BEA WebLogic Server™

Extending the Administration Console

Release 7.0
Document Revised: June 28, 2002

Extending the Administration Console

Part Number	Date	Software Version
N/A	June 28, 2002	BEA WebLogic Server Version 7.0

Contents

3. Using Localization in a Console Extension

About This Document

This document describes how to create a custom extension to the WebLogic Server Administration Console.

This document covers the following topics:

- Chapter 1, "Extending the Administration Console" Describes the steps to create an Administration Console extension.
- Chapter 2, "Using the Console Extension Tag Library" Describes the JSP tag library used to create an Administration Console extension.
- Chapter 3, "Using Localization in a Console Extension" Describes how to use localization in an Administration Console extension.

What You Need to Know

This document is intended mainly for application developers who are interested in creating an extension to the WebLogic Server Administration console. Familiarity with the WebLogic Server platform, Java programming, WebLogic Server Mbeans, XML, and JavaServer Pages is required.

e-docs Web Site

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

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

The following BEA WebLogic Server documents contain information that is relevant to creating Administration Console extensions.

- WebLogic Server System Administration Guide. at http://e-docs.bea.com/wls/docs70/adminguide/index.html
- Programming WebLogic JSP at http://e-docs.bea.com/wls/docs70/jsp/index.html.
- The Sun Microsystems, Inc. Java site at http://java.sun.com/

Contact Us!

Your feedback on the BEA WebLogic Server documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Server documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Server 5.0 release.

If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSupport at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.

Convention	Item
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

Convention	Item

. . .	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Extending the Administration Console

This document describes how to extend the BEA WebLogic Server Administration Console. By extending the Administration Console, you can create your own console screens that appear along with the standard console pages. The following sections provide procedures for extending the console:

Overview of Extending the Administration Console

The BEA WebLogic Server Administration Console is a browser-based graphical user interface that you use to manage a WebLogic Server Domain. For more information about the Administration Console, see About the Administration Console in the *Administration Console Online Help*. You extend the Administration Console by adding screens and navigation elements that appear along with the supplied system Administration Console screens.

A console extension can provide functionality not included in the standard Administration console or an alternate interface for existing functionality. For example, you can use a console extension to:

- Provide custom management of applications deployed on WebLogic Server.
- Manage third-party systems.
- Manage a custom security provider. (For more information, see Writing Console Extensions for Custom Security Provider.)
- Provide customized monitoring and management screens for a WebLogic Server domain.

Creating an Administration Console extension requires intermediate knowledge of Java programming, JavaServer Pages (JSP), HTML, and WebLogic Server Mbeans. Mbeans are Java objects used for system administration of WebLogic Server domains. For more information about WebLogic MBeans and WebLogic Server system administration infrastructure, see System Administration Infrastructure in the *Administration Guide*.

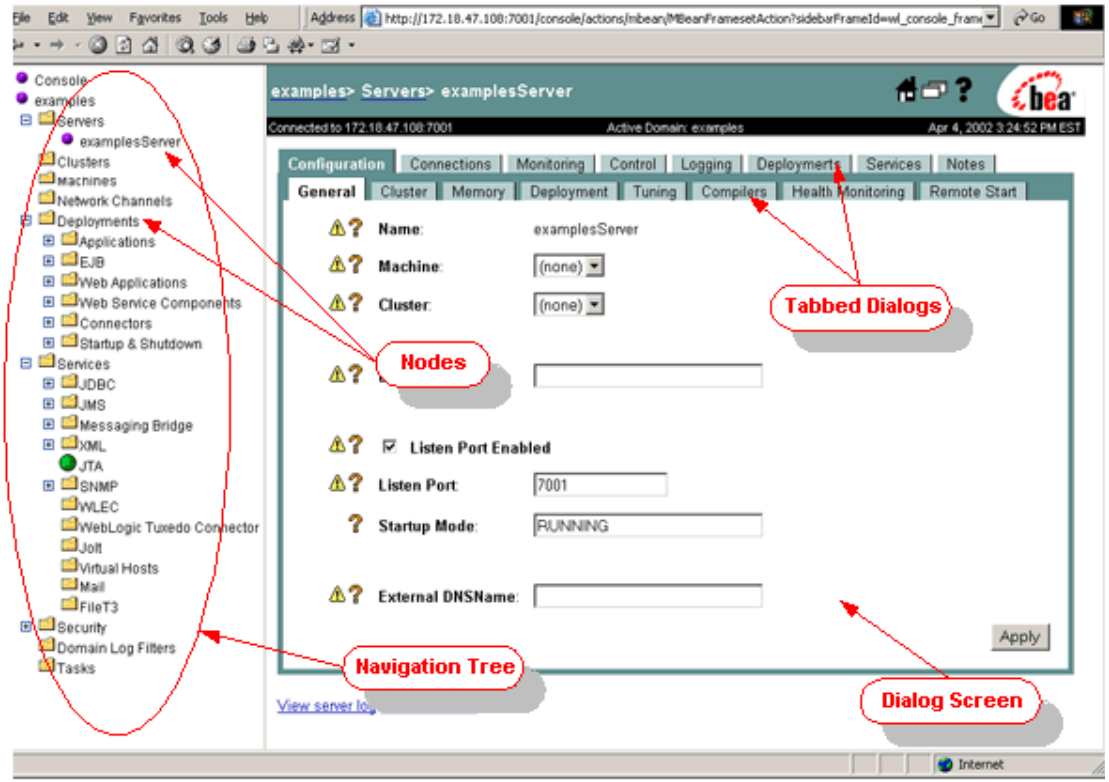
A complete code example of an Administration Console extension is available from the BEA dev2dev web site. Click on *Sample Administration Console Extension (WLS 7.0)* and then download the `ConsoleExtensionExample.zip` file. You will need files from this example to create your console extension.

Visual Elements of an Administration Console Extension

An Administration Console extension can contain the following visual elements, as shown in Figure 1-1:

- *Navigation Tree*. The navigation tree is a Java applet that allows users to navigate among the console dialog screens.
- *Nodes*. Nodes are branches of the navigation tree. A node can contain other nodes or can call dialog screens that are displayed in the right pane of the Administration Console. Your console extension can add one or more nodes to the navigation tree.
- *Tabbed Dialogs*. You can create up to 2 levels of tabbed dialogs. Your console extension screens are part of a dialog.
- *Dialog Screens*. Dialog Screens are displayed when a user selects one of the tabbed dialogs. Dialog Screens are where the functionality of your console extension appears.

Figure 1-1 Administration Console Visual Elements



Programmatic Elements of a Console Extension

To create a console extension, you create the following programmatic elements:

- A Web Application that contains the elements described in this section, and any additional Java classes or JSP tag libraries that you require to implement your extended console screen.
- A Java class that defines a new node in the Administration Console navigation tree where a link to your console extension appears. You can also use this class

to initialize functionality required for your console extension. This class implements an interface that is part of the WebLogic API. You register this class in the Web Application using a special context parameter.

- A JavaServer Page (JSP) that defines the behavior of the new node in the navigation tree and (optionally) defines additional nodes that appear under your new node. You can also define menu options that appear when users right-click on the node.
- One or more JSPs that defines your console dialog screen(s). A JSP tag library is supplied that allows you to construct a tabbed interface, similar to the standard Administration Console screens. You have the option of utilizing an standard HTML style sheet that helps you create screens with a similar look and feel as the standard console pages.
- (Optional) Localization catalogs you can use to look up localized strings for text and labels that appear in your console extension. Localization catalogs are constructed using XML.

Main Steps to Create an Administration Console Extension

The following steps are required to create an Administration Console extension:

1. Create a Java class that defines your Administration Console Extension. This class defines where your console extension appears in the navigation tree and can provide additional functionality required by your extension. See "Implementing the NavTreeExtension Interface" on page 1-6.
2. Define the behavior of the Navigation tree. In this step you can define multiple nodes that appear under the node you define in step 1. You can also define right-click menus and actions. See "Setting Up the Navigation Tree" on page 1-8
3. Write JSPs to display your console extension screens. You may use localized text by looking up strings in a localization catalog. A supplied tag library allows you to create tabbed dialog screens similar to those in the standard Administration Console and to access the localization catalogs. See "Writing the Console Screen JSPs" on page 1-11.

4. Package your JSPs, catalogs, and Java classes as a Web Application. See "Packaging the Administration Console Extension" on page 1-15.
5. Deploy the Web Application containing your console extension on the Administration Server in your WebLogic Server domain. See "Deploying an Administration Console Extension" on page 1-18.

Implementing the NavTreeExtension Interface

To define your console extension, write a Java class that extends `weblogic.management.console.extensibility.Extension` and implements `weblogic.management.console.extensibility.NavTreeExtension`. For a sample of this Java class, see "Sample Java Class for Implementing the NavTreeExtension Interface" on page 1-19.

Note: If you are creating a console extension for a custom security provider, implement the `weblogic.management.console.extensibility.SecurityExtension` interface instead of the `weblogic.management.console.extensibility.NavTreeExtension` interface. (For more information, see Writing Console Extensions for Custom Security Provider.)

To write the Java class:

1. Decide where (that is, under which node) in the navigation tree you want your console extension to appear. Each node in the console is associated with an MBean object. By associating your extension with one of these MBean objects using the steps in this procedure, your console extension appears as a new node under one of these existing nodes. (MBeans are Java objects used for configuring a WebLogic Server domain.)

   Your choice of where to place the node(s) representing your console extension should be determined by the functionality of your console extension. For example, if your console extension is related to WebLogic Server instances in a domain, place your console extension node under the Servers node by associating your extension with the `ServerMBean` (`weblogic.management.configuration.ServerMBean`).

   Your console extension will appear under each instance of a configured object that appears under a node. For instance, if you select the Servers node,

(`ServerMBean`) your extension will appear under each configured server in your domain. (For a list of MBeans, see the Javadocs for the `weblogic.management.congfiguration` package.)

If you want your extension to appear at the top (domain) level of the navigation tree, associate your extension with the `DomainMBean` (`weblogic.management.configuration.DomainMBean`). Your extension will only appear once because only one instance of a domain is displayed in the console.

2. Add an `import` statement for the MBean class associated with your console extension. The navigation tree node where you access your console extension appears as a child of the node for this Mbean. For example:

   ```
   import weblogic.management.configuration.DomainMBean.
   ```

3. Add the following additional `import` statement:

   ```
   import weblogic.management.console.extensibility.
   NavTreeExtension;
   ```

4. If required for the functionality of your console extension, you may want to add the following `import` statements:

   ```
   import weblogic.management.console.extensibility.Catalog;
   import weblogic.management.console.extensibility.Extension;
   import javax.servlet.jsp.PageContext;
   ```

5. Declare the class name of this class. For example:

   ```
   final public class ExampleConsoleExtension extends Extension
   implements NavTreeExtension
   ```

6. Add a public constructor, without arguments. For example:

   ```
   public ExampleConsoleExtension() {}
   ```

7. Define the `getNavExtensionFor()` method. When the Administration Console initializes itself, it calls this method, passing in the name of the associated MBean as the `Object` argument as it constructs each node of the navigation tree.

   In this method, test to see if the `Object` argument is an instance of the MBean associated with a node under which your console extension should appear. If the `Object` is an instance of this MBean, the method should return a URL to a JSP page that defines the behavior of the node in the navigation tree, otherwise the method should return `null`. For example:

```
public String getNavExtensionFor(Object key)
{
  if (key instanceof DomainMBean) {
     System.out.println(
           "\nFound an instance of the DomainMbean\n");
     return "domain_navlink.jsp";
  }
  return null;
}
```

In the above example, when the Administration Console constructs the node for the `DomainMBean` it runs this method, passing in the name of a Domain as the `Object` argument. Because the `Object` is an instance of the `DomainMBean`, the method returns the URL `domain_navlink.jsp`.

Note: If you are creating a console extension for a custom security provider, do not define the `getNavExtensionFor()` method. Instead, define one of the methods described under Replacing Custom Security Provider-Related Administration Console Dialog Screens Using the SecurityExtension Interface. For more information, see the Javadocs for the SecurityExtension interface.

The `System.out.println` statements are optional and serve only to display the message to standard out.

8. You may need to call methods of the `weblogic.managment.console.extensions.Extension` class to implement the functionality of your console extension. The exact usage required is beyond the scope of this document. For more information, see the Javadocs for the `weblogic.management.console.extensibility` package.

9. Compile the class so that it appears in the `WEB-INF/classes` directory of the Web Application containing your console extension. To compile the class, set up your development environment to include the WebLogic Server classes. For more information, see Establishing a Development Environment.

Setting Up the Navigation Tree

The `getNavExtensionFor()` method in the Java class that you wrote (as described in the "Implementing the NavTreeExtension Interface" on page 1-6) returns a URL for each console extension node that appears in the navigation tree. This URL points to a JSP that defines the behavior of this node. In this JSP, you can define:

- Additional sub-nodes that appear as children of the a node.
- An icon that appears to the left of the node's label.
- A right-click menu. Items on the menu can call a URL that is displayed in the console. You can also define separators (a horizontal line in the menu list) that display in the right-click menu.

To create a JSP that defines a navigation tree node:

1. Create a new JSP file whose name matches the URL returned from the `getNavExtensionFor()` method, for example `domain_navlink.jsp`.
2. Save the JSP file in the top-level directory of the Web Application containing your console extension.
3. Add this taglib statement:

   ```
   <%@ taglib uri='console_extension_taglib.tld' prefix='wl' %>
   ```

4. (Optional) If you need access to an object in this JSP, add the following JSP tag:

   ```
   <wl:extensibility-key
   id='domainKey'
   class='MyObjectClass' />
   ```

   Where *`MyObjectClass`* is the Java class name of the object you want to access.

   For more information, see "<wl:extensibility-key> Tag" on page 2-8.

5. Add one or more `<wl:node>` tags. These tags describe nodes that appear in the navigation tree. You can nest `<wl:node>` tags to create child nodes. You can use the following attributes of the `<wl:node>` tag to define the appearance and functionality of this node: `url`, `label`, `labelId`, `icon`, `expanded`, `target`, and `font`. For details on using these attributes see "<wl:node> Tag" on page 2-2.

   The label attribute defines the displayed name of the tab. If you want to localize this name you can use the labelId attribute to look up the name in the localization catalog. For more information on localization, see "Using Localization in a Console Extension" on page 3-1.

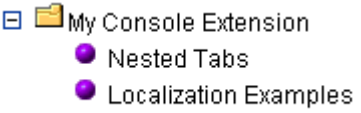
   The icon attribute points to an image file and displays the image as an icon for this node in the navigation tree. Image files for use as icons are available in the sample application, in the `extension_files/images` directory. (See step 1. in "Packaging the Administration Console Extension".)

   For example:

```
<wl:node
   label='<%="My Console Extension"%>'
   icon='/images/folder.gif'
   expanded='true'>

   <wl:node
      label='Nested Tabs'
      icon='/images/bullet.gif'
      url='/dialog_domain_example.jsp'>
   </wl:node>

   <wl:node label='Localization Examples'
      icon='/images/bullet.gif'>
   </wl:node>

</wl:node>
```

The above code will result in the navigation tree nodes shown in Figure 1-2.

Figure 1-2 Navigation Tree Nodes



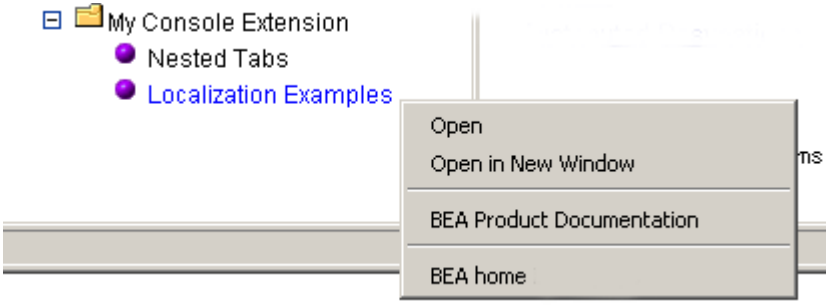
6. (Optional) Add one or more `<wl:menu>` tags to create right-click menu options. You can define the following attributes for the `<wl:menu>` tag: `label`, `labelId`, `url`, and `target`. For more information, see "<wl:menu> and <wl:menu-separator> Tags" on page 2-4. For example, to add `<wl:menu>` tags to the "Localization Examples" node defined in the previous example in previous step, use the following code:

```
...

   <wl:node
      label='Localization Examples'
      icon='/images/bullet.gif'>
      <wl:menu
         label='BEA Product Documentation'
         url='http://e-docs.bea.com/index.html'
         target='_blank'/>
      <wl:menu-separator/>1

       <wl:menu
           label='BEA home'
```

```
            url='http://www.bea.com'
             target='_blank'/>
    </wl:node>

...
```

The above code creates the right-click menu shown in Figure 1-3.

Figure 1-3 Navigation Nodes with Right-Click Menu



Writing the Console Screen JSPs

The actual dialog screens generated by your console extension appear in the right pane of the Administration Console when a user clicks on your extension's node in the navigation tree. To create these screens, you write a JSP using the supplied JSP tag library. (For reference information on the tag library, see "Using the Console Extension Tag Library" on page 2-1.) The JSP that is displayed is determined by the `url` attribute of the `<wl:node>` tags in the JSP you created in the Setting Up the Navigation Tree section.

You create the dialog screens using one or more tabbed dialogs that you define using the `<wl:tab>` JSP tag. These tabs can also contain nested sub-tabs, but only one level of nesting is supported. Each tab has a text label that you can specify explicitly or, you can specify a label ID that you can use to look up a localized version of the tab's label in a localization catalog.

With in each tab (that is, within a pair of `<wl:tab>...</wl:tab>` tags) you can use JSP and HTML coding to create the functionality of your console extension. Any text that appears in these can also be localized by looking up text from a localization catalog. For more information on using localization (the ability to display your console extension in multiple languages), see "Using Localization in a Console Extension" on page 3-1.

Note: You can use a variety of programming techniques to create the user interface of your extension. These techniques are beyond the scope of this document. For more information, see:

- Programming WebLogic JSP
- Programming WebLogic JMX Services

The following procedure creates a basic JSP that displays the screen for your console extension:

1. Create a new JSP file whose name matches the URL specified with the `url` attribute of the `<wl:node>` tag that calls this screen, for example, `domain_dialog.jsp`.
2. Save the JSP file in the top-level directory of the Web Application containing your console extension.
3. Insert this `taglib` statement at the top of the JSP file:

   ```
   <%@ taglib uri='console_extension_taglib.tld' prefix='wl' %>
   ```

4. Insert HTML and JSP blocks into the JSP file. The display of your console extension is defined by HTML code and JSP code that is translated into HTML code, therefore wrap your display code in the following set of HTML tags:

   ```
   <html>
    <head>
      <wl:stylesheet/>
    </head>
    <body>
       <div class='content'>
           <wl:dialog>
              (Insert <wl:tab> statements here.)
           </wl:dialog>
       </div>
     </body>
   </html>
   ```

The `<wl:stylesheet/>` tag in the `<head>...</head>` block is optional. When included, this tag formats your text so that it is consistent with standard WebLogic Server Administration Console pages.

5. Add one or more `<wl:tab>` tags in the JSP file (place these tags between the `<wl:dialog>...</wl:dialog>` tags). Each `<wl:tab>` tag defines a tabbed screen that appears in the right panel of the Administration Console. You can nest one or more tabs with in a top-level tab, but only one level of nesting is supported.

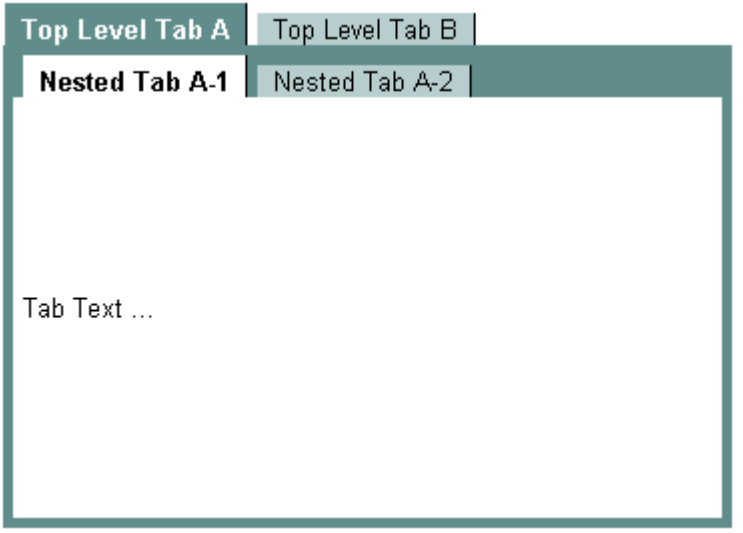
   You can define the following attributes for each `<wl:tab>` tag: `name`, `label`, and `labelId`. Each `<wl:tab>` tag requires a closing (`</wl:tab>`) tag. For more information on using these attributes, see "<wl:tab> Tag" on page 2-6. You write the HTML and JSP code that displays the body of your console extension dialog screen within a `<wl:tab>` block. You can also localize the label displayed for the tab. For more information, see "Localizing Tab Labels" on page 3-10.

   For example, the following code creates two top-level tabs, each containing two nested tabs (see Figure 1-4 to see how these tabs look in the console):

```
<wl:tab name='TopLevelTabA' label='Top Level Tab A'>

   <wl:tab name='NestedTabA1' label='Nested Tab A-1'>
       (Insert your JSP and/or HTML code
       for displaying your console extension here.)
   </wl:tab>

   <wl:tab name='NestedTabA2' label='Nested Tab A-2'>
      (Insert your JSP and/or HTML code
       for displaying your console extension here.)
   </wl:tab>

</wl:tab>

<wl:tab name='TopLevelTabB' label='Top Level Tab B'>

   <wl:tab name='NestedTabB1' label='Nested Tab B-1'>
       (Insert your JSP and/or HTML code
       for displaying your console extension here.)
   </wl:tab>

   <wl:tab name='NestedTabB2' label='Nested Tab B-2'>
      (Insert your JSP and/or HTML code
       for displaying your console extension here.)
   </wl:tab>

</wl:tab>
```

Note: This procedure creates a basic JSP defining a console extension. There are additional tags not shown here that supply other functionality such as localization. These tags are described in other sections.

Figure 1-4 Nested Tabs



Localizing the Administration Console Extension

The preceding main steps have omitted any discussion of localization procedures you can use to display your extension in multiple languages. The procedure to localize your console extension includes using special JSP tags, writing localization catalogs and writing the `index.xml` file that lists all your localization catalogs. You package the `index.xml` file and the catalog files in the Web Application that defines your console extension, as described in the next section, "Packaging the Administration Console Extension" on page 1-15.

For a complete discussion of localization, see "Using Localization in a Console Extension" on page 3-1.

Packaging the Administration Console Extension

You package the JSPs and Java classes for your console extension as a J2EE Web Application for deployment on the Administration Server in a WebLogic Server domain.

To package your console extension:

1. Download the Sample Administration Console Extension from the BEA dev2dev web site. Click on *Sample Administration Console Extension (WLS 7.0)* and then download the `ConsoleExtensionExample.zip` file. You will need files from this example to create your console extension.

2. Unzip the `ConsoleExtensionExample.zip` file into a temporary directory on your hard drive. This `extension_files` directory in this archive contains the following files required for your console extension:

   - `console_extension_taglib.tld`
   - `images/folder.gif`
     (required only if you used these icons in the navigation tree)
   - `images/bullet.gif`
     (required only if you used these icons in the navigation tree)
   - `deployment_descriptor_templates/web.xml`
     (a template you can modify for your console extension)
   - `deployment_descriptor_templates/weblogic.xml`
     (a template you can modify for your console extension)

   Copy the above files into the locations indicated as you follow these steps to package your console extension. Figure 1-5 describes the correct locations in the Web Application for these files.

3. Write the `web.xml` deployment descriptor for the Web Application. (You can start with the template provided in the `ConsoleExtensionExample.zip` file.) `web.xml` deployment descriptor declares the name of your console extension class and the supplied JSP tag library. You can use any text editor to create the deployment descriptor, or you can use the WebLogic Builder tool, included with your WebLogic Server distribution. For more information, see WebLogic Builder Online Help.

   Your `web.xml` deployment descriptor must contain the elements shown in the following example. (Substitute the name of the class you created in step 5. in

"Implementing the NavTreeExtension Interface" for *MyConsoleExtension*. Be sure to include the full package name:

```
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <display-name>
     My Weblogic Console Example Extension
  </display-name>
  <context-param>
    <param-name>weblogic.console.extension.class</param-name>
    <param-value>MyConsoleExtension</param-value>
  </context-param>

<taglib>
  <taglib-uri>console_extension_taglib.jar</taglib-uri>
  <taglib-location>
     WEB-INF/console_extension_taglib.tld
  </taglib-location>
</taglib>

</web-app>
```

You may also need to add other elements required by your console extension.

4. Write the `weblogic.xml` deployment descriptor. (You can start with the template provided in the `ConsoleExtensionExample.zip` file.) The `weblogic.xml` descriptor contains an entry that allows your console extension to share the same security context as the overall Administration Console. You can use any text editor to create the deployment descriptor, or you can use the WebLogic Builder tool, included with your WebLogic Server distribution. For more information, see WebLogic Builder Online Help.

   Your `weblogic.xml` deployment descriptor must contain the elements shown in the following example:

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA
Systems, Inc.//DTD Web Application 7.0//EN"
"http://www.bea.com/servers/wls700/dtd/weblogic700-web-jar.dtd"
>

<weblogic-web-app>

   <session-descriptor>
     <session-param>
       <param-name>CookieName</param-name>
       <param-value>ADMINCONSOLESESSION</param-value>
```

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

</weblogic-web-app>
```

You may need also to add other elements required by your console extension.

5. Copy the `console_extension_taglib.tld` file from the sample application (see step 2. in "Packaging the Administration Console Extension") to the `WEB-INF` directory of your console extension Web Application.

6. Arrange the components of your console extension, including the `web.xml` and `weblogic.xml` deployment descriptors you created in steps 1 and 2, as shown in the following directory structure example, adding any additional JSP, HTML, tag library descriptors, Java classes, or image files required for your console extension in the locations indicated:

Figure 1-5 Sample Directory Layout for Console Extension

```
+---MyConsoleExtensionWebApp
|   |   dialog_domain_example.jsp
|   |   dialog_example.jsp
|   |   domain_navlink.jsp
|   |   server_navlink.jsp
|   |   *.html, *.jsp
|   |
|   +---images
|   |       bullet.gif
|   |       smiley.gif
|   |       *.gif, *.jpg
|   |
|   \---WEB-INF
|       |   console_extension_taglib.tld
|       |   *.tld
|       |   web.xml
|       |
|       +---catalogs
|       |       english.xml
|       |       german.xml
|       |       index.xml;
|       |       japanese.xml
|       |
|       \---classes
|                MyConsoleExtension.class
```

7. Package the application as a `.war` archive. For example, using the directory layout shown in Figure 1-5, switch to the `MyConsoleExtensionWebApp` directory and issue the following command:

```
jar cvf MyConsoleExtension.war .
```

You can also deploy your Web Application in "exploded" format without making the `.war` archive file. Deploying in exploded format can be helpful while you are developing your console extension. See Deploying an Administration Console Extension.

Deploying an Administration Console Extension

After you create the Web Application containing your console extension, deploy it on the Administration Server of your WebLogic Server domain. For information on Deploying Web Applications, see WebLogic Server Deployment in *Developing WebLogic Server Applications.*

You must re-start the administration server after deploying the Web Application for your console extension to function. If you revised the JSPs that define the Navigation Tree nodes, you must re-start the Administration Server for the change to be reflected in the navigation tree.

After re-starting the Administration Console, if you modify any of the JSPs that define dialog screens, you can see the changes by re-deploying the Web Application. redeploy the Web Application, use the Administration Console:

1. Select the Deployments --> Web Applications node in the Navigation Tree.
2. Select the name of your console extension.
3. Click the Deploy tab.
4. Click the Redeploy button.

Sample Java Class for Implementing the NavTreeExtension Interface

```
package weblogic.management.console.extensibility.example;


import javax.servlet.jsp.PageContext;
import weblogic.management.configuration.DomainMBean;
import weblogic.management.console.extensibility.Catalog;
import weblogic.management.console.extensibility.Extension;
import weblogic.management.console.extensibility.NavTreeExtension;

/**
 * <p>Sample implementation for a console extension.</p>
 */
final public class ExampleConsoleExtension extends Extension
implements NavTreeExtension {

  // Constructor

  /**
   * A public  constructor without arguments is required.
   */
  public ExampleConsoleExtension() {}

  // ============================================================
  // NavTreeExtension implementation

  public String getNavExtensionFor(Object key)
  {
      if (key instanceof DomainMBean) {
          System.out.println("==\n== Found instance of
DomainMbean\n==");
          return "domain_navlink.jsp";
      }
      return null;
  }

  // ============================================================
  // Optional Extension methods

   /**
   * <p>The example does not need to perform any special
   * initialization work, so this method only sends out a message to
```

```
   * let us know that the console found the extension.</p>
   * <p>You can use this method to perform any initialization
required
   * by your console extension.</p>
   */
  public void initialize()
  {
    System.out.println("==\n== Example Extension for Domain
Initialized!\n==");
  }

  /**
   * <p>Returns the name of the extension by looking it up in the
   * localization catalog. it.</p>
   */
  public String getName(PageContext context)
  {
    return Catalog.Factory.getCatalog(context).
      getText("example.extension.name");
  }

  /**
   * <p>Just provide a brief description of this extension.</p>
   */
  public String getDescription(PageContext context)
  {
    return Catalog.Factory.getCatalog(context).
      getText("example.extension.description");
  }
}
```

CHAPTER

2 Using the Console Extension Tag Library

The following sections describe the attributes of the console extension tags and provide a sample usage for each:

Overview of the Console Extension JSP Tag Library

A JSP tag library is supplied with the WebLogic Server distribution for use in creating console extensions. The tag library allows you to:

- Create new nodes in the Administration Console navigation tree.
- Create right-click options for the nodes in the Administration Console navigation tree.
- Create a tabbed interface for displaying your console extension.
- Localize (render text in an alternate language) the text displayed in the Navigation Tree and in your console extension tree.

Tag Library Attribute Reference

The following sections describe the usage of each tag in the Console Extension Tag Library.

<wl:node> Tag

Use the `<wl:node>` tag to create new nodes in the Administration Console navigation tree.

The following example demonstrates the usage of the `<wl:node>` tag:

Listing 2-1 Sample Usage of the <wl:node> Tag

```
<wl:node
   label='<%="My Console Extension"%>'
```

```
   icon='/images/smiley.gif'
   expanded='true'>

   <wl:node
      label='My 1st nested node'
      icon='/images/bullet.gif'
      url='/dialog_domain_example.jsp'>
   </wl:node>

   <wl:node label='My 2nd nested node'
      icon='/images/bullet.gif'>
   </wl:node>

</wl:node>
```

Table 2-1 Attributes of the <wl:node> Tag

Attribute	Description
`icon`	The URL of an image file (.gif or .jpg) that is displayed in the navigation tree for this node. The URL may be an absolute URL, (for example, `http://somesite.com/images/myIcon.gif`) or a URL relative to the Web Application containing the console extension (for example, `/images/myIcon.gif`.).
`label`	The text label displayed for this node. Do not use this attribute if you define the `labelId` attribute
`labelId`	The Catalog ID of the localized text label for this node. Do not use this attribute if you define the `label` attribute.
url	The URL of the page that should be displayed in the Administration Console when the user clicks this node. The URL may be an absolute URL, (for example, `http://somesite.com/myPage.jsp`) or a URL relative to the Web Application containing the console extension.

Attribute	Description
target	The name of a browser frame where the URL specified in the `url` attribute should be displayed. If this attribute is not specified, the URL is displayed in the right pane of the Administration Console. You may use any name you choose or one of the following keywords: ■ `_top` - Displays the URL in the same browser window that displays the console, replacing the console. ■ `_blank` - Displays the page specified by the `url` attribute in a new browser window.
expanded	Set to `true` or `false`. If set to `true`, the node appears expanded (all child nodes are visible) when the console first loads. Default is `false`.
font	Font used to display the node's text label. Support for fonts is browser-dependent.

<wl:menu> and <wl:menu-separator> Tags

Use the `<wl:menu>` to create menus and actions that users access by right-clicking on nodes in the navigation tree defined with the `<wl:node>` tag. The `<wl:menu-separator>` tab inserts a separator line in the right-click menu.

Listing 2-2 Usage of the <wl:menu> and <wl:menu-separator> Tags

```
...

   <wl:node
      label='My 2nd nested node'
      icon='/images/bullet.gif'>
      <wl:menu
         label='BEA Product Documentation'
         url='http://e-docs.bea.com/index.html'
         target='_blank'/>
      <wl:menu-separator/>

       <wl:menu
```

```
            label='BEA home page'
            url='http://www.bea.com'
             target='_blank'/>

   </wl:node>

...
```

The above code creates a right-click menu under the “My 2nd Nested Node” entry in the navigation tree.

Table 2-2 Attributes of the <wl:menu> Tag

Attribute	Description
`label`	Text label that appears for this menu item. Do not use this attribute if you define the `labelId` attribute.
`labelId`	The Catalog ID of the localized text label for this menu item. Do not use this attribute if you define the `label` attribute.
`url`	Absolute URL or a URL relative to the Web Application root for a page to be displayed in the console
target	The name of a browser frame where the URL specified in the `url` attribute should be displayed. If this attribute is not specified, the URL is displayed in the right pane of the Administration Console. You may use any name you choose or one of the following keywords: `_top` - Displays the URL in the same browser window that displays the console, replacing the console. `_blank` - Displays the page specified by the `url` attribute in a new browser window.

The `<wl:menu-separator>` tag has no attributes.

<wl:tab> Tag

Use the <wl:tab> tag to create a tabbed interface in your console extension. You can create nested tabbed screens by nesting a <wl:tab> tag within another <wl:tab> tag. Only one level of nesting is supported.

The following example demonstrates the usage of the <wl:tab> tag:

Listing 2-3 Sample Usage for the <wl:tab> Tag

```
<wl:tab name='TopLevelTabA' label='Top Level Tab A'>

   <wl:tab name='NestedTabA1' label='Nested Tab A-1'>
       (Insert your JSP and/or HTML code
       for displaying your console extension here.)
   </wl:tab>

   <wl:tab name='NestedTabA2' label='Nested Tab A-2'>
      (Insert your JSP and/or HTML code
       for displaying your console extension here.)
   </wl:tab>

</wl:tab>

<wl:tab name='TopLevelTabB' label='Top Level Tab B'>

   <wl:tab name='NestedTabB1' label='Nested Tab B-1'>
       (Insert your JSP and/or HTML code
       for displaying your console extension here.)
   </wl:tab>

   <wl:tab name='NestedTabB2' label='Nested Tab B-2'>
      (Insert your JSP and/or HTML code
       for displaying your console extension here.)
   </wl:tab>

</wl:tab>
```

Table 2-3 Attributes of the <wl:tab> Tag

Attribute	Description
`name`	The name of the tab. Do not use the period (`.`) character in the name. If you do not specify the `labelId` attribute, the console looks for an entry in the localization catalog with the form tab + *`name`* For example, if you set *`name`* to `config`, the console labels the tab by looking up localized text located in the catalog using the ID `tab.config`.
`label`	The exact text that appears as the title of the tab. Do not define this attribute if you define the `labelId` attribute. Use this attribute only if you are not using a localization catalog.
`labelId`	The catalog ID for a localized label for the tab, if different from the name attribute. This text is looked up in the localization catalog. Do not define this attribute if you define the `label` attribute.

<wl:dialog> Tag

The `<wl:dialog>` Tag demarcates a section of the JSP that defines tabbed console screens. `<wl:tab>` tags must appear with in a `<wl:dialog>` block.

Listing 2-4 Sample Usage of the <wl:dialog> Tab

```
...
<wl:dialog>
  <wl:tab>
     ....(Insert code for tabbed dialog screen here.)
  </wl:tab>
</wl:dialog>
...
```

The `<wl:dialog>` tag has no attributes.

<wl:stylesheet> Tag

Insert the <wl:stylesheet> tag with the HTML <head> block to specify that your console screens use the same display styles (fonts, colors, etc.) as the standard Administration Console.

Listing 2-5 Sample Usage of the <wl:stylesheet> Tab

```
<html>
 <head>
   <wl:stylesheet/>
 </head>
...
```

The <wl:dialog> tag has no attributes.

<wl:extensibility-key> Tag

The <wl:extensibility-key> tag creates a scripting variable that represents a Java object. You can use this tag in the JSP that defines the Navigation tree.

Note: The <wl:extensibility-key> tag cannot be used in a JSP that defines a console dialog screen.

Listing 2-6 Sample Usage of the <wl:extensibility-key> Tag

```
<wl:extensibility-key
  id='domainKey'
  class='weblogic.management.configuration.DomainMBean' />

<%="Configuration Version is" +
domainKey.getConfigurationVersion()%>
```

Attribute	Description
id	Name of the scripting variable.
class	Java class of the scripting variable.

<wl:text> Tag

Use the <wl:text> tag to display text from the localization catalog.

Listing 2-7 Sample Usage of the <wl:text> Tag

```
<wl:text textId='Text.3' textParamId='Param.1' />
<p>
<wl:text textId='Text.2' textParam="Blue"/>
```

Table 2-4 Attributes of the <wl:text> Tag

Attribute	Description
style	(Optional) HTML Style class used to display the text.
text	The actual text you want to display. Do not define this attribute if you define the textID attribute.
textId	The catalog ID for the localized text you want to display. This text is looked up in the localization catalog. Do not define this attribute if you define the text attribute.
textParamId	The localization catalog ID of text that is substituted for any occurrence of the string {0} in text retrieved from the catalog. Do not define this attribute if you define the textParam attribute.

Attribute	Description
`textParam`	A literal string that is substituted for any occurrence of the string `{0}` in text retrieved from the catalog. Do not define this attribute if you define the `textParamID` attribute.

Using the Tag Library in a Console Extension

To use the console extensibility tag library:

1. Add the following section to the `web.xml` deployment descriptor of the Web Application containing your console extension:

```
<taglib>
  <taglib-uri>console_extension_taglib.jar</taglib-uri>
  <taglib-location>
     WEB-INF/console_extension_taglib.tld
  </taglib-location>
</taglib>
```

2. Copy the `console_extension_taglib.tld` file to the `WEB-INF` directory of the Web Application containing your console extension.

3. Insert this taglib statement at the top of each JSP that uses the console extensibility tag library:

```
<%@ taglib uri='console_extension_taglib.jar' prefix='wl' %>
```

3 Using Localization in a Console Extension

The following sections describe how to localize the text in your Administration Console Extension:

Overview of Console Extension Localization

Localizing your Administration Console extension allows you to present your console extension in a variety of languages. You store the localized text in special localization catalog files, one for each language you want to present. You also describe all of the catalog files in a file called `index.xml`. You package the `index.xml` file and the catalog files in the Web Application that defines your console extension, as described in "Packaging the Administration Console Extension" on page 1-15.

You can localize the following parts of your console extension:

- Labels for nodes in the navigation tree

- Labels for right-click menu options in the navigation tree
- Labels for tabs in the console screens
- Any text you want to display in the console screens.

How the Console Determines Which Localization Catalog to Use

To determine which localization catalog to use to display the console, the console application uses a combination of the language preference set in the Administration Console, the language and country specified in the user's Web browser, and settings in the `index.xml` file. The `index.xml` file (see "Writing the index.xml File" on page 3-5) lists all the available localization catalogs and associates them with a language preference, and one or more locale settings. Locale settings include Country and Language attributes.

The console application checks the language preference set in the Console --> Preferences screen in the Administration Console. If a language is specified, the console looks at the `name` attribute of the `<catalog>` element in `index.xml` file to find the localization catalog associated with the language.

Note: If no language has explicitly been set in the Console --> Preferences screen, English will show as the default in the Language field.) If no preference is set, the console application looks at the `<locale>` elements in the `index.xml` file to find a Country and Language that matches the values set in the user's Web browser.

Main Steps for Console Extension Localization

To use localized text in your console extension:

1. Create an XML catalog file for each language you want to present. These XML files contain an ID string that you use to reference the localized text. This ID string is the same in each language catalog. The ID string points to a block of text that is retrieved from the catalog and displayed in the console.

   For detailed instructions, see "Writing a Localization Catalog" on page 3-3

2. Create an index.xml file that lists all of your localization catalogs.

   For detailed instructions, see "Writing the index.xml File" on page 3-5.

3. Display localized text using JSP tags in the Console Extension Tag Library. Each JSP tag has an attribute you use to specify the catalog ID. For more information, see "Using localized text in JSPs" on page 3-8.

4. Package your JSPs, catalogs, and Java classes as a Web Application. See "Packaging the Administration Console Extension" on page 1-15.

5. Deploy the Web Application containing your console extension on the Administration Server in your WebLogic Server domain. See "Deploying an Administration Console Extension" on page 1-18

Writing a Localization Catalog

You write localization catalogs using XML notation. You can use an XML editor or any plain-text editor. For a sample catalog, see "Sample Localization Catalog" on page 3-8.

To write a localization catalog:

1. Create the localization catalog as a plain text file and save it in the `WEB-INF/catalogs` directory of the Web Application containing your console extension. The file name of the catalog must match the `file` attribute of the `<catalog>` element in `index.xml` (see "Writing the index.xml File" on page 3-5) that defines this localization catalog.

   The catalog file must contain the following base XML elements:

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

```
(Insert your catalog data here.)

</catalog>
```

2. Create the localized text entries.
   - To localize single words or short phrases, see "Localizing Single Words or Phrases" on page 3-4.
   - To localize longer blocks of text, see "Localizing Long Blocks of Text" on page 3-5.
3. Create an entry for the catalog in the `index.xml` file in the `WEB-INF/catalogs` directory. For instructions, see "Writing the index.xml File" on page 3-5.

Localizing Single Words or Phrases

To create localized text for single words or phrases:

1. Use the `<textlist>` element. Within a `<textlist>` element you can create multiple entries of localized text as name/value pairs in the form:

   ```
   textid = localized text string
   ```

2. Wrap all of the localized text entries in a `<textlist>` element with the following XML:

   ```
   <![CDATA[
   ...
   ]]>
   ```

For example:.

```
<textlist>
<![CDATA[

example.mytext      = This is localized text from the catalog.
example.title       = Console Extensibility Example
example.tab.extra   = Extra
example.tab.1       = Extension Catalog Text
example.tab.2       = Console Catalog Text
```

```
example.tab.3       = Programmatic
]]>
</textlist>
```

Localizing Long Blocks of Text

To create localized text for long text strings:

1. Use the `<text>` element, specifying the localization catalog ID using the `id` attribute. For example:

   ```
   <text id='example.error-message'>
   ```

2. Insert the localized text between the `<text>` and `</text>` tags. You may use HTML tags within this text.

3. Wrap the localized text with the following XML:

   ```
   <![CDATA[
   ...
   ]]>
   ```

For example:

```
<text id='example.error-message'>
<![CDATA[
An <b>error</b> has occured.
<p> Please consult the documentation for more information.
]]>
</text>
```

Writing the index.xml File

The `index.xml` file maps a language to a catalog file and provides other details about the display of the localized text.

To create the `index.xml` file:

1. Create the `index.xml` file as a plain-text file in the `WEB-INF/classes` directory of the Web Application containing your console extension.

The catalog file must contain the following base XML elements:

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

(Insert your catalog definitions here.)

</index>
```

2. Create a `<catalog>` element for each catalog that you define. Within this element, define the following attributes:

Attribute	Description
`name`	The name of the Language. For more information, see "How the Console Determines Which Localization Catalog to Use" on page 3-2
`file`	The path and filename of the catalog file to use for this language. Specify the path relative to the `index.xml` file. (The `index.xml` file should be located in the `WEB-INF/catalogs` directory of the Web Application containing your console extension.
`charset`	The name of the character set to use to display the localized text. Use a standard character set name as defined by the World Wide Web Consortium (W3C).
`encoding`	The name of the encoding to use when displaying the localized text. Use a standard encoding as defined by the World Wide Web Consortium (W3C).

3. Create one or more `<locale>` sub-elements within the `<catalog>` element. The Administration Console application matches the settings in the user's Web browser with these attributes to determine which catalog to use. (For more information, see "How the Console Determines Which Localization Catalog to Use" on page 3-2.)

Attribute	Description
`country`	Country as specified at http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt.
`language`	Language as specified at http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

4. Create additional `<catalog>` and `<locale>` elements for each language catalog in your console extension.

Sample Localization Catalog

Listing 3-1 english.xml

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


<catalog>

<text id='example.error-message'>
<![CDATA[
An <b>error</b> {0} has occured.
<p> Please consult the documentation for more information.
]]>
</text>


<textlist>
<![CDATA[

example.mytext      = This is localized text from the catalog.
example.title         = Console Extensibility Example
example.copyright     = Copyright 2001 BEA Systems.
example.dialog        = This is an example dialog for the server
named {0}.
example.tab.extra     = Extra
example.tab.1         = Extension Catalog Text
example.tab.2         = Console Catalog Text
example.tab.3         = Programmatic
example.tab.4         = Switch on the Fly
]]>
</textlist>
</catalog>
```

Using localized text in JSPs

The following sections discuss the JSP code you use to localize various elements of your console extension.

Localizing the Navigation Tree Nodes

To create a localized node in the navigation tree, use the `labelId` attribute of the `<wl:node>` tag. For more information, see "<wl:node> Tag" on page 2-2.

For example. the following code creates a node whose label is looked up in the localization catalog under the ID `node1`:

```
<wl:node
   labelId='node1'
   icon='/images/bullet.gif'
   url='/dialog_domain_example.jsp'>
</wl:node>
```

Localizing Right-Click Menus

To create a localized right-click menu, use the `labelId` attribute of the `<wl:menu>` tag. For more information, see "<wl:menu> and <wl:menu-separator> Tags" on page 2-4.

For example. the following code creates a node with a right-click menu. The labels for the menu items are looked up in the localization catalog using the ID `beaDocs` and `beaHome`:

```
<wl:node
   label='My 2st nested node'
   icon='/images/bullet.gif'>
   <wl:menu
      labelId='beaDocs'
      url='http://e-docs.bea.com/index.html'
      target='_blank'/>
   <wl:menu-separator/>

    <wl:menu
        labelId='beaHome'
        url='http://www.bea.com'
         target='_blank'/>

</wl:node>
```

Localizing Tab Labels

To create a localized label for a tabbed dialog, use the `labelId` attribute of the `<wl:tab>` tag. For more information, see "<wl:tab> Tag" on page 2-6.

For example, the following code uses a localized label for a tab. The label is looked up in the localization catalog using the ID `tab.1`:

```
<wl:tab name='LocalizedTextTab' labelId='tab.1'>
      The tab label for this tab comes from the catalog.
</wl:tab>
```

If you do not specify the `labelId` attribute, the console looks for an entry in the localization catalog with the form

tab + name

Localizing Text in Console Dialogs

To create localized text in your console screen, use the `textId` and the `textParamId` attributes of the `<wl:text>` tag as described in the next two sections. For details on this tag, see "<wl:text> Tag" on page 2-9.

Localizing Text

To localize text, use the `textId` attribute of the `<wl:text>` tag to look up text in the localization catalog. For example, the following code looks up the ID `LocalizedText.1` in the localization catalog:

```
<wl:tab name='LocalizedTextTab' labelId='tab.2'>
    <wl:text textId='LocalizedText.1'/>
</wl:tab>
```

Localizing Parameters

You can also localize parameters. Parameters are substituted for the string `{0}`, when the string is stored in the localization catalog.

For example, in the following code, the localized text stored in the catalog under the ID `LocalizedParam.1` will be appear in place of the string `{0}`, which is stored under the ID `LocalizedText.3`.

```
<wl:text
  textId='LocalizedText.3'
  textParamId='LocalizedParam.1' />
```