BEA WebLogic Server®

Extending the Administration Console

Version 9.0
Revised: July 22, 2005

Contents

Introduction and Roadmap

The BEA WebLogic Server® Administration Console is a browser-based graphical user interface that you use to manage a WebLogic Server domain. Because it uses the WebLogic Portal® framework to render its user interface, the process of extending the Administration Console is similar to creating or editing an existing WebLogic Portal application.

Administration Console extensions enable you to add content to the WebLogic Server Administration Console, replace content, and change the logos, styles and colors without modifying the files that are installed with WebLogic Server. For example, you can add a portlet that provides custom monitoring and management facilities for your applications.

The following sections describe the contents and organization of this guide—*Extending the Administration Console*.

- "Document Scope and Audience" on page 1-1
- "Guide to this Document" on page 1-2
- "Related Documentation" on page 1-2
- "New and Changed Console-Extension Features in This Release" on page 1-3

Document Scope and Audience

This document is a resource for software vendors who embed (and rebrand) WebLogic Server in their products, software vendors who develop security providers or other resources that extend the functionality of WebLogic Server, and J2EE application developers who want to provide custom monitoring and configuration features for their applications.

It is assumed that the reader is already familiar with using Java, JavaServer Pages, and Apache Struts to develop J2EE Web applications. This document emphasizes a hands-on approach to developing a limited but useful Administration Console extension. For information on applying Administration Console extensions to a broader set of management problems, refer to documents listed in "Related Documentation" on page 1-2.

Guide to this Document

- This chapter, Introduction and Roadmap, introduces the organization of this guide.
- Chapter 2, "Understanding Administration Console Extensions," introduces the building blocks for creating Administration Console extensions.
- Chapter 3, "Rebranding the Administration Console," describes how to create a WebLogic Portal Look and Feel and deploy it as an Administration Console extension.
- Chapter 4, "Adding Portlets and Navigation Controls," describes how to add portlets that contain simple, static content to the Administration Console.
- Chapter 5, "Creating Portlets That Match the Administration Console," describes how to create an extension that uses the Administration Console's JSP templates, styles, and JSP tag library.

Related Documentation

This section provides links to documentation that describes the technologies used by the Administration Console. The more you understand these technologies, the more complex extensions you can create.

For information on the WebLogic Portal framework, see:

- "How Do the WebLogic Portal Framework and WebLogic Portal Differ?" on page 2-2
- *Portal User Interface Framework Guide* at http://e-docs.bea.com/wlp/docs81/lookandfeel/index.html
- *White Paper: WebLogic Portal Framework* at http://e-docs.bea.com/wlp/docs81/whitepapers/netix/index.html

For information on Apache Struts, see *The Apache Struts Web Application Framework* at http://struts.apache.org/.

For information on JavaServer Pages, see *JavaServer Pages Technology* at http://java.sun.com/products/jsp/index.jsp.

For information on Apache Beehive Page Flows, see *Page Flows: Getting Started* at http://incubator.apache.org/beehive/pageflow/getting_started.html.

New and Changed Console-Extension Features in This Release

The WebLogic Server Administration Console has been completely re-designed in this release. Prior to 9.0, the Administration Console used JSPs and its own framework to render its user interface; it did not use the WebLogic Portal framework, Apache Struts, or Apache Beehive.

Because the new architecture is so different, WebLogic Administration Console extensions built for prior releases of WebLogic Server will not function in 9.0. While you might be able to re-use the content and some of the logic in the JSPs that you created in previous Administration Console extensions, BEA no longer supports the APIs or JSP tag libraries that it provided for extensions prior to 9.0. In addition, you must now define portlets as the container for your JSPs.

If you added a node to the Administration Console navigation tree in a previous release, see "Add Nodes to the Domain Structure Portlet (Optional)" on page 4-11 for information on how to do so in 9.0.

If you localized the text in your Administration Console extension prior to this release, see "Create and Use a Message Bundle" on page 5-3 for information on how to do so in 9.0.

Understanding Administration Console Extensions

Administration Console extensions enable you to add content to the WebLogic Server Administration Console, replace content, and change the logos, styles and colors without modifying the files that are installed with WebLogic Server. For example, you can add a portlet that provides custom monitoring and management facilities for your applications.

The Administration Console is a J2EE Web application that uses the WebLogic Portal framework, Apache Beehive, Apache Struts, Java Server Pages (JSP), and other standard technologies to render its user interface (UI) and content. It also uses the WebLogic Portal framework to enable extensions.

The following sections describe Administration Console extensions:

What Is an Administration Console Extension?

An Administration Console extension is a JAR file that contains the resources for a section of a WebLogic Portal Web application. When you deploy the extension, the Administration Console creates a union of the files and directories in its WAR file with the files and directories in the extension JAR file. Once the extension has been deployed, it is a full member of the

Administration Console: it is secured by the WebLogic Server security realm, it can navigate to other sections of the Administration Console, and if the extension modifies WebLogic Server resources, it participates in the change control process.

The two key components of an Administration Console extension are:

- The resources that make up your section of the Administration Console.

  At a minimum, you must provide the following resources:

  – A JSP or HTML file that contains the content you want to display.

  – A portlet XML file that defines a WebLogic Portal portlet, which is a container for JSPs, HTML files, and other types of content. Some portlets provide borders and minimize/maximize controls; some do not.

  For more complex extensions, you can provide any of the following additional resources:

  – Java classes.

  – Configuration files and Java classes for Apache Struts applications.

  – Support files for Apache Beehive Page Flows.

  – XML files that describe other types of WebLogic Portal UI controls, such as tabs and subtabs (see "UI Controls in the Administration Console" on page 2-3).

  The Administration Console does not support WSRP portlets or portlets based on JSR 168.

- A NetUI Extension XML file that describes the location in the UI in which you want your extension to display.

How Do the WebLogic Portal Framework and WebLogic Portal Differ?

The WebLogic Portal framework provides basic support for rendering the UI. The full WebLogic Portal product provides the framework and additional features such as personalization, interaction management, content management, and the ability for end users to customize their portal desktops.

If your BEA product license includes only WebLogic Server, then you can use the WebLogic Portal framework when creating Administration Console extensions. If you want your own Web applications to provide a portal interface, you can purchase the WebLogic Portal product.

UI Controls in the Administration Console

The UI for the Administration Console is rendered by groups of specialized WebLogic Portal components called UI controls. For example, one group of UI controls renders the two-column layout that you see after you log in to the Administration Console. Other groups render individual tabs in the tabbed interface.

Many of these UI controls are identified by unique labels, and your NetUI Extension XML file specifies the labeled control that you want to append or replace. You can also use these labels with WebLogic Server JSP tags to forward requests to specific UI controls. If a UI control is not identified by a label, you cannot extend it or forward to it. You must either interact with its labeled ancestor control or a labeled child control.

The following sections describe the labeled UI controls in the Administration Console:

- "Types of UI Controls" on page 2-3
- "The Desktop" on page 2-4
- "The Home Book and Page" on page 2-6
- "The ContentBook" on page 2-7
- "Summary of the Administration Console UI Controls" on page 2-10

Types of UI Controls

The following is a list of the types of UI controls that the Administration Console labels:

- Desktop

  Contains the Look and Feel for the Administration Console and the top-level book. It provides little functionality beyond entitlement checking and aggregating other controls.

- Book

  Aggregates a set of pages or other books. It can contain an optional menu control that provides navigation among its pages and books.

- Page

  Displays portlets or books. It arranges its content based on the definitions in a layout.

- Layout

  Defines a grid in the UI. Each column in the grid is called a placeholder, and each placeholder can host zero or more portlets or books.

  Most pages in the Administration Console use a single column layout, but one of the top pages uses a two-column layout to create the left column that contains the Change Center, Domain Structure, and other portlets, and the right column that contains the tabbed interface.

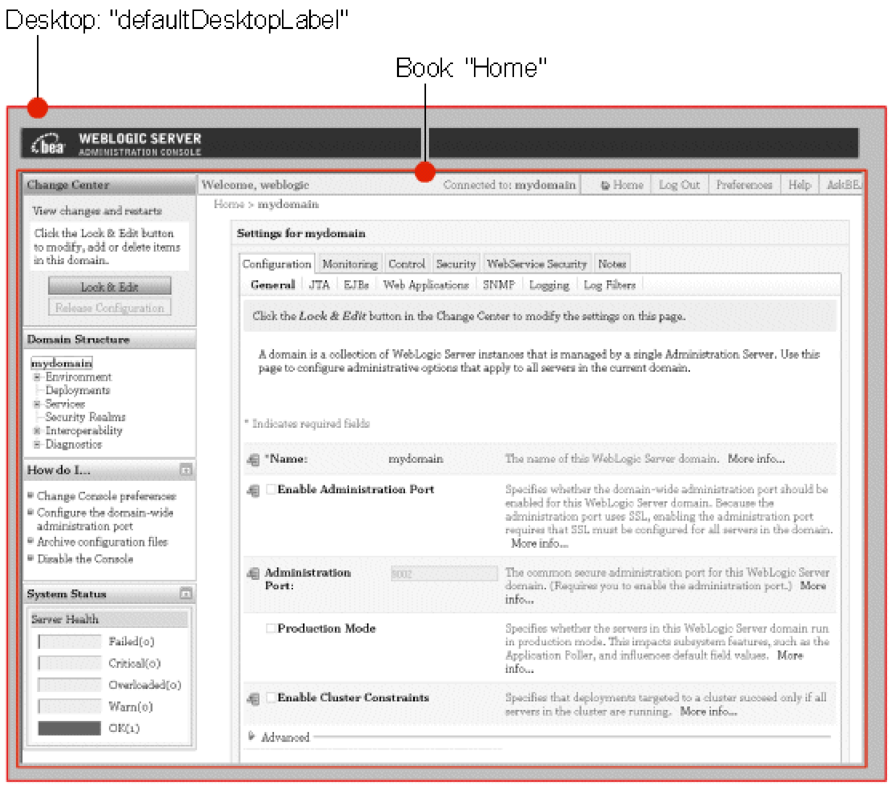
- Portlet

  Defines static and dynamic content to display. You can add portlets to the Administration Console that contain HTML pages and JSP files, or that forward to Struts actions or Beehive Page Flows.

The Desktop

Every WebLogic Portal Web application must have at least one desktop control, and the Administration Console supports **only** one. Its label is `defaultDesktopLabel` (see Figure 2-1).

Figure 2-1 The Desktop



View changes and restarts

Click the Lock & Edit button to modify, add or delete items in this domain.

mydomain
- Environment
- Deployments
- Services
- Security Realms
- Interoperability
- Diagnostics

- Change Console preferences
- Configure the domain-wide administration port
- Archive configuration files
- Disable the Console

Failed(0)
Critical(0)
Overloaded(0)
Warn(0)
OK(1)

Settings for mydomain

Click the *Lock & Edit* button in the Change Center to modify the settings on this page.

A domain is a collection of WebLogic Server instances that is managed by a single Administration Server. Use this page to configure administrative options that apply to all servers in the current domain.

* Indicates required fields

*Name:	mydomain	The name of this WebLogic Server domain. More info...
Enable Administration Port		Specifies whether the domain-wide administration port should be enabled for this WebLogic Server domain. Because the administration port uses SSL, enabling the administration port requires that SSL must be configured for all servers in the domain. More info...
Administration Port:	9002	The common secure administration port for this WebLogic Server domain. (Requires you to enable the administration port.) More info...
Production Mode		Specifies whether the servers in this WebLogic Server domain run in production mode. This impacts subsystem features, such as the Application Poller, and influences default field values. More info...
Enable Cluster Constraints		Specifies that deployments targeted to a cluster succeed only if all servers in the cluster are running. More info...

Extending the Desktop

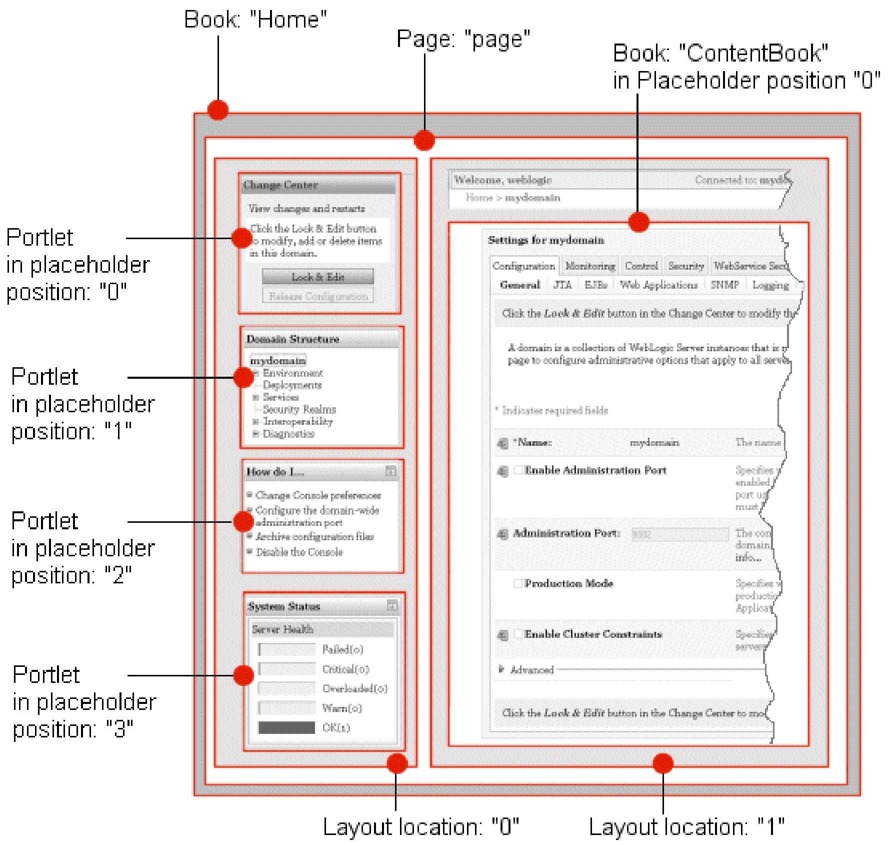
You can use Administration Console extensions to replace the desktop's Look and Feel, but you cannot replace the top-level book. To replace the Look and Feel, use the `desktop-extension` element in the netuix-extension XML file (see desktop-extension in the *NetUI Extension Schema Reference*):

```
<desktop-extension>
    <look-and-feel-content title="myLookandFeel"
      ...
   />
</desktop-extension>
```

The Home Book and Page

The top-level book in the Administration Console is identified by the label `Home`. It contains a single page (labeled `page`) within which resides all of the Administration Console content (see Figure 2-2).

Figure 2-2 The Home Book and Page



Click the Lock & Edit button to modify, add or delete items in this domain.

Click the *Lock & Edit* button in the Change Center to modify th

A domain is a collection of WebLogic Server instances that is
page to configure administrative options that apply to all serve

Click the *Lock & Edit* button in the Change Center to mo

The `page` page uses a two-column layout. The left column (layout location 0) contains portlets that provide essential services when using the Administration Console. The right column (layout location 1) contains:

- Portlets:

  - The topmost portlet displays a welcome message and contains buttons that launch online help and other services.
  - The second portlet displays breadcrumbs, which are a series of hypertext links that keep a history of your navigation in the Administration Console.
  - A third portlet is hidden by default and displays error messages and other status messages.
- A book named `ContentBook`. See "The ContentBook" on page 2-7.

Extending the Home Book

The simplest extensions within the `Home` book add portlets to either column of its `page` page. For example, below the System Status portlet, you can add a portlet that monitors your applications.

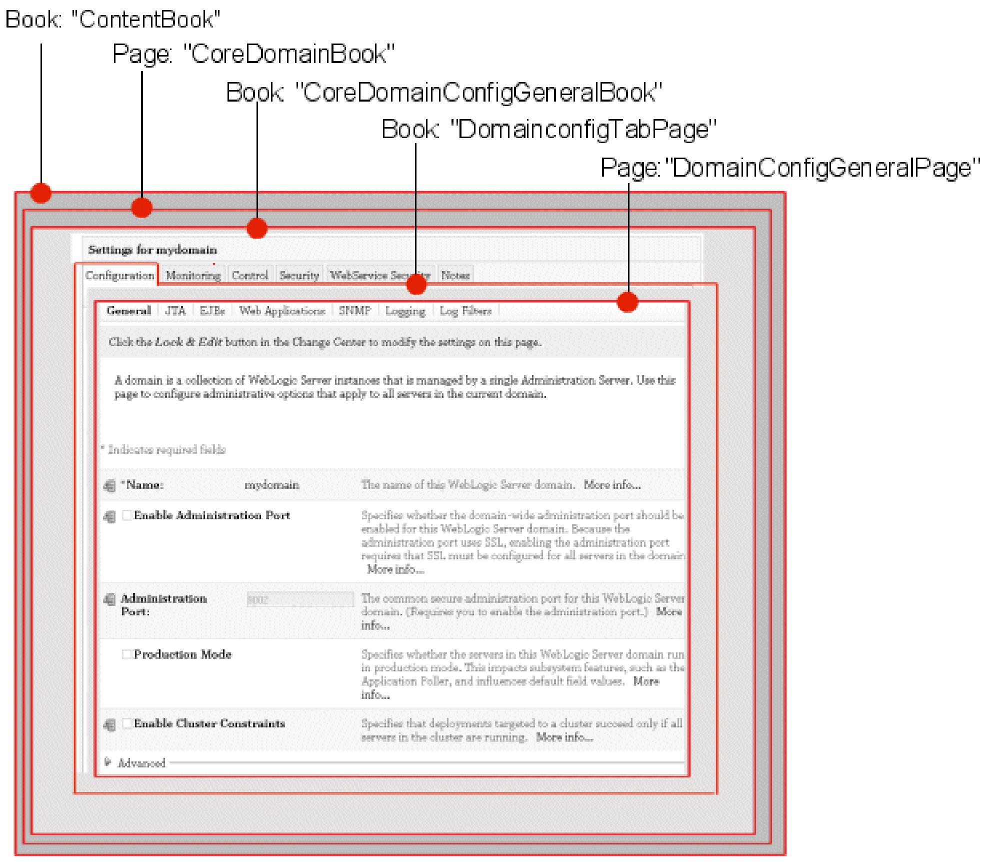
To extend the `page` page, use the `page-extension` element in the netuix-extension XML file (see page-extension in the *NetUI Extension Schema Reference*):

```
<page-extension>
    <page-location>
      <parent-label-location label="page"/>
      ...
   </page-location>
   ...
</page-extension>
```

The ContentBook

The `ContentBook` is a book that contains over 40 pages, but it displays only one page at a time. Navigational controls throughout the Administration Console determine which page is displayed. For example, if a user clicks on "mydomain" in the portlet whose user-visible title is Domain Structure (NavTreePortlet), the `ContentBook` displays the page that contains controls for the configuration a domain (see Figure 2-3).

Figure 2-3 The ContentBook


Book: "ContentBook"
Page: "CoreDomainBook"
Book: "CoreDomainConfigGeneralBook"
Book: "DomainconfigTabPage"
Page: "DomainConfigGeneralPage"
Configuration | Monitoring | Control | Security | WebService Security | Notes
General | JTA | EJBs | Web Applications | SNMP | Logging | Log Filters
Advanced

Settings for mydomain

Click the *Lock & Edit* button in the Change Center to modify the settings on this page.

A domain is a collection of WebLogic Server instances that is managed by a single Administration Server. Use this page to configure administrative options that apply to all servers in the current domain.

* Indicates required fields

***Name:**	mydomain	The name of this WebLogic Server domain. More info...
☐ **Enable Administration Port**		Specifies whether the domain-wide administration port should be enabled for this WebLogic Server domain. Because the administration port uses SSL, enabling the administration port requires that SSL must be configured for all servers in the domain. More info...
Administration Port:	9002	The common secure administration port for this WebLogic Server domain. (Requires you to enable the administration port.) More info...
☐ **Production Mode**		Specifies whether the servers in this WebLogic Server domain run in production mode. This impacts subsystem features, such as the Application Poller, and influences default field values. More info...
☐ **Enable Cluster Constraints**		Specifies that deployments targeted to a cluster succeed only if all servers in the cluster are running. More info...

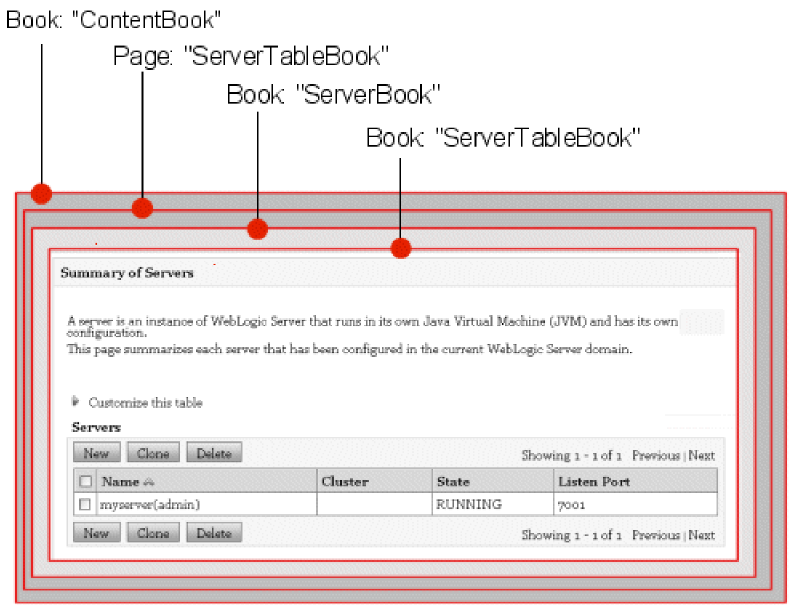
Each page in `ContentBook` contains a single content-specific book. This content-specific book contains multiple books, some of which use a singleLevelMenu control to display a tabbed interface. Each of the tabs in Figure 2-3 (Configuration, Monitoring, Control, Security, Web Services Security, and Notes) is rendered by a book that uses the singleLevelMenu control.

Note: The `CoreDomainBook` contains additional labeled controls that are omitted for the sake of brevity.

Each book that displays a tabbed interface contains at least one page. If the book contains multiple pages, each page uses a singleLevelMenu control to display a subtab. In Figure 2-3, the `DomainconfigTabPage` book renders the Configuration tab. Each of the subtabs in this book is a page. For example, the `DomainConfigGeneralPage` is the page that contains the portlet that renders the General subtab.

Some content-specific books do not display a tabbed interface. Figure 2-4 shows the `ServerBook`, which does not display a tabbed interface.

Figure 2-4 ServerTableBook



Summary of Servers

A server is an instance of WebLogic Server that runs in its own Java Virtual Machine (JVM) and has its own configuration.
This page summarizes each server that has been configured in the current WebLogic Server domain.

Servers

Showing 1 - 1 of 1 Previous | Next

☐	Name	Cluster	State	Listen Port
☐	myserver(admin)		RUNNING	7001

Showing 1 - 1 of 1 Previous | Next

Extending the ContentBook

The simplest extensions within the `ContentBook` add a tabbed book to a content-specific book or add a subtab. To extend the `ContentBook`, use the `book-extension` element in the netuix-extension XML file (see book-extension in the *NetUI Extension Schema Reference*).

For example, the following XML stanza adds a tabbed book to `CoreDomainGeneralBook`:

```
<book-extension>
   <book-location>
      <parent-label-location label="CoreDomainGeneralBook"/>
      <book-insertion-point action="append"/>
   </book-location>
</book-extension>
```

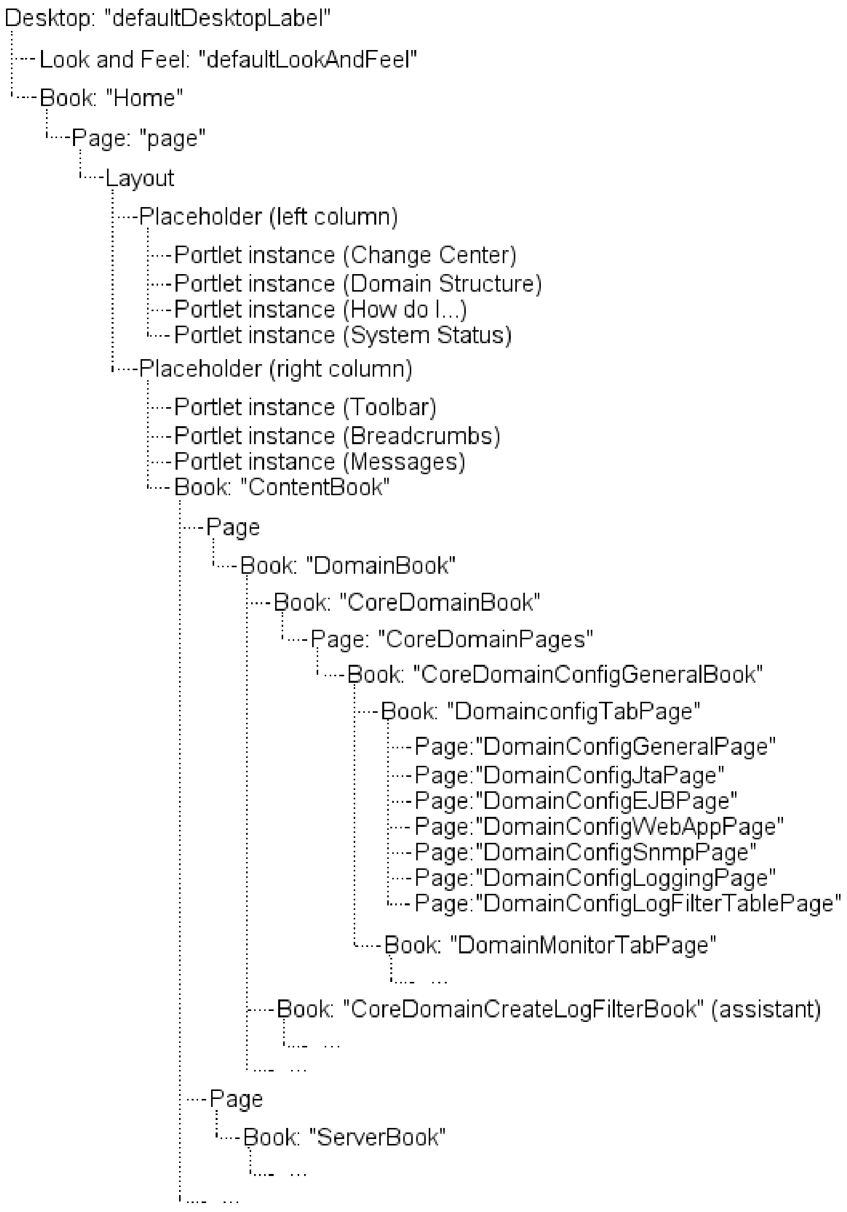
The following XML stanza adds a subtab to the `DomainconfigTabPage` book:

```
<book-extension>
   <book-location>
      <parent-label-location label="DomainconfigTabPage"/>
      <book-insertion-point action="append"/>
   </book-location>
</book-extension>
```

Summary of the Administration Console UI Controls

Figure 2-5 shows the top levels of the Administration Console's labeled UI controls. For a complete list of labeled UI controls, including all of the content-specific books, download and install a Look and Feel that causes the Administration Console to display labels for its controls. See "Deploy a Development Look and Feel to See UI Control Labels" on page 4-15.

Figure 2-5 Summary of the UI Control Hierarchy



```
Desktop: "defaultDesktopLabel"
 ---Look and Feel: "defaultLookAndFeel"
 ---Book: "Home"
     ---Page: "page"
         ---Layout
             ---Placeholder (left column)
                 ---Portlet instance (Change Center)
                 ---Portlet instance (Domain Structure)
                 ---Portlet instance (How do I...)
                 ---Portlet instance (System Status)
             ---Placeholder (right column)
                 ---Portlet instance (Toolbar)
                 ---Portlet instance (Breadcrumbs)
                 ---Portlet instance (Messages)
                 ---Book: "ContentBook"
                     ---Page
                         ---Book: "DomainBook"
                             ---Book: "CoreDomainBook"
                                 ---Page: "CoreDomainPages"
                                     ---Book: "CoreDomainConfigGeneralBook"
                                         ---Book: "DomainconfigTabPage"
                                             ---Page:"DomainConfigGeneralPage"
                                             ---Page:"DomainConfigJtaPage"
                                             ---Page:"DomainConfigEJBPage"
                                             ---Page:"DomainConfigWebAppPage"
                                             ---Page:"DomainConfigSnmpPage"
                                             ---Page:"DomainConfigLoggingPage"
                                             ---Page:"DomainConfigLogFilterTablePage"
                                         ---Book: "DomainMonitorTabPage"
                                             ---- ...
                             ---Book: "CoreDomainCreateLogFilterBook" (assistant)
                                 ---- ...
                             ---- ...
                     ---Page
                         ---Book: "ServerBook"
                             ---- ...
                     ---- ...
```

The Look and Feel in the Administration Console

A Look and Feel is a collection of images, cascading style sheets, XML files, and other file types that control the physical appearance of a portal application. For example, it defines the fonts and colors used by the Administration Console, the layout of portal components, and the navigation menus.

Creating a simple Look and Feel that contains your company's logos, fonts, and color scheme requires you to copy a sample Look and Feel that WebLogic Server provides and then replace the logos and some cascading style sheet (CSS) definitions. Making complex changes to the WebLogic Server Look and Feel, such as changing the layout of portal components and navigation menus, requires an advanced knowledge of WebLogic Portal Look and Feels. If you have a license for WebLogic Workshop, you can use its Look and Feel editor to make these complex changes. For more information about Look and Feels, see the *Portal User Interface Framework Guide*.

Note: Because the Administration Console uses only the WebLogic Portal Framework, it supports only a single Look and Feel. Portal applications that use the entire set of features available with a license for the BEA WebLogic Portal product can support multiple Look and Feels that are personalized based on user or group ID.

JSP Templates and Tag Libraries

Most of the content in the Administration Console is rendered by JSPs, and most of these JSPs import one of several template JSPs to provide a basic structure for the content. In addition, the JSPs use several JSP tag libraries to render such UI features as tables, data-entry boxes, and buttons.

Your console extensions can follow this same pattern and use the Administration Console's JSP templates and tag libraries. For example, by using JSP tags and templates, less than 45 lines of code are needed to generate the Server table page in Figure 2-4 (see Listing 2-1).

Listing 2-1 JSP for Server Table Page

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/console-html.tld" prefix="wl" %>
<%@ taglib uri="/WEB-INF/fmt.tld" prefix="fmt" %>
<%@ taglib uri="/WEB-INF/beehive-netui-tags-template.tld"
prefix="beehive-template" %>

<fmt:setBundle basename="core" var="current_bundle" scope="page"/>
```

```
<beehive-template:template templatePage="/layouts/tableBaseLayout_netui.jsp">

   <beehive-template:section name="configAreaIntroduction">
      <fmt:message key="core.server.servertable.introduction"
            bundle="${current_bundle}"/>
   </beehive-template:section>

   <beehive-template:section name="table">
      <wl:table name="extensionForm"
             property="contents"
             showcheckboxes="true"
             captionEnabled="true"
             controlsEnabled="true"
             checkBoxValue="handle"
             bundle="core"
             formEnabled="true"
             singlechange="false">

         <wl:caption>
            <fmt:message key="server.table.caption"
                 bundle="${current_bundle}"/>
         </wl:caption>
         <wl:column property="name" label="server.table.label.name">
            <wl:column-dispatch perspective="configuration-page"/>
         </wl:column>
         <wl:column property="clusterName"
            label="server.table.label.cluster"/>
         <wl:column property="machineName" label="server.table.label.machine"/>
      </wl:table>

   </beehive-template:section>

</beehive-template:template>
```

WebLogic Server JSP Templates

Table 2-1 describes the JSP templates that you can use for your Administration Console extensions. All of the templates are located in the `/layouts` directory, which is relative to the `WEB-INF` directory of the Administration Console. WebLogic Server does not publish the templates themselves, but "Creating Portlets That Match the Administration Console" on page 5-1 describes how to use them.

If these templates do not meet your needs, you can create your own templates and structure the content directly in your JSP.

Table 2-1 Administration Console JSP Templates

Template	Description
`tableBaseLayout_netui.jsp`	The Administration Console uses this template for all of its JSPs that render a single table (see Figure 2-4). To create the overall structure of the document, the template outputs an HTML table with two rows. The first row contains everything in the including document's `<beehive-template:section name="configAreaIntroduction">` tag, which is usually the document's introductory text. The second row contains everything in the including document's `<beehive-template:section name="table">` tag, which is usually a table that displays a list of WebLogic Server resources and a button bar for working with the resources. Listing 2-1 uses this template.
`configBaseLayout_netui.jsp`	The Administration Console uses this template for all of its JSPs that render an introductory description, an HTML form, and Save and Cancel buttons (see Figure 2-3). The template output depends on whether the user has privileges to modify the domain's configuration. If a user has permission, the template outputs an HTML table with four rows. The first and last rows display Save and Cancel buttons along with a message indicating whether the user has a lock on the configuration and can make changes. If a user does not have permission, the table does not contain these rows. The second row contains everything in the including document's `<beehive-template:section name="configAreaIntroduction">` tag, which is usually the document's introductory text. The third row contains everything in the including document's `<beehive-template:section name="form">` tag, which is a form that provides user-input controls and descriptions.

JSP Tag Libraries

For each of the tag libraries in Table 2-2, the Administration Console provides runtime support by default. If you want development support for these libraries (for example, if you use an integrated development environment that provides code completion for JSP tags), you must configure your development environment to include these tags. (See "Set Up Your Development Environment" on page 5-7.)

Note: You can create custom tag libraries or use additional tag libraries, but you must include all of the necessary support files for custom tag libraries in your extension JAR file. See *Programming WebLogic JSP Tag Extensions*.

Table 2-2 Included JSP Tag Library Support

Tag Library	Contains
`console-html.tld`	WebLogic Server JSP tags for creating HTML forms and tables that match the functionality of the forms and tables in the Administration Console. Use these tags only to extend the WebLogic Server Administration Console. The documentation for this tag library is in the *WebLogic Server 9.0 JSP Tags Reference*.

`beehive-netui-tags-template.tld`
`beehive-netui-tags-databinding.tld`
`beehive-netui-tags-html.tld`
	Apache Beehive JSP tags for associating JSPs with a JSP template, binding data, and generating basic HTML tags.
	You can download the Beehive distribution, which includes the tag libraries and documentation from http://incubator.apache.org/beehive/downloads.html.

Table 2-2 Included JSP Tag Library Support

Tag Library	Contains
`c.tld` `fmt.tld`	JavaServer Pages Standard Tag Library (JSTL) tags which provide core functionality common to many JSP applications. You can download the JSTL distribution from http://java.sun.com/products/jsp/jstl/downloads/index.html. The documentation for these tag libraries is in the *JSTL Tag Library Reference*.

`struts-bean.tld`
`struts-html.tld`
`struts-logic.tld`
`struts-nested.tld`
`struts-template.tld`
`struts-tiles.tld`
	Apache Struts tags for interacting with the Struts framework.
	You can download the Struts distribution from http://struts.apache.org/download.cgi.
	The documentation for these tag libraries is available from http://struts.apache.org/.

Example: How an Extension Finds and Displays Content

The following steps describe how the portal framework uses an extension's source files to find and display data. The steps describe an extension that uses WebLogic Server JSP tags and Struts actions:

1. The NetUI Extension file names a portlet file to be loaded.

   For example, the following stanza in a NetUI Extension file describes the location in which to display the extension and a portlet file to load into the location:

```
<page-extension>
   <page-location>
      <parent-label-location label="page"/>
      <page-insertion-point layout-location="0"
          placeholder-position="0"
      />
   </page-location>
   <portlet-content
      content-uri="/portlets/desktop/desktop_view.portlet"
       title="My Extension"
      orientation="top" default-minimized="false"
      instance-label="PortletExtensionInstanceLabel"
   />
</page-extension>
```

If your extension adds books or pages to the Administration Console, the NetUI Extension file names a portal include file (instead of a portlet file) that defines the books or pages. The portal include file names a portlet file to be loaded.

2. The portlet file names a Struts action to run.

   For example, the following stanza from a portlet file defines a portlet and names a Struts action to run:

```
<portal:root>
   <netuix:portlet
      definitionLabel="MyPortlet"
      title="my.portlet.title">
   <netuix:strutsContent module="/mycompany"
      action="MyExtensionAction"
      refreshAction="MyExtensionAction"/>
   </netuix:portlet>
</portal:root>
```

   You can create portlets that load JSPs or Beehive Page Flows instead of running a Struts action.

3. The Struts action populates a form bean (usually with data from WebLogic Server MBeans) and then forwards to a JSP.

4. The JSP uses JSP tags to display data in the form bean.

CHAPTER 3

Rebranding the Administration Console

This section describes how to create a WebLogic Portal Look and Feel and deploy it as an Administration Console extension. The extension enables you to replace some or all of BEA's logos, colors, and styles in the Administration Console.

Figure 3-1 illustrates the process. The steps in the process, and the results of each are described in Table 3-1. Subsequent sections detail each step in the process.

Figure 3-1 Administration Console Extension Development Overview

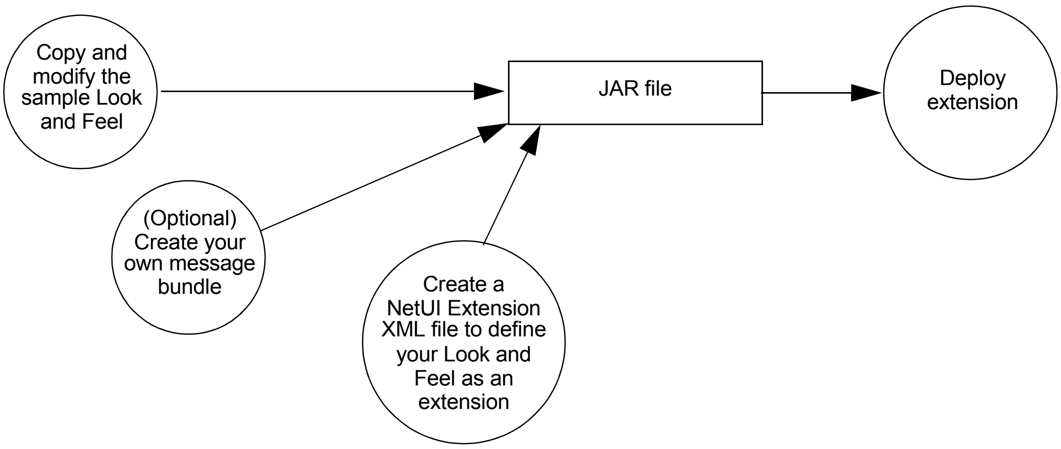

Copy and modify the sample Look and Feel
JAR file
Deploy extension
(Optional) Create your own message bundle
Create a NetUI Extension XML file to define your Look and Feel as an extension

Table 3-1 Model MBean Development Tasks and Results

Step	Description	Result
1. "Copy and Modify the Sample Look and Feel: Main Steps" on page 3-2.	BEA installs a sample Look and Feel that you use as a starting point. Replace the images and styles in this sample with your own.	A Look and Feel that contains your logos and styles.
2. (Optional) "Create a Message Bundle" on page 3-6.	If you want to change the text messages displayed in the banner, login, and login error pages, create your own message bundle and modify the pages to use messages from your bundle.	Localized properties files that contain your messages.
3."Modify the Sample NetUI Extension File" on page 3-9.	The NetUI Extension file is the deployment descriptor for your extension. It describes the locations of files and directories in your Look and Feel.	A deployment descriptor for your extension.
4. "Archive and Deploy the Extension" on page 3-10.	Archive the Look and Feel extension in a JAR file and copy it to your domain's `console-ext` directory.	When the Administration Console starts in your domain, it uses the Look and Feel extension that is in the domain's `console-ext` directory instead of the Look and Feel that BEA packages and installs.

Copy and Modify the Sample Look and Feel: Main Steps

To create a simple extension that replaces the BEA logos and colors with your own:

1. Copy all files from the following directory into your own development directory:

   `WL_HOME/samples/server/medrec/console-extension`

   where *`WL_HOME`* is the directory in which you installed WebLogic Server.

2. Change the name of the `medrec` directory under *`dev-dir`*`/framework/skins` and *`dev-dir`*`/framework/skeletons` to a name that you choose.

   where *`dev-dir`* is the name of your development directory.

3. "Modify the Administration Console Banner" on page 3-3.

4. "Modify Colors, Fonts, Buttons, and Images" on page 3-4.
5. "Modify Themes for the Change Center and Other Portlets" on page 3-5.
6. "Modify the Login and Error Page" on page 3-6.

Making more complex changes to the WebLogic Server Look and Feel, such as changing the layout of portal components and navigation menus, requires an advanced knowledge of WebLogic Portal Look and Feels. If you have a license for WebLogic Workshop, you can use its Look and Feel editor to make these complex changes. For more information, see the *Portal User Interface Framework Guide*.

Modify the Administration Console Banner

To overwrite the MedRec Look and Feel's image files with your company's image files:

1. To replace the logo in the Administration Console banner, save your own logo file as `dev-dir/framework/skins/medrec/images/banner_logo.gif`.

   To prevent the need to resize the banner frame, do not make your image any taller than 42 pixels.

2. To replace the ALT text for the logo, open `dev-dir/framework/skeletons/medrec/header.jsp` and replace `Your Message Here - <bean:message key="login.wlsident"/>` with your text.

   If you want to provide localized strings, use the JSTL `<fmt:message>` tag. See "Create a Message Bundle" on page 3-6.

3. To change the background color of the banner, replace the following image file with one of the same size but that contains a different color: `dev-dir/framework/skins/medrec/images/banner_bg.gif`

To make more complex modifications, you can change the JSP and styles that render the banner. The `dev-dir/framework/skeletons/medrec/header.jsp` file determines the contents of the Administration Console banner. Within `header.jsp`, the style `bea-portal-body-header-logo` specifies the name and location of the logo file. The style `bea-portal-body-header` specifies the name and location of an image file that is used as the banner background. Both of these styles are defined in `dev-dir/framework/skins/medrec/css/body.css`.

Modify Colors, Fonts, Buttons, and Images

The Administration Console uses several cascading style sheets (CSS) to specify its fonts and colors. To change these styles, open the style sheet and change the style's definition. Table 3-2 summarizes the CSS files that the Administration Console uses. All of these files are located in the `*dev-dir*/framework/skins/medrec/css` directory.

Table 3-2 CSS Files That Define General Colors and Fonts

CSS File	Description
`wls.css`	Contains WebLogic Server styles for the following areas: • General definitions for elements such as `body`, `a`, `h1`, and `h2` • Data tables • Form fields • WebLogic Server form buttons • Error messages • Toolbar content • Breadcrumbs content • General styles for How Do I..., System Status, and Change Center portlets
• `body.css` • `book.css` • `button.css` • `form.css` • `layout.css` • `portlet.css` • `window.css`	Contain WebLogic Portal framework styles for the following areas (some of which are not used by the Administration Console): • Portal header and footer • Book, page, and menu styles • Button styles • Form, input, and text area styles • Layout and placeholder styles • Portlet styles

The buttons in the Administration Console use a repeating background image to render the blue fade (and grey for inactive buttons). The image files for these buttons are located in the following directory:

```
dev-dir/framework/skins/medrec/images
```

Modify Themes for the Change Center and Other Portlets

Several portlets in the Administration Console use a theme, and you can change the definitions of these themes. Themes are similar to Look and Feels but the scope of a theme is limited to a section of a portal, such as a book, page, or portlet. A theme can be used to change the look and feel of the components of a portal without affecting the portal itself.

For example, the Change Center portlet uses its own theme to distinguish its buttons from the other form buttons in the Administration Console.

To change the color of a theme's buttons or title bars, change the images and styles in the theme's `skins` directory. Table 3-2 summarizes the directories that contain CSS files and images for theme skins. All of these directories are under the *`dev-dir`*`/framework/skins/medrec` directory. For information about modifying skin themes, see Creating Skins and Skin Themes in *WebLogic Workshop Online Help*.

Table 3-3 Skins for Administration Console Themes

Skin Directory	Description
`wlsbreadcrumbs`	Defines fonts and spacing for the breadcrumbs portlet, which displays above the tabbed interface and provides a navigation history.
`wlschangemgmt`	Defines buttons, fonts, title bar background, and spacing for the Change Center portlet.
`wlsmessages`	Defines buttons, fonts, title bar background, and spacing for the messages portlet, which displays only when the Administration Console has validation or confirmation messages.
`wlsnavtree`	Defines buttons, fonts, title bar background, and spacing for the NavTreePortlet.
`wlsquicklinks`	Defines buttons, fonts, title bar background, and spacing for the How Do I... portlet.
`wlsstatus`	Defines buttons, fonts, title bar background, and spacing for the System Status portlet.
`wlstoolbar`	Defines fonts and spacing for the breadcrumbs portlet, which displays in the banner and contains the Home, Help, and AskBEA buttons.
`wlsworkspace`	Defines borders, spacing, and background colors of the books and pages in the ContentBook area of the Administration Console.

Each theme is made up of a skin **and** a skeleton. The skeleton defines the overall structure of the portlet contents. The definition for each theme's skeleton is under the `dev-dir/framework/skeletons/medrec` directory. For information about modifying skeleton themes, see Creating Skeletons and Skeleton Themes in *WebLogic Workshop Online Help*.

Modify the Login and Error Page

The login page asks users to enter a user ID and password. The login error page displays if users enter invalid data. Both of these pages are displayed before the Administration Console loads its portal desktop. Therefore, these pages do not use the portal's Look and Feel and their image and stylesheet files are not under the `dev-dir/framework` directory. Table 3-4 summarizes the files and directories that determine the appearance of the login and login error pages.

Table 3-4 Files for the Login and Login Error Page Appearance

File	Description
`dev-dir/common/login.css`	Defines fonts and spacing for the login page.

`dev-dir/images/`
`login_banner_bg.gif`
`login_banner_right.gif`
`login_banner.gif`
`login_bottom.gif`
	Images for the login page.

`dev-dir/login/`
`LoginError.jsp`
`LoginForm.jsp`
	Render the login and login error pages.
	If you want to change the text that these pages display, modify the `<fmt:message/>` JSP tags to point to messages in your own message bundle. See "Create a Message Bundle" on page 3-6.

Create a Message Bundle

In the banner, login, and login error pages, the Administration Console uses JSTL tags to load text messages from localized properties files. For example, to display the window title in `LoginForm.jsp`:

1. The `<fmt:setBundle basename="global" var="current_bundle" scope="page"/>` tag in `LoginForm.jsp` sets the current message bundle to `global`.

This JSP tag looks in `WEB-INF/classes` for files with the following name pattern: `bundle[-locale].properties`.

The default properties file for this bundle is `WEB-INF/classes/global.properties`. If the Web browser or operating system specifies a different locale, then the JSP tag would load `WEB-INF/classes/global_locale.properties`.

2. The `<fmt:message key="window.title" bundle="${current_bundle}" />` tag opens the `global.properties` file and renders the text that is identified by the `window.title` key:

   ```
   window.title=BEA WebLogic Server Administration Console
   ```

If you want to change these messages, you can create your own properties files and modify the JSP tags to use your bundle.

Table 3-5 describes the text messages that the banner, login, and login error pages display.

Table 3-5 Messages in Banner, Login, and Login Error Pages

File	Message Key and Value

dev-dir/login/LoginForm.jsp	`window.title=BEA WebLogic Server Administration Console`
	`login.wlsident=BEA WebLogic Server Administration Console`
	`login.welcome2=Log in to work with the WebLogic Server domain`
	`login.username=Username:`
	`login.password=Password:`
	`login.submit=Log In`

dev-dir/login/LoginError.jsp	`window.title=BEA WebLogic Server Administration Console`
	`login.wlsident=BEA WebLogic Server Administration Console`
	`loginerror.authdenied=Authentication Denied`
	`loginerror.passwordrefused=The username or password has been refused by WebLogic Server. Please try again.`
	`login.username=Username:`
	`login.password=Password:`
	`login.submit=Log In`

dev-dir/framework/skeleton/medrec/header.jsp	`window.title=BEA WebLogic Server Administration Console`

To provide your own messages for the banner, login, and login error pages:

1. Change the name of the message bundle in the login and login error pages:
   a. Open the files listed in Table 3-5.
   b. In the `<fmt:setBundle basename="global" var="current_bundle" scope="page"/>` tag, change the value of the `basename` attribute.

BEA recommends that you include your company name in the bundle name. For example:

```
<fmt:setBundle basename="mycompany" var="current_bundle"
scope="page"/>
```

2. Create the following directory:
   `dev-dir/WEB-INF/classes`

3. In this directory, create a text file named `bundle.properties`

   where *`bundle`* is the bundle name that you specified in the previous step. For example, `mycompany.properties`.

4. Copy and paste into your properties file the message keys and values from Table 3-5. Delete any duplicate key/value pairs.

5. Modify the values in your properties file.

6. To create localized properties, save your properties file under the following name:
   `bundle[_locale].properties`

   where *`locale`* is a locale code supported by `java.util.Locale`. See `Locale` in the *J2SE API Specification*.

   For example, `mycompany_ja.properties`.

Modify the Sample NetUI Extension File

A NetUI Extension file is the deployment descriptor for your Look and Feel. It contains the names and locations of the files in your Look and Feel, and it causes the Administration Console to replace its Look and Feel with yours. For more information, see the *NetUI Extensions Schema Reference*.

The sample file is in the following location:
`dev-dir/WEB-INF/netuix-extension.xml`

To modify this file:

1. Open the file in a validating XML editor (recommended) or a text editor.

2. In the `<provider-info>` element, change the information to describe your Look and Feel, developer contact and support URL.

   The information in this element has no programmatic significance. It is intended to help your technical support team keep track of your software modifications.

3. In the `<look-and-feel-content>` element:

   a. In the `title`, `skin`, and `skeleton` attributes, replace the `medrec` value with the name of the directory you chose in step 2 in "Copy and Modify the Sample Look and Feel: Main Steps" on page 3-2.

   b. In the `definitionLabel` and `markupName` attributes, replace the `medrec` value with the name of the directory you chose in step 2 or use some other string. These attributes are required by the portal framework, but are not used in a Look and Feel extension.

Archive and Deploy the Extension

If you are creating additional extensions for the Administration Console, all extensions must be archived in a single JAR file.

To archive and deploy your extensions:

1. Archive your development directory into a JAR file. The name of the JAR file has no programmatic significance, so choose a name that is descriptive for your purposes.

   The contents of your *`dev-dir`* directory must be the root of the archive; the *`dev-dir`* directory name itself must not be in the archive. If you use the Java `jar` command to create the archive, enter the command from the *`dev-dir`* directory. For example:

   ```
   c:\dev-dir\> jar -cf my-extension.jar *
   ```

2. Copy the JAR file into each domain's *`domain-dir`*`/console-ext` directory, where *`domain-dir`* is the domain's root directory.

3. Restart the Administration Server for each domain.

Adding Portlets and Navigation Controls

In the Administration Console, all content is contained within portlets, so even the most minimal extension must define a portlet (and content for the portlet). You can add your portlet directly to the desktop, but if you want your portlet to display as a tab or subtab in the `ContentBook`, you must define books or pages to contain it. Your extension can also add a node to the NavTreePortlet, which enables users to navigate to your portlet directly from the desktop.

This section describes how to add portlets that contain simple, static content to the Administration Console. For information on adding portlets with dynamic content, such as portlets that launch Apache Struts actions, see "Creating Portlets That Match the Administration Console" on page 5-1.

Figure 4-1 illustrates the process. The steps in the process, and the results of each are described in Table 4-1. Subsequent sections detail each step in the process.

Figure 4-1 Adding Portlets and Navigation Controls Development Overview

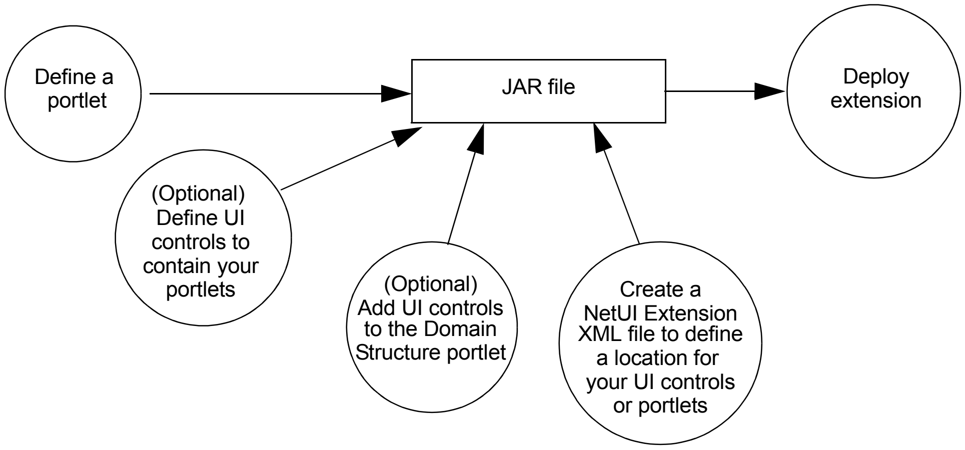

Define a portlet
JAR file
Deploy extension
(Optional) Define UI controls to contain your portlets
(Optional) Add UI controls to the Domain Structure portlet
Create a NetUI Extension XML file to define a location for your UI controls or portlets

Table 4-1 Model MBean Development Tasks and Results

Step	Description	Result
1. "Define a Portlet" on page 4-3.	Create an XML file to define a portlet that the portal framework can instantiate. A portlet definition includes instructions on which type of data to load: JSPs, HTML files, Struts actions, or Beehive Page Flows. The portal's Look and Feel determines whether the portlet provides borders and minimize/maximize controls.	A `.portlet` XML file.
2."Define UI Controls (Optional)" on page 4-5.	If you want your portlet to display in a tab, subtab, or in some other location within `ContentBook`, create an XML file that defines a page or book.	A `.pinc` XML file.

Table 4-1 Model MBean Development Tasks and Results

Step	Description	Result
3. "Add Nodes to the Domain Structure Portlet (Optional)" on page 4-11.	You can create a link from the NavTreePortlet to any book or page in your extension. WebLogic Server provides default support for appending control names to the end of the existing navigation tree. If you want to insert nodes in specific locations, or if you want to create a node tree, you create your own Java classes that describe the node and node location.	Additional entries in the `.pinc` XML file. Optionally, Java classes that give you more control over the node that you are adding.
4. "Specify a Location for Displaying Portlets or UI Controls" on page 4-15.	Create an XML file that describes whether you want your portal to display next to a labeled UI control or to replace the control.	A `netuix-extension.xml` file.
5. "Archive and Deploy the Extension" on page 4-20.	Organize your extension files into a standard Web application directory structure. Then archive the directory tree as a JAR file and copy it to your domain's `console-ext` directory.	A JAR file that automatically deploys.

Define a Portlet

To create `.portlet` XML file that defines a portlet:

1. Create a development directory that will serve as the root of your Administration Console extension. Consider creating a subdirectory to contain all of the portlet XML files in your extension. For example, `*dev-dir*/PortalConfig`
   where `*dev-dir*` is your development directory.

   Subsequent sections will instruct you to add other files and directories to your development directory. For more information, see "Archive and Deploy the Extension" on page 4-20.

2. Copy the code from Listing 4-1 and paste it into a new text file in `*dev-dir*/portlets` or some other directory below `*dev-dir*`.

   Consider using the following naming convention:
   `*content-name*.portlet`

where *`content-name`* is the name of a JSP or HTML file that the portlet contains. For example, if the portlet contains a JSP file named `monitorEJB.jsp`, then name the portlet XML file `monitorEJB.portlet`.

3. Replace the values in Listing 4-1 as follows:
   - *`Label`*. Provide a unique identifier that the portal framework uses to identify this portlet.
   - (optional) *`Title`*. Provide a default title that this portlet displays in its title bar when the Look and Feel causes the title bar to be displayed. You can override this default for each instance of this portlet (see "Create a NetUI Extension XML File" on page 4-17).
   - *`URI`*. Specifies the absolute path and file name of the JSP that the portlet contains starting from the root of the extension.

     For example:
     `/monitoring/monitorEJB.portlet`
     `/monitoring/monitorEJB.JSP`

   For more information about portlet XML files, see the *Portal Support Schema Reference*.

Listing 4-1 Template for a Simple .portlet XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root xmlns:html="http://www.w3.org/1999/xhtml-netuix-modified/1.0.0"
   xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
   xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/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-support-1_0_0.xsd">

   <netuix:portlet definitionLabel="Label" title="Title" >
      <netuix:content>
         <netuix:jspContent contentUri="URI"/>
      </netuix:content>
   </netuix:portlet>

</portal:root>
```

Define UI Controls (Optional)

The `ContentBook` is made up of a hierarchy of books and pages. All portlets in `ContentBook` display within a book or a page, and if you want to add portlets to the `ContentBook`, you must create a book or page that conforms to the existing hierarchy:

- To create a top-level tab (such as a sibling of Domains: Configuration), you create a book that contains one or more pages. Each page contains a portlet.
- To create a subtab of an existing tab (such as a sibling of Domains: Configuration: General), you create a page that contains a portlet.

Save the definitions of your books and pages in one or more portal include (`.pinc`) files. Create one `.pinc` file for each hierarchical grouping of controls. For example, create one `.pinc` file for a book that creates a top-level tab and its subtabs. Create another `.pinc` file for a page that adds a subtab to an existing WebLogic Server tab. The root element of a `.pinc` file (`portal:root`) can have only one direct child element; the child element can have multiple children.

The following sections describe creating books and pages:

- "Create a Top-Level Tab That Contains a Subtab" on page 4-5
- "Create a Top-Level Tab That Does Not Contain a Subtab" on page 4-8
- "Create a Subtab in a WebLogic Server Top-Level Tab" on page 4-9
- "Create a Control Without Tabs or Subtabs" on page 4-10
- "Navigating to a Custom Security Provider Page" on page 4-10

Create a Top-Level Tab That Contains a Subtab

To create a portal include (`.pinc`) XML file that defines a top-level tab and one subtab:

1. Copy the code from Listing 4-2 and paste it into a new text file. Save the file in a directory below *`dev-dir`*.

   For example, *`dev-dir`*`/PortalConfig/MyApp.pinc`
   where *`dev-dir`* is your development directory. For more information, see "Archive and Deploy the Extension" on page 4-20.

2. Replace the values in Listing 4-2 as follows:

   – *`Book-Label`*. Provide a unique identifier that the portal framework and WebLogic Server JSP tags use to forward requests to the book. This is the same type of label that

WebLogic Server provides for many of its UI controls. See "UI Controls in the Administration Console" on page 2-3.

- `Book-Title`. Provide either the text that users see as the name of the tab or a key in a message bundle that you have created. The Administration Console assumes that this value is a key, but if it cannot find the key in the specified bundle, it displays the value.

  The Administration Console looks in the extension's `/WEB-INF/classes` directory for message bundles.
- `Bundle`. Specify the name of a message bundle that you have created. See "Create and Use a Message Bundle" on page 5-3.
- `Page-Label`. Provide a unique identifier that the portal framework and WebLogic Server JSP tags use to forward requests to the page.
- `Page-Title`. Provide either the text that users see as the name of the subtab or a key in a message bundle that you have created.
- `Metadata-Type` and `Metadata-ID`. If you want to use the Administration Console's `<wl:column-dispatch>` JSP tag to create a hypertext link that forwards to this page, include a `<netuix:meta>` element and supply values for `Metadata-Type` and `Metadata-ID`. See column-dispatch in the *WebLogic Server JSP Tag Reference*.
- `Portlet-Instance-Title`. This value is required by the portal support schema, but it is not used in this context. The portal framework displays the value in the portlet's title bar, but the Administration Console's Look and Feel specifies that portlets at this level of the hierarchy do not display title bars.
- `Portlet-Instance-Label`. Provide a unique identifier that the portal framework and WebLogic Server JSP tags use to forward requests to the portlet instance.
- `Portlet-URI`. Specify the path and file name of a portlet file that you created (see "Define a Portlet" on page 4-3). The path must be relative to the root of the portal Web application.

  For example:
  `/monitoring/monitorEJB.portlet`

3. To create additional subtabs, add `netuix:page` elements as siblings to the `netuix:page` element in Listing 4-2.

For more information about portal include XML files, see the *Portal Support Schema Reference*.

Note the use of the following elements in the `.pinc` file:

- `netuix:singleLevelMenu` renders one subtab for each page in the book. The book's parent UI control (which Listing 4-2 assumes is provided by WebLogic Server) is responsible for generating a top-level tab for the book.
- `netuix:meta name="breadcrumb-context" content="handle"` adds the page's title to the history of visited pages (breadcrumbs) after a user has visited the page. The breadcrumbs display on the desktop above `ContentBook`.

Listing 4-2 Template for a .pinc File That Defines a Top-Level Tab with Subtabs

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root
   xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0"
   xmlns:html="http://www.w3.org/1999/xhtml-netuix-modified/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
   xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/portal/support
      /1.0.0 portal-support-1_0_0.xsd">

   <netuix:book markupName="book" markupType="Book"
      definitionLabel="Book-Label" title="Book-Title">
      <netuix:singleLevelMenu markupType="Menu" markupName="singleLevelMenu"
         skeletonUri="singlelevelmenu_children.jsp"/>
      <netuix:meta name="skeleton-resource-bundle" content="Bundle"/>
      <netuix:content>
         <netuix:page markupName="page" markupType="Page"
            definitionLabel="Page-Label" title="Page-Title"
            skeletonUri="/framework/skeletons/default/wlsworkspace/
               page_content.jsp">
            <netuix:meta name="Metadata-Type" content="Metadata-ID"/>
            <netuix:meta name="breadcrumb-context" content="handle"/>
            <netuix:meta name="skeleton-resource-bundle" content="Bundle"/>
            <netuix:content>
               <netuix:gridLayout columns="1" markupType="Layout"
                  markupName="singleColumnLayout">
                  <netuix:placeholder flow="vertical" markupType="Placeholder"
                     markupName="singleColumn_columnOne">
                     <netuix:portletInstance markupType="Portlet"
                        title="Portlet-Instance-Title"
                        instanceLabel="Portlet-Instance-Label"
                        contentUri="Portlet-URI"/>
                  </netuix:placeholder>
               </netuix:gridLayout>
            </netuix:content>
         </netuix:page>
         <!-- Add additional netuix:page elements here -->
      </netuix:content>
   </netuix:book>
</portal:root>
```

Create a Top-Level Tab That Does Not Contain a Subtab

To create a portal include (`.pinc`) XML file that defines a top-level tab and no subtabs (such as Domains: Notes):

1. Copy the code from Listing 4-3 and paste it into a new text file.

   For example, *`dev-dir`*`/PortalConfig/MyApp.pinc`
   where *`dev-dir`* is your development directory. For more information, see "Archive and Deploy the Extension" on page 4-20.

2. Replace the values in Listing 4-3 as follows:

   – *`Page-Label`*. Provide a unique identifier that the portal framework and WebLogic Server JSP tags use to forward requests to the page.

   – *`Page-Title`*. Provide either the text that users see as the name of the subtab or a key in a message bundle that you have created.

   – *`Bundle`*. Specify the name of a message bundle that you have created. See "Create and Use a Message Bundle" on page 5-3.

   – *`Portlet-Instance-Title`*. This value is required by the portal support schema, but it is not used in this context. The portal framework displays the value in the portlet's title bar, but the Administration Console's Look and Feel specifies that portlets at this level of the hierarchy do not display title bars.

   – *`Portlet-Instance-Label`*. Provide a unique identifier that the portal framework and WebLogic Server JSP tags use to forward requests to the portlet instance.

   – *`Portlet-URI`*. Specify the path and file name of a portlet file that you created (see "Define a Portlet" on page 4-3). The path must be relative to the root of the portal Web application.

     For example:
     `/monitoring/monitorEJB.portlet`

Note that Listing 4-3 defines a page, not a book, so the Administration Console Look and Feel will render the page as a tab with no subtabs.

Listing 4-3 Template .pinc File that Creates a Top-Level Tab with No Subtabs

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root
   xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/1.0.0"
   xmlns:html="http://www.w3.org/1999/xhtml-netuix-modified/1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
   xsi:schemaLocation="http://www.bea.com/servers/netuix/xsd/portal/support
      /1.0.0 portal-support-1_0_0.xsd">
   <netuix:page markupName="page" markupType="Page"
      definitionLabel="Page-Label" title="Page-Title"
      skeletonUri="/framework/skeletons/default/wlsworkspace/
         page_content.jsp">
         <netuix:meta name="skeleton-resource-bundle" content="Bundle"/>
         <netuix:content>
            <netuix:gridLayout columns="1" markupType="Layout"
               markupName="singleColumnLayout">
            <netuix:placeholder flow="vertical" markupType="Placeholder"
               markupName="singleColumn_columnOne">
               <netuix:portletInstance markupType="Portlet"
                  title="Portlet-Instance-Title"
                  instanceLabel="Portlet-Instance-Label"
                  contentUri="Portlet-URI"/>
            </netuix:placeholder>
         </netuix:gridLayout>
      </netuix:content>
   </netuix:page>
</portal:root>
```

Create a Subtab in a WebLogic Server Top-Level Tab

To add a subtab to an existing WebLogic Server tab, use the same code from Listing 4-3.

The location that you specify for the control determines whether the Administration Console renders it as a top-level tab or a subtab. For example, if you specify that the control is a child of the `DomainConfigGeneralPage` book, then it displays as a subtab of the Domains: Configuration tab. If you specify that the control is a child of the `CoreDomainConfigGeneralBook` book, then it displays as a top-level tab and a sibling of the Domains: Configuration tab.

See "Specify a Location for Displaying Portlets or UI Controls" on page 4-15.

Create a Control Without Tabs or Subtabs

There is no requirement for books and pages in `ContentBook` to be accessible by tab or subtab. Many WebLogic Server pages that display summary tables are accessible from the NavTreePortlet but not from the tabbed interface (see Figure 2-4).

Any of the code listings in the previous sections can be located in a parent control that does not render tabs or subtabs for its children. See "Specify a Location for Displaying Portlets or UI Controls" on page 4-15.

Navigating to a Custom Security Provider Page

If you created a custom security provider and used WebLogic MBeanMaker to create MBeans to manage your provider, the Administration Console automatically generates pages to display the provider's configuration data. It also generates a link to your provider pages from the Security: Providers table.

However, you can create your own pages to customize this display. If you create your own pages, you need to redirect the link in the Security: Providers table from the pages that the Administration Console generates to your custom pages.

To redirect the link, include the following element as a child of your page's `<netuix:page>` element:

```
<netuix:meta type="configuration" content="MBean-class-name"/>
```

where *`MBean-class-name`* is the fully qualified name of your provider's MBean class.

For example:

```
<netuix:page markupName="page" markupType="Page"
   definitionLabel="SimpleSampleAuthorizerAuthorizerConfigCommonTabPage"
   title="tab.common.label"
   skeletonUri="/framework/skeletons/default/wlsworkspace
      /page_content.jsp">
   <netuix:meta name="configuration"
   content="examples.security.providers.authorization.simple.
      SimpleSampleAuthorizerMBean"/>
   <netuix:content>
...
```

Add Nodes to the Domain Structure Portlet (Optional)

You can create a link from the portlet whose user-visible title is Domain Structure (NavTreePortlet) to any book or page in your extension. The following sections describe adding nodes to the NavTreePortlet:

- "Append a Single Node" on page 4-11
- "Append or Insert Nodes or Node Trees" on page 4-11

Append a Single Node

To append a node that links to one of your book or page controls, add the following attribute and attribute value to the `<netuix:book>` or `<netuix:page>` element:

```
backingFile="com.bea.console.utils.NavTreeExtensionBacking"
```

For example:

```
<netuix:book definitionLabel="MyAppTableBook" title="MyApp.title"
   markupName="book"
   markupType="Book"
   backingFile="com.bea.console.utils.NavTreeExtensionBacking"
>
```

The NavTreePortlet displays the value of the book or page element's `title` attribute as the link text. If the `title` attribute value is a key in your message bundle, the NavTreePortlet displays the localized value mapped to the key.

If you specify this `backingFile` attribute and value for multiple controls in your extension, the NavTreePortlet appends each node entry in the order in which you have declared the controls in the NetUI Extension file.

Append or Insert Nodes or Node Trees

If you want to control the location in which your node is added to the NavTreePortlet, or if you want to add a node that contains other nodes (a node tree):

1. Create a `NavTreeBacking` Java class.

   This class defines one or more nodes and specifies the location for your nodes. See "Create a NavTreeBacking Class" on page 4-12.

2. Add the following attribute and attribute value to the `<netuix:book>` or `<netuix:page>` element in your portal include (`.pinc`) file:
   `backingFile="`*`your-NavTreeBacking-class`*`"`

   where *`your-NavTreeBacking-class`* is the fully-qualified name of the class you created in step 1.

   If your `NavTreeBacking` Java class creates a node tree, add the `backingFile` attribute to the control that is at the top of the node tree. For example, if you are adding a node tree for a book and its pages, add the `backingFile` attribute to the `<netuix:book>` element.

Create a NavTreeBacking Class

The Java class that defines your nodes must extend `com.bea.console.utils.NavTreeExtensionBacking`. This class is already available in the WebLogic Server runtime environment. However, for support in your development and compiling environment, you must add the following JARs to your environment's classpath:

`WL_HOME``/server/lib/consoleapp/webapp/WEB-INF/lib/console.jar`
`WL_HOME``/server/lib/consoleapp/webapp/WEB-INF/lib/netuix_servlet.jar`

where *`WL_HOME`* is the location in which you installed WebLogic Server.

To create a `NavTreeBacking` class (see Listing 4-4):

1. Extend `com.bea.console.utils.NavTreeExtensionBacking`.

2. Override the `NavTreeExtensionBacking.getTreeExtension( PageBackingContext ppCtx, String extensionUrl)` method.

   The portal framework passes the `ppCtx` and `extensionUrl` parameter values to your `NavTreeBacking` class when it loads a control that contains the `backingFile="`*`your-NavTreeBacking-class`*`"` attribute.

   `PageBackingContext` is a class in the portal framework that contains data about a portal control and methods for retrieving the data. For example, it contains the value of a control's `definitionLabel`. The `ppCtx` instance that is passed describes the control that invoked the `getTreeExtension()` method. See `com.bea.netuix.servlets.controls.page.PageBackingContext` in the *WebLogic Portal 8.1.4 API Reference*.

   The `extensionUrl` value is a URL to the control that invoked the `getTreeExtension()` method.

   In your implementation of `getTreeExtension()`:

a. Construct a `com.bea.jsptools.tree.TreeNode` object for the node that you want to add.

   Use the following constructor for a parent node (or for a node that contains no children):
   `TreeNode(String nodeId, String nodeName, String nodeUrl)`
   where:

   `nodeId` is the value of the control's `definitionLabel`. You can use `PageBackingContext.getDefinitionLabel()` to get this value for the `PageBackingContext` instance that was passed as a parameter to your method. Alternatively, you can enter the `definitionLabel` value that is in the control's `.pinc` file.

   `nodeName` is the text that you want to display in the NavTreePortlet.

   `nodeURL` is a URL to the control. Supply *`extensionUrl`* as the value of this parameter.

b. If you want to add a tree of nodes, construct additional `TreeNode` objects as children of the top `TreeNode`.

   For each child node, use the following constructor:
   ```
   TreeNode(String nodeId, String nodeName,
      String nodeUrl, TreeNode parent)
   ```
   where:

   `nodeId` is the value of the control's `definitionLabel`. You can **not** use `PageBackingContext.getDefinitionLabel()` to get this value because the `PageBackingContext` available to this method is for the parent node. Instead, you must enter the `definitionLabel` value that is in the control's `.pinc` file.

   `nodeName` is the text that you want to display in the NavTreePortlet.

   `nodeURL` is a URL to the control. Supply the following value:
   `?_nfpb=true&_pageLabel=definitionLabel`
   where *`definitionLabel`* is the `definitionLabel` of the page or book to which you want to link.

   `parent` is any `TreeNode` that you have constructed. You can create multiple levels in your node tree by specifying a parent that is a child of node higher up in the hierarchy.

c. Pass the parent `TreeNode` object to the constructor for `com.bea.console.utils.NavTreeExtensionEvent`.

   The event is broadcast to a listener in the NavTreePortlet.

Use the following constructor:
```
NavTreeExtensionEvent(String pageLabel, String url,
   String parentPath, TreeNode node, int ACTION)
```
where:

`pageLabel` is the same `nodeID` value that you used when constructing the `TreeNode` object for the parent node.

`url` is the same `nodeURL` value that you used when constructing the `TreeNode` object for the parent node.

`parentPath` is the name of the node under which you want your node to display. Use `/` (slash) to represent the root of the navigation tree in the NavTreePortlet.

For example, if you want your node or node tree to display at the top level, specify `/`. If you want your node to display as a child of Environments, specify `/Environments`.

`node` is the parent TreeNode that you created in step a.

`ACTION` is `NavTreeExtensionEvent.APPEND_ACTION`. For information about other possible actions, see `NavTreeExtensionEvent` in the *WebLogic Portal 8.1.4 API Reference*.

d. Return the `NavTreeExtensionEvent` object that you constructed.

3. Save the compiled class in a package structure under your extension's `WEB-INF/classes` directory.

Listing 4-4 Example NavTreeExtensionBacking Class

```
package com.mycompany.consoleext;

import com.bea.netuix.servlets.controls.page.PageBackingContext;
import com.bea.jsptools.tree.TreeNode;
import com.bea.console.utils.NavTreeExtensionBacking;
import com.bea.console.utils.NavTreeExtensionEvent;

public class CustomNavTreeExtension extends NavTreeExtensionBacking {

   public NavTreeExtensionEvent getTreeExtension(PageBackingContext ppCtx,
      String extensionUrl){
     /*
      * Construct a TreeNode for the control that has invoked this method.
      */
      TreeNode node = new TreeNode(ppCtx.getDefinitionLabel(),
         ppCtx.getDefinitionLabel(),extensionUrl);
```

```
        /*
         * Construct a child TreeNode.
         */
         TreeNode node1 = new TreeNode("MyAppGeneralTabPage",
            "MyApp General",
            "?_nfpb=true&_pageLabel=MyAppGeneralTabPage",node);

        /*
         * Add the parent node (which includes its child) below the
         * Environment node in the NavTreePortlet.
         */
         NavTreeExtensionEvent evt =
           new NavTreeExtensionEvent(ppCtx.getDefinitionLabel(),extensionUrl,
           "/Environment",node, NavTreeExtensionEvent.APPEND_ACTION);

         return evt;
    }
}
```

Specify a Location for Displaying Portlets or UI Controls

All locations for displaying your portlets or UI controls must be specified as relative to existing controls in the Administration Console. For example, you can specify that your portlet displays on the desktop below the System Status portlet.

The following sections describe how to specify a location for your portlets or UI controls:

- "Deploy a Development Look and Feel to See UI Control Labels" on page 4-15
- "Create a NetUI Extension XML File" on page 4-17

Deploy a Development Look and Feel to See UI Control Labels

Most of the Administration Console UI controls are identified by a `definitionLabel`, and it is this value that you use for relative positioning of your UI control. WebLogic Server provides a Look and Feel that reveals the labels in the Administration Console user interface.

Note: If you plan only to add a portlet to the desktop, you do not need to deploy the development Look and Feel.

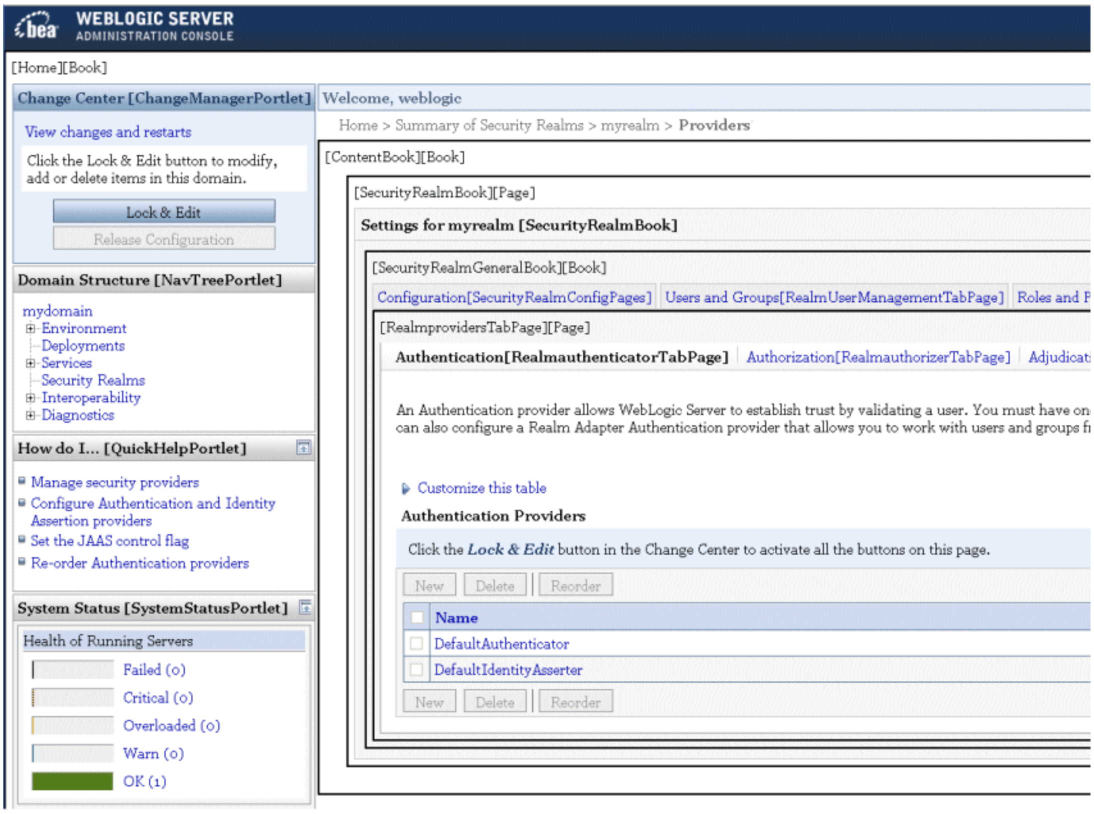
To use this Look and Feel:

1. Download the Look and Feel archive from Code Library on the dev2dev Web site.

2. Save the Look and Feel archive (`devlaf-1.0.jar`) in *`domain-root`*`/console-ext` where *`domain-root`* is the root directory of a domain in your development environment.
3. Restart the domain's Administration Server.
4. Log in to the Administration Console.

Each labeled control displays the value of its `definitionLabel` in brackets ([]) next to its user-visible title. In a separate pair of brackets, the control displays whether it is a book or a page. See Figure 4-2.

Figure 4-2 A Control Label in the Administration Console User Interface



[Home][Book]

Change Center [ChangeManagerPortlet]

View changes and restarts

Click the Lock & Edit button to modify, add or delete items in this domain.

Lock & Edit

Release Configuration

Domain Structure [NavTreePortlet]

mydomain
- Environment
- Deployments
- Services
- Security Realms
- Interoperability
- Diagnostics

How do I... [QuickHelpPortlet]

- Manage security providers
- Configure Authentication and Identity Assertion providers
- Set the JAAS control flag
- Re-order Authentication providers

System Status [SystemStatusPortlet]

Health of Running Servers

Failed (0)
Critical (0)
Overloaded (0)
Warn (0)
OK (1)

Welcome, weblogic

Home > Summary of Security Realms > myrealm > **Providers**

[ContentBook][Book]

[SecurityRealmBook][Page]

Settings for myrealm [SecurityRealmBook]

[SecurityRealmGeneralBook][Book]

Configuration[SecurityRealmConfigPages] | Users and Groups[RealmUserManagementTabPage] | Roles and P

[RealmprovidersTabPage][Page]

Authentication[RealmauthenticatorTabPage] | Authorization[RealmauthorizerTabPage] | Adjudicat

An Authentication provider allows WebLogic Server to establish trust by validating a user. You must have on can also configure a Realm Adapter Authentication provider that allows you to work with users and groups fr

Customize this table

Authentication Providers

Click the *Lock & Edit* button in the Change Center to activate all the buttons on this page.

New | Delete | Reorder

	Name
☐	DefaultAuthenticator
☐	DefaultIdentityAsserter

New | Delete | Reorder

Create a NetUI Extension XML File

A NetUI Extension XML file (`netuix-extension.xml`) is the deployment descriptor for your extension. It declares each parent UI control in your extension and the location in which you want it to display. For more information, see the *NetUI Extensions Schema Reference*.

To create a NetUI Extension XML file (see Listing 4-5):

1. Create an XML file named `netuix-extension.xml` and save it in
   `dev-dir``/WEB-INF`
   where *`dev-dir`* is your development directory. For more information, see "Archive and Deploy the Extension" on page 4-20.

2. Create a `<weblogic-portal-extension>` root element.

3. (Optional) Create a `<provider-info>` element to describe your extension.

   This element is for your information only. The portal framework does not use the data in this element.

4. Add the following element:
   `<portal-file>/console.portal</portal-file>`

   This required element specifies the name and relative location of the Administration Console's `.portal` file, which is the portal that you are extending.

5. To add a portlet to the Administration Console desktop, create the following stanza:

```
<page-extension>
   <page-location>
      <parent-label-location label="page"/>
      <page-insertion-point layout-location="0"
         placeholder-position="0"/>
   </page-location>
   <portlet-content
      content-uri="portlet-URI" title="title"
      orientation="top" default-minimized="false"
      instance-label="portlet-instance-label"/>
</page-extension>
```

   where:

   – ***`portlet-URI`*** is the path and file name of your `.portlet` file. The path must be relative to the root of the portal Web application.

   – ***`title`*** is the title that displays in the portlet's title bar. If you specify a null value, the portal framework uses the title that you defined in the `.portlet` file.

- ***`portlet-instance-label`*** is a unique identifier that the portal framework and WebLogic Server JSP tags use to forward requests to the portlet instance.

You can change the values of the `layout-location` and `placeholder-position` attributes to change the location of your portlet. For more information, see the *NetUI Extensions Schema Reference*.

6. To add a control that renders a tab, create the following stanza:

```
<book-extension>
   <book-location>
      <parent-label-location label="Admin-Console-Book-Label"/>
      <book-insertion-point action="append"/>
   </book-location>
   <book-content content-uri="pinc-URI"/>
</book-extension>
```

where:

- ***`Admin-Console-Book-Label`*** is the `definitionLabel` of an Administration Console book control that renders tabs for its child books.
- ***`pinc-URI`*** is the path and file name of your `.pinc` file that defines the book control for your tab (and optional subtabs). The path must be relative to the root of the portal Web application.

7. To add a control that renders a subtab in an existing tab, create the same stanza as the previous step, where:

- ***`Admin-Console-Book-Label`*** is the `definitionLabel` of an Administration Console book control that renders subtabs for its child pages.
- ***`pinc-URI`*** is the path and file name of your `.pinc` file that defines the page control for your subtab. The path must be relative to the root of the portal Web application.

8. Save the file as *`dev-dir`*`/WEB-INF/netuix-extension.xml` where *`dev-dir`* is the root directory of your extension.

Listing 4-5 Example netuix-extension.xml File

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-portal-extension
   xmlns="http://www.bea.com/servers/portal/weblogic-portal/8.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bea.com/servers/portal/weblogic-portal/
      8.0 netuix-extension-1_0_0.xsd">
   <provider-info>
      <title>My Extension</title>
      <version>1.0</version>
      <description>Inserts a portlet on the desktop, a tab next to
           Domains:Configuration, and a subtab under Domains: Configuration.
      </description>
      <author>Me</author>
      <last-modified>02/03/2005</last-modified>
      <support-url>http://www.mycompany/support/index.jsp</support-url>
   </provider-info>

   <portal-file>/console.portal</portal-file>

   <!--Adds a portlet to the console desktop -->
   <page-extension>
      <page-location>
         <parent-label-location label="page"/>
         <page-insertion-point layout-location="0" placeholder-position="0"/>
      </page-location>
      <portlet-content content-uri="/portlets/desktop/desktop_view.portlet"
         title="My App Status" orientation="top" default-minimized="false"
          instance-label="PortletExtensionInstanceLabel"
      />
   </page-extension>

   <!--Adds a tab to the Domain tabs -->
   <book-extension>
      <book-location>
         <parent-label-location label="CoreDomainConfigGeneralBook"/>
         <book-insertion-point action="append"/>
      </book-location>
      <book-content content-uri="/page/page.pinc"/>
   </book-extension>

<!-- Adds a subtab to the Domain: Configuration tab-->
   <book-extension>
      <book-location>
         <parent-label-location label="DomainconfigTabPage"/>
         <book-insertion-point action="append"/>
      </book-location>
      <page-content content-uri="/page/notespage.pinc"/>
   </book-extension>

</weblogic-portal-extension>
```

Archive and Deploy the Extension

To archive and deploy your extension:

1. Arrange your XML files, JSPs, Java classes in a directory tree that meets the requirements of a Web application.

   For example, the `netuix-extension.xml` file, which is the deployment descriptor for your extension, must be in a directory named:
   `root-dir/WEB-INF`

   If you are using a Java class to add nodes to the NavTreePortlet, the package structure must start in the following directory:
   `root-dir/WEB-INF/classes`

   If you are using a message bundle, the properties files must be in the following directory:
   `root-dir/WEB-INF/classes`

   All other resources can be in any directory structure that you choose, as long as it is below the extension's root directory (and as long as relative links from the `netuix-extension.xml` file and other XML files are correct).

2. Archive your extension directory into a JAR file. The name of the JAR file has no programmatic significance, so choose a name that is descriptive for your purposes.

   The contents of your *root-dir* directory must be the root of the archive; the *root-dir* directory name itself must not be in the archive. If you use the Java `jar` command to create the archive, enter the command from the *root-dir* directory. For example:
   ```
   c:\root-dir\> jar -cf my-extension.jar *
   ```

3. Copy the JAR file into each domain's `domain-dir/console-ext` directory, where *domain-dir* is the domain's root directory.

4. Restart the Administration Server for each domain.

Error Output During Deployment

If the Administration Console encounters deployment errors, it outputs error and warning messages to standard out and to the Administration Server's server log file.

If you do not see error or warning messages and you do not see your extension in the Administration Console, you might have named the wrong parent UI control in your `netuix-extension.xml` file. For example, if you name a parent UI control that does not render tabs for its children, then your extension is deployed but there is no menu control for accessing it.

Creating Portlets That Match the Administration Console

This section describes how to add a portlet that uses the Administration Console's JSP templates, styles, and user input controls. For example, you can add portlets that render your content as one of the following:

- A table in the `ContentBook` that summarizes the resources you have provided and that enables users to navigate to a specific resource or to invoke actions on the resource from the table. (See Figure 2-4 for an example of a WebLogic Server table.)
- A tab in the `ContentBook` that enables users to monitor or configure resources that you have provided.

Figure 5-1 illustrates the process. The steps in the process, and the results of each are described in Table 5-1. Subsequent sections detail each step in the process.

Figure 5-1 Administration Console Extension Development Overview

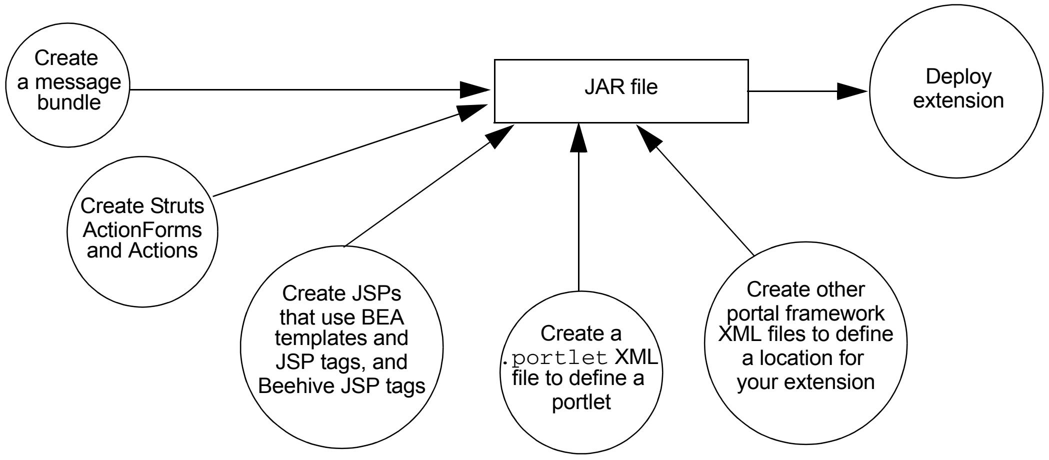

Create a message bundle
Create Struts ActionForms and Actions
Create JSPs that use BEA templates and JSP tags, and Beehive JSP tags
Create a .portlet XML file to define a portlet
Create other portal framework XML files to define a location for your extension
JAR file
Deploy extension

Table 5-1 Model MBean Development Tasks and Results

Step	Description	Result
1. "Create and Use a Message Bundle" on page 5-3.	Create a text file that contains a name/value pair for each text string that you want to display in your extension.	One or more `.properties` files.
2. "Create Struts ActionForms and Actions" on page 5-4.	To render HTML forms and tables that function as forms, the Administration Console uses JSP tags that load data from Java beans. To submit user input, the JSP tags forward to Struts Actions. Some JSP tags require the use of Struts ActionForms to populate the Java beans and make them available to the JSP. If you use Administration Console JSP tags, you must create your own Struts ActionForms and Actions.	A Struts configuration file, Java beans, and Java classes that implement `org.apache.struts.action.ActionForm` and `org.apache.struts.action.Action`.
3. "Create JSPs that Use BEA Templates and JSP Tags" on page 5-7.	WebLogic Server provides JSP templates that you can import into your JSPs. It also provides a JSP tag library to render the same UI controls that the Administration Console uses.	JSPs that match the Administration Console styles and structure.

Table 5-1 Model MBean Development Tasks and Results

Step	Description	Result
4. "Create a .portlet XML File" on page 5-14.	Create a portlet that causes Struts to instantiate your ActionForm. The ActionForm populates Java beans and forwards to a JSP.	A `.portlet` XML file that defines a portlet and configures it to launch a Struts action.
5."Create Other Portal Framework Files and Deploy the Extension" on page 5-15.	Create XML files that define a location for your extension. Then archive your extension files as a JAR file and copy it to your domain's `console-ext` directory.	A `.pinc` XML file that defines a page or book control (optional), a `netuix-extension.xml` file that describes where to locate your extension, and a JAR file that automatically deploys.

Create and Use a Message Bundle

BEA recommends that you define all of the text strings that your JSP displays in a message bundle.

To create and use a message bundle:

1. Create a text file that contains name/value pairs (properties) for each string you want to display. Use the equal sign (=) as the delimiter between the name and value, and place each property on its own line.

   For example:
   ```
   myextension.myTab.introduction=This page provides monitoring data for
   my application.
   myextension.myTab.TotalServletHits.label=Total hits for my servlet.
   ```

2. Save the file as `*dev-dir*/WEB-INF/classes/*bundle*.properties` where

   – *`dev-dir`* is the root directory of your extension

   – *`bundle`* is a unique value (do not use `global`, which is the name of a WebLogic Server bundle). Consider using your company name as the value for *`bundle`*.

3. Save each localized version of the properties file as
   `*dev-dir*/WEB-INF/classes/*bundle_locale*.properties`

   where *`locale`* is a locale code supported by `java.util.Locale`. See `Locale` in the *J2SE API Specification*.

   For example, `mycompany_ja.properties`.

Make sure to provide one file named *`bundle`*`.properties`; this is the default file that is used if a user has not specified a locale.

4. To use the bundle in your JSPs:

   a. Import the JSTL *`fmt.tld`* tag library:
   ```
   <%@ taglib uri="/WEB-INF/fmt.tld" prefix="fmt" %>
   ```

   b. Declare the name of your bundle:
   ```
   <fmt:setBundle basename="bundle" var="current_bundle" scope="page"/>
   ```
   where *`bundle`* is the name you used in the previous step.

   c. When you want the JSP to output a string, use the following JSP tag:
   ```
   <fmt:message key="property-name" bundle="${current_bundle}"/>
   ```

   For example:
   ```
   <fmt:message key="myextension.myTab.introduction"
   bundle="${current_bundle}"/>
   ```

Create Struts ActionForms and Actions

All WebLogic Server JSP tags that display and submit forms (and tables that function as forms) assume that Apache Struts is the controller agent. The JSP tags use Java beans that are populated by Struts ActionForms (form beans) and submit user input to a Struts Action. For information on Apache Struts, see *The Apache Struts Web Application Framework* at http://struts.apache.org/.

Some tags, such as `<wl-extension:table>`, require the form bean to contain a property that is a collection of Java beans. Each bean in the collection describes a single object or component that is rendered as a row in the table. For information on each Administration Console JSP tag and the Java bean properties that it requires, see *WebLogic Server JSP Tags Reference*.

If your extension manages WebLogic Server resources, create Struts ActionForms that populate Java bean properties with attribute values from WebLogic Server Managed Beans (MBeans). Similarly, create Struts Actions that set MBean values or invoke MBean operations. For information on working with WebLogic Server MBeans, see *Developing Custom Management Utilities with JMX* and *Developing Manageable Applications with JMX*.

Create a Named Struts Module

A Struts module is a Struts configuration file, message bundle, and other related resources. All Struts applications must have one default module. As of Struts 1.1, applications can have multiple modules, each of which must be uniquely named. For more information, see the Struts Newbie FAQ at http://struts.apache.org/faqs/newbie.html#modules.

Because the WebLogic Server Administration Console already uses the default module, you must create your own module for your extension. If you follow WebLogic Server naming conventions, the Administration Console will automatically discover your module and register it with the Apache Struts controller servlet.

To create a named Struts module for your Administration Console extension:

1. Save your Struts configuration file as follows:
   `dev-dir/WEB-INF/struts-auto-config-MyModule.xml`
   where:
   - `dev-dir` is the root of your Administration Console
   - `MyModule` is a name that you have chosen for your module. Consider using the name of your company to avoid possible naming conflicts
2. When you define portlets that forward to your Struts actions, include the `module="/MyModule"` attribute in the portlet's `<netuix:portlet>` element (see "Example: The Form in the Domains: Configuration: General Portlet" on page 5-5).

For information on using additional features of Struts modules, see Configuring Applications in the *Apache Struts User Guide*.

Example: The Form in the Domains: Configuration: General Portlet

The Domains: Configuration: General portlet renders a form that provides configuration data for the domain. When the Administration Console loads this portlet:

1. The portlet forwards to the Struts `DomainConfigGeneral` action path, which refers to an ActionForm that is defined in a Struts configuration file. The configuration file and associated resources are in a Struts module named `core`.

   Below is an excerpt from the portlet's `.portlet` file:

```
<netuix:portlet definitionLabel="CoreDomainDomainConfigGeneralPortlet"
   title="core.domain.domainconfiggeneral.portlet.title">
   <netuix:strutsContent module="/core"
                        action="DomainConfigGeneral"
                        refreshAction="DomainConfigGeneral"/>
</netuix:portlet>
```

2. When the Struts controller servlet receives the request from the portlet, it:

a. Instantiates the form bean that is named by the `<action path="DomainConfigGeneral">` element in the Struts configuration file. (See Listing 5-1.)

   This form bean, `domainConfigGeneralForm`, contains three properties:

   `name`, which identifies the bean

   `handle`, which is the name of an Action class. The `<wl-extension:form>` JSP tag invokes this Action when a user submits the form.

   `domainConfigGeneral`, which is a Java bean whose properties correspond to attributes in the WebLogic Server `DomainMBean`.

b. Launches the ActionForm class that is specified by the `<action path="DomainConfigGeneral">` element in the Struts configuration file.

   This class, `com.bea.console.actions.core.domain.DomainConfigGeneralAction`, connects to an MBean server, gets the value of the `DomainMBean` attributes, and populates the `domainConfigGeneral` Java bean with the MBean attribute values.

3. After the ActionForm instantiates and populates a form bean, it loads the `DomainConfigGeneralForm.jsp` and makes the form bean available to the JSP.

Listing 5-1 ActionForm for the Domains: Configuration: General Portlet

```
...

<form-bean name="domainConfigGeneralForm"
   type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="name"
      type="java.lang.String"/>
   <form-property name="handle"
      type="com.bea.console.handles.Handle"/>
   <form-property name="domainConfigGeneral"
      type="com.bea.console.cvo.core.domain.DomainConfigGeneralBean"/>
</form-bean>

...

<action path="/DomainConfigGeneral"
   type="com.bea.console.actions.core.domain.DomainConfigGeneralAction"
   name="domainConfigGeneralForm"
   parameter="tab"
   scope="request"
```

```
    validate="false">
    <forward name="success"
       contextRelative="true"
       path="/jsp/core/domain/DomainConfigGeneralForm.jsp"/>
</action>
```

Create JSPs that Use BEA Templates and JSP Tags

Most portlets in the Administration Console JSPs that are based on the `tableBaseLayout_netui` and `configBaseLayout_netui` templates. For information about these templates, see "WebLogic Server JSP Templates" on page 2-13.

The following sections describe how to create JSPs that use these templates:

- "Set Up Your Development Environment" on page 5-7
- "Create a Table JSP" on page 5-7
- "Create a Configuration JSP" on page 5-10

Set Up Your Development Environment

The Administration Console JSP templates require the use of JSP tags from the JSP Standard Tag Library (JSTL), the BEA Administration Console Extension Tag Library, and the Apache Beehive Page Flows Tag Library.

The WebLogic Server runtime environment already provides these tag libraries. For development support, add the following tag libraries to your development environment:

WL_HOME`/server/lib/consoleapp/webapp/WEB-INF/beehive-netui-tags-html.jar`
WL_HOME`/server/lib/consoleapp/webapp/WEB-INF/fmt.tld`
WL_HOME`/server/lib/consoleapp/webapp/WEB-INF/console-html.tld`

Create a Table JSP

A table JSP uses a table to summarize a set of resources and provides navigation to those resources (see Figure 2-4). If your extension manages multiple resources or multiple instances of a single resource, create a table JSP.

Note: Tables can contain buttons and checkboxes, which enable you to select one or more rows and invoke an action for the selected items. For information about this feature, see `<wl-extension:table>` in *WebLogic Server JSP Tags Reference*.

Before you create a table JSP, create a Struts ActionForm that populates a form bean and forwards to your JSP. (See "Create Struts ActionForms and Actions" on page 5-4.) The form bean must contain the following minimal properties:

- A property named `name`, which identifies the form bean
- A property that contains an `Array` of Java beans. Each of these Java beans (row beans) must contain one property for each table row that you want to render.

  For example, to render the Servers table in the Administration Console, each server instance in the domain is represented by its own row bean. The row bean's properties define each server's name, listen port, listen address and other information.

To create a table JSP (see Listing 5-2):

1. Create a JSP and save it in your development directory. Consider creating a subdirectory to contain all of the JSPs in your extension. For example, *`dev-dir`*`/jsp`
   where *`dev-dir`* is your development directory. For more information, see "Archive and Deploy the Extension" on page 4-20.
2. Import JSP tag libraries by including the following tags:
   ```
   <%@ taglib uri="/WEB-INF/console-html.tld" prefix="wl-extension" %>
   <%@ taglib uri="/WEB-INF/fmt.tld" prefix="fmt" %>
   <%@ taglib uri="/WEB-INF/beehive-netui-tags-template.tld"
   prefix="beehive-template" %>
   ```
   For information about these tag libraries, see "JSP Tag Libraries" on page 2-15.
3. (Optional) If you plan to use `<fmt:message>` tags to display localized text, use `<fmt:setBundle/>` to specify the name of the message bundle.

   This `<fmt:setBundle/>` tag enables you to specify the bundle name once, and then refer to this value from `<fmt:message>` tags by variable.
4. Declare the JSP template for tables by creating the following opening tag:
   ```
   <beehive-template:template
      templatePage="/layouts/tableBaseLayout_netui.jsp">
   ```
   Do not close the tag yet. All other JSP tags in a table JSP are nested in this template tag.
5. Create a `<beehive-template:section name="configAreaIntroduction">` tag. Inside this tag, provide an introductory sentence or paragraph that describes the table. This description is rendered above the table.
6. Create the following opening tag:
   ```
   <beehive-template:section name="table">
   ```

Do not close the tag yet.

7. Create an opening `<wl-extensions:table>` tag and specify values for the following minimal attributes:
   - `name`, specify the name of the form bean that you configured for this table.
   - `property`, specify the name of the form-bean property that contains row beans.
   - `bundle`, (optional) specify the name of a message bundle that contains localized names of your column headings.
   - `captionEnabled`, (optional) specify "true" to generate a title above the table.
8. If you specified "true" for the `captionEnabled` attribute, create a `<wl-extension:caption>` tag. Inside this tag, provide a caption for the table.
9. For each property in the row bean that you want to display in the table, create a `<wl-extension:column>` tag and specify values for the following attributes:
   - `property`, specify the name of the row bean property
   - `label`, specify a key in your message bundle to display as the column heading
10. If you want the text in a column to be a hyperlink, nest a `<wl-extension:column-link>` tag in the `<wl-extension:column>` tag. For the `<wl-extension:column-link>` tag's portlet attribute, specify the `definitionLabel` of the portlet to which you want to link.

    Instead of using `<wl-extension:column-link>` and the `definitionLabel` of a portlet, you can use `<wl-extension:column-dispatch>`, the value of metadata that you have defined for a portlet, and the value of an additional property that you must include in this table's form bean. Using metadata specify a linking target enables you to forward to all portlets that contain the specified metadata. See column-dispatch in the *WebLogic Server JSP Tag Reference*.
11. Close the `<wl-extension:table>`, `<beehive-template:section>`, and `<beehive-template:template>` tags.

Listing 5-2 Example: Simple Table JSP

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/console-html.tld" prefix="wl" %>
<%@ taglib uri="/WEB-INF/fmt.tld" prefix="fmt" %>
<%@ taglib uri="/WEB-INF/beehive-netui-tags-template.tld"
prefix="beehive-template" %>
```

```
<fmt:setBundle basename="core" var="current_bundle" scope="page"/>

<beehive-template:template templatePage="/layouts/tableBaseLayout_netui.jsp">

   <beehive-template:section name="configAreaIntroduction">
      <fmt:message key="core.server.servertable.introduction"
            bundle="${current_bundle}"/>
   </beehive-template:section>

   <beehive-template:section name="table">
      <wl-extension:table name="extensionForm"
             property="contents"
             captionEnabled="true"
            bundle="core">

         <wl-extension:caption>
            <fmt:message key="server.table.caption"
                 bundle="${current_bundle}"/>
         </wl-extension:caption>
         <wl-extension:column property="name" label="server.table.label.name">
            <wl-extension:column-link portlet="MyPortletID"/>
         </wl-extension:column>
         <wl-extension:column property="clusterName"
            label="server.table.label.cluster"/>
         <wl-extension:column property="machineName"
            label="server.table.label.machine"/>
      </wl-extension:table>

   </beehive-template:section>

</beehive-template:template>
```

Create a Configuration JSP

A configuration JSP uses a form to display the configuration of a resource and to enable users to modify the configuration (see Figure 2-3). You can create a read-only configuration JSP that enables users to monitor a resource.

The Administration Console JSP tags can render standard HTML input controls, such as text, text-areas, and radio buttons.

Before you create a configuration JSP, create a Struts ActionForm that populates a form bean and forwards to your JSP. (See "Create Struts ActionForms and Actions" on page 5-4.) The form bean must contain the following minimal properties:

- A property named `name`, which identifies the form bean

- A property that contains a Java bean. The Java bean must contain one property for each HTML input control that you want to render.

To create a configuration JSP (see Listing 5-3):

1. Create a JSP and save it in your development directory. Consider creating a subdirectory to contain all of the JSPs in your extension. For example, `dev-dir/jsp`
   where `dev-dir` is your development directory. For more information, see "Archive and Deploy the Extension" on page 4-20.

2. Import JSP tag libraries by including the following tags:

   ```
   <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
   <%@ taglib uri="/WEB-INF/console-html.tld" prefix="wl-extension" %>
   <%@ taglib uri="/WEB-INF/fmt.tld" prefix="fmt" %>
   <%@ taglib uri="/WEB-INF/beehive-netui-tags-template.tld"
   prefix="beehive-template" %>
   ```

   For information about these tag libraries, see "JSP Tag Libraries" on page 2-15.

3. (Optional) If you plan to use `<fmt:message>` tags to display localized text, use `<fmt:setBundle/>` to specify the name of the message bundle.

   This `<fmt:setBundle/>` tag enables you to specify the bundle name once, and then refer to this value from `<fmt:message>` tags by variable.

4. Declare the JSP template for configuration pages by creating the following opening tag:

   ```
   <beehive-template:template
   templatePage="/layouts/configBaseLayout_netui.jsp">
   ```

   Do not close the tag yet. All other JSP tags in a configuration JSP are nested in this template tag.

5. Create a `<beehive-template:section name="configAreaIntroduction">` tag. Inside this tag, provide an introductory sentence or paragraph that describes the form. This description is rendered above the form.

6. Create the following opening tag:

   ```
   <beehive-template:section name="form">
   ```

   Do not close the tag yet.

7. Indicate that the next set of JSP tags output XHTML by creating the following tag:

   ```
   <html:xhtml/>
   ```

8. Create an opening `<wl-extension:template name="/WEB-INF/templates/form.xml">` tag.

   The JSP tags that render forms can be used to render different types of forms. All form-related JSP tags output XML and the `<wl-extension:template>` tag specifies an XSLT template to transform the XML to XHTML. WebLogic Server provides the following templates:

   – `/WEB-INF/templates/form.xml`, which creates a form that matches Administration Console configuration pages (such as Domains: Configuration: General). This template also generates a button that submits the form.

   – `/WEB-INF/templates/assistant.xml`, which creates a form that matches Administration Console assistants.

   – `/WEB-INF/templates/tablePreferences.xml`, which creates a form that matches pages used to customize table displays in the Administration Console

9. Create an opening `<wl-extension:form>` and specify values for the following attributes:

   – `action`, specify the path of a Struts action that is invoked when a user submits this form. The Struts module that defines the action path is specified in the request.

   – `bundle`, (optional) specify the name of a message bundle that contains localized names of your column headings.

   – `readOnly`, (optional) specify "true" to make this form read-only (for example, if you are displaying read-only monitoring data).

10. For each property in the form bean that you want to display in the form, create a `<wl-extension>` tag corresponding to the type of control that you want to render (see *WebLogic Server JSP Tags Reference*):

   – `<wl-extension:checkbox>`

   – `<wl-extension:chooser-tag>`

   – `<wl-extension:hidden>`

   – `<wl-extension:password>`

   – `<wl-extension:radio>`

   – `<wl-extension:select>`

   – `<wl-extension:text>`

   – `<wl-extension:text-area>`

Alternatively, you can use `<wl-extension:reflecting-fields>`, which generates an HTML input tag for each property in a form bean. For example, for a bean property that contains a `java.lang.String`, the tag generates a text control; for a boolean, it generates a checkbox. This tag uses the default form bean, which is passed to the JSP in the request.

11. To generate text on the page that describes to users the purpose of each control, include the `inlineHelpId` attribute in each `<wl-extension>` tag in the previous step.
12. Close the `<wl-extension:form>`, `<beehive-template:section>`, and `<beehive-template:template>` tags.

Listing 5-3 Example: Simple Configuration JSP

```
<%@ page language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/console-html.tld" prefix="wl" %>
<%@ taglib uri="/WEB-INF/fmt.tld" prefix="fmt" %>
<%@ taglib uri="/WEB-INF/beehive-netui-tags-template.tld"
   prefix="beehive-template" %>

<fmt:setBundle basename="core" var="current_bundle" scope="page"/>
<beehive-template:template templatePage="/layouts/configBaseLayout_netui.jsp">
   <beehive-template:section name="configAreaIntroduction">
      <fmt:message key="core.server.serverconfiggeneral.introduction"
           bundle="${current_bundle}"/>
   </beehive-template:section>

   <beehive-template:section name="form">
      <html:xhtml/>
      <wl-extension:template name="/WEB-INF/templates/form.xml">
         <wl-extension:form action="/CoreServerServerConfigGeneralUpdated"
bundle="core">
            <wl-extension:text property="serverConfigGeneral.Name"
               labelId="core.server.serverconfiggeneral.name.label"
                inlineHelpId="core.server.serverconfiggeneral.name.
                    label.inlinehelp" />
             <wl-extension:select
property="serverConfigGeneral.selectedMachine"
                 labelId="core.server.serverconfiggeneral.machine.label"
                 inlineHelpId="core.server.serverconfiggeneral.machine.
                    label.inlinehelp">
                  <wl-extension:optionsCollection
                     property="serverConfigGeneral.availableMachines"
                     label="label" value="value"/>
              </wl-extension:select>
         </wl-extension:form>
      </wl-extension:template>
   </beehive-template:section>
</beehive-template:template>
```

Create a .portlet XML File

A `.portlet` XML file defines a WebLogic Portal portlet, which is a container for content such as JSPs and Struts actions. All JSPs in the Administration Console must be contained in portlets. For more information about portlet XML files, see the *Portal Support Schema Reference*.

To create a portlet that contains a JSP based on WebLogic Server template and its required Struts actions:

1. Copy the code from Listing 5-4 and paste it into new text file.

   Consider creating a subdirectory to contain all of the portlet XML files in your extension. For example, `dev-dir/PortalConfig`
   where `dev-dir` is your development directory. For more information, see "Archive and Deploy the Extension" on page 4-20.

   Also consider adopting the following naming convention:
   `JSP-file-name-no-extension.portlet`

   where `JSP-file-name-no-extension` is the name of the JSP file that the portlet contains. For example, if the portlet contains a JSP file named `monitorEJB.jsp`, then name the portlet XML file `monitorEJB.portlet`.

2. Replace the values in Listing 5-4 as follows:

   – `Label`. Provide a unique identifier that the portal framework uses to identify this portlet.

   – (optional) `Title`. Provide a default title that this portlet displays in its title bar if the Look and Feel causes the title bar to be displayed.

   – `Struts-module`. Specifies the Struts module that defines the ActionForm that you use for table and configuration JSPs. The value that you enter must be part of an XML file name that you save in `/WEB-INF`.

     For example, if you specify "`myModule`" for `Struts-module`, the Struts controller servlet looks in the following location for action paths:
     `/WEB-INF/struts-auto-config-myModule.xml`

- *action-path*. Specifies the path to a Struts action that is defined in your Struts module. The action must create an ActionForm and then forward to your table or configuration JSP. See "Create Struts ActionForms and Actions" on page 5-4.
- *refresh-action-path*. Specifies the action to invoke on subsequent requests for this portlet (for example, the user agent refreshes the document).

Note that this .portlet does not specify the name of your table or configuration JSP. Instead, the name of this JSP is specified in the Struts configuration file.

Listing 5-4 Template for a Simple .portlet XML File

```
<?xml version="1.0" encoding="UTF-8"?>
<portal:root xmlns:html="http://www.w3.org/1999/xhtml-netuix-modified/1.0.0"
   xmlns:portal="http://www.bea.com/servers/netuix/xsd/portal/support/1.0.0"
   xmlns:netuix="http://www.bea.com/servers/netuix/xsd/controls/netuix/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-support-1_0_0.xsd">

   <netuix:portlet definitionLabel="Label" title="Title" >
     <netuix:strutsContent module="Struts-module"
          action="action-path"
          refreshAction="refresh-action-path"/>
   </netuix:portlet>

</portal:root>
```

Create Other Portal Framework Files and Deploy the Extension

You can add your portlet directly to the desktop, but if you want your portlet to display as a tab or subtab in the ContentBook, you must define books or pages to contain it. In addition, you must create a netuix-extension.xml file which specifies where to locate your portlet, books, and pages and which functions as the deployment descriptor for your extension.

See "Adding Portlets and Navigation Controls" on page 4-1.