
Placeholder Width	Read−only. Displays the width set for the placeholder. This value is read from the .layout file for the page's selected Layout Type.

Related Topics

Properties for All Portal Components

Desktop Properties

Header and Footer Properties

Book and Page Properties

Portlet Properties

Portlet Properties

This topic describes the properties you can set on portlets in the Portal and Portlet designers.

Portlet Properties in the Portal Designer

Portlet Properties in the Portlet Designer

The term "hint" in the property descriptions means available capabilities that are not supported in the default skeletons provided with the WebLogic Workshop Portal Extensions.

Portlet Properties in the Portal Designer

When you select a portlet instance in the Portal Designer (as opposed to opening the .portlet source file), the following properties appear in the Property Editor window.

Title	Required. Enter a title only if you want to override the default title provided by the .portlet file. The title is used in the portlet titlebar.
Instance Label	Required. A single portlet, represented by a .portlet file, can be used multiple times in a portal. Each use of that portlet is a portlet instance, and each portlet instance must have a unique ID, or Instance Label. A default value is entered automatically, but you can change the value. Instance Labels are necessary for inter−portlet communication between Java Page Flow portlets. Also, portlets must have Instance labels for entitlements and delegated administration.
Portlet URI	Required. The path (relative to the project) of the parent .portlet file. For example, if the file is stored in <project>\myportlets\my.portlet, the Portlet URI is /myportlets/my.portlet.
Theme	Optional. Select a theme to give the portlet a different look and feel than the rest of the desktop.
Orientation	Optional. Hint to the skeleton to position the portlet titlebar on the top, bottom, left, or right side of the portlet. You must build your own skeleton to support this property. Following are the numbers used in the .portal file for each orientation value: top=0, left=1, right=2, bottom=3. Change the value for this property only if you want to override the default orientation provided by the .portlet file.
Default Minimized	Optional. Select "true" for the portlet to be minimized when it is rendered. The default value is "false." Change the value for this property only if you want to override the default value provided by the .portlet file.

Portlet Properties in the Portlet Designer

When you open a portlet in the Portlet Designer, the following properties appear in the Property Editor window:

JSP Content	
Content URI	Required. The path (relative to the project) to the JSP or HTML file to be used for portlet's content. The Content URI was set when the portlet was created. You can use this property to point to a different file. For example, if the content is stored in <project>/myportlets/my.jsp, the Content URI is /myportlets/my.jsp.
Error Page URI	Optional. The path (relative to the project) to the JSP or HTML file to be used for the portlet's error message if the main content cannot be rendered. For example, if the error page is stored in <project>/myportlets/error.jsp, the Content URI is /myportlets/error.jsp.
Backing File	Optional. If you want to use a class for preprocessing (for example, authentication) prior to rendering the portlet, 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. See the Tutorial Portal in the Portal Samples for examples of backing files.
Thread Safe	Optional. Performance setting for books, pages, and portlets that use backing files. When Thead Safe is set to "true," an instance of a backing file is shared among all books, pages, or portlets that request the backing file. You must synchronize any instance variables that are not thread safe. When Thread Safe is set to "false," a new instance of a backing file is created each time the backing file is requested by a different book, page, or portlet.
Portlet Properties	
Title	Required. Enter the title that will appear in the portlet's titlebar. You can override this title in each instance of the portlet (in the Portal Designer).
Orientation	Optional. Hint to the skeleton to position the portlet titlebar on the top, bottom, left, or right side of the portlet. You must build your own skeleton to support this property in the .portal file. Following are the numbers used in the .portal file for each orientation value: top=0, left=1, right=2, bottom=3. You can override the orientation in each instance of the portlet (in the Portal Designer).

Packed	Optional. Rendering hint that can be used by the skeleton to render the portlet in either expanded or packed mode. You must build your own skeleton to support this property. When packed="false" (the default), the portlet takes up as much horizontal space as it can. When packed="true," the portlet takes up as little horizontal space as possible. From an HTML perspective, this property is most useful when the window is rendered using a table. When packed="false," the table's relative width would likely be set to "100%." When packed="true," the table width would likely remain unset.
Definition Label	Required. Unique identifier for the portlet. A default value is entered automatically, but you can change the value. Each portlet must have a unique Definition Label. Definition Labels can be used to navigate to portlets. Also, components must have Definition Labels for entitlements and delegated administration.
Default Minimized	Required. Select "true" for the portlet to be minimized when it is rendered. The default value is "false."
Render Cacheable	Optional. To enhance performance, set to "true" to cache the portlet. For example, portlets that call Web services perform frequent, expensive processing. Caching Web service portlets greatly enhances performance. Do not set this to "true" if you are doing your own caching.
Cache Expires (seconds)	Optional. When the "Render Cacheable" property is set to "true," enter the number of seconds in which the portlet cache expires.
Fork Render	Optional. Intended for use by a portal administrator when configuring or tuning a portal. Setting to "true" tells the framework that it should attempt to multithread render the portlet. This property can be set to true only if the "Forkable" property is set to "true."
Forkable	Optional. Lets a portlet developer determine whether or not the portlet is allowed to be multithread rendered. When set to "true," a portal administrator can use the "Fork Render" property to make the portlet multithread rendered.
Client Classifications	Optional. Select the multichannel devices in which the portlet can be viewed. The dialog list of devices comes from <project>\WEB−INF\client−classifications.xml. In the Manage Portlet Classifications dialog: 1. Select whether you want to enable or disable classifications for the portlet. 2. Move the client classifications you want to enable or disable from the Available Classifications to the Selected Classifications, and click OK.

	When you disable classifications for a portlet, the portlet is automatically enabled for the classifications you did not select for disabling.
Portlet Titlebar	
Icon URI	Optional. The path (relative to the project) to the graphic to be used in the portlet titlebar. You must create a skeleton to support this property.
Help URI	Optional. The path (relative to the project) to the portlet's help file.
Edit URI	Optional. The path (relative to the project) to the portlet's edit page.
Can Maximize	Optional. If set to "true," the portlet can be maximized.
Can Minimize	Optional. If set to "true," the portlet can be minimized.
Can Delete	Optional. If set to "true," the portlet can be deleted from a page.
Mode Properties (Available when you add a mode to a portlet)	
Content URI	Required. The path (relative to the project) to the JSP or HTML file to be used for portlet's mode content, such as the edit page. For example, if the content is stored in <project>/myportlets/editPortlet.jsp, the Content URI is /myportlets/editPortlet.jsp.
Backing File	Optional. If you want to use a class for preprocessing (for example, authentication) prior to rendering the portlet's mode page (such as the edit page), 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. See the Tutorial Portal in the Portal Samples for examples of backing files.
Error URI	Optional. The path (relative to the project) to the JSP or HTML file to be used for the error message if the portlet's mode page cannot be rendered. For example, if the error page is stored in <project>/myportlets/errorPortletEdit.jsp, the Content URI is /myportlets/errorPortletEdit.jsp.
Name	Enter the name of the mode, such as "Edit".
Visible	Optional. Makes the mode icon (such as the edit icon) in the titlebar or menu invisible (false) or visible (true). Set Visible to "false" when, for example, you want to provide an edit URL in a desktop header.
Presentation Properties	
Presentation Class Presentation ID Presentation Style Skeleton URI	See Properties for All Portal Components.
Content Presentation	Optional. The primary uses are to allow content scrolling and content height−setting.

Style	For scrolling, enter one of the following attributes: • overflow−y:auto − Enables vertical (y−axis) scrolling • overflow−x:auto − Enables horizontal (x−axis) scrolling • overflow:auto − Enables vertical and horizontal scrolling For setting height, enter the following attribute: height:200px where 200px is any valid HTML height setting. You can also set other style properties for the content as you would using the Presentation Style property. The properties are applied to the component's content/child tag.
Content Presentation Class	Optional. The primary uses are to allow content scrolling and content height−setting. For scrolling, enter a stylesheet class that uses one of the following attributes: • overflow−y:auto − Enables vertical (y−axis) scrolling • overflow−x:auto − Enables horizontal (x−axis) scrolling • overflow:auto − Enables vertical and horizontal scrolling For setting height, enter a stylesheet class that uses the following attribute: height:200px where 200px is any valid HTML height setting. You can also set other style properties for the content as you would using the Presentation Class property. The properties are applied to the component's content/child tag.

Related Topics

Properties for All Portal Components

Desktop Properties

Header and Footer Properties

Book and Page Properties

Placeholder Properties