BEA WebLogic Personalization Server

Developer's Guide

BEA WebLogic Personalization Server 3.2
Document Edition 3.2
August 2001

WebLogic Personalization Server Developer’s Guide

Document Edition	Date	Software Version
3.2	August 2001	BEA WebLogic Commerce Server 3.2

Contents

3. Foundation Classes and Utilities

4. Creating and Managing Content

5. Developing Portlets

6. Building a Custom Portal Step-by-Step

7. Using the Catalog Application in a Portal

8. Localizing Applications with the Internationalization Tags

9. WebLogic Personalization Server Schema

10. JSP Tag Library Reference

About This Document

This document explains how to use the BEA WebLogic Personalization Server™ to create personalized applications for use in an e-commerce site.

This document includes the following topics:

- Chapter 1, "Overview of Personalization Development," provides developer components and utilities that enable developers to create personalized applications. The pieces documented in this guide include the Advisor, Foundation classes and utilities, and JSP tag reference.
- Chapter 2, "Creating Personalized Applications with the Advisor," recommends content and performs several important functions in creating a personalized application, including searching for content, tying the other core personalization services together, and matching content to user profiles.
- Chapter 3, "Foundation Classes and Utilities," describes the Foundation, a set of miscellaneous utilities to aid JSP and Java developers in the development of personalized applications using the WebLogic Personalization Server. Its utilities include JSP files and Java classes that can be used by JSP developers to gain access to functions provided by the server and helpers for gaining access to Advisor services.
- Chapter 5, "Developing Portlets," provides developers with indepth information about creating the portlets that are included in your portal.
- Chapter 6, "Building a Custom Portal Step-by-Step," is a tutorial for building your own custom e-commerce portal.
- Chapter 7, "Using the Catalog Application in a Portal,"describes how to add some of the functionality of the WebLogic Commerce Server to your portal. This chapter also discusses deploying your application as a Web application.

- Chapter 8, “Localizing Applications with the Internationalization Tags,”provides a simple framework that allows access to localized text and messages. The internationalization (I18N) framework is accessible from JSP through a small I18N tag library.
- Chapter 9, “WebLogic Personalization Server Schema,” documents the database schema for the WebLogic Personalization Server.
- Chapter 10, “JSP Tag Library Reference,” describes the JSP tags included with WebLogic Personalization Server that allow developers to create personalized applications without having to program using Java.

What You Need to Know

This document is intended for business analysts, Web developers, and Web site administrators involved in setting up an e-commerce site using BEA WebLogic Personalization Server. It assumes a familiarity with related Web technologies as described below. The topics in this document are organized primarily around development goals and the tasks needed to accomplish them, specifically:

- **Java Server Page (JSP) developer** creates JSPs using the tags provided or by creating custom tags as needed.
- **System analyst, or database administrator** writes rules, designs schemas, optimizes SQL and monitors usage.
- **System administrator** installs, configures, deploys, and monitors the Web application server.
- **Java developer** extends or modifies the Enterprise Java Bean (EJB) components that make up the WebLogic Personalization Server engine, if that level of customization is desired.

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.beasys.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 Personalization 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 Personalization 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/.

Contact Us!

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

In your e-mail message, please indicate the release number of the WebLogic Personalization Server documentation you are using.

If you have any questions about this version of BEA WebLogic Personalization Server, or if you have problems installing and running BEA WebLogic Personalization Server, contact BEA Customer Support through BEA WebSUPPORT at **www.beasys.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.
italics	Indicates emphasis or book titles.

Convention	Item

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames 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. *Examples*: 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 Overview of Personalization Development

WebLogic Personalization Server provides developers with the ability to create personalized applications for e-commerce Web sites. This developer's guide provides information about the Advisor, Foundation Classes and Utilities, and JSP tags.

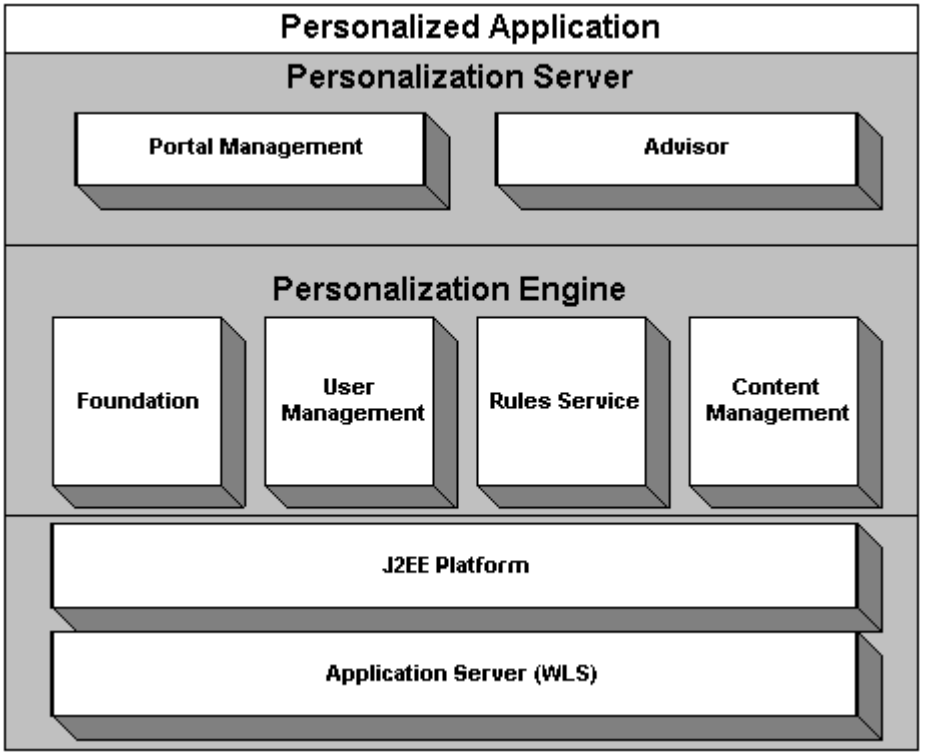
This topic includes the following sections:

- Personalization Server Run-Time Architecture
  - Advisor
  - Portal Management
  - User Management
  - Content Management
  - Rules Management
  - Foundation Classes and Utilities
- JSP Tags
- Integration of External Components
- Support for Native Types

Personalization Server Run-Time Architecture

The WebLogic Personalization Server (WLPS) run-time architecture is designed to support a variety of personalized applications. These applications can be built on the portal/portlet infrastructure, on the tags and EJBs supplied by the Advisor, and on select tags and EJBs supplied by other personalization server components.

The following high-level architecture picture may be used to visualize the relationships between the components.



The personalized application is one built by the developer to use the personalization components. It may consist of a portal instance with JSP portlets, a set of traditional JSP pages or servlets, and/or code that accesses EJB objects directly.

Advisor

The Advisor component is the primary interface to the most common operations that personalized applications will use. It provides access through tags or a single EJB session bean. Specific functionality provided by the Advisor includes classifying users, selecting content based on user properties, and querying content management directly. The Advisor uses the Foundation, User Management, Rules Service, and Content Management components.

Portal Management

The Portal Management component provides tags and EJB objects to support creating a framework of portals and portlets. It is configured using the Portal Administration Tools and has embedded JSP fragments built by the developer.

User Management

The User Management component supports the run-time access of users, groups, and the relationships between them. The notion of property sets is embedded within the user and group property access scheme. This component is set up using the User Management Administration tools and supports access via JSP tags or direct access to EJB objects. A Unified User Profile may be built by the developer, extending the User EJB object, to provide custom data source access to user property values.

Content Management

The Content Management component provides the run-time API by which content is queried and retrieved. The functionality of this component is accessible via tags. The content retrieval functionality is provided using either the provided reference implementation or Documentum content retrieval products.

Rules Management

The Rules Management component is the run-time service that runs the rule sets that are built in the Rules Management Administration Tool. This component is accessible only via the functionality of the Advisor tags. This component uses the JRules run-time library to make decisions.

Foundation Classes and Utilities

The Foundation is a set of miscellaneous utilities to aid JSP and Java developers in the development of personalized applications using the WebLogic Personalization Server. Its utilities include JSP files and Java classes that can be used by JSP developers to gain access to functions provided by the server and helpers for gaining access to Advisor services.

JSP Tags

The JSP tags included with WebLogic Personalization Server (Table 1-1) allow developers to create personalized applications without having to program using Java.

Table 1-1 JavaServer Page (JSP) Tags Overview

Library	Tag	Description
Advisor	`<pz:contentQuery>`	Provides content based on search expression query syntax.
	`<pz:contentSelector>`	Provides content based on results of a content selector rule and subsequent content query.
	`<pz:div>`	Turns a user-provided piece of content on or off based on the results of a classifier rule.
Content Management	`<cm:getProperty>`	Retrives the value of the specified content metadata property.
	`<cm:printDoc>`	Inlines the raw bytes of a document object in to the JSP output stream.
	`<cm:printProperty>`	Inlines the value of the specified content metadata property as a string.
	`<cm:select>`	Selects content based on a search expression query syntax.
	`<cm:selectById>`	Retrieves content using the content's unique identifier.
Flow Manager	`<fm:getApplicationURI>`	Gets the Flow Manager.
	`<fm:getCachedAttribute>`	Gets an attribute out of the session/global cache.
	`<fm:setCachedAttribute>`	Sets an attribute in the session/global cache.
	`<fm:removeCachedAttribute>`	Removes an attribute from the session/global cache.
	`<fm:getSessionAttribute>`	Gets an attribute out of the HttpSession.
	`<fm:setSessionAttribute>`	Sets an attribute in the HttpSession.

Table 1-1 JavaServer Page (JSP) Tags Overview (Continued)

	`<fm:removeSessionAttribute>`	Removes an attribute from the HttpSession.
Internationalization	`<i18n:localize>`	Defines the language, country, variant, and base bundle name to be used throughout a page when accessing resource bundles via the `<i18n:getmessage>` tag. Also allows a character encoding and content type to be specified for a JSP.
	`<i18n:getMessage>`	Used in conjunction with the `<i18:localize>` tag to retrieve localized static text or messages from a JspMessageBundle.
Portal Management	`<esp:eval>`	Evaluates a conditional attribute of a portlet. An example of a conditional attribute is `isMinimizeable`.
	`<esp:get>`	Retrieves a String attribute of a portlet.
	`<esp:getGroupsForPortal>`	Retrieves the names of the groups associated with a Portal.
	`<esp:monitorSession>`	Disallows access to a page if the session is not valid or if the user has not logged in.
	`<esp:portalManager>`	Provides the ability to do `create`, `get`, `getColumnInfo`, `update`, and `remove` actions on a Portal object.
	`<esp:portletManager>`	Provides the ability to do `create`, `get`, `getArranged`, `update`, and `remove` actions on a Portlet object.
	`<esp:props>`	Used to get a property from the Portal Properties bean, whose deployment descriptor contains default values used by the Portal Administration Tool.
Property Sets	`<ps:getPropertyNames>`	Used to get a list of property names given a property set.

Table 1-1 JavaServer Page (JSP) Tags Overview (Continued)

	`<ps:getPropertySetNames>`	Used to get a list of property sets given a property set type.
User Management (Profile)	`<um:getProfile>`	Retrieves the Unified User Profile object.
	`<um:getProperty>`	Gets the value for the specified property from the current user profile in the session.
	`<um:getPropertyAsString>`	Works exactly like the `<um:getProperty>` tag above, but ensures that the retrieved property value is a `String`.
	`<um:removeProperty>`	Removes the property from the current user profile in the session.
	`<um:setProperty>`	Sets a new value for the specified property for the current user profile in the session.
(Group-User Management)	`<um:addGroupToGroup>`	Adds the group corresponding to the provided `childGroupName` to the group corresponding to the provided `parentGroupName`.
	`<um:addUserToGroup>`	Adds the user corresponding to the provided `userName` to the group corresponding to the provided `parentGroupName`.
	`<um:changeGroupName>`	Adds the user corresponding to the provided `userName` to the group corresponding to the provided `parentGroupName`.
	`<um:createGroup>`	Creates a new `com.beasys.commerce.axiom.contact.Group` object.
	`<um:createUser>`	Creates a new persisted User object with the specified username and password.

Table 1-1 JavaServer Page (JSP) Tags Overview (Continued)

	`<um:getChildGroupNames>`	Returns the names of any groups that are children of the given group.
	`<um:getChildGroups>`	Retrieves an array of `com.beasys.commerce.axiom.contact.Group` objects that are children of the Group corresponding to the provided `groupName`.
	`<um:getGroupNamesForUser>`	Retrieves a `String` array that contains the group names matching the provided search expression and corresponding to groups to which the provided user belongs.
	`<um:getParentGroupName>`	Retrieves the name of the parent of the `com.beasys.commerce.axiom.contact.Group` object associated with the provided `groupName`.
	`<um:getTopLevelGroups>`	Retrieves an array of `com.beasys.commerce.axiom.contact.Group` objects, each of which has no parent group.
	`<um:getUsernames>`	Retrieves a String array that contains the usernames matching the provided search expression.
	`<um:getUsernamesForGroup>`	Retrieves a `String` array that contains the usernames matching the provided search expression and correspond to members of the provided group.
	`<um:removeGroup>`	Removes the `com.beasys.commerce.axiom.contact.Group` object corresponding to the provided `groupName`.
	`<um:removeGroupFromGroup>`	Removes a child group from a parent group.

Table 1-1 JavaServer Page (JSP) Tags Overview (Continued)

	`<um:removeUser>`	Removes the `com.beasys.commerce.axiom.contact.User` object corresponding to the provided `userName`.
	`<um:removeUserFromGroup>`	Removes a user from a group.
(Security)	`<um:login>`	Authenticates a user/password combination.
	`<um:logout>`	Ends the current user's WebLogic Server session. This is independent of the FlowManager's user session tracking, and should be used in combination with the `<um:login>` tag.
	`<um:setPassword>`	Updates the password for the user corresponding to the provided username.
Personalization Utilities	`<es:counter>`	Creates a `for loop` construct.
	`<es:date>`	Gets a date and time formatted string based on the user's time zone preference.
	`<es:forEachInArray>`	Iterates over an array.
	`<es:isNull>`	Checks to see if a value is null. If the value type is a `String`, also checks to see if the `String` is empty.
	`<es:monitorSession>`	Disallows access to a page if the session is not valid or if the user is not logged in.
	`<es:notNull>`	Checks to see if a value is not `null`. If the value type is a `String`, also checks to see if the `String` is not empty.
	`<es:preparedStatement>`	Creates a JDBC prepared statement.
	`<es:simpleReport>`	Creates a two-dimensional array out of a simple query.
	`<es:transposeArray>`	Transposes a standard [row][column] array to a [column][row] array.

Table 1-1 JavaServer Page (JSP) Tags Overview (Continued)

	`<es:uriContent>`	Pulls content from a URL.
WebLogic Utilities	`<wl:process>`	Provides a attribute-based flow control construct.
	`<wl:repeat>`	Used to iterate over a variety of Java objects, as specified in the set attribute.

Integration of External Components

A range of external components either come already integrated into the WebLogic Personalization Server, or can be integrated easily by a developer as extensions to the core components. A specific set of components that are known to be widely useful are described in Table 1-2. Other custom component integrations are possible given the JSP and EJB basis for the WebLogic Personalization Server, but the entire range of possibilities is not addressed here.

Table 1-2 Useful External Components the Personalization Server

External Component	Out-of-the-Box Support	Methods and Notes
DBMS	Integrated and tested with Cloudscape, Oracle 8.0.5, and 8.1.5.	Uses standard WebLogic Server JDBC connection pools.
LDAP authentication	Can be set up automatically using administration tools and property files.	Uses WebLogic Server security realms.
LDAP retrieval of user and group information	Can be set up automatically using administration tools.	Built into EJB persistence for User entity bean.

Table 1-2 Useful External Components the Personalization Server (Continued)

Legacy database of users	None.	Requires Unified User Profile extension of User entity bean.
Content Management engine	Reference implementation provided.	Provides API/SPI support from Documentum.
Legacy content database	None.	Requires either extension of Document entity bean or custom implementation of content management SPI.
Rules engine	JRules engine provided.	API/SPI with only JRules supported at this time as a valid service.

Support for Native Types

WebLogic Personalization Server supports the native types shown in Table 1-3.

Table 1-3 Native Types

Supported Type	Java Class	Notes
Boolean	java.lang.Boolean	Comparators: ==, !=
Integer	java.lang.Number	Comparators: ==, !=, <, >, <=, >=
Float	java.lang.Double	Comparators: ==, !=, <, >, <=, >=
Text	java.lang.String	Comparators: ==, !=, <, >, <=, >=, like

Table 1-3 Native Types (Continued)

Datetime	java.sql.Timestamp	Comparators: ==, !=, <, >, <=, >=
UserDefined	Defined by developer	Comparators: N/A User-defined properties may be programmatically set and gotten, but are not supported in the tools, rules, or content query expressions.

Any property can be a multi-value of a specific single native type as well. This is implemented as a java.util.Collection. Comparators for multi-values are contains and containsall, although the rules development tool will only allow the use of contains. The values possible as part of a multi-value may be restricted to a valid set, using the Property Set management tools.

2 Creating Personalized Applications with the Advisor

The Advisor recommends content by matching content to user profiles and producing a personalized application for the user. In essence, the Advisor ties together all the other services and components in the system to deliver personalized content.

This topic includes the following sections:

What Is the Advisor?

Content personalization allows Web developers to tailor applications to users. Based on data gathered from user profile, Request, and Session objects, the Advisor coordinates the delivery of personalized content to the end user.

The Advisor Delivers Content to a Personalized Application

The Advisor delivers content to a personalized application based on a set of rules and user profile informaton. It can retrieve any type of content from a Document Management system and display it in a JSP or use it in a servlet.

The Advisor ties together all the services and components in the system to deliver personalized content. The Advisor component includes a JSP tag library and an Advisor EJB (stateless session bean) that access the WebLogic Personalization Server's core personalization services including:

- User Profile Management
- Rules Service
- Content Management
- Foundation

The tag library and session bean contain personalization logic to access these services, sequence personalization actions, and return personalized content to the application.

This architecture allows the JSP developer to take advantage of the personalization engine using the Advisor JSP tags. In addition, a Java developer can access the underlying Personalization EJB and its features via the public Advisor bean interface. (For more information, see the API documentation in the *Javadoc*.) Think of the Advisor as sitting on top of the core services to provide a unified personalization API.

The Advisor gathers information from the user profile provided by the User Management component, submits that information to the Rules Service, runs the resulting queries against the document management system used in the Content Management component, and returns the content to the JSP.

The Advisor recommends document content for the following items:

- Web content included or excluded as determined by a user's classification using rules-based matching against user profile information. For more information about classifying users, see "Classifying Users with the JSP <pz:div> Tag" on page 2-5 and "Classifying Users with the Advisor Session Bean" on page 2-10.
- Documents returned by document attribute searches. For more information about searching for content, see "Selecting Content with the <pz:contentQuery> JSP Tag" on page 2-6 and "Selecting Content with the Advisor Session Bean" on page 2-12.
- Documents returned by content selectors using rules-based matching against user profile information. For more information about rules-based matching, see "Matching Content to Users with the <pz:contentSelector> JSP Tag" on page 2-7 and "Matching Content to Users with the Advisor Session Bean" on page 2-13.

The Advisor Provides Information About User Classifications

In addition to supplying content to a personalized application, the Advisor can also provide information about user classifications. For example, an application can ask the Advisor if, based on predefined rules, the current user is classified as a *Premier Customer* or an *Aggressive Investor*, and take action accordingly. The Advisor accomplishes this classification by gathering relevant user profile information, submitting it to the Rules Service, and turning on or off the supplied content based on the results of the rules execution.

You Can Use the Advisor in One of Two Ways

- **Using the JSP tags**. Developers will probably find it most useful to use the JSP tags when building typical pages. The tags provide ways to switch content on

and off based on user classification, return content based on a static query, and match content to users based on rules that execute a content query. The JSP tags that perform these tasks are: <pz:div> , <pz:contentSelector>, and <pz:contentQuery>.

- **Using the Advisor session bean**. The page or application developer may use the Advisor session bean directly in place of the tags, if desired. The Advisor session bean recommends content to personalized applications by matching advice requests with registered advislets that perform recommendations.

Creating Personalized Applications with the Advisor JSP Tags

The Advisor provides three JSP tags to help developers create personalized applications. These tags provide a JSP view to the Advisor session bean and allow developers to write pages that retrieve personalized data without writing Java source code.

Note: You must insert the following JSP directive into your JSP code to use the Advisor's <pz:div> and <pz:contentSelector> tags. The <pz:contentQuery> tag does not require that you extend the class.

```
<%@ page extends="com.beasys.commerce.axiom.p13n.jsp.P13NJspBase"
%>
```

- The <pz:div> tag turns user-provided content on or off based on the results of a classifier rule being executed. If the result of the classifier rule is `true`, it turns the content on; if `false`, it turns the content off.

  Note: The system turns on the content by inserting the content residing between the start and end <pz:div> tags in the JSP code. This content can include any valid JSP, including HTML tags, other JSP tags, and scriptlets. If the classifier rule returns `false`, the system skips the content between the start and end <pz:div> tags.

- The <pz:contentQuery> tag provides content attribute searching for content in a Content Manager. It returns an array of `Content` objects that a developer can handle in numerous ways.

Note: For more information about how WebLogic Personalization Server manages content, see Chapter 4, "Creating and Managing Content."

- The `<pz:contentSelector>` tag recommends content if a user matches the classification part of a content selector rule. When a user matches, the personalization engine executes a content query defined in the rule and returns the content back to the JSP page.

  Note: For information about defining a content selector rule, see "Creating a content selector rule" under "Creating and Managing Rules" in the *WebLogic Personalization Server User's Guide*.

In addition to using JSP tags to create personalized applications, you can work directly with the Advisor bean. For more information about using the bean, see "Creating Personalized Applications with the Advisor Session Bean" on page 2-8.

Classifying Users with the JSP <pz:div> Tag

The `<pz:div>` tag turns user-provided content on or off based on the results of a classifier rule being executed. If the result of the classifier rule is `true`, it turns the content on; if `false`, it turns the content off.

Note: For information about creating classifier rules, see the topic "Creating a classifier rule" in "Creating and Managing Rules" in the *WebLogic Personalization Server User's Guide.*

This example executes the *PremierCustomer* classifier rule and displays an alert to premier customers in the HTML page's output.

```
<%@ taglib uri="pz.tld" prefix="pz" %>
.
.
.
<pz:div
ruleSet="jdbc://com.beasys.commerce.axiom.reasoning.rules.RuleShe
etDefinitionHome/AcmeRules" rule="PremierCustomer">
    <p>Please check out our new Premier Customer bonus program…<p>
</pz:div>
```

You can also use the Advisor bean directly to classify users. For more information, see "Classifying Users with the Advisor Session Bean" on page 2-10.

Selecting Content with the <pz:contentQuery> JSP Tag

The <pz:contentQuery> tag provides content attribute searching for content in a Content Manager. It returns an array of Content objects that a developer can handle in numerous ways.

Note: For information about using the <pz:contentQuery> JSP tag, see "<pz:contentQuery>" on page 10-4. This tag provides similar functionality to the <cm:select> tag.

The following example executes a query against the content management system to find all content where the author attribute is *Hemmingway* and displays the Document titles found:

```
<%@ taglib uri="pz.tld" prefix="pz" %>
.
.
.
<pz:contentQuery id="docs"
contentHome="com.beasys.commerce.axiom.document.DocumentManager"
query="author = 'Hemmingway'" />

<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.beasys.commerce.axiom.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

Note: For more information about these JSP tags, see "<cm:printProperty>" on page 10-16 and "<es:forEachInArray>" on page 10-73.

You can also use the Advisor bean directly to select content. For more information, see "Selecting Content with the Advisor Session Bean" on page 2-12.

Matching Content to Users with the <pz:contentSelector> JSP Tag

The `<pz:contentSelector>` recommends content if a user matches the classification part of a content selector rule. When a user matches based on a rule, the Advisor executes the query defined in the rule to retrieve content.

Notes: For more information about this tag, see "<pz:contentSelector>" on page 10-6.

For information about creating classifier rules, see "Creating a Classifier Rule" in the *WebLogic Personalization Server User's Guide.*

The following example asks the Advisor for content specific to premier customers and then displays the `Document` titles as the results.

```
<%@ taglib uri="pz.tld" prefix="pz" %>
.
.
.
<pz:contentSelector id="docs" ruleSet="jdbc://com.beasys.
    commerce.axiom.reasoning.rules.
    RuleSheetDefinitionHome/AcmeRules"
    rule="PremierCustomerSpotlight"
    contentHome="com.beasys.commerce.axiom.document.
    DocumentManager" />
<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.beasys.commerce.axiom.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

You can also use the Advisor bean directly to match content to users. For more information, see "Matching Content to Users with the Advisor Session Bean" on page 2-13.

Creating Personalized Applications with the Advisor Session Bean

Java developers can work directly against the Advisor bean through a set of APIs to create personalized applications. This process provides an alternative to using the JSP tags to call into the bean.

Note: Refer to the API documentation in the *Javadoc* for more information about using the session bean to create personalized applications.

The following steps and Figure 2-1 provide a general overview of the process involved for an application to get advice from the Advisor.

1. Create an instance of the Advisor session bean.
2. Use the Advisor's `createTemplate` factory method to create a Request object.

   This method also determines the best advislet to use for the request by mapping `theKindOfRequest` to the best fit advislet.
3. Set the required and optional inputs for the `Request` object.
4. Call the `advise` method.

   The Advisor calls the best advislet to make the recommendation. The advislet determines the recommendations and the Advisor then passes the `AdviceResults` object back to the application.
5. The personalized application extracts the recommendation from the `AdviceResults` object and uses it in the application.

When a personalized application requests advice from the Advisor, the Advisor bean delegates the request to a registered advislet that can handle the request. The Advisor's job is to determine which registered advislet is best suited for making recommendations for the request, based on the advice request type.

The Advisor uses the advice request type to determine which registered advislet to delegate the advice request to. The advislet then makes the recommendations and returns the advice results back to the Advisor. This design encapsulates all of the advice logic into the advislet and allows advislets to be specialized.

Figure 2-1 Mapping a Request to an Adviselet

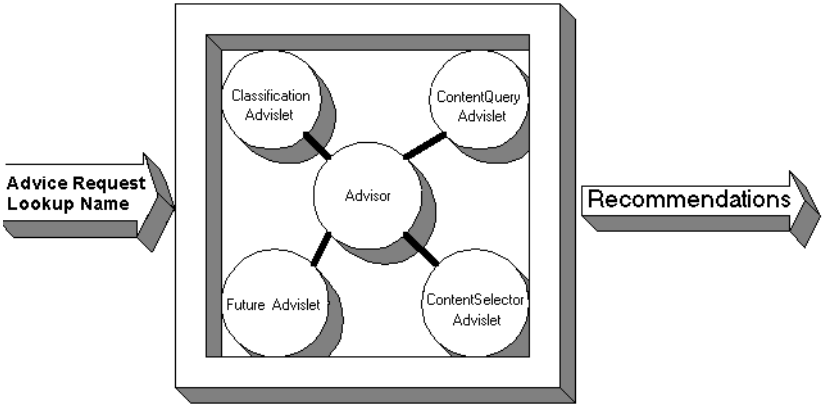


Table 2-1 shows the logic the Advisor uses to determine how to map a recommendation request to an advislet. Note that some combinations are not valid. For example, you cannot send a `<pz:bea.rules>` technique request with a ContentQueryAdviceRequest.

Table 2-1 Mapping an Advise Request to an Advislet

Kind of Advice Request	Inferred Advislet
ClassificationAdviceRequest	ClassificationAdvislet Uses rules-based matching with an inference engine that classifies a user.
ContentSelectorAdviceRequest	ContentSelectorAdvislet ■ Uses rules-based matching with an inference engine to classify a user. ■ Determines if the user matches the classification. ■ Selects content based on a content query.
ContentQueryAdviceRequest	ContentQueryAdvislet Performs a content attribute search with a content management system.

Classifying Users with the Advisor Session Bean

For classification requirements beyond what the JSP tags provide, or to use classification in a servlet, developers can use the Advisor EJB directly. The following sequence describes the process of asking the Advisor for a classification (refer to the *Javadoc* for API details).

Note: All classes used here reside in the `com.beasys.commerce.axiom.*` packages.

1. Create an instance of the Advisor session bean.
2. Call the Advisor's `createTemplate` method to get the correct `AdviceRequest` object. In this case, it should return a `ClassificationAdviceRequest`.
3. Set the required objects on the `ClassificationAdviceRequest`. These include the:
   - `Session` object (retrieved from `P13NJspHelper.createP13NSession(HttpServletRequest)`)
   - `User` object (retrieved from `P13NJspHelper.createP13NProfile(HttpServletRequest)`)
   - `Request` object (retrieved from `P13NJspHelper.createP13NRequest(HttpServletRequest)`)
   - `java.sql.Timestamp` object representing *now*
   - rule set name (For more information, see "What Are Rule Sets?" in the *WebLogic Personalization Server User's Guide*.)
   - rule name (For more information, see "Creating and Managing Rules" in the *WebLogic Personalization Server User's Guide*.)
   - `Successor` object (for example, the user's group).
4. Call the `advise` method on the Advisor.
5. The Advisor returns a subclass of `AdviceResults`. In this case, it should return a `ClassificationAdviceResults` object. If the classification object exists in the results, the classification is `true`. If the object is `null`, the classification is not true.

A basic example of using the bean for classification might look like the following:

Note: This code is just a model and is not complete. The complete example resides in the following files:
`<WLPS_installation_directory>/server/public_html/portals/repository/portlets/advisor_ejb_example.jsp`
`<WLPS_installation_directory>/src/examples/advisor/ClassificationExample.java`

```
try
{
    ClassificationAdviceRequest request = null;
    AdviceRequest arequest = anAdvisor.createRequestTemplate
        ("ClassificationAdviceRequest");
    request = (ClassificationAdviceRequest)arequest;
    HttpServletRequest someRequest = (HttpServletRequest)
        pageContext.getRequest();
    ConfigurableEntity user = P13NJspHelper.createP13NProfile(httpRequest);
    request.setUser(user);

    request.setHttpRequest(P13NJspHelper.createP13NRequest(httpRequest));
    request.setHttpSession(P13NJspHelper.createP13NSession(httpRequest));
    request.setNow(new Timestamp(System.currentTimeMillis()));
    request.setRuleSet(ruleset);
    request.setRule(rule);

    AdviceResults result = anAdvisor.advise(request);
    Classification classification = ((ClassificationAdviceResults)result).
        getClassification();
    return classification != null;
    }
catch(Exception e)
{
    e.printStackTrace();
}
```

Note: You can also use the JSP `<pz:div>` tag to classify users. (For more information, see “Classifying Users with the JSP <pz:div> Tag” on page 2-5.)

Selecting Content with the Advisor Session Bean

For content selection requirements beyond what the JSP tags provide, or to use classification in a servlet, developers can use the Advisor EJB directly. The following sequence describes the process of asking the Advisor for content (refer to the *Javadoc* for API details).

1. Create an instance of the Advisor session bean.
2. Call the Advisor's `createTemplate` method to get the correct `AdviceRequest` object. In this case, it should return a `ContentQueryAdviceRequest`.
3. Set the parameters on the `ContentQueryAdviceRequest`, including:
    - *`contentHome`* (required): the JNDI name to find a content home
    - *`query`* (required): the query to run against the system
    - *`sortBy`* (optional)
    - *`max`* (optional)
4. Call the `advise` method on the Advisor.
5. The Advisor returns a subclass of `AdviceResults`. In this case, it should return a `ContentQueryAdviceResults` object, from which you can retreive an array of `Content` objects.

A basic example of using the bean for a content query might look like the following:

```
try
{
    AdviceRequest arequest = anAdvisor.createRequestTemplate(
        "ContentQueryAdviceRequest");

    request = (ContentQueryAdviceRequest)arequest;
    request.setQuery(query);
    request.setMax(max);
    request.setSortBy(sortby);
    request.setContentHome(home);

    AdviceResults result = anAdvisor.advise(request);
    Collection docs = ((ContentQueryAdviceResults)result).getContent();
    if (docs==null)
    {
        return new Content [0];
```

```
    }
    return (Content[])docs.toArray(new Content[docs.size()]);

}
catch(Exception e)
{
    e.printStackTrace();
}
```

Note: You can also use the JSP `<pz:contentQuery>` tag to select content. (For more information, see "Selecting Content with the <pz:contentQuery> JSP Tag" on page 2-6.)

Matching Content to Users with the Advisor Session Bean

For content matching requirements beyond what the JSP tags provide, or to use content selection in a servlet, developers can use the Advisor EJB directly. The following sequence describes the process of asking the Advisor for content (refer to the *Javadoc* for API details).

Note: All classes used here reside in the `com.beasys.commerce.axiom.p13n.*` packages.

1. Create an instance of the Advisor session bean.
2. Call the Advisor's `createTemplate` method to get the correct `AdviceRequest` object. In this case, it should return a `ContentSelectorAdviceRequest`.
3. Set the required objects on the `ClassificationAdviceRequest`. These include the:
    - `Session` object (retrieved from P13NJspHelper.createP13NSession(HttpServletRequest))
    - `User` object (retrieved from P13NJspHelper.createP13NProfile(HttpServletRequest))
    - `Request` object (retrieved from P13NJspHelper.createP13NRequest(HttpServletRequest))
    - `java.sql.Timestamp` object representing *now*

- rule set name (For more information, see "What Are Rule Sets?" in the *WebLogic Personalization Server User's Guide*.)
- rule name (For more information, see "Creating and Managing Rules" in the *WebLogic Personalization Server User's Guide*.)
- `Successor` object (that is, the user's group)
- *`contentHome`* (required): the JNDI name to find a content home
- *`query`* (required): the query to run against the system
- *`sortBy`* (optional)
- *`max`* (optional)

4. Call the `advise` method on the Advisor.
5. The Advisor returns a subclass of `AdviceResults`. In this case, it should return a `ContentQueryAdviceResults` object, from which you can retreive an array of `Content` objects.

A basic example of using the bean for content selection might look like the following.

Note: This code is just a model and is not complete. The complete example resides in the following files:
`<WLPS_installation_directory>/server/public_html/portals/repository/portlets/advisor_ejb_example.jsp`
`<WLPS_installation_directory>/src/examples/advisor/ContentSelectorExample.java`

```
try
{
    AdviceRequest arequest = anAdvisor.createRequestTemplate
        ("ContentSelectorAdviceRequest");

    request = (ContentSelectorAdviceRequest)arequest;
    HttpServletRequest someRequest =
        (HttpServletRequest)pageContext.getRequest();
    ConfigurableEntity user = P13NJspHelp.createP13NProfile(someRequest);

    request.setUser(user);
    request.setHttpRequest(P13NJspHelper.createP13NRequest(someRequest));
    request.setHttpSession(P13NJspHelper.createP13NSession(someRequest));
    request.setNow(new Timestamp(System.currentTimeMillis()));
    request.setRuleSet(ruleset);
    request.setRule(selector);
    request.setMax(max);
    request.setSortBy(sortby);
    request.setContentHome(home);
    request.setQuery(query);

    AdviceResults result = anAdvisor.advise(request);
    Collection docs = ((ContentQueryAdviceResults)result).getContent();
}
catch(Exception e)
{
    e.printStackTrace();
}
```

Note: You can also use the JSP `<pz:contentSelector>` tag to match content to users. (For more information, see "Matching Content to Users with the <pz:contentSelector> JSP Tag" on page 2-7.)

3 Foundation Classes and Utilities

The Foundation is a set of miscellaneous utilities to aid JSP and Java developers in the development of personalized applications using the WebLogic Personalization Server. Its utilities include JSP files and Java classes that JSP developers can use to gain access to functions provided by the server, and helpers for gaining access to the Advisor services.

This topic includes the following sections:

- Flow Manager
  - Hot Deployment
  - Dynamic Flow Determination and Handling
  - Property Set Usage
  - Webflow
  - Accessing Your Application via the Flow Manager
- Repository
- HTTP Handling
- Personalization Request Object
  - Default Request Property Set
- Personalization Session Object
  - Default Session Property Set
- Utilities

- JspHelper
- JspBase
- P13NJspBase
- ContentHelper
- CommercePropertiesHelper

- Utilities in commerce.util Package
  - ExpressionHelper
  - TypesHelper

Flow Manager

The Flow Manager is a servlet implementation that allows the hot deployment of applications within the WebLogic Application Server. Flow Manager also adds flexibility to navigation through the system by allowing navigation information to move off the JSP page and into a single point of control. Using a destinationdeterminer and a destinationhandler, the Flow Manager dynamically determines a destination for a given page request and dynamically handles it.

Note: The Flow Manager replaces the functionality previously supplied by the Portal Service Manager and JSP Service Manager. All the functionality of the service managers now reside within the Flow Manager. The JSP Service Manager and the Portal Service Manager have been deprecated.

Hot Deployment

Hot deployment allows you to register a portal or non-portal application without restarting the server. To add a new portal with the Flow Manager, you simply create a new instance of a property set. The changes become visible according to a configurable refresh setting (ttl).

Dynamic Flow Determination and Handling

The Flow Manager allows the determination of page routing to be centralized on the server based on an application's needs. To define properties of your unique application, you will create a property set of type `APPLICATION_INIT`. (See "Property Set Usage" on page 3-6.) There are three required values:

- destinationdeterminer—An implementation of the `com.beasys.commerce.foundation.flow.DestinationDeterminer` interface.
- destinationhandler—An implementation of the `com.beasys.commerce.foundation.flow.DestinationHandler` interface.
- ttl—How long (in milliseconds) before reloading the application init property set.

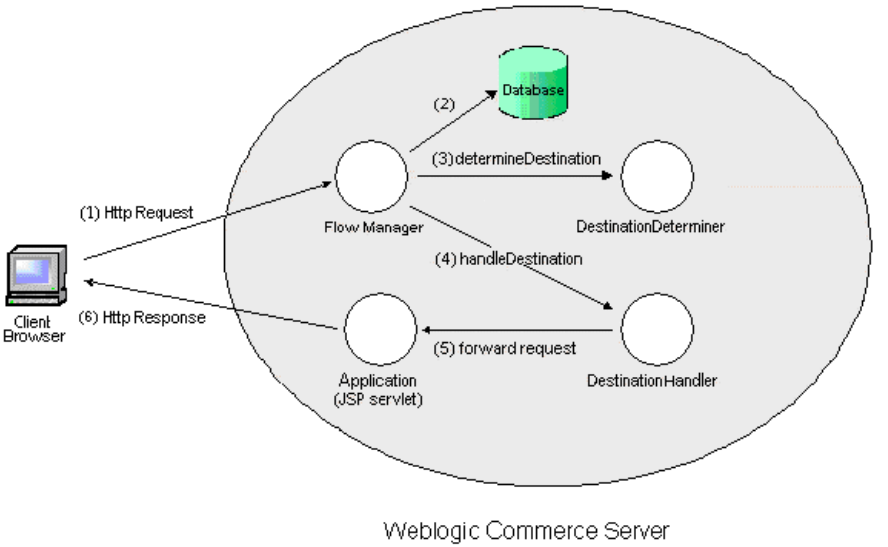
How the FlowManager Works

When WebLogic Personalization Server is installed, the Flow Manager servlet is registered with the WebLogic Server in the `weblogic.properties` file:
`weblogic.httpd.register.application=com.beasys.commerce.foundation.flow.FlowManager`

To access the servlet, a client browser makes an HTTP request. For example:
http://localhost:7501/application/exampleportal

In this example, "application" is the registered servlet (the Flow Manager), and "exampleportal" is the `APPLICATION_INIT` property set that you defined.

The following diagram illustrates how the Flow Manager handles the request.



Let's look at the diagram one step at a time, using our example.

1. A client browser makes an HTTP request via a form submission, hyperlink, etc.

   In this example, the request is for the exampleportal at http://localhost:7501/application/exampleportal.

   WebLogic Server (WLS) routes the request to the servlet registered in `weblogic.properties` with the name "application," which is the Flow Manager.

2. The request is analyzed within the servlet, and the path-info is pulled out. The path-info is the name of the property set to retrieve.

   In our example, the Flow Manager extracts the string "exampleportal" from the URL.

   The property set is retrieved from the database (or the cache).

   Using the SchemaManager, the Flow Manager reads the Application Init property set of that name from the database. The Flow Manager reads the properties named "destinationdeterminer" and "destinationhandler" from the property set and instantiates each class.

Note: Implementations of these classes are to be provided by the application developer, as needed.

3. The Flow Manager then calls the destinationdeterminer defined in the property set, using the `DestinationDeterminer.determineDestination` method.

   In this example, the PortalDestinationDeterminer class does not find a `DESTINATION_URI` in the request and the user is not logged in, so it retrieves the `"defaultdest"` property and returns the destination string `"/portals/example/portal.jsp"` to the Flow Manager.

4. The Flow Manager then calls the `DestinationHandler.handleDestination` method. The destination returned from the previous call is passed on to the destinationhandler defined in the property set.

5. In this example, the portal uses the `ServletDestinationHandler` which calls the `requestDispatcher.forward` method, passing execution control to the portal.jsp servlet.

6. Finally, application processing proceeds in the servlet which uses the response object to return data to the client browser.

Property Set Usage

The Property Set Management Administration Tools include a class of property sets called Application Initialization Property Sets. To support non-portal based personalized applications, the Flow Manager uses `_DEFAULT_APP_INIT`. For portals, the Flow Manager uses the`_DEFAULT_PORTAL_INIT` property set. For more information, see the topic "`_DEFAULT_PORTAL_INIT` Property Set" in the chapter Creating and Managing Portals in the *WebLogic Personalization Server User's Guide*.

The following three properties support the Flow Manager:

Property Name	Required	Description
destinationdeterminer	Yes	Used by Flow Manager to determine JSP page navigation.
destinationhandler	Yes	Used by Flow Manager to execute JSP page navigation.
ttl	Yes	Time to live determines (in milliseconds) how often the Flow Manager reloads the application init property set from the database.

destinationdeterminer Property

The destinationdeterminer evaluates an HTTP request and determines which servlet to route it to.

The value provided for this property should be the name of a class that implements the `com.beasys.commerce.foundation.flow.DestinationDeterminer` interface. If appropriate, use a default implementation provided by WebLogic Personalization Server or WebLogic Commerce Server. Otherwise, develop your own implementation according to the needs of your application.

destinatationhandler Property

Given a destination route, the destinationhandler is responsible for invoking the requested processing.

The value provided for this property should be the name of a class that implements the `com.beasys.commerce.foundation.flow.DestinationHandler` interface. If appropriate, use a default implementation provided by WebLogic Personalization Server or WebLogic Commerce Server. Otherwise, develop your own implementation according to the needs of your application.

ttl (time-to-live) Property

Time-to-live (ttl) represents how often (in milliseconds) the Flow Manager reloads the application init property set from the database. This allows you to make property set changes visible while the portal is running.

Note: To force immediate reloading of the property set, append the "flowReset" argument to your URL, like this:
http://localhost:7001/application/exampleportal?flowReset=true

Creating a New Property Set

1. Open the Administration Tools Home page. Click the Property Set Management icon to open the Property Set Management screen.
2. From the main Property Set Management screen, click Create.
3. Name the new property set you are creating (100 character maximum). The name of the property set should be the same as the name you used to create the portal, or the name you will use to access the application.
4. Enter a description of the property set (255 character maximum).
5. From the Copy Properties From drop-down list, select
   `APPLICATION_INIT._DEFAULT_PORTAL_INIT` (for a portal)
   or
   `APPLICATION_INIT._DEFAULT_APP_INIT` (for a non-portal application).
6. From the Property Set Type drop-down list, select Application Init.
7. Click the Create button.
8. At the top of the page, in red, you will see the message “Property Set creation was successful.” (Or, you will see an error message indicating why the property set was not created.)
9. Click Back to return to the main Property Set Management screen.

Set Parameters for Your Portal or Application

1. From the Property Set Management Home page, under the Application Initialization Property Sets heading, click the name of the property set you just created.
2. A Property Set page comes up, allowing you to set parameters.
3. **Note:** For non-portal applications, skip this step.
   To edit the portal name, click the Edit button to the right of the "portal name" property. Change the default value from `UNKNOWN` to the name of your portal, as you created it in Portal Management.
4. Edit the destinationdeterminer property. Either accept the default, or edit to provide your own implementation of these classes.
5. Edit the destinationhandler property. Either accept the default, or edit to provide your own implementation of these classes.
6. Customize any other properties you choose. For information about customizing properties in portals, see "Creating and Managing Portals" in the *WebLogic Personalization Server User's Guide*.
7. When you have finished setting properties, click the Finished button at the bottom of the page.

Webflow

Webflow is a mechanism that controls the flow of a user session by determining which pages are displayed in a browser. The Flow Manager provides the basic infrastructure to support the Webflow functionality. On the WebLogic Personalization Server, Webflow does a simple dispatch to a target destination. When a request comes in from the browser, a destinationdeterminer looks for a `dest` parameter on the URL and grabs what `dest` asks for.

The WebLogic Commerce Server extends the Flow Manager with the addition of a Webflow properties file. By setting parameters, you can determine how Webflow reacts to events and which pieces of business logic to execute. When a request comes into WebLogic Commerce Server from a browser, Webflow looks for the `origin` and `event` parameters in the `webflow.properties` file and grabs what the properties file asks for.

The Webflow scheme provides a good example of centralized routing information. It provides an implementation of the destinationdeterminer which uses a properties file resource as a state table to determine the routing destination. For more information about the Webflow implementation in the WebLogic Commerce Server, see the *Webflow and Pipeline Management Guide*.

Accessing Your Application via the Flow Manager

The exact URL you use depends upon whether or not you have deployed your application as a Web application. WebLogic Personalization Server includes two sample configurations of the Acme Demo Portal, both as a Web application/Web archive deployment and a non-Web application configuration. For more information, see "Deploying a Portal as a Web Application" on page 7-2.

Using Flow Manager with a Web Application

The URL for Acme Demo Portal accessed as a Web Archive (WAR) application is: http://localhost:7501/portal

The "portal" portion of the URL is the context name for the Web application as defined in the `weblogic.properties` file:

```
weblogic.httpd.webApp.portal=C:/WebLogicCommerceServer3.2/server/
webapps/examples/portal/portal.war
```

Within the portal.war, the `web-inf/web.xml` file includes `<servlet>` and `<servlet-mapping>` entries for the Flow Manager, associating all URL accesses starting with `"application/*"` with the Flow Manager's class.

Note: To use hot deployment here, the additional portal code deployments must live in the Web application directory, so the same context name ("portal") can be used to access them.

In the above URL example, the HTTP request defaults to `index.jsp`, as defined in the `<welcome-file-list>` element in the Web application `web.xml` file. When the HTTP request is routed to the Flow Manager, it extracts the path information "exampleportal" from the URL and retrieves the property set of the same name from the server (or cache). The "destinationdeterminer" and "destinationhandler" properties are used to instantiate the supporting implementations, and processing proceeds as described above.

Using Flow Manager with a non-Web Application

The URL for Acme Demo Portal accessed as a non-Web application is:

http://localhost:7501/application/exampleportal

The Flow Manager accesses a non-Web application in almost the same way it accesses a Web application. There are two primary differences:

- There is no web-application context path
- The definition of the Flow Manager servlet is drawn from the WebLogic properties file:
  `weblogic.httpd.register.application=com.beasys.commerce.foundation.flow.FlowManager`

Otherwise, processing proceeds as described above (see “Using Flow Manager with a Web Application” on page 3-9).

Repository

The repository feature allows you to specify a single directory to contain files that otherwise would have to be replicated several times.

The administration pages for components take advantage of the repository feature to store images shared between components. Each HTML reference to an image is wrapped by the `ToolsJspBase.fixupRelativeURL` method. This method first looks in the path-relative directory for the image specified in the argument. If not found there, the *`repositorydir`* specified in the `weblogic.properties` file (for the wlpsadmin servlet) is searched for the image.

For portals, the default portal (Acme) implementation has its files contained in a folder named `repository` and specifies a `repositorydir=/portal/repository`. In an extreme example, a second portal which only differed from Acme in one file, say `portal.jsp`, would be created by creating a new directory named `extremeExample` and by adding one file (`portal.jsp`) to it. All files supporting the `extremeExample` portal which were not found in its `workingdir` will be fetched from the repository directory.

For Web applications, the repository directory must reside inside the Webapp context.

HTTP Handling

Both the `<pz:div>` and `<pz:contentselector>` tag implementations send `HttpRequest` and `Session` information to the Advisor.

The Advisor includes helper classes that transform an `HttpRequest` and `Session` into serializable personalization surrogates for their HTTP counterparts. These surrogates are compatible with the Personalization Rules Service which uses these objects to execute classifier and content selector rules.

Personalization Request Object

In order to use `HttpRequest` parameters in requests to the rules service, they must be wrapped in a Personalization `Request` object (`com.beasys.commerce.axiom.p13n.http.Request`) before they can be set on the appropriate `AdviceRequest` (see the Javadoc API documentation). While the `HttpRequest` object can be wrapped by directly calling the Personalization `Request` constructor, it is recommend that developers use the `createP13NRequest` helper method on `P13NJspBase` (`com.beasys.commerce.axiom.p13n.jsp.P13NJspBase`) for this purpose. See the Javadoc API documentation for more information.

Caution: The tag implementations for the `<pz:div>` and `<pz:contentSelector>` tags create the Personalization `Request` surrogate for the `HttpRequest` before calling the Advisor bean, so JSP developers need not worry about the details of the `Request` object. Only developers accessing the `PersonalizationAdvisor` bean directly need to wrap the `HttpRequest` object explicitly.

In order to avoid confusing results on `getProperty` method calls, developers need to know the algorithm used in the `getProperty` method implementation for determining the value of the property requested . When the `Request's getProperty` method is called (for example, by a rules engine), the system uses the following algorithm to find the property:

1. The `getProperty` method first looks in the `HttpRequest's` attributes for the property.
2. If not found, `getProperty` looks for the property in the `HttpRequest` parameters.
3. If not found, `getProperty` looks in the HTTP headers.
4. If not found, `getProperty` looks in the `Request` methods (`getContentType`, `getLocale`, etc.).
5. If not found, `getProperty` uses the *`scopeName`* parameter to find a schema entity for a `Request` schema group name and, if the schema is found, uses the default value in the schema.
6. If not found, `getProperty` uses the default value passed into the method call.

Default Request Property Set

For Rules developers to write rules for classifier rules that contain conditions based on an `HttpRequest`, there must be a property set defined for the `HttpRequest`. By default, WebLogic Personalization Server ships with a default request property set for the standard `HttpRequest` properties. Developers adding properties to the request programatically will need to add those properties to the default property set in order for them to be available to the rules editor and service.

The default `Request` properties include the following

Request Property Name	Associated Request Method
Request Method	request.getMethod()
Request URI	request.getRequestURI()
Request Protocol	request.getProtocol()
Servlet Path	request.getServletPath()
Path Info	request.getPathInfo()
Path Translated	request.getPathTranslated()
Locale	request.getLocale()
Query String	request.getQueryString()
Content Length	request.getContentLength()
Content Type	request.getContentType()
Server Name	request.getServerName()
Server Port	request.getServerPort()
Remote User	request.getRemoteUser()
Remote Address	request.getRemoteAddr()
Remote Host	request.getRemoteHost()
Scheme	request.getAuthType()

Request Property Name	Associated Request Method
Authorization Scheme	request.getScheme()
Context Path	request.getContextPath()
Character Encoding	request.getCharacterEncoding()

Personalization Session Object

In order to use HTTP Session parameters in requests to the rules service, they must be wrapped in a Personalization `Session` object (`com.beasys.commerce.axiom.p13n.http.Session`) before they can be set on the appropriate `AdviceRequest` (see the Javadoc API documentation). While the `HttpSession` object can be wrapped by directly calling the Personalization `Session` constructor, using the `createP13NSession` helper method on `P13NJspBase` (`com.beasys.commerce.axiom.p13n.jsp.P13NJspBase`)is recommended. See the Javadoc API documentation for more information.

The tag implementations for the `<pz:div>` and `<pz:contentselector>` tags create the Personalization Session surrogate for the HTTP Session before calling the Advisor bean, so JSP developers need not worry about the details of the `HttpSession` object. Only developers accessing the `PersonalizationAdvisor` bean directly need to wrap the `HttpSession` object explicitly.

Default Session Property Set

For Rules developers to write rules that contain conditions based on an HTTP session, there must be a property set defined for the HTTP session. WebLogic Personalization Server ships with a default session property that contains no values set as a placeholder. There are no default `Session` property set values. Developers adding properties to the session programatically will need to add those properties to the default property set in order for them to be available to the rules editor and service.

The Personalization `Session` object retrieves the session values from the Service Manager (see "Repository" on page 3-11) for the current thread and clones them so they can be used on a remote machine.

The Personalization Session uses the following algorithm to find a property:

1. It first looks in its own cloned HTTP Session properties.
2. If it does not find the property, it locates the schema for the Personalization Session for the `scopeName` method parameter.
3. If it still does not find the property, it uses the *`scopeName`* parameter to find a schema entity for the `Session` schema group name and, if the schema is found, uses the default value in the schema.
4. If it still does not find the property, it uses the default value passed into the `getProperty` method call.

Utilities

You can view more detailed documentation for the utilities listed here in the Javadoc API documentation.

JspHelper

`JspHelper` provides get methods to the `JspServiceManager` URI, the working directory, the home page, and the current page. It also provides set and get methods for session values and JSP destinations.

Note: Some of these methods assume that the JspServiceManager model is being used.

JspBase

`JspBase` acts as a base class for all JSP pages that use a `JspServiceManager`. A wide variety of important methods are provided:

- `Get` methods for the TrafficURI, working directory, repository directory, default destination, RequestURI, default successor, home page, and current page.
- Methods to create URLs, and fixup (fully qualified) URLs.
- Methods to override the destination tag.
- Methods to set and get logged-in status.
- Methods to get, set, and remove session values.
- A method to convert HTML special characters to HTML entities.
- Methods to set the user and successor.

P13NJSPHelper

`P13NJspBase` provides convenience methods to developers writing JSP pages (including but not limited to portals and portlets) that include personalized content. It provides methods for wrapping HTTP `Request` and `Session` objects into their personalization surrogates, and a method for retrieving the current Profile (User, Group, and so on) for an application.

P13NJspBase

`P13NJspBase` acts as a base class for all personalized JSP pages. This class extends `JspBase`.

ContentHelper

`ContentHelper` simplifies the life of the developer using the Content Management component. Methods are provided to get an array of content given a search object, to get the length of a piece of content. Constants for the default `Content` and `Document` homes are also provided.

CommercePropertiesHelper

`CommercePropertiesHelper` allows easy access to the commerce.properties file's properties. Methods are provided to return the values of a given keys as various data types. Also provided is a method to return all keys that start with a given string as a string array. For example, use the method to find all of the keys that start with *personalization.portal*.

Utilities in commerce.util Package

ExpressionHelper

`ExpressionHelper` handles dealing with `Expression`, `Criteria`, and `Logical` objects. It contains methods for parsing query strings into `Expressions`, joining `Expressions` into `Logicals`, normalizing `Expressions`, changing `Expressions`, `Logicals`, and `Criteria` into `Strings`, and turning `Expressions` into `String` trees for debugging purposes.

TypesHelper

`TypesHelper` provides a set of constants corresponding to the types and operators used in the configurable entity properties. Methods are provided to get string representations of the type names, to determine a type from a `java.sql.Type`, and to get the list of comparison operators for a certain type.

CHAPTER

4 Creating and Managing Content

The Content Manager provides content and document management capabilities for use in personalization services. The Content Manager works with files or with content managed by third-party vendor tools from Documentum and Interwoven.

This topic includes the following sections:

- What Is the Content Manager?
  - Using Third-party Tools
  - Constructing Queries Using Java
  - Differences Between Content Management and Document Management
  - Using the Document Servlet
  - JSP Tags
- Configuring the Content Manager
  - Configuring the DocumentSchema EJB Deployment Descriptor
  - Configuring the DocumentManager EJB Deployment Descriptor
  - Setting Up Connection Pools
  - Configuring WebLogic Commerce Properties
  - Using the Show Document Servlet
  - Querying Document Content
  - Structuring a Query
  - Using Comparison Operators to Construct Queries

- Using the BulkLoader to Load File-based Content
- Using Content Management JSP Tags

What Is the Content Manager?

The Content Manager run-time subsystem provides access to content via both tags and EJBs. The Content Management tags allow a JSP developer to receive an enumeration of Content objects by querying the content database directly using a search expression syntax.

The Content Manager component works alongside the other components to deliver personalized content, but does not have a GUI-based tool for edit-time customization. The content engine behind the `ContentManager` may be set up to be the reference implementation, provided out of the box, or Documentum. The Content Management component supports querying that returns content from a content repository using several methods:

- **Search for content by metadata—**Boolean logic searching evaluates content that matches a metadata/operator/value criteria.
- **Retrieve content by ID—**the system allows retrieval of raw bytes of content data—either in blocks or in its entirety—through the content's known identifier.
- **Query content metadata by ID—**the system, through the known identifier of a content piece, can query the metadata describing the content piece. Several metadata attributes provide information about the content. The query language maps some attribute names onto explicit attributes of the `Content` or `Document` objects the query searches. Queries searching for `Content` objects support the following case-sensitive explicit attribute names:
  - *identifier*: Corresponds to the unique `String` identifier of the `Content` (that is, the getIdentifier method).
  - *mimeType*: Corresponds to the `String` MIME type of the `Content` (that is, the `getMimeType` method).
- Queries searching for `Document` objects support the following additional case-sensitive explicit attribute names:

- *size*: Corresponds to the `Long` size of the document in bytes (that is, the `getSize` method). Documents without file bytes will have a size of 0 or less.
- *version*: Corresponds to the `Integer` version number of the document (that is, the `getVersion` method).
- *author*: Corresponds to the `String` identifier of the author of the document (that is, the `getAuthor` method).
- *creationDate*: Corresponds to the `Timestamp` of when the document was created (that is, the `getTimestamp` method).
- *modifiedBy*: Corresponds to the `String` identifier of the individual who last modified the document (that is, the `getModifiedBy` method).
- *modifiedDate*: Corresponds to the `Timestamp` of when the document was last modified (that is, the `getModifiedDate` method).
- *lockedBy*: Corresponds to the `String` identifier of the individual who has the document locked (that is, the `getLockedBy` method).
- *description*: Corresponds to the `String` description of the document (that is, the `getDescription` method).
- *comments*: Corresponds to any `String` comments about the document (that is, the `getComments` method).

Note: All other attribute names in queries are considered implicit metadata properties.

- **Get content schema by name—**the document management system (DMS) contains a set of named schemas that describe a set of non-standard metadata attributes. Each piece of content in the DMS is associated with one of these schemas and each schema specifies valid attributes
- **Get content schema names—**a user can query the system for a list of all schema names a DMS supports.

Note: See “Querying Document Content” on page 4-14 for more information about queries.

Using Third-party Tools

BEA partners with third-party vendors to add flexibility to the WebLogic Personalization Server. The Content Manager works with Interwoven's Teamsite/OpenDeploy product and Documentum's 4i product. Both these products provide robust, content-creation management solutions while the Content Manager personalizes and serves the content to the end user.

How Do I Choose What Content Management Tools to Use?

- WebLogic Personalization Server **BulkLoader—**for sites with limited content personalization needs and existing metatagged HTML, WebLogic Personalization Server includes a command-line utility called the BulkLoader. The BulkLoader can parse a directory of HTML files and store their URL address and metadata attributes in a JDBC store. The BulkLoader automatically creates the schema for these attributes.

  Interwoven Teamsite/OpenDeploy Integration—for customers who have larger amounts of content and want more control over the publishing and tagging of content, WebLogic Personalization Server provides integration with the Interwoven Teamsite/OpenDeploy product. Teamsite is a content capture, versioning, staging, and publishing system. Teamsite templates define the metadata attributes for the content; customers can use the templates to categorize the documents during the creation process. Teamsite is not a run-time engine; it does not actually get queried by the WebLogic Personalization Server. Once content is captured, the Interwoven OpenDeploy workflow capability works with the WebLogic Personalization Server to publish content to a well-known location and the metadata to our JDBC store.

  A WebLogic Personalization Server customer can purchase the Interwoven Teamsite product for content management and publishing. The extensions for Interwoven are available from the BEA download center as one of the WebLogic Commerce Servers download options.

- **Documentum 4i Integration—**as a compatible document management solution, BEA provides integration with the Documentum 4i product. Documentum products manage the metadata and documents in their own repositories.

  Content developers working with Documentum create metadata capture tools in Documentum 4i. When they complete the authoring process and check in their documents to Documentum, they must tag the content. Depending on

administrative options, this content can be available immediately to the WebLogic Personalization Server for display on the customer's e-commerce site.

WebLogic Personalization Server customers can purchase Documentum 4i to manage the documents or content for their e-commerce site. The JDBC driver for Documentum 4i is available from the BEA download center as one of the WebLogic Commerce Servers download options.

Constructing Queries Using Java

To construct queries using Java syntax instead of using the query language supplied with the Content Management component, refer to the Javadoc API documentation.

Note: Use the constants in TypesHelper when calling `Logical.setLogical` and `Criteria.setComparator`.

The `ContentManager` session bean is the primary interface to the functionality of the Content Management component. Using a `ContentManager` instance, content is returned based on a Search object with an embedded `Expression`. An `Expression` is a Boolean tree of arbitrary depth, with other sub-`Expressions` as nodes. The `Expression` interface is meant to be abstract, where the actual instances are `Logical` or `Criteria` interfaces. As an example, the expression `color == 'red' && price > 50` would consist of a `Logical` with the value *and* that has as children two `Criteria`.

Differences Between Content Management and Document Management

`Content` objects include metadata about the content. Metadata provides a means to query and match content with users by allowing the system to retrieve content based on the metadata that describes the content. In general, some kind of content management system provides services such as retrieval of content and content authoring services including creation, editing, versioning, and workflow.

`Documents` are a specialized type of `Content` that provide two methods for retrieval: a metadata-searching mechanism and retrieval of the pure bytes of the document's file. `Documents` should include additional explicit metadata properties related to the file

and its versioning, including its size, name, path, author, and version. A document management system usually provides document-based services for documents that reside in the system's repository.

WebLogic Personalization Server provides the entire `Content` object model; however, it only provides the `Document` object as a concrete implementation (subclass) of the `Content` class.

Using the Document Servlet

The Content Management component includes a servlet capable of outputting the contents of a `Document` object. This servlet is useful when streaming the contents of an image that resides in a content management system or to stream a document's contents that are stored in a content management system when an HTML link is selected. The servlet supports the following Request/URL parameters:

Request Parameter	Required	Description
contentHome	Maybe	If the *`contentHome`* initialization parameter is not specified, then this is required and will be used as the JNDI name of the DocumentHome. If the *`contentHome`* initialization parameter is specified, this is ignored.
contentId	No	The string identifier of the Document to retrieve. If not specified, the servlet looks in the `PATH_INFO`.
blockSize	No	The size of the data blocks to read. The default is 8K. Use 0 or less to read the entire block of bytes in one operation.

The servlet only supports `Documents`, not other subclasses of `Content`. It sets the *`Content-Type`* to the `Document's` mimeType and, the *`Content-Length`* to the `Document's` size, and correctly sets the *`Content-Disposition`*, which should present the correct filename when the file is saved from a browser.

Example 1: Usage in a JSP

This example searches for news items that are to be shown in the evening, and displays them in a bulleted list.

```
<cm:select
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME%>" max="5"
sortBy="creationDate ASC, title ASC"
query="type = 'News' && timeOfDay = 'Evening' && mimetype like
'text/*' " id="newsList" />

<ul>
    <es:forEachInArray array="<%=newsList%>" id="newsItem"
    type="com.beasys.commerce.axiom.content.Content">
        <li><a href="/showDocServlet/<cm:printProperty
        id="newsItem" name="identifier" encode="url"/>
       &contentHome=<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME%>">
        <cm:printProperty id="newsItem" name="title"
        encode="html"/></a>
    </es:forEachInArray>
</ul>
```

Example 2: Usage in a JSP

This example searches for image files that match keywords that contain *bird* and displays the image in a bulleted list.

```
<cm:select
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME%">"
max="5" sortBy="name" id="list" query="Keywords like '*birds*' &&
mimeType like 'image/*'" />
<ul>
    <es:forEachInArray array="<%=list%>" id="img"
    type="com.beasys.commerce.axiom.content.Content">
       <li><img src="/showDocServlet?contentId=<cm:printproperty
       id="img" name="identifier" encode="url"/>
       &contentHome=<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME%>">
    </es:forEachInArray>
</ul>
```

JSP Tags

The Content Management component includes the following four JSP tags. These tags allow a JSP developer to include non-personalized content in a HTML-based page. Note that none of the tags support or use a body.

- The `<cm:select>` tag uses only the search expression query syntax to select content. See the "JSP Tag Library Reference" in the WebLogic Personalization Server *Developer's Guide* for more information.
- The `<cm:selectById>` tag retrieves content using the content's unique identifier. See the "JSP Tag Library Reference" in the WebLogic Personalization Server *Developer's Guide* for more information.
- The `<cm:printProperty>` tag inlines the value of the specified Content metadata property as a string. See the "JSP Tag Library Reference" in the WebLogic Personalization Server *Developer's Guide* for more information.
- The `<cm:printDoc>` tag inlines the raw bytes of a Document object into the JSP output stream. See the "JSP Tag Library Reference" in the WebLogic Personalization Server *Developer's Guide* for more information.

Configuring the Content Manager

The DocumentSchema EJB and DocumentManager EJB deployment descriptors handle the configuration for the Content Management component. To use the reference implementation document repository, you need to configure the EJB deployment descriptors and also set up two WebLogic Server JDBC connection pools.

Once the deployment descriptor has been written, just build the EJBs as you normally would, then add the resulting JAR file to your `ejb.deploy` entry in the `weblogic.properties` file.

Configuring the DocumentSchema EJB Deployment Descriptor

The logic for loading DocumentSchema EJBs is handled via a `SmartBMP`. The Schema EJB implementation loads the `SmartBMP` object from a class name specified in the EJB environment in the EJB's deployment descriptor. The EJB environment variable is *`SmartBMPClass`*. The value must be the fully qualified class name of the `SmartBMP` to use. This `SmartBMP` must be capable of populating a `SchemaImpl` object with `PropertyMetaData` objects.

To use the reference implementation document management system, set *`SmartBMPClass`* to `com.beasys.commerce.axiom.document.SPISchemaSmartBMP` and specify the following EJB environment variables in the document EJB deployment descriptor:

- *`SmartBMPUpdate`*: Set to `false`.
- *`UseDataSource`*: Controls whether `jdbc/docPool` (`true`) or `DocPoolURL` (`false`) is used to get connections. Defaults to `true`.
- *`DocPoolURL`*: Specifies the JDBC URL to the document JDBC connection to use (if `UseDataSource` is `false`). Should point to a connection pool. For example: `jdbc:weblogic:pool:docPool`.
- *`DocPoolDriver`*: Specifies the JDBC driver class to use to connect to the `DocPoolURL`. This is optional. If not specified, the EJB will try to determine the appropriate JDBC driver class from the DocPool URL.
- *`jdbc/docPool`*: A Data Source reference to the document JDBC connection Pool (see the topic "Setting Up Connection Pools" on page 4-11). This should correspond to the Data Source attached to the WebLogic connection pool that uses the document reference implementation JDBC driver.
- *`jdbc/commercePool`*: A DataSource reference to the `weblogic.jdbc.jts.commercePool`, which should be attached to the WebLogic connection pool `commercePool`.

Other `SmartBMP` classes for other document management systems will possibly require more and/or different EJB environment variables.

Configuring the DocumentManager EJB Deployment Descriptor

The `DocumentManagerSession` EJB simply hides the details of getting to the `Document` and `DocumentSchema` EJBs. It understands the following environment variables in its deployment descriptor:

- *`PropertyCase`*: This sets how the `DocumentImpl` modifies incoming property names. If this is *lower*, all property names are converted to lowercase. If this is *upper*, all property names are converted to uppercase. If this is anything else or not specified, property names are not modified. Use *lower* or *upper* if the `SmartBMP` class expects everything in a certain case (for example, the Documentum `SmartBMP` expects everything in lowercase). For the document reference implementation, do not specify the *`PropertyCase`*.
- *`jdbc/docPool`*: A Data Source reference to the document JDBC connection Pool (see the topic "Setting Up Connection Pools" on page 4-11). This should correspond to the Data Source attached to the WebLogic connection pool that uses the document reference implementation JDBC driver.
- *`ejb/ContentHome`*: EJB reference to the Document Home to which this should delegate for non-readOnly access.

Note: Since the Document EJB is deprecated for read access, this will eventually no longer be required.

- *`ejb/SchemaHome`*: EJB reference to the Schema Home to which this should delegate for Schema information.
- *`UseDataSource`*: Controls whether `jdbc/docPool` (`true`) or `DocPoolURL` (`false`) is used to get connections. Defaults to `true`.
- *`DocPoolURL`*: Specifies the JDBC URL to the document JDBC connection to use (if `UseDataSource` is `false`). Should point to a connection pool. For example: `jdbc:weblogic:pool:docPool`.
- *`DocPoolDriver`*: Specifies the JDBC driver class to use to connect to the `DocPoolURL`. This is optional. If not specified, the EJB will try to determine the appropriate JDBC driver class from the DocPoolURL.

Setting Up Connection Pools

For the document reference implementation, set up a specialized WebLogic connection pool and DataSource which will be used by the DocumentManager via the `jdbc/docPool` reference. (See the topic "Configuring the DocumentManager EJB Deployment Descriptor" on page 4-10.)

For example, if the connection pool name is *`docPool`*:

- The *`URL`* should be `jdbc:beasys:docmgmt:com.beasys.commerce.axiom.document.ref.RefDocumentProvider`.
- The *`driver`* should be `com.beasys.commerce.axiom.document.jdbc.Driver`. It should not be configured to use a test_table, although it can be allowed to shrink. The driver supports the following properties:
  - *`jdbc.url:`* (Required) Specifies the JDBC URL of the database. The connection in this pool opens a connection to this JDBC URL. This property probably should refer to another, non-specialized JDBC connection pool, although it can be any JDBC URL.
  - *`jdbc.driver`*: Specifies a JDBC driver class name to load.
  - *`jdbc.isPooled`*: If `true`, then the system assumes the JDBC URL in `jdbc.url` is a pooling connection URL and connections will open and close as needed. If `false`, then this connection opens one connection via the `jdbc.url` and uses that for its lifetime. If the `jdbc.url` starts with `jdbc:weblogic:pool` or `jdbc:weblogic:jts`, then this property automatically becomes `true`.
  - *`docBase:`* (Required) Specifies the document base of the document files. The IDs in the database use file paths relative to this directory and must exist when the connection is created. To operate in a cluster or a multi-server environment, you must either replicate the files on the machines or the put the `docBase` directory on a shared volume.
  - *`schemaXML`*: Specifies the file or directory where the XML schema (following the doc-schemas.dtd) resides. Either the `schemaXML` property or the `iw.schemaBase` property is required, although the schemas under *`schemaXML`* take precedence if both are specified. The `schemaXML` property has the same constraints as the *`docBase`* property when used in a cluster.

Note: If *schemaXML* is a directory, the connection will recurse under it and load all files ending in `.xml` (`*.xml`).

Note: If *schemaXML* is a file, the connection loads it.

- *iw.schemaBase*: Specifies the directory in which the InterWoven `datacapture.cfg` files reside. The connection recurses through this directory, loading all `datacapture.cfg` files it finds. Either the `iw.schemaBase` or `schemaXML` property is required, although you can specify both. The `iw.schemaBase` property has the same constraints as the `docBase` property when used in a cluster.

- Set up a non-transactional DataSource pointing to the pool. The name of the DataSource should be the same as that configured with the DocumentManager and Schema.

All other properties are passed with `jdbc.url` when the Driver Manager opens a database connection.

Example Connection Pool Entry

The following example shows a sample configuration in the `weblogic.properties` file.

```
weblogic.jdbc.connectionPool.docPool=\
url=jdbc:beasys:docmgmt:com.beasys.commerce.axiom.document.ref.Re
fDocumentProvider,\
driver=com.beasys.commerce.axiom.document.jdbc.Driver,\
loginDelaySecs=1,\
initialCapacity=1,\
maxCapacity=5,\
capacityIncrement=1,\
allowShrinking=true,\
shrinkPeriodMins=15,\
refreshMinutes=10,\
    props=jdbc.url=jdbc:weblogic:pool:commercePool;\
    jdbc.isPooled=true;\
    docBase=C:/WeblogicCommerce/docBase;\
    schemaXML=C:/WeblogicCommerce/docSchemas;\
    iw.schemaBase=C:/iw-home/templatedata
weblogic.allow.reserve.weblogic.jdbc.connectionPool.docPool=every
one
weblogic.jdbc.DataSource.weblogic.jdbc.pool.docPool=docPool
```

Configuring WebLogic Commerce Properties

Use a ContentManager or DocumentManager with `<cm:select>` or `<cm:selectById>` to retrieve Content or Documents. The default DocumentManager is deployed at `com.beasys.commerce.axiom.document.DocumentManager`.

To help with the JNDI names, the ContentHelper class has the following six constants:

`DEF_CONTENT_HOME`
: Specifies the default deployed ContentHome.

`DEF_CONTENT_MANAGER_HOME`
: Specifies the default deployed ContentManagerHome.

`DEF_CONTENT_SCHEMA_HOME`
: Specifies the default deployed SchemaHome for Content.

`DEF_DOCUMENT_HOME`
: Specifies the default deployed DocumentHome.

`DEF_DOCUMENT_MANAGER_HOME`
: Specifies the default deployed DocumentManagerHome.

`DEF_DOCUMENT_SCHEMA_HOME`
: Specifies the default deployed SchemaHome for Document.

The values of those constants are read from the `weblogiccommerce.properties` file from the values for the following properties:

`DEF_CONTENT_HOME`
: `commerce.home.content.ContentHome`

`DEF_CONTENT_MANAGER_HOME`
: `commerce.home.content.ContentManagerHome`

`DEF_CONTENT_SCHEMA_HOME`
: `commerce.home.content.ContentSchemaHome`

`DEF_DOCUMENT_HOME`
: `commerce.home.document.DocumentHome`

`DEF_DOCUMENT_MANAGER_HOME`
: `commerce.home.document.DocumentManagerHome`

`DEF_DOCUMENT_SCHEMA_HOME`
: `commerce.home.document.DocumentSchemaHome`

Therefore, in any <cm:select>, <cm:selectById>, <pz:contentQuery> or <pz:contentSelector> tags, define the contentHome (or contenthome) parameter to use a ContentManagerHome or DocumentManagerHome.

Example:

The News Index and News Viewer portlets use the default deployed DocumentManager and can be used as a reference. The JSPs are located in the server/public_html/portals/repository/portlets directory in the news_index.jsp, news_viewer.jsp and content_titlebar.jsp files.

Using the Show Document Servlet

To operate the Show Document servlet, it should be registered with WebLogic Server. The class name of the servlet is com.beasys.commerce.content.ShowDocServlet. To register it with WebLogic, add a line similar to the following to your weblogic.properties files:

```
weblogic.httpd.register.showDocServlet=\
    com.beasys.commerce.content.ShowDocServlet
```

Reference the class in the URL as /showDocServlet. To change the URL reference, change /showDocServlet. For example, to specify the URL as /myapp/doc-shower, enter the following in the weblogic.properties file:

```
weblogic.httpd.register.myapp/doc-shower=\
    com.beasys.commerce.content.ShowDocServlet
```

Querying Document Content

There are several way to query the document management system. To query the system, you construct a query expression, then pass the expression to any one of these:

- JSP tags (see “Using Content Management JSP Tags” on page 4-25.)
- ContentHelper (see the Javadoc API documentation)
- ContentManager (see the Javadoc API documentation)
- ContentHome (see the Javadoc API documentation)

Structuring a Query

WebLogic Personalization Server queries use a syntax similar to the SQL string syntax that supports basic Boolean-type comparison expressions, including nested parenthetical queries. In general, the template for use includes a metadata property name, a comparison operator, and a literal value. The basic query uses the following template:

```
attribute_name comparison_operator literal_value
```

Note: Consult the Javadoc API documentation on `com.beasys.commerce.util.ExpressionHelper` for more information about the query syntax.

Several constraints apply to queries constructed using this syntax:

- String literals must be enclosed in single quotes.
  - `'WebLogic Server'`
  - `'football'`
- Date literals can be created via a simplistic `toDate` method that takes one or two `String` arguments (enclosed in single quotes). The first, if two arguments are supplied, is the `SimpleDateFormat` format string; the second argument is the date string. If only one argument is supplied, it should include the date string in 'MM/dd/yyyy HH:mm:ss z' format.
  - `toDate('EE dd MMM yyyy HH:mm:ss z', 'Thr 06 Apr 2000 16:56:00 MDT')`
  - `toDate('02/23/2000 13:57:43 MST')`
- Use the `toProperty` method to compare properties whose names include spaces or other special characters. In general, use `toProperty` when the property name does not comply with the Java variable-naming convention that uses alphanumeric characters.
  - `toProperty ('My Property') = 'Content'`
- To include a scope into the property name, use either `scope.propertyName` or the `toProperty` method with two arguments.
  - `toProperty ('myScope', 'myProperty')`

  Note: The reference document management system ignores property scopes.

- Use \ along with the appropriate character(s) to create an escape sequence that includes special characters in string literals.
  - `toProperty ('My Property\'s Contents') = 'Content'`
- Additionally, use Java-style Unicode escape sequences to embed non-ASCII characters in string literals.
  - Description like `'*\u65e5\u672c\u8a9e*'`

  Note: The query syntax can only contain ASCII and extended ASCII characters (0-255).

  Note: Use `ExpressionHelper.toStringLiteral` to convert an arbitrary string to a fully quoted and escaped string literal which can be put in a query.
- The *now* keyword—only used on the literal value side of the expression—refers to the current date and time.
- Boolean literals are either `true` or `false`.
- Numeric literals consist of the numbers themselves without any text decoration (like quotation marks). The system supports scientific notation in the forms (for example, *1.24e4* and *1.24E-4*).
- An exclamation mark (!) can be placed at an opening parenthesis to negate an expression.
  - `!(keywords contains 'football') || (size >= 256)`
- The Boolean `and` operator is represented by the literal `&&`.
  - `author == 'james' && age < 55`
- The Boolean `or` operator is represented by the literal `||`.
  - `creationDate > now || expireDate < now`

The following examples illustrate full expressions:

Example 1:

```
((color='red' && size <=1024) || (keywords contains 'red' &&
creationDate < now))
```

Example 2:

```
creationDate > toDate ('MM/dd/yyyy HH:mm:ss', '2/22/2000 14:51:00')
&& expireDate <= now && mimetype like 'text/*'
```

Using Comparison Operators to Construct Queries

To support advanced searching, the system allows construction of nested Boolean queries incorporating comparison operators. The following table summarizes the comparison operators available for each metadata type. (For more information about the native types supported in WebLogic Personalization Server, see Support for Native Types in the *Developer's Guide* chapter Overview of Personalization Development.)

Operator Type	Characteristics
Boolean (==, !=)	Boolean attributes support an equality check against Boolean.TRUE or Boolean.FALSE.
Numeric (==, !=, >, <, >=, <=)	Numeric attributes support the standard equality, greater than, and less than checks against a `java.lang.Number`.
Text (==, !=, >, <, >=, <=, like)	Text strings support standard equality checking (case sensitive), plus lexicographical comparison (less than or greater than). In addition, strings can be compared using wildcard pattern matching (that is, the *`like`* operator), similar to the SQL LIKE operator or DOS prompt file matching. In this situation, the wildcards will be * (asterisk) to match any string of characters and ? (question mark) to match any single character. Interval matching (for example, using [ ]) is not supported. To match * or ? exactly, the quote character will be \ (backslash).
Datetime (==, !=, >, <, >=, <=)	Date/time attributes support standard equality, greater than, and less than checks against a `java.sql.Timestamp`.
Multi-valued Comparison Operators (contains, containsall)	Multi-valued attributes support a *`contains`* operator that takes an object of the attribute's subtype and checks that the attribute's value contains it. Additionally, multi-valued attributes support a *`containsall`* operator, which takes another collection of objects of the attribute's subtype and checks that the attribute's value contains all of them. Single-valued operators applied to a multi-valued attribute should cause the operator to be applied over the attribute's collection of values. Any value that matches the operator and operand should return `true`. For example, if the multi-valued text attribute *`keywords`* has the values *BEA*, *Computer*, and *WebLogic* and the operand is *BEA*, then the < operator returns `true` (*BEA* is less than *Computer*), the > operator returns `false` (*BEA* is not greater than any of the values), and the == operator returns `true` (*BEA* is equal to *BEA*).

Operator Type	Characteristics
User Defined Comparison Operators	Currently, no operators can be applied to a user-defined attribute.

Note: The search parameters and expression objects support negation of expressions via a bit flag (!).

Note: The reference document management system has only single-value Text and Number properties. All implicit properties are single-value Text.

Using the BulkLoader to Load File-based Content

WebLogic Personalization Server provides no run-time tools to load metadata information from a content database. However, the server provides a command-line utility, the BulkLoader, that descends a directory hierarchy, parses the HTML-style `<meta>` tags, reverses the metadata content contained within the `<meta>` tags into schema information, and loads the resulting documents into the reference implementation database.

The BulkLoader is a command-line application that is capable of loading document metadata into the reference implementation database from a directory and file structure. The BulkLoader parses the document base and `weblogic.properties` and loads all the document metadata so that the Content Management component can search for documents. The BulkLoader supports all document types, not just HTML documents.

Command Line Usage

The BulkLoader class allows a number of command-line switches:

```
java com.beasys.commerce.axiom.document.loader.BulkLoader
  [-/+verbose] [-/+recurse] [-/+delete] [-/+metaparse] [-/+cleanup]
  [-/+hidden] [-/+inheritProps] [-schemaName <name>] [-encoding <encoding>]
  [-properties <name>] -conPool <name> [-schema <name>] [+schema]
  [-match <pattern>] [-ignore <pattern>] [-htmlPat <pattern>]
  [-d <dir>] [-mdext <ext>] [--]
  [files... directories...] [-filter <filter class>] [+filters]
```

`-verbose`

Emits verbose messages.

`+verbose`
: Runs quietly [default].

`-recurse`
: Recurses into directories [default].

`+recurse`
: Does not recurse into directories.

`-delete`
: Removes document from database.

`+delete`
: Inserts documents into database [default].

`-metaparse`
: Parses HTML files for `<meta>` tags [default].

`+metaparse`
: Does not parse HTML files for `<meta>` tags.

`-cleanup`
: If specified, this only performs a table cleanup using the `-d` argument as the document base. (All files will need to be under that directory.)

`+cleanup`
: Turns off table cleanup (do a document load) [default].

`-hidden`
: Specifies to ignore hidden files and directories [default].

`+hidden`
: Specifies to include hidden files and directories.

`-inheritProps`
: Specifies to have metadata properties be inherited when recursing [default].

`+inheritProps`
: Specifies to have metadata properties not be inherited when recursing.

`-htmlPat <pattern>`
: Specifies a pattern for determining which files are HTML files when determining whether to do the `<meta>` tag parse. This can be specified multiple times. If none are specified, `*.htm` and `*.html` are used.

`-properties <name>`
: Specifies the location of the `weblogic.properties` file which should contain the `connectionPool` definition. Defaults to `weblogic.properites` in the current directory.

`-conPool <name>`
: Specifies the `connectionPool` name from the properties file from which the BulkLoader should get the connection information.

`-schema <name>`
: Specifies the path to the schema file the BulkLoader will generate (defaults to `document-schema.xml`).

`+schema`
: If specified, then no schema file will be created.

`-schemaName <name>`
: Specifies the name of the schema generated by the BulkLoader. Defaults to "LoadedData".

`-encoding <name>`
: Specifies the file encoding to use. Defaults to your system's default encoding. (See your JDK documentation for the valid encoding names.)

`-match <pattern>`
: Specifies a file pattern the BulkLoader should include. This can be specified multiple times. If none are specified, all files and directories are included.

`-ignore <pattern>`
: Specifies a file pattern the BulkLoader should not include. This can be specified multiple times.

`-d <dir>`
: Specifies the docBase that non-absolute paths will be relative to. If not specified, `"."` (current directory) is used.

`-mdext <ext>`
: Specifies the file name extension for metadata property files. The value should starts with a `"."` (defaults to `.md.properties`).

`-filter <filter class>`
: Specifies the class name of a `LoaderFilter` to run files through. This can be specified multiple times to add to the list of Loader Filters.

`+filters`
: Clears the current list of Loader Filters. (This will clear the default filters as well.)

`--`
: Everything after this is considered a file or directory.

How the BulkLoader Finds Files

The following sequence describes how the BulkLoader locates files:

1. The BulkLoader starts by looking at the list of files and directories specified from the command line.
   - If no files or directory are specified, it uses only the `docBase` specified by the `-d` option. It then loops over the list of files and directories.
   - If it finds a directory and `+recurse` is specified, then it stops.
   - If it finds a directory and recursion is turned on (the default or with `-recurse`), then the BulkLoader loops over the files and directories contained within that directory.

   Note: If the file or directory is not an absolute path, then it is assumed to be relative to the `docBase` specified by the `-d` option.

2. To determine if the BulkLoader should process a file or directory, it checks to see if the file is marked as a hidden file.

   Note: If it is a hidden file (or directory) and the `+hidden` option was not specified, then the file or directory is ignored.

3. If the file or directory does not exist or is not readable by the user executing the BulkLoader, a warning is displayed and the file or directory is ignored.

4. If the file or directory is a file, then it is loaded.

5. If the loaded object is a directory and recursion is enabled, then the files and directories under the directory are retrieved by filtering against the `-match` and `-ignore` options.

   Note: The `-match` and `-ignore` options only apply to files and directories not listed on the command line; in other words, they apply only to those found by recursing into a directory. The patterns specified with the `-match` and `-ignore` options (and the `-htmlPat` options, for that matter) should be DOS-style patterns: '*' matches any set of characters, '?' matches any one character. Sets of characters (for example, *[aceg]*) are not supported.

6. If the subfile or directory name matches any of the patterns specified by a `-ignore` option, the subfile or directory is ignored.

7. If the subfile or directory is a directory, then it is included.

8. If the subfile or directory is a file and no -match options were specified, then it will be included; if at least one -match option is supplied, then the filename must match at least one of -match patterns.

   Note: Files with an extension matching the extension specified by -mdext (*.md.properties* by default) are always ignored.

How the BulkLoader Finds Metadata Properties

As the BulkLoader is finding files and directories, it will also attempt to load metadata property files. Whenever the BulkLoader encounters a directory that it will process, it looks for a file called dir.<*mdext*> where <*mdext*> is the extension specified by the -mdext option. Therefore, the default filename it looks for is dir.md.properties. If this file exists and is readable by the user, the BulkLoader loads it as a Java-style properties file of name=value properties. If the directory is actually a subdirectory entered because +recurse was not specified and the +inheritProps option is not specified, then the properties from dir.md.properties will be added to the properties from the parent directories. All files in the directory gain these metadata properties.

When the BulkLoader finds a file which is to be included and loaded, it looks for a file whose name is the original filename appended with the -mdext extension. So, by default, if the file is called image.gif, the BulkLoader looks for a file called image.gif.md.properties. If that file exists and is readable, the BulkLoader loads those properties into the directory's properties (and possibly the parent directories' as well).

Next, if the file is an HTML file and the +metaparse option was not specified, then the BulkLoader will parse the HTML, looking for <meta> tags and <title> tags. The BulkLoader determines if a file is an HTML file by using the filename patterns specified by the -htmlPat options. If no -htmlPat patterns are specified, then *.htm and *.html are used. The BulkLoader will load into the file's properties any <meta> tags that contain name and content values found anywhere in the file (not just in the HTML head section). Additionally, it will pull the title from the <title></title> and set it as "title".

Finally, the BulkLoader will pass the file to the loadProperties method of each registered LoaderFilter (the -filter option). The LoaderFilter may assign additional metadata to the file. When the BulkLoader starts up, it looks for a com/beasys/commerce/axiom/document/loader/loader.properties file in the classpath. From that, it looks for a loader.defFilters property. This is the colon-separated list of LoaderFilter class names the BulkLoader should always

load. Unless that file is modified, the BulkLoader will load an `ImageLoaderFilter`, which will pull the width and height from `*.gif`, `*.jpg`, `*.png`, and `*.xbm` image files.

In summary, the BulkLoader gathers metadata for a document from the following sources (in this order):

1. The parent directories `dir.md.properties` file.
2. The file's directory's `dir.md.properties` file.
3. The file's`.md.properties` file.
4. If the file is an HTML file, then it uses `<meta>` tags.
5. The list of LoaderFilters.

From there, the ID of the document in the database will be the file path, relative to the `docBase` specified by the `-d` option. If the file path is not relative to the `docBase`, then it will be relative to the path from the command line. The file size will be retrieved from the file. The *`mimeType`* will be determined by the file's extension. The *`modifiedDate`* in the database will become the current time (since that is when the document is being modified in the database).

Cleaning Up the Database

If the `-cleanup` option is specified, the BulkLoader will not actually load any documents. Instead, it will attempt to clean up and update the database tables. It will first query the database, looking for any metadata entries that do not have corresponding document entries. For each of those, it will create a document entry. It will then go over each document entry and update the size, modified date, and possibly the MIME type (if the MIME type is not in the database) based upon the files in the `docBase` specified with the `-d` option.

Loading Internationalized Documents

The BulkLoader accepts a `-encoding <enc>` option. When this is specified, the BulkLoader will use that encoding to open all HTML files to find `<meta>` tags.

For example, if the files under the Unicode files directory were saved in the Unicode encoding, you could do:

```
java com.beasys.commerce.axiom.document.loader.BulkLoader -verbose
-properties weblogic.properties -conPool commercePool -schema
```

`dmsBase\schemas\unicode-files.xml -d dmsBase unicode-files -encoding Unicode`. When -encoding is specified, the generated schema XML file will be in the UTF-8 encoding (since some metadata property names might not be ASCII), which the run-time engine can read in. (Note: UTF-8 is a superset of ASCII and can be mostly read by common text editors.)

When `-encoding` is specified, all HTML files the BulkLoader encounters will be opened with the specified encoding. Therefore, either the encoding must be a superset of all the files' encodings (for example, ISO8859_1 is a superset of ASCII, where as Unicode is not) or the BulkLoader might not be able to correctly pull out the `<meta>` tag information. It is recommended to either save all documents in a single encoding or to run the BulkLoader against only certain directories at a time (for example, put all the Big5 files in one directory).

The list of available encoding names is contained in the documentation for your JDK, or the documentation for the tool which created the file. If you are not creating files containing non-ASCII characters, this should not affect you.If you want to check if the BulkLoader is correctly parsing your HTML file, you can use the `com.beasys.commerce.axiom.document.loader.MetaParser` class. For example:
`java com.beasys.commerce.axiom.document.loader.MetaParser unicode.htm unicode` would print out the `<meta>` tags found in the `unicode.htm` file, assumed to be Unicode encoded. Of course, any non-ASCII character probably will not print correctly to your console window, but you can tell what it thinks it found.

Generating Schema Files

Additionally, the BulkLoader supports a `-schemaName <name>` argument which controls the name of the schema in the generated XML file; this in turn affects the name of the Content Property Sets which appear in the rules editor. If not specified, it defaults to "LoadedData."

After loading all the documents on the list, if the `+schema` option is not specified, the BulkLoader will output a XML file containing the schema information and following the doc-schemas DTD. The BulkLoader will output a single schema which contains entries for all the metadata attributes it finds over the entire load.

If `+schema is` specified, then no schema file will be created.

Using Content Management JSP Tags

To use the Content Management JSP tags, ensure that the `cm.tld` file resides in the `WEB-INF` directory of your `WAR` files or in your document root.

Content Cache

The `<cm:select>` and `<cm:selectById>` tags support a session-based, per-user Content cache for content searches. To enable this, both tags support the following parameters. All three tag parameters can be JSP run-time expressions.

`useCache`
: Set to `true` or `false`. The default is `false`. If `true`, then the ContentCache will be used (it will be stored in the user's HttpSession).

`cacheId`
: The ID name to use to cache the Content under. Internally, the cache is implemented as a Map; this will become the key. If this is not specified, than the `id` parameter of the tag will be used.

`cacheTimeout`
: The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back in the cache. Use -1 for no timeout (always use the cached Content); 0 to immediately timeout the cached Content.

Additionally, the `commerce.content.cache.useSoftHashMap` property in the `weblogiccommerce.properties` controls whether the ContentCaches internally use SoftReferences to cache the Content. SoftReferences should allow the garbage collector to clear portions of the caches as memory is needed. This defaults to `false`.

Note: The Windows NT 1.2.2 JVM supports SoftReference quite well; however, the Solaris Production 1.2.1_04 JVM always immediately clears SoftReferences, thereby eliminating their usefulness as a caching mechanism. It is recommended that `commerce.content.cache.useSoftHashMap` be set to `true` under Windows NT, but set to `false` under Solaris.

Example:

The News Index and News Viewer portlets use content caching and can be used as a reference. The JSPs are located in the `server/public_html/portals/repository/portlets` directory in the `news_index.jsp, news_viewer.jsp` and `content_titlebar.jsp` files.

readOnly Content Tag

The `<cm:select>` and `<cm:selectById>` tags support an optional `readOnly` parameter.

If not specified, the default value (from `ContentHelper.DEF_CONTENT_READONLY`, which is loaded from the `commerce.content.defaultReadOnly` property in the `weblogiccommerce.properties` file) is used. Additionally, a `ContentHelper.getContent()` method and a `ContentManager.getContent()` method take a `readOnly` flag.

In the reference implementation, invoking the `getContent()` method without the `readOnly` parameter invokes the new `getContent()` method, passing in `ContentHelper.DEF_CONTENT_READONLY`. If `readOnly` is `true`, then non-EJB instances of Content and Document objects can be returned; with the reference implementation, non-EJB instances will be returned.

Note: This can help performance and prevent deadlocks. It is highly recommended that `commerce.content.defaultReadOnly` be set to `true`, and that, if `readOnly` is specified, it be specified `true`.

Object Interfaces

The interfaces `ConfigurableEntityRemote`, `ContentRemote`, `DocumentRemote`, `UserRemote` and `GroupRemote` extend both EJBObject and their respective non-EJBObject interfaces. For example:

`ConfigurableEntityRemote` extends `EJBObject` and `ConfigurableEntity`.
`DocumentRemote` extends `ContentRemote`, which extends `ConfigurableEntityRemote`.
`UserRemote` and `GroupRemote` extend `ConfigurableEntityRemote`.

In this fashion, session beans, tags and utility methods can return either lightweight objects or the EJB instances, depending upon usage and parameters.

Note: In general, you should only typecast to the non-EJB interfaces (that is, `ConfigurableEntity`, *not* `ConfigurableEntityRemote`).

5 Developing Portlets

An integral part of any portal solution is the portlet application. This chapter explains what you need to know to create a portlet application.

This topic includes the following sections:

- Introduction
- Creating a Portlet Application
  - Defining the Portlet JSP
- Working Within the Portal Framework
  - Extending the PortalJspBase Class
  - Accessing Portal Session Information
  - Sending Requests Through the Flow Manager
  - Using URL Links in Your Portlet
  - HTML Form Processing
  - Retrieving the Home Page
  - Retrieving the Current Page
  - Setting the Request Destination
  - Tracking User Login Status
  - Loading Content from an External URL
  - Using Example Portlets
  - HTML Tables versus HTML Frames

Introduction

Generally, a main portal page is organized into smaller display areas. Using the WebLogic Personalization Server, the portal developer can create a main page layout, with flexible methods for determining custom headers, footers, look and feel elements, and the primary content areas.

The most information-rich part of the main page consists of a set of portlets, laid out in columns. Each portlet is a small content area, provided to display a particular type of information. These portlets are developed especially for each portal and are written in JSP, so there is great flexibility in what can be displayed. There is a standard set of development guidelines, coupled with portal services, to ensure portal and portlets are well-behaved.

The primary way dynamic functionality of the personalization components is made available to portlets is via custom JSP tags resident in tag libraries. These tags hide much of the internal run-time complexity of the Personalization Server, presenting a small, well-defined interface to its functions. Portlets may also access certain types of personalization EJBs directly, using embedded Java to access Personalization Server functionality.

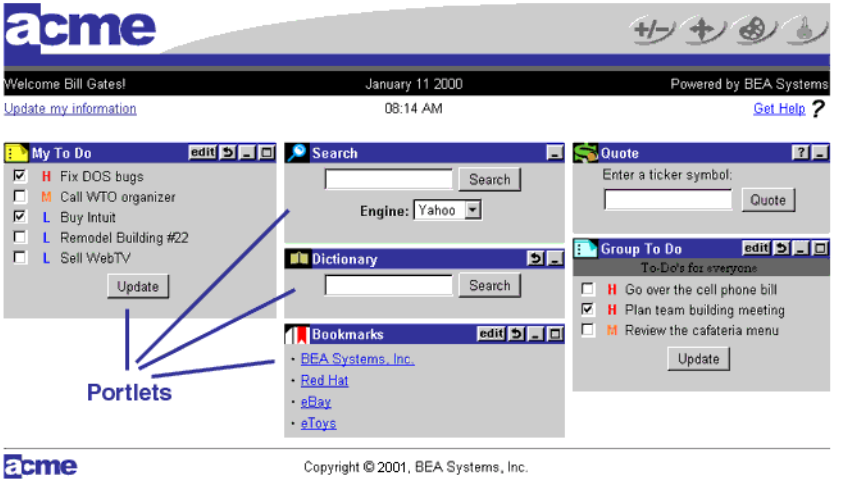
Each portlet may have a series of custom pages with specific functions associated with it, accessed via button clicks on the portlet. An edit page may make available to the user HTML input elements, in which the user can enter data on preferences specific to that portlet. A full page (or pages) version may be brought up to show an arbitrary amount of detail. A help page can be set up. The portlet may also be maximized, minimized, or floated in its own window.

What Is a Portlet?

From the end user point-of-view, a portlet is a specialized content area that occupies a small “window” in the portal page. For example, a portlet can contain travel itineraries, business news, local weather, or sports scores. The user can personalize the content, appearance, and position of the portlet according to the profile preferences set by the administrator and group to which the user belongs. The user can also edit, maximize, minimize, or float the portlet window.

The following figure shows how portlets appear in a portal Home page:

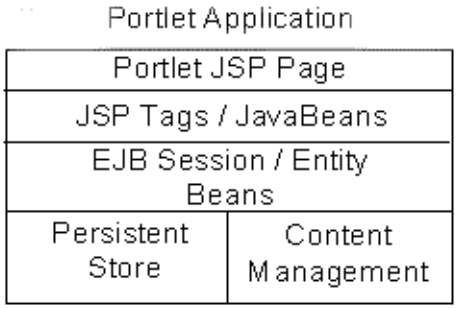
Figure 5-1 Portlet HomePage View



From a server application point-of-view, a portlet is a content component implemented as a JSP that defines the static and dynamic content for a specific content subject (weather, business news, etc.) in the portal page. The portlet JSP generates dynamic HTML content from the server by accessing data entities or content adapters implemented using the J2EE platform. The Portlet JSP then displays the content in the portal.

Note: All of the portlets in a portal are included in a *single* HTML page, through the use of the `<jsp:include>` action.

Figure 5-2 Portal Application Programming Model



The diagram shown above defines the portal application programming model. This programming model includes JSP, JSP tags, JavaBeans, EJBs, data stores, and content management stores. The portlet JSP contains static HTML and JSP code. This JSP code uses application or content specific JSP tags and/or JavaBeans to access dynamic application data through EJBs, content adapters, and legacy system interfaces. Once this data is retrieved, the portlet JSP applies HTML styling to it and the generated HTML is returned in the HTTP request to the client HTTP client.

Creating a Portlet Application

To create a portlet application, you should be a J2EE developer with a background in JavaServer Pages (JSP), JavaScript and HTML, and have a knowledge of Enterprise Java Beans.

The portlet application is a JSP that contains code responsible for retrieving personalized content and rendering it as HTML.

Once you have created your portlets, you can associate them with one or more portals. Therefore, you must create your portlet applications before using the Portal Administration Tool to create and define your portal.

Defining the Portlet JSP

The portal treats portlets as components or HTML fragments, not as entire HTML documents. The portal relies on the portlet application to create an HTML fragment for its portlet content. The portal renders the portlet's content in the portal page according to the personalization rules (the row and column position, colors, etc.) for the portal, group, and user levels.

When creating a portlet application, keep the following items in mind to ensure that your portlets run efficiently:

- Avoid using forms in a portlet that update the data within the portlet. This causes the entire portal to refresh its data which can be very time consuming. For more information on using an HTML form in a portlet, see HTML Form Processing.
- Place items that require heavy processing in an edit page or a maximized URL. Otherwise, the portal must wait for the portlet to process which considerably slows down the painting of the portal.

To define your portlet JSP:

1. Create a JSP for your portlet content.
2. Create JSPs for the portlet banner, header, footer, alternate header, alternate footer, help page, and edit URL as needed.

Note: You do not need to create a JSP for the portlet title bar because it is included in the WebLogic Personalization Server (`public.html/portals/repository/titlebar.jsp`). The portlet title bar displays the appropriate portlet title bar icons and the name of the portlet you defined in the Portal Administration Tool.

Note: Avoid using the following HTML tags in your portlet content page. The HTML generated by the portlet content page is an HTML fragment contained in a larger portal HTML page, not a separate HTML document.

- `<html></html>`
- `<header></header>`
- `<body></body>`
- `<meta></meta>`
- `<title></title>`

3. Use the following portlet layout guidelines.

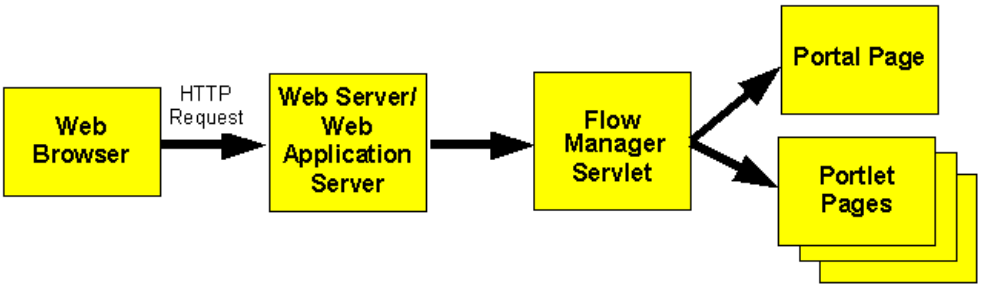
Table 5-1 Portlet Layout Guidelines

Layout Attribute	Recommendation
Content Height	There are no restrictions on height as long as the content fits in your portal page.
Column Width	Take into account that the width of your portlet is controlled by the portal(s) it is associated with. A portal lays out your portlet content in a column based on portal, group, and user personalization rules. As a result, the width of your portlet should be well behaved.
Content Wrapping	Allow wrapping for all portlet content. Do **not** use the **NOWRAP** attribute in table cells.
Titlebar Icon Height	The image height attribute in titlebar.jsp is set to 20.
Titlebar Icon Width	The image width in titlebar.jsp is set to 27.

Working Within the Portal Framework

The portal framework consists of JavaServer Pages, JSP tag libraries, EJBs, Java servlets, and other supporting Java objects. The main Java servlet is the Flow Manager. The Flow Manager receives all incoming HTTP requests and dispatches each request to the appropriate destination URL. As a result, all access to your portal pages is controlled by the Flow Manager. The following diagram shows where the Flow Manager fits in the portal framework.

Figure 5-3 Portal Framework



Extending the PortalJspBase Class

It is recommended that your portlet JSP extend the framework's `PortalJspBase` Java class. This class contains many convenience methods which perform general tasks for your portlet JSP page, such as accessing session information, the *traffic uri*, and user login information.

To extend the `PortalJspBase` class, include the following code at the top of your portlet JSP:

```
<%@ page
extends="com.beasys.commerce.portal.admin.PortalJspBase"%>
```

Accessing Portal Session Information

The portal session information you can access from the `PortalJspBase` class are listed in the following table which lists the name, type, and description for each session value. For more information, see the Portal API Documentation.

Table 5-2 PortalJspBase Class Session Values

Session Value Name	Type	Description
`PortalAdminConstants.PORTAL_NAME`	`String`	The name of the portal associated with the current request.

Table 5-2 PortalJspBase Class Session Values (Continued)

Session Value Name	Type	Description
`JspConstants.SERVICEMANAGER_SUCCESSOR`	`String`	The name of the successor associated with the current session. The successor profile properties are used for those properties not specified by the user.
`JspConstants.SERVICEMANAGER_USER`	`String`	The name of the user associated with the current session.
`UserManagmentConstants.PROFILE_USER`	`Config-urable Entity`	The user profile associated with the current request or the session.
`UserManagmentConstants.PROFILE_SUCCESSOR`	`Config-urable Entity`	The group profile associated with the current request or the session.
`UserManagmentConstants.PROFILE_SUCCESSOR_UID`	`Long`	Unique IDs for the configurable entities.
`UserManagmentConstants.PROFILE_USER_UID`	`Long`	Unique IDs for the configurable entities.

You can retrieve the portal session information described above through the following `PortalJspBase` methods:

- `public Object getSessionAttribute(String` *`aName`*`, HttpServletRequest` *`aRequest`*`)`
- `public void setSessionAttribute(String` *`aName`*`, Object` *`aValue`*`, HttpServletRequest` *`aRequest`*`)`
- `public void removeSession(String` *`aName`*`, HttpServletRequest` *`aRequest`*`)`

You can set the portal session's `SERVICEMANAGER_USER` and `SERVICEMANAGER_SUCCESSOR` through the following `JspBase` methods:

- `public static void setUser(String` *`aUser`*`, HttpServletRequest aRequest)`
- `public static void setSuccessor(String` *`aSuccessor`*`, HttpServletRequest` *`aRequest`*`)`
- `public static void setUserAndSuccessor(String aUser, String` *`aSuccessor`*`, HttpServletRequest` *`aRequest`*`)`

Sending Requests Through the Flow Manager

Remember that all HTTP requests and responses are sent to the Flow Manager servlet. Therefore, your portlet HTML must refer to the Flow Manager's URL for URL links and HTML form processing.

Using URL Links in Your Portlet

If your portlet contains links to a JSP page that is not a portlet, use the following `PortalJspBase` method to create your URL and to guarantee that the HTTP request is sent to the service manager URL:

```
public String createURL(HttpServletRequest aRequest, String
destination, String parameters)
```

The destination should be a relative or qualified file location in the form such as `example/mytodo.jsp`, or `/yourportal/example/mytodo.jsp`. The path is relative to the `documentRoot`, as specified in `weblogic.properties`. Parameters should be a string such as `column=4&row=5`.

Note: Parameter values should already be encoded as you would for any HTTP request. Example: `String parms = "column=" + java.net.URLEncoder.encode("4");`

Because of the way the JSP engine handles `jsp:forward` and `jsp:include`, you must *fixup* the relative URLs in your portlet, especially relative links to images. The Web browser thinks the root for relative links is the directory in which the Flow Manager resides and not your portlet's directory.

To fix up relative URLs, use the following `ToolsJspBase` method:

```
public static String ToolsJspBase fixupRelativeURL(String aURL,
HttpServletRequest aRequest)
```

where *aURL* is the destination URL to fix up and *aRequest* is the current HTTP request. In your JSP page, use the following method to code a `fixup`:

```
<img src="<%=fixupRelativeURL("images/quote.gif",
request)%>"width="50" height="35" border="0">
```

Note: For the repository feature to work with `jsp:include` and `jsp:forward`, use *reconcile file* to determine the correct location of the file that is included or forwarded. For example:

```
<jsp:forward page="<%=reconcileFile(request, "login.jsp")%>" />
```

HTML Form Processing

If your portlet contains an HTML form, send all requests to the Flow Manager and set the destination request parameter.

To process HTML forms:

1. Set the form action to `action=getTrafficURI(request)`. This sends the form action request to the Flow Manager. This calls the `PortalJspBase` method:

   `public String getTrafficURI(HttpServletRequest aRequest)`

   The following example shows the use of the HTML form action to send a form request to the Flow Manager:

   ```
   <form method="post" action="<%=getTrafficURI(request)%>">
   ```

2. Set the destination request parameter in the HTTP post request. This tells the Flow Manager where to dispatch the request.

   To set the request destination for HTML forms, enter the following code within your form in your JSP page:

   ```
   <input type="hidden" name="<%=DESTINATION_TAG%>"
   value="example/mytodo.jsp">
   ```

   Note: Do not go through the Flow Manager for HTTP requests to other servers.

Retrieving the Home Page

The Flow Manager sets the Home page for each portal in the Portal Framework session information. The Home page is registered as an initial argument for Flow Manager servlet in weblogic.properties. Use the following `PortalJspBase` method call to retrieve the Home page:

```
public String getHomePage(HttpServletRequest aRequest)
```

Retrieving the Current Page

You can also retrieve the current page from the Portal Framework session information by using the following `PortalJspBase` method:

```
public String getCurrentPage(HttpServletRequest aRequest)
```

Note: When you maximize a portlet, the current page changes to `fullscreenportlet.jsp`.

Setting the Request Destination

When routing a request through the Flow Manager, you must specify the destination that should receive the request. The destination can be relative to the current page (portal.jsp, full-screen portlet.jsp, etc.) or a fully qualified path from the document root.

Note: The `DESTINATION_TAG` constant is available in `PortalJspBase`.

If your portlet contains links to other portal pages, use the following `PortalJspBase` method to create your URL and to guarantee that the HTTP request is sent to the service manager URL:

```
public String createURL(HttpServletRequest aRequest, String
destination, String parameters)
```

The destination should be a relative or qualified file location in the form such as `example/mytodo.jsp`, or `/yourportal/example/mytodo.jsp`.

In some cases, you may need to override the request parameter used by the Flow Manager. For example, use an override destination if your page contains a form that needs to be validated and forwarded elsewhere after validation. Use the following `PortalJspBase` method in your JSP page:

```
public void setOverrideDestination(HttpServletRequest req, String
dest)
```

To set the request destination for HTML forms, enter the following code within your form in your JSP page:

```
<input type="hidden" name="<%=DESTINATION_TAG%>"
value="example/mytodo.jsp">
```

Tracking User Login Status

You can log the user in or out and track whether a user is currently logged in.

Use the following `PortalJspBase` method to track the user login status of a portal session:

```
public void setLoggedIn(HttpServletRequest aRequest,
HttpServletResponse aResponse, boolean aBool)

public Boolean getLoggedIn(HttpServletRequest aRequest)
```

Loading Content from an External URL

According to the JSP specification, a JSP processed by a JSP engine must be relative to the server in which the JSP engine is running, requiring that all of your portlets reside in your portal server and not on an external Web site. However, you can use the `uricontent` tag to download the contents of an external URL into your portlet. If you download the contents of a URL into your portlet, you need to fully qualify the images located on the remote server because the relative links contained within the remote URL will not be found unless fully qualified.

Use the following method to load content from an external URL:

```
<es:uricontent id="uriContent"

               uri="http://www.beasys.com/index.html">
<%
out.print(uriContent);
%>

</es:uricontent>

The sample <es:uricontent> tag is available in
public_html/portals/repository/portlets/_uri_example.jsp
```

Using Example Portlets

The /server/public_html/portals/repository/portlets directory of the WebLogic Personalization Server contains example portlets. The following table lists the name of each example portlet, its description, and its associated files.

Caution: The example portlets are intended for illustration purposes only and should not be used for production code.

Table 5-3 Example Portlet Descriptions and Associated Files

Example Portlet	Description
_uri_example.jsp	Demonstrates how to implement the uricontent tag to import contents from another URL on the Internet.
bookmarks.jsp	Displays the bookmarks associated to the current user. ■ bookmarks_edit.jsp–edit screen for the bookmarks. ■ images/pt_bookmark.gif–bookmark icon for the portlet titlebar.
definedportals.jsp	Displays the portals defined in the system. Uses the <es:foreachinarray>, <es:simplereport>, and <wl:sqlquery tags>.
definedportlets.jsp	Displays the portlets defined in the system. Uses the <es:foreachinarray>, <es:simplereport>, and <wl:sqlquery tags>.
dictionary.jsp	Demonstrates how to redirect a portlet to an external site. ■ images/pt_dictionary.gi –dictionary icon for the portlet titlebar
generic_todo.jsp	For a complete generic_todo.jsp example, see Using the Default Implementation.

Table 5-3 Example Portlet Descriptions and Associated Files (Continued)

Example Portlet	Description
news_index.jsp	Demonstrate use of `<cm:>` tags.
`news_viewer.jsp`	Display content driven from `content_index.jsp`. (Use in conjunction with `content_index.jsp`.)
`grouptodo.jsp`	Displays a `Group To Do List`. ■ `todo.js` –statically included file that does not run by itself. It requires user information from `grouptodo.jsp`. ■ `grouptodo_edit.jsp`–edit URL for `grouptodo.jsp`. ● `todo_edit.jsp`–statically included file that does not run by itself. It requires user information from `grouptodo_edit.jsp`. ■ `grouptodobanner.jsp`–banner for the `grouptodo.jsp`. ■ `images/pt_group_list.gif`–Group To Do List icon for the portlet title bar.
`mytodo.jsp`	Displays a `My To Do List`. ■ `todo.jsp`–statically included file that does not run by itself. It requires user information from `mytodo.jsp`. ■ `mytodo_edit.jsp`–edit URL for `mytodo.jsp`. ● `todo_edit.jsp`–statically included file that does not run by itself. It requires user information from `mytodo_edit.jsp`. ■ `images/pt_my_list.gif`–My To Do List icon for the portlet title bar.
`quote.jsp`	Demonstrates how to redirect a portlet to an external site. ■ `images/pt_quote.gif`–Quote icon for the portlet title bar.

Table 5-3 Example Portlet Descriptions and Associated Files (Continued)

Example Portlet	Description
`search.jsp`	Demonstrates how to redirect a portlet to an external site. ■ `images/pt_search.gif`–Search icon for the portlet title bar.

HTML Tables versus HTML Frames

BEA WebLogic Commerce Server does not prevent the use of any kind of HTML, including HTML frames. You will see that the demos and examples that ship with the product are all reference implementations which use HTML tables.This tabular style is not a requirement of the product—you can write HTML however you like. If you choose to use HTML frames, keep the following considerations in mind:

- When using frames, performance may be an issue, as each HTML frame is a separate page request.
- Since HTTP is stateless, managing state between HTML frames is difficult. In a single application, using a table allows a single HTTP request to be made with all the portlets gathered in one request.

If you choose to use frames, you will need to write HTML code to layout the portlets.

6 Building a Custom Portal Step-by-Step

This chapter is a tutorial for building your own custom e-commerce portal. It assumes minimal knowledge of BEA products, and some knowledge of HTML and JSP. If you are new to WebLogic Server 5.1 (WLS) and WebLogic Commerce Server 3.2 (WLCS), and want to get up to speed quickly, this chapter is for you.

It is recommended, but not required, that you review the Personalization Tour before proceeding with this chapter.

This topic has the following sections:

- Introduction
- Creating the Framework for Your Custom Portal
- Simple Customizations
- Writing Your Own Portlets
- Advanced Portlet Functionality
- Other Customization Techniques
- Framework Files

Note: Throughout this chapter, the environment variable WL_COMMERCE_HOME is used to indicate the directory in which you installed the WebLogic Commerce Server 3.2 and WebLogic Personalization Server 3.2 software.

Introduction

Internet portals are a key part of many e-commerce applications. Portals provide an entry point to the Internet as well as value-added services such as searching and application integration. The WebLogic Personalization Server allows you to quickly assemble both Business-to-Consumer and Business-to-Business portals that require personalized application content on the Internet.

The WebLogic Personalization Server enables Web developers to create portal Web pages and personalized application content for each portal user. The WebLogic Personalization Server uses JSPs, a part of the J2EE specification, in conjunction with a special library of JSP tags, standard HTML, Enterprise Java Beans (EJB), portal end user and the Portal Administration Tools, and a preconfigured database to store portal component entities.

Terminology

Before you can begin building your portal, familiarize yourself with the following terminology.

%WL_COMMERCE_HOME%
The folder in which you installed WebLogic Personalization Server 3.2.

portal
This word has a specific meaning when working with the WebLogic Commerce Server product. A portal is a page that is intended to be the starting point for a user on a site. Furthermore, this chapter assumes that you will be using the "Portal Framework" included with WebLogic Personalization Server to build your portal.

Portal Framework
A collection of prebuilt JSP pages included with the WebLogic Personalization Server distribution that provide the core functionality for portals. They are located in `%WL_COMMERCE_HOME%/server/public_html/portals/repository`.

When using the framework, your pages will have a common layout. In this layout, a page's real estate will be divided into three main areas: a header, a

content area, and a footer. The header resides at the top of the page and typically contains a full-sized logo for the site plus some navigation features. The footer resides at the bottom of the page and typically contains legal notices, copyright information, and a small logo. The middle section, the content area, contains any number of small independent components called portlets. The JSPs included in the Portal Framework manage the layout of these portlets on the page.

Note: You are not required to use the Portal Framework. You may build your site from scratch, although, it is not recommended for new users of the system.

portlet

A JSP page that is displayed within a portal page. There is a one-to-many relationship between a portal and its portlets. Each portlet should provide a limited piece of functionality. For example, imagine an information portal where one portlet gives the weather report, another provides a stock ticker, another the top news stories, and another that shows yesterday's sports scores.

administration tool

WebLogic Personalization Server ships with its own Administration Tool. The focus of this chapter is on development; it will not provide detailed instructions on how to use the tool. If you have questions on how to use the Administration Tool, refer to Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*

example portal

The name given to a sample portal implementation included with the WebLogic Personalization Server distribution. This example is built on top of the Portal Framework. If you took the Personalization Tour, you worked with the example portal. It is branded with an "Acme" logo. The files for this example portal coexist in the same folder with the files used for the Portal Framework. The difference is the Portal Framework files are generic, while the example portal files are specific to the example.

property set

WebLogic Personalization Server supports the storage of collections of data called property sets. These sets may be associated with users, groups, or sites. In this chapter, you will need to create and edit a property set that describes your portal. This kind of property set is called an "Application Init" property set and describes properties such as your portal's name, its working directory, and the Home page.

How to Use This Chapter

The WebLogic Personalization Server includes a Portal Administration Tool that allows you to quickly build a basic portal using the Portal Framework. Techniques for using the Portal Administration Tool are documented in the chapter Creating and Managing Portals in the *WebLogic Personalization Server User's Guide*. That chapter also includes step-by-step instructions for building the Acme Demo Portal. This tutorial will not repeat the information presented in the *User's Guide.*

The goal of this chapter is to get you started building your own custom portal. It will cover many techniques for customizing the Portal Framework. This chapter also provides many small projects which will demonstrate how to use these techniques. The code fragments used to build these projects are included.

However, this chapter does not explain what every line of code does in these samples. It provides general guidance in understanding how an example works, but the details are left as an exercise to the user. The reason for this is that the best way to learn how to develop with WebLogic Personalization Server is to reverse engineer code written by others. Once you get each example working, spend some time experimenting with the code. A good rule of thumb is to not proceed to the next example until you know what each line of code does in the previous example.

Creating the Framework for Your Custom Portal

This section describes how to build a custom portal. At the end of this section, you will have created a copy of the example portal (which uses the Portal Framework) which you will alter as you build your custom site. It is important that you use the example portal as a base since it does provide extensive functionality. Later, when you gain familiarity with the product, you can re-engineer your custom site one piece at a time.

Note: It is not recommended for new WebLogic Personalization Server developers to attempt to build a portal from scratch.

This section will walk you through the process one step at a time. It is primarily intended to help you get the framework of your custom portal up and running and does not attempt to explain the details of this process. In later sections, you will be introduced to the details in a more rigorous way.

Installing WebLogic Personalization Server

If you have already installed WebLogic Personalization Server 3.2, begin this procedure at step 14.

To install WebLogic Personalization Server, follow these steps:

1. Download and install JDK 1.2.1 or greater.
2. Restart your machine as required by the JDK installer.
3. Download the WebLogic 5.1 installer from the BEA Web site if you do not have the WebLogic Server 5.1 CD.
4. Obtain a valid WebLogic Server license file. If your site does not already have one, you can obtain an evaluation license from the BEA Web site. Rename it to `WebLogicLicense.xml`.
5. Run the WebLogic Server 5.1 installer and complete the WebLogic Server installation.

6. Install Service Pack 6 or greater for WebLogic Server 5.1. You can download the latest service pack from the BEA Web site. After unpacking it, you will need to manually update certain files and your classpath. Do this carefully. Making a mistake here can cause errors later in this process.

   Note: There are two errors in the Service Pack 6 install instructions. 1) Be sure to copy `weblogic510sp6.jar` and `weblogic510sp6boot.jar` to the `lib` subfolder and **not** the root folder. 2) Copy `weblogic-tags-510.jar` to the `lib` folder. For more information, consult the *WebLogic Commerce Server with WebLogic Personalization Server Installation Guide.*

7. Copy your WebLogic Server license file into the `license` subfolder in your WebLogic Server installation. WebLogic Server needs to verify that you have a proper license before it will start up. It looks for the license in the classpath, and the `license` folder is in the classpath.

8. Test WebLogic Server by clicking on `startWebLogic.cmd` (`startWebLogic.sh` for UNIX users). It should start up and print "WebLogic started."

   Note: It is very important to look at the console window and inspect the output for exceptions. If any exceptions occurred during startup, you will need to resolve the problem before WebLogic Server will work properly.

   Note: Windows users should modify the properties of their console window to extend the screen buffer to at least 500 lines. Accomplish this by right-clicking on your console window's title bar and choosing "Edit..." from the pop-up menu. Go to the Layout tab and change the Screen Buffer height to be at least 500. When you click OK, it will ask if you want to apply these changes for future windows of the same title. Check this option.

9. Shut down the server by attaching to the server using the WebLogic console and issuing a Shutdown command.

   Note: You may also shut down the server by pressing CTRL-C in the console window in which the server is running. This method is not recommended in production environments since it may cause database connections to be consumed until a database server reboot.

10. Download the WebLogic Commerce Server installer if you do not have the WebLogic Personalization Server 3.2 CD.

11. Obtain a valid WebLogic Personalization Server license file. If your site does not already have one, you may obtain an evaluation license from the BEA Web site. Rename it to `WebLogicCommerceLicense.xml`.

12. Run the WebLogic Personalization Server 3.2 installer and complete the WebLogic Personalization Server installation.

13. Copy your WebLogic Personalization Server license file into the `license` subfolder in your WebLogic Personalization Server installation. WebLogic Personalization Server needs to verify that you have a proper license before it will start up. It looks for the license in the classpath, and the `license` folder is in the classpath.

14. Test WebLogic Personalization Server by clicking on `StartCommerce.bat` (`StartCommerce.sh` for UNIX users). It should start up and print "WebLogic started." **Do not** shut down the server.

    Note: It is very important to look at the console window and inspect the output for Java exceptions. If any exceptions occurred during startup, you will need to resolve the problem before WebLogic Personalization Server will work properly.

    Note: If you have errors related to `foundation.jar` not loading properly, your WebLogic Server Service Pack update did not succeed. Go back and reinstall the Service Pack.

You have completed the installation of WebLogic Personalization Server 3.2. Now you will proceed with setting up the framework for your custom portal.

Setting Up the Portal Framework

To set up the framework for your custom portal, follow these steps:

1. Create a new subfolder in `%WL_COMMERCE_HOME%/server/public_html/portals` and give the folder the name of your new portal. This new folder will contain the resources specific to your portal. This chapter will assume you have named this folder eTestPortal. This name will also refer to the name of the portal.

   Note: Do not use spaces in this folder name or you will have complications in step 6 below.

2. Create two new subfolders in your new portal folder. Name one `images` and the other `portlets`.

3. Log into the WebLogic Personalization Server Administration Tool. If you installed WebLogic Personalization Server with the default settings, you can use this URL in a browser that is invoked on the same machine as the server: `http://localhost:7501/tools`. The default username is `administrator` and the default password is `password`.

4. Click the triple-can icon on the Property Set Management title bar. This will take you into the Property Set Management Administration Tool.

5. To register your eTestPortal, you need to create a new property set. Click the Create button on the title bar.

6. You are presented with a form.

   a. In the Name field, enter `eTestPortal`.

   b. In the Description field, enter something like `My test portal`.

   c. In the Copy Properties From drop-down list, select `APPLICATION_INIT._DEFAULT_PORTAL_INIT`.

   d. Finally, in the Property Set Type drop-down list, enter `Application Init`.

   e. Once you have completed the form, click the Create button.

   Note: Make sure you type this exactly as you see it. It is case sensitive and spaces should not be used.

7. You have just created a property set that will be used to register your application. Click the Back button.

   Note: Clicking Back will fail with an "Authorization Failed" message if your browser does not allow cookies. In this case, you must change your browser settings to allow cookies for the Administration Tool to function properly.

8. You now need to edit your new property set. Click the eTestPortal name in the Application Initialization property set list. This will invoke the property set editor for the eTestPortal property set.

9. Change the `defaultdest`, `homepage` and `workingdir` properties to point to your portal's folder. For example, `/portals/example/...` should be changed to `/portals/eTestPortal/....` This establishes the location of the files for your portal.

   Note: Do not change the `repositorydir` property.

10. Edit the portalName property. This must exactly match the name of the portal you are creating; in this example it should be eTestPortal.

11. Return to the Home page. Click Finished, then click Home.

12. Click the blue and red monitor icon on the Portal Management title bar. This will take you into the Portal Management Administration Tool.

13. Click the Create button on the Portals title bar. This will allow you to register your new portal with the server.

14. A form will appear.

    a. In the Portal Name form control, enter `eTestPortal` exactly as you did while editing the property set. Leave the rest of the controls as they are by default.

    b. Click Create. This should succeed. You have now successfully registered your new portal.

    c. Click Back to return to the Portal Manager page.

15. Click your portal's name underneath the portal's title bar. This will take you to the portal editor tool.

16. By default, no portlets will be included in your portal. You should add a few portlets. Click the +/- icon on the associated portlets title bar. You will now be directed to the portlet association page.

17. Make a few portlets available, and at least a couple of portlets visible. Click Save and then Back when finished.

    Note: Not all of the displayed portlets may be valid for your new portal. If any portlets were added for another portal on the server, these portlets may have a different relative path from `%WL_COMMERCE_HOME%/server/public_html/portals`. Adding a portlet located in another portal will give your users a run-time error. To avoid this problem, use only the portlets located in the

`%WL_COMMERCE_HOME%/server/public_html/portals/repository/portlets` subfolder (consult the list below).

The following list describes the available portlets.

Defined Portals—displays the portals defined in the system. Uses the `<es:forEachInArray>` and `<es:simpleReport>` tags.

Defined Portlets—displays the portlets defined in the system. Uses the `<es:forEachInArray>` and `<es:simpleReport>` tags.

News Index—demonstrates the use of Content Management tags.

News Viewer—displays content driven from `content_index.jsp`. Use in conjunction with News Index.

Quote—displays stock quotes. Demonstrates how to redirect a portlet to an external site.

Dictionary—demonstrates how to redirect a portal to an external site.

Search—demonstrates how to redirect a portal to an extend site.

Group To Do List*—displays a Group To Do list. Requires user to be logged in.

My To Do List*—displays a My To Do list. Requires user to be logged in.

Bookmarks*—displays the bookmarks associated with the current user. Requires user to be logged in.

18. Test your portal site as follows:

    Open a browser window and type in `http://localhost:7501/application/eTestPortal`.

    a. Replace `localhost` in this URL with the name of the machine the server is running on if it is different from the browser's machine.

    b. Replace `7501` if you changed the listening port of WebLogic Server during the WebLogic Personalization Server installation.

    c. Replace `eTestPortal` with the name of the property set you created.

    In your browser, you should see a Web page with an Acme logo. You should see all the portlets which you defined as being visible in the Portlet Administration page. Portlets that require a user login (marked with an ***** above) will not display until you are logged in.

Note: It is important to look at the console window to make sure exceptions are not being thrown.

Troubleshooting

If you do not see the portal page or have exceptions, make sure you have not made the following common mistakes:

Problem: Server not responding

a. Server is not running—make sure you have a console window with the Commerce Server running. It must output "WebLogic Server started." before it will accept connections.

b. Server is running on non-default port—this chapter assumes that your server is running on port 7501. Check the `weblogic.system.listenPort` property in your `%WL_COMMERCE_HOME%/weblogic.properties` file. The number assigned to this property is the server's port number.

c. Server is running on different machine—this chapter assumes that your server is running on the same machine as your browser. If not, replace the name `localhost` in your browser URLs with the name of the machine on which the server is running.

Problem: Server returns error

a. `PORTAL_NAME` not defined—when creating your property set, you must replace the default value of the `PORTAL_NAME` property.

b. `PORTAL_NAME` does not match portal name—the `PORTAL_NAME` property in your property set (step 6) must match exactly the name you give your portal when creating the portal in the Portal Manager (step 27).

c. `repositorydir` incorrect—you should have modified several paths in your property set (step 22) but not `repositorydir`. The `repositorydir` property should be `/portals/repository`.

19. Explore your new site. Create a new user account by clicking on the key icon in the top right-hand side of the page. Build a personalized Web page for your new user.

You have completed the first step in building your own custom portal.

Repository Directory

An important concept to understand is the repository directory. The repository directory (specified by your `repositorydir` property in your property set) is the location where the server looks to find a resource if it cannot find it in your working directory (specified by your `workingdir` property in your property set). If you followed the previous instructions, you did not populate your working directory with any files. Therefore, when you navigated to your site in "Setting up the Portal Framework," step 17, the server failed to find the files (specifically, your Home page `portal.jsp`) in your working directory and so it found them in your repository directory instead. It is important that you understand how this works.

Before you proceed, take a few moments now to read back through this section and review the steps you followed. Although you may not understand why each step was necessary, it is helpful to have a clear understanding of what the steps were. Also, spend some time looking in the repository directory to see what resources are provided by default. The repository directory contains the Portal Framework and resources specific to the Acme portal and its portlets.

The rest of this chapter is devoted to making incremental changes to your copy of the example portal so that it is transformed into your own custom portal. In the process, you will replace many pieces of the example portal, though your site will still be based on the Portal Framework.

Simple Customizations

The previous section described how to establish a platform for your custom portal. This section assumes that you successfully completed this process. Whereas the last section had a strictly-defined process, the following sections are project based. It is recommended that you do these projects in order, although it is not required. Each project will list any prerequisite projects.

Note: Remember that your browser may cache pages. If you make modifications to your site and they do not appear in your browser, then click Refresh. This will circumvent the cache and get an updated page from the WebLogic Personalization Server server. If this fails to work, you should flush your browser's disk cache, as follows:

- **IE users:** from the Tools menu, choose Internet Options. From the General tab of the dialog box, click the Delete Files button.
- **Netscape users:** from the Edit menu, choose Preferences. From the browse tree, click Advanced, then Cache. Click the Clear Memory Cache and Clear Disk Cache buttons.

If problems continue, try switching to a different browser (IE/Netscape) to view the page.

Project 1: Customizing the Acme Logos

Start by removing the Acme logo and replacing it with your own brand image, as follows:

1. Open the `%WL_COMMERCE_HOME%/server/public_html/portals/repository/images` folder using your preferred file system navigator. The file listing will show all of the graphics used in your custom portal. Spend some time opening these graphics so you gain familiarity with the graphical components of your site.

2. Copy the files `logo.gif` and `logo_small.gif` to the `images` subfolder in your eTestPortal folder. By doing this, you are creating your own copy of these images outside of the repository. Therefore, when these images are used in your portal, they come from your working directory and not the repository.

3. Launch your preferred image editing application. This application must support reading and writing the GIF file format.

4. Open the file called `logo.gif` located in your `%WL_COMMERCE_HOME%/server/public_html/portals/eTestPortal/images` subfolder using your image editor application.

5. When the image opens, you will notice that it is the logo image that appears at the top of your custom portal page.

6. Through whatever means is best for you, replace the Acme logo with your brand image. You may do this either by authoring a new image in your image editing application or using an existing GIF file with the same approximate dimensions as the original Acme logo. Ultimately, you will need to have your GIF format image named `logo.gif` in your `/eTestPortal/images` subfolder.

7. There is another image called `logo_small.gif` located in your `images` subfolder that also needs to be updated. This is simply a smaller version of the main logo and is used on the bottom of your portal pages. Update this image as you did the first image.

8. Be sure you have updated the files in the `/eTestPortal/images` folder and that the new GIFs have exactly the same name as the original GIFs.

9. Use your browser to display the first page of your custom portal (refer to "Setting Up the Portal Framework" on page 6-7).

   Note: WebLogic Personalization Server will automatically detect the new images and load them in, so you do not need to restart the server. Remember to click Refresh in your browser. If after clicking Refresh, you do still do not see your new image, you may have to flush your browser's disk cache.

Project 2: Customizing the Choice of Portlets

In the previous section, you were asked to randomly choose a set of portlets to assign to your portal. Now that you are up and running, it is time to revisit the choice of portlets you made. Log on to the Administration Tool. Go to the Portal Manager and double click the name of your portal under the Portals title bar. You will see the portal edit page. Click the +/- icon on the portlets title bar. The portlet selection editor will appear.

Experiment with the controls and choose the portlets that you want to include. At this time, you can choose from only the prebuilt portlets. In later sections you will be building your own portlets. When choosing a portlet for your site, you may also specify whether it is visible by default, not visible but available, or not available at all.

For more information, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*

Project 3: Customizing the Layout of Portlets

If you look carefully at your portal, you will notice that your portal has three main sections: a title bar, a container in the middle for a number of smaller components, and a footer. The smaller components in the middle space are called "portlets." Each portlet is written as an independent JSP or HTML file. Each portlet is responsible for its own contents, while the portal page is responsible for laying the portlets out in columns.

You can customize this layout in two ways:

1. You can specify how many columns the portal uses to arrange portlets (1, 2 or 3 columns are valid).
2. You can choose which portlets appear in each of the columns.

Note: As an administrator you may define how the layout appears by default, but the user may override your choices.

Log on to the Administration Tool. Go to the Portal Manager and then click the name of your portal under the Portals title bar. You will see the portal edit page. By editing the portal definition, you may change the number of columns used to display your portal. To change which portlets are in which column, you need to use the Layout editor.

For more information, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide*.

Project 4: Describing Your Users

Your portal site will most likely have a number of categories of users. WebLogic Personalization Server recognizes this by supporting a feature called User Groups. With this feature, you can define a hierarchical set of groups to which you can assign users. Additionally, you will may have a number of predefined users that you would like to initially set up. As part of this process, you will want to assign these people to one or more of the groups you have set up. This project details how to build this into your portal.

1. Log on to the Administration Tool.
2. Navigate into the User Manager.
3. Click Create on the Users title bar to create a number of new users.
4. Return to the User Manager and create a number of new groups by clicking Create from the Groups title bar.
5. Return to the User Manager and click the text "Groups" from the Groups title bar, which will take you to the Group editor. Edit the groups you have created, and in doing so assign some users to each of the groups.
6. Finally, go to the Portal Manager and edit your custom portal. In the Associated Groups section, add your new groups to your portal.

For more information, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide*.

Writing Your Own Portlets

"Creating the Framework for Your Custom Portal" on page 6-5 instructed you how to get up and run with your custom portal by copying the example portal. "Simple Customizations" on page 6-13 showed you how to alter the logo images and use the Administration Tool to customize your portal. Now, this section will show you how to build your own functionality into your custom portal.

In this section, you will see how to build static portlets and portlets that change based on state information retrieved from WebLogic Personalization Server. Once you have mastered the essence of portlet writing, you will learn about advanced functionality such as maximized porlets and inter-portlet communication in the next section, Advanced Portlet Functionality.

Note: In the following projects, you will be registering your portlets and adding them to your portal using the Administration Tool. When doing these activities or when editing the portlet definition after creation, you may not see the changes to your portal in your browser even after flushing the browser cache. This is due to server-side caching. In order to force a rebuild of your portal page, you must log in if you are logged out, or log out if already logged in to see your changes take effect.

Project 5: Building a Static Portlet

The fundamentals of portlet writing are not difficult. As you will see, the construction of a static portlet is quite easy. The complexity of portlets come when they become dynamic. The first portlet project is a static portlet.

When you view your portal, your browser is displaying an HTML page to you. If you "View Source" on your portal, you will not see any JavaServer Page tags, although JSP was used to author the document. This is because the server processed the JSP input and output the HTML to your browser. In exactly the same way, before your portlet is placed on the portal page, it is processed and converted into HTML if it was not already. With this in mind, think of a portlet as just a small Web page.

For this static portlet project, you are not going to use a JSP. You will build just an HTML fragment that will get included into the portal page. What follows is your first portlet.

welcome.html

```
<p align=center>
<h1>Welcome!</h1><br>

This portal contains the projects built by following the WLCS
tutorial.
</p>
```

Students of HTML will recognize this as just a simple static HTML fragment. There really is nothing special about this HTML.

To make the HTML fragment above into a portlet, follow these steps:

1. With your file system navigation tool, navigate to your custom portal folder. Then, enter the `portlets` subfolder. Here, create a new file called `welcome.html` with a text editor. This HTML file you just created will hold your first portlet.

   Note: Windows users using WordPad or Microsoft Word should Save As a text document; otherwise, extra unwanted characters will be dumped into the text stream.

2. Copy the HTML fragment above into `welcome.html`. This file should contain no other text. Save this file to disk.

3. Log into the WebLogic Personalization Server Administration Tool. For more information on completing this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*

4. Click the blue and red monitor icon on the Portal Management title bar. This will take you into the Portal Management Administration Tool.

5. Click the Create button on the Portlets title bar. This will allow you to register your new portlet with the server.

   Note: Be sure to click the Create button on the Portlet title bar and not the one on the Portal title bar.

6. You are presented with a form. There are two required fields which you must fill in, the rest you should leave with the defaults. For Portlet Name, enter `Welcome`. For Content URL, enter `portlets/welcome.html`. Press the Create button, and then press the Back button. You have now successfully registered your new portlet with the server.

7. Add the portlet to your custom portal and make it visible. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*

8. Use your browser to display the first page of your custom portal, following the procedure described in "Creating the Framework for Your Custom Portal" on page 6-5. You should see your Welcome portlet.

You have now added new functionality to WebLogic Personalization Server.

Project 6: Building a Simple Dynamic Portlet

Now that you see how easy it is to build portlets, your next step is to add some dynamic behavior to your portlet. For this, you will need to create a JavaServer Page file, not HTML. With the power of JSP, you can query the WebLogic Personalization Server for information, and vary the output depending on the results of your queries.

In this project, you will build a portlet that will detect if the user of the browser is logged on to the portal. If not, this portlet will display a message to the user, asking the user to log on. If the user is logged on, the portlet will instead display the user's login name.

You will also accomplish the dynamic functionality by using methods included in the portlet JSP base class. All portlet JSPs should extend `com.beasys.commerce.portal.admin.PortalJspBase`.This class contains many convenience methods which perform general tasks for your portlet JSP page, such as accessing session information and user login information. To achieve this, begin your portlet JSP files with the following line:

```
<%@ page
extends="com.beasys.commerce.portal.admin.PortalJspBase"%>
```

Once you have extended `PortalJSPBase`, you have access to many methods from your JSP file, including `getLoggedIn()` and `getSessionValue()`. Instead of explaining exactly what these methods do here, look instead at the following JSP fragment

isloggedon.jsp

```
<%@ page extends="com.beasys.commerce.portal.admin.PortalJspBase"
%>

<p>
<%
    // getLoggedIn() returns true if the user is logged in
    if ( getLoggedIn(request) )
    {
%>
       You are currently logged in as
       <%= getSessionValue(
com.beasys.commerce.axiom.jsp.JspConstants.SERVICEMANAGER_USER,
              request)
       %>
       . Please make yourself at home.
<%
    }
    else
    {
%>
       You are not currently logged on. Please click the
       key icon at the top right-hand corner of the page to
       log onto this site.
<%
    }
%>
</p>
```

This code is the complete text for your first dynamic portlet.

To implement this project, follow these steps:

1. With your file system navigation tool, navigate to your custom portal folder. Then, enter the `portlets` subfolder. Here, create a new file called `isloggedon.jsp` with a text editor. This JSP file you just created will hold your first dynamic portlet.
2. Copy the JSP fragment above into `isloggedon.jsp`. This file should contain no other text. Save this file to disk.

3. Log into the WebLogic Personalization Server Administration Tool. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*
4. Click the blue and red monitor icon on the Portal Management title bar. This will take you into the Portal Management Administration Tool.
5. Click the Create button on the Portlets title bar. This will allow you to register your new portlet with the server.
6. You are presented with a form. There are two required fields which you should fill in, the rest you should leave with the defaults. For Portlet Name, enter `Logged on?` For Content URL, enter `portlets/isloggedon.jsp`. Click Create, then click Back. You have now successfully registered your new portlet with the server.
7. Add the portlet to your custom portal and make it visible. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*
8. Use your browser to display the first page of your custom portal, following the procedure described in "Creating the Framework for Your Custom Portal" on page 6-5. You should see your "Logged on?" portlet. See how the text in the portlet changes depending on whether you are logged into the portal or not.

You have now finished your first dynamic portlet.

Project 7: Building a Second Dynamic Portlet

In project 6, you built a simple dynamic portlet using functionality provided by extending PortalJSPBase. Take some time to look at other functionality provided by PortalJSPBase. Once you have done this, you are ready to begin working with another dynamic technique available to your JavaServer Page portlets, JSP tags.

Included with the WebLogic Personalization Server is a set of tag libraries that enable your JSPs access to the full power of the personalization engine. The five tag libraries are as follows:

- Personalization Advisor: uri = "pz.tld"
- Content Management: uri = "cm.tld"

- Portal Management: uri = “esp.tld“
- User Management: uri = “um.tld“
- Personalization Utilities: uri = “es.tld“

For more information, see Chapter 10, “JSP Tag Library Reference.” For full details on how tag libraries work in the JSP language, consult a JavaServer Page handbook. After viewing this sample tag library portlet, you will see that tag libraries are quite easy to use.

Each one of these tag libraries supports a number of tags. In this project, you will use tags from the User Management and Personalization Utilities tag libraries. This portlet will output the name of all users of your portal. Next to each username, it will output the e-mail address of that user. A detailed description of how this code works is not provided here. Hopefully, reading the code and consulting the *WebLogic Personalization Server Developer’s Guide* is sufficient.

One point about the code should be made. You will notice that for every username retrieved by calling <um:getUsersnames>, there is a call to <um:getProfile>. It is necessary to explain why this line of code is needed. At any time during the processing of a portlet, exactly one user profile is in scope. Calls like <um:getProperty> and <um:setProperty> refer to the user profile in scope. In this project the code must iterate through the list of usernames and query the profile associated with each user. Therefore, before a call to <um:getProperty> is made, the profile for the user must be loaded into scope by calling <um:getProfile>. And at the end of this JSP, the original user profile must be loaded back into scope to avoid causing problems with other portlets.

EmailList.jsp

```
<%-- include the tag libraries we need --%>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="um.tld" prefix="um" %>

<%-- extend PortalJSPBase to get some base functionality --%>
<%@ page
       extends="com.beasys.commerce.portal.admin.PortalJspBase"
%>

<%
    // get the name of the current user
    String originalUserName = (String)getSessionValue(
com.beasys.commerce.axiom.jsp.JspConstants.SERVICEMANAGER_USER,
```

```
        request);
    if ( originalUserName == null ) originalUserName = "";

    // get the name of the portal
    String portalName =
       (String)getSessionValue(PORTAL_NAME, request);
    if ( portalName == null ) portalName = "";
%>

<%-- ask WLCS to put a list of the user names in string array
"userNameList" --%>
<um:getUsernames id="userNameList" result="namesResult"/>

<table border=1 cellspacing=1 align="center">

<tr>
       <th colspan=2>Portal Users</th>
</tr>

<es:forEachInArray id="curUser" type="String"
             array="<%=userNameList%>"
             counterId="curIndex">

<tr>
       <%-- This section is evaluated once for
             every user in userNameList --%>

       <%-- Output the name of the user for this row --%>
       <td><%=curUser%></td>

       <%-- Output the email address of the curUser --%>
       <td>

             <um:getProfile profileKey="<%=curUser%>"
                     scope="request"
             />
             <um:getProperty id="email"
                    propertySet="<%=portalName%>"
                    propertyName="<%=PROFILE_EMAIL%>"
             />
             <%=email%>
       </td>
</tr>
</es:forEachInArray>

<%
if (getLoggedIn(request)) {
%>

       <um:getProfile
       profileKey="<%=originalUserName%>" scope="request" />
```

```
<%
}
%>

</table>
```

This code is the complete text for your second dynamic portlet.

To implement this project, follow these steps:

1. With your favorite file system navigation tool, navigate to your custom portal folder. Then, enter the `portlets` subfolder. Here, create a new file called `EmailList.jsp` with a text editor. This JSP file you just created will hold your first dynamic portlet.

2. Copy the JSP fragment above into `EmailList.jsp`. This file should contain no other text. Save this file to disk.

3. Log into the WebLogic Personalization Server Administration Tool. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*

4. Click the blue and red monitor icon on the Portal Management title bar. This will take you into the Portal Management Administration Tool.

5. Click the Create button on the Portlets title bar. This will allow you to register your new portlet with the server.

6. You are presented with a form. There are two required fields which you should fill in, the rest you should leave with the defaults.

   a. For Portlet Name, enter `Email List`.

   b. For Content URL, enter `portlets/EmailList.jsp`.

   Click Create, then click Back. You have now successfully registered your new portlet with the server.

7. Add the portlet to your custom portal and make it visible. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide*.

8. Use your browser to display the first page of your custom portal, following the procedure described in"Creating the Framework for Your Custom Portal"step 18. You should see your "Email List" portlet.

9. Now add some more users to your portal. Do this by clicking the key icon in the upper right-hand corner of your portal. This will bring up the sign-on page. Under the New User title bar, click Create. Follow the instructions to add new users.

10. Return to your portal page and see the new users appear in the portlet.

You have now seen the three major techniques for building a portlet: static HTML, dynamic behavior based on extending PortalJSPBase, and dynamic behavior based on the WebLogic Personalization Server tag libraries.

Advanced Portlet Functionality

In the previous section, you learned how to build portlets. This section continues with portlets and demonstrates how to use more portlet features.

Project 8: Adding a Maximized URL

This project will walk you through how to build a maximized version of your portlet. In the default case, your portlet cannot be maximized. If you allow your portlet to be maximized but do not provide a maximized URL, WebLogic Personalization Server will simply use your normal-sized portlet content when maximized. In most cases, you will want to take advantage of the extra space afforded by being maximized and alter your portlet content to include more information. This project shows how to do this. It assumes you completed "Project 7: Building a Second Dynamic Portlet". You will not change the portlet created in that project, but you will add a new maximized JSP.

The first step of this project is to create the new portlet content JSP for the maximized state. Since in the maximized state your portlet has more screen real estate, you can display more information. In this case, for each user displayed, you will show more columns of information. The following is the maximized JSP:

EmailListMax.jsp

```
<%-- include the tag libraries we need --%>
<%@ taglib uri="es.tld" prefix="es" %>

<%@ taglib uri="um.tld" prefix="um" %>

<%-- extend PortalJSPBase to get some base functionality --%>
<%@ page
extends="com.beasys.commerce.portal.admin.PortalJspBase"
%>

<%
       // get the name of the current user
       String originalUserName = (String)getSessionValue(
com.beasys.commerce.axiom.jsp.JspConstants.SERVICEMANAGER_USER,
              request);
       if ( originalUserName == null ) originalUserName = "";
```

```
        // get the name of the portal
        String portalName =
        (String)getSessionValue(PORTAL_NAME,request);
        if ( portalName == null ) portalName = "";
%>

<%-- ask WLCS to put a list of the user names in
        string array "userNameList" --%>
<um:getUsernames id="userNameList" result="listresult" />

<table border=1 cellspacing=1 align="center">

<tr>
        <th colspan=7>Portal Users</th>
</tr>

<es:forEachInArray id="curUser" type="String"
        array="<%=userNameList%>" counterId="curIndex">

<tr>
        <%-- This section is evaluated once for
                every user in userNameList --%>

        <%-- Output the name of the user for this row --%>
        <td><%=curUser%></td>

        <um:getProfile profileKey="<%=curUser%>" scope="request" />



        <%-- Output the email address of the curUser --%>

        <td>
                <%-- Load curUser's profile into scope -->
                <um:getProperty id="email"
                        propertySet="<%=portalName%>"
                        propertyName="<%=PROFILE_EMAIL%>"/>
                <%=email%>
        </td>

        <%-- Output the first name of the curUser --%>
        <td>
                <um:getProperty id="first"
                        propertySet="<%=portalName%>"
                        propertyName="<%=PROFILE_FIRST%>"/>
                <%=first%>
        </td>

        <%-- Output the last name of the curUser --%>
        <td>
                <um:getProperty id="last"
```

```
                        propertySet="<%=portalName%>"
                        propertyName="<%=PROFILE_LAST%>"/>
                <%=last%>
        </td>

        <%-- Output the address of the curUser --%>
        <td>
                <um:getProperty id="address"
                        propertySet="<%=portalName%>"
                        propertyName="<%=PROFILE_ADDRESS%>"/>
                <%=address%>
        </td>

        <%-- Output the city of the curUser --%>
        <td>
                <um:getProperty id="city"
                        propertySet="<%=portalName%>"
                        propertyName="<%=PROFILE_CITY%>"/>
                <%=city%>
        </td>

        <%-- Output the state of the curUser --%>
        <td>
                <um:getProperty id="state"
                        propertySet="<%=portalName%>"
                        propertyName="<%=PROFILE_STATE%>"/>
                <%=state%>
        </td>

        <%-- Output the zip code of the curUser --%>
        <td>
                <um:getProperty id="zip"
                        propertySet="<%=portalName%>"
                        propertyName="<%=PROFILE_ZIP%>"/>
                <%=zip%>
        </td>
</tr>
</es:forEachInArray>

<%
if (getLoggedIn(request)) {
%>
        <um:getProfile profileKey="<%=originalUserName%>"
                scope="request" />
<%
}
%>
</table>
```

To specify the above code as your maximized portlet content, follow these steps:

1. With your file system navigation tool, navigate to your custom portal folder. Then, enter the `portlets` subfolder. Here, create a new file called `EmailListMax.jsp` with a text editor. This JSP file you just created will hold your maximized portlet code.

2. Copy the JSP fragment above into `EmailListMax.jsp`. This file should contain no other text. Save this file to disk.

3. Log into the WebLogic Personalization Server Administration Tool. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*

4. Click the blue and red monitor icon on the Portal Management title bar. This will take you into the Portal Management Administration Tool.

5. Under the Portlets title bar, click the EmailList portlet you created in "Project 7: Building a Second Dynamic Portlet" on page 6-21.

6. You will be presented with a form.

   a. Change the state of the "Maximizable" check box so that it is checked.

   b. In the "Maximized URL" edit field, enter the `EmailListMax.jsp` file you created in step 1 (in this case: `portlets/EmailListMax.jsp`).

   Click Save, then click Back.

7. Open another browser window and navigate to your portal, following the procedure described in "Setting Up the Portal Framework" on page 6-7. Your portlet should be visible, and will have a square-like icon on its title bar. This indicates that it is maximizable.

   Note: Remember to log out and log in to force the server to rebuild the portal page.

8. Click the maximize icon, which will cause the portlet to be maximized. The `EmailListMax.jsp` page will load, listing each user, plus more information per user than what is displayed when the portlet is of normal size.

Project 9: Changing the Look of a Maximized Portlet

Project 9 assumes that you have completed "Project 8: Adding a Maximized URL." The goal of this example is to specify a non-default header and footer to be used when your portlet is maximized. In Project 8, the portlet used the provided `alternateheader.jsp` and `alternatefooter.jsp` since you did not override the defaults. In some cases the defaults are sufficient, but for the purposes of demonstration this project will define its own.

EmailListMaxHeader.jsp

```
<p>
<h1>Email List Portlet</h1>
<hr/>
</p>
```

EmailListMaxFooter.jsp

```
<p>
<hr/>
<i>Creating a Custom Portal</i> Tutorial
</p>
```

These code examples are just simple samples. You may elaborate on the design if you wish.

To replace the alternate header and footer with the code above, follow these steps:

1. With your file system navigation tool, navigate to your custom portal folder. Create two new files called `EmailListMaxHeader.jsp` and `EmailListMaxFooter.jsp` with a text editor. This JSP files you just created will hold the header and footer for your maximized portlet.

   Note: Be sure to create these files in your portal's root folder and not in your portlets subfolder.

2. Copy the first JSP fragment above into `EmailListMaxHeader.jsp`, and the second into `EmailListMaxFooter.jsp`. These files should contain no other text. Save these files to disk.

3. Log into the WebLogic Personalization Server Administration Tool. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide.*

4. Click the blue and red monitor icon on the Portal Management title bar. This will take you into the Portal Management Administration Tool.

5. Under the Portlets title bar, find the EmailList portlet you created in Project 7 and click on it.

6. You will presented with a form.

   a. In the Alternate Header URL, enter `EmailListMaxHeader.jsp`.

   b. In the Alternate Footer URL, enter `EmailListMaxFooter.jsp`.

   Click Save, then click Back.

7. Open another browser window and navigate to your portal. Your portlet should be visible, and will have a square-like icon on its title bar. Click the icon, which will cause the portlet to be maximized.

8. `EmailListMax.jsp` will load, along with `EmailListMaxHeader.jsp` and `EmailListMaxFooter.jsp`.

   Note: Remember to log in and log out to force the server to rebuild the portal page.

This concludes the customization of your portlet's maximized state. You may also experiment with creating an editable version of your portlet, and even a help window associated with your portlet.

Project 10: Inter-portlet Communication

In some instances, you will want to have actions in one portlet affect the display of another portlet. This is called inter-portlet communication. To implement this functionality, imagine writing Java Beans or database calls where one portlet persists data and another reads the data. This would work, but requires more effort than necessary. Unless you need to pass large amounts of data between portlets, you should follow a simpler approach. This approach is demonstrated here.

Before you read the code, you must understand that there is an object that is shared between the Portal Framework and all of the portlets. This object can be used to pass data between all of these entities. This object is the HTTP request. You may set parameters in the request in one portlet, then forward the request back to the portal, which will ultimately forward the request to all portlets. The end result is that the HTTP request can serve as a message passing mechanism for portlets. This project shows you how to exploit this.

The goal of this project is to create two portlets. The first portlet, called "User Index," displays a ranges of usernames in the system. For example, it will show "Allan to Carl, Chuck to Elmer, Francis to Irene,..." If the user clicks on a name range, the second portlet will refresh and show detailed information about each username in that range. The second portlet is called "User Index Details."

The following is the code for both portlets. Take special note of how the "href" is constructed in `UserIndex.jsp`, and how the parameters are retrieved in `UserIndexDetails.jsp`.

UserIndex.jsp

```
<%-- include the tag libraries we need --%>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="um.tld" prefix="um" %>

<%-- extend PortalJSPBase to get some base functionality --%>
<%@ page
 extends="com.beasys.commerce.portal.admin.PortalJspBase"%>

<%
    // get the name of the current user
    String originalUserName = (String)getSessionValue(
com.beasys.commerce.axiom.jsp.JspConstants.SERVICEMANAGER_USER,
        request);
    if ( originalUserName == null ) originalUserName = "";

    // get the name of the portal
    String portalName = (String)getSessionValue(PORTAL_NAME,
        request);
    if ( portalName == null ) portalName = "";
%>

<%-- ask WLCS to put a list of the user names in string
        array "userNameList" --%>
<um:getUsernames id="userNameList" result="listresult" />
```

```
<table border=1 cellspacing=1 align="center">

<tr>
       <th>Portal Users Index</th>
</tr>

<%
int divisor = 5;
boolean isRowTerminated = true;
String persistCurUser = null;
%>

<es:forEachInArray id="curUser" type="String"
       array="<%=userNameList%>" counterId="curIndex">
       <%-- This section is evaluated once for every user in
              userNameList --%>

<%
       persistCurUser = curUser;

       // start the cell if this is the first user in a range
       if (curIndex.intValue()%divisor == 0)
       {
              // beginning of range
              isRowTerminated = false;
%>
              <%-- THIS IS WHERE THE DATA IS PASSED --%>
              <tr><td>
              <a href="<%=response.encodeURL(createURL(request,
                     getHomePage(request),
                     ("userIndexStartIndex=" + curIndex
                     + "&userIndexDivisor=" + divisor
                     )))%>"
              >
              <%=curUser%> to
<%
       }

       // finish cell if this is the last user in range or
       // the last user in the list
       if (curIndex.intValue()%divisor == divisor-1)
       {
              // end of range
              isRowTerminated = true;
%>
              <%-- Output the name of the user for this row --%>
              <%=curUser%></a></td></tr>
<%
       }
```

```
%>
</es:forEachInArray>
<%
       if (!isRowTerminated)
       {
              // terminate the row if it ended on a
              // non-divisor boundary
%>
              <%-- Output the name of the user for this row --%>
              <%=persistCurUser%></a></td></tr>
<%
       }
%>
</table>
```

UserIndexDetails.jsp

```
<%-- include the tag libraries we need --%>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="um.tld" prefix="um" %>

<%-- extend PortalJSPBase to get some base functionality --%>
<%@ page
extends="com.beasys.commerce.portal.admin.PortalJspBase"%>

<%
    // get the name of the current user
    String originalUserName = (String)getSessionValue(
com.beasys.commerce.axiom.jsp.JspConstants.SERVICEMANAGER_USER,
              request);
    if ( originalUserName == null ) originalUserName = "";

    // get the name of the portal
    String portalName =
              (String)getSessionValue(PORTAL_NAME, request);
    if ( portalName == null ) portalName = "";
%>

<%
// GET THE PARAMETERS PASSED IN, if they exist

       int startIndex = 0;
       String startIndexString =
                     request.getParameter("userIndexStartIndex");
       if (startIndexString != null)
       {
              try
              {
```

```
                        startIndex =
                                Integer.parseInt(startIndexString);
                }
                catch (Exception e) {
                        System.out.println("UserIndexDetail.jsp - "+
                        "startIndex parse error: "+startIndexString);
                }
        }

        int divisor = 0;
        String divisorString =
                         request.getParameter("userIndexDivisor");
        if (divisorString != null)
        {
                try
                {
                        divisor = Integer.parseInt(divisorString);
                }
                catch (Exception e) {
                        System.out.println("UserIndexDetail.jsp - "+
                        " divisor parse error: "+divisorString);
                }
        }

        if (divisor == 0)
        {
%>
        Click a name range in the User Index portlet to display
        information about each user in the range.
<%
        }
        else // divisor !=0
        {
%>

<%-- ask WLCS to put a list of the user names in string array
        "userNameList" --%>
        <um:getUsernames id="userNameList" result="listresult" />

        <table border=1 cellspacing=1 align="center">

        <tr>
        <th colspan=2>Portal Users</th>
        </tr>

        <es:forEachInArray id="curUser" type="String"
                array="<%=userNameList%>" counterId="curIndex">
<%
        if ((curIndex.intValue() >= startIndex) &&
```

```
         (curIndex.intValue() < startIndex+divisor))
        {
%>
        <tr>
        <%-- This section is evaluated once for every user in
               userNameList --%>

        <%-- Output the name of the user for this row --%>
        <td><%=curUser%></td>

        <%-- Output the email address of the curUser --%>
        <td>
               <um:getProfile profileKey="<%=curUser%>"
                      scope="request" />
               <um:getProperty id="email"
                propertySet="<%=portalName%>"
                propertyName="<%=PROFILE_EMAIL%>"
               />
               <%=email%>
        </td>
        </tr>
<%
        }
%>
        </es:forEachInArray>

<%
        if (getLoggedIn(request)) {
%>
               <um:getProfile profileKey="<%=originalUserName%>"
                      scope="request" />
<%
        }

        } // from else clause of if (divisor == 0)
%>

</table>
```

To use the above code as your portlets, follow these steps:

1. With the file system navigation tool, navigate to your custom portal folder. Navigate into your “portlets” subfolder. Here, create two new files called `UserIndex.jsp` and `UserIndexDetails.jsp` with a text editor. These JSP files you just created will hold the two portlets that will communicate with each other.

2. Copy the first JSP fragment above into `UserIndex.jsp`, and the second into `UserIndexDetails.jsp`. These files should contain no other text. Save these files to disk.

3. Log into the WebLogic Personalization Server Administration Tool. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide*.

4. Click the blue and red monitor icon on the Portal Management title bar. This will take you into the Portal Management Administration Tool.

5. Click the Create button on the Portlets title bar. This will allow you to register your new portlets with the server.

6. You are presented with a form. There are two required fields which you should fill in, the rest you should leave with the defaults.

   a. For Portlet Name, enter `User Index`.

   b. For Content URL, enter `portlets/UserIndex.jsp`.

   Click Create. You have now successfully registered your first new portlet with the server.

7. You still are presented with the form. You should now overwrite the values you entered in step 6.

   a. For Portlet Name, and enter `User Index Details`.

   b. For Content URL, enter `portlets/UserIndexDetails.jsp`.

   Click Create. You have now successfully registered your second new portlet with the server.

8. Add the portlets to your custom portal and make them visible. For more information about this step, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide*.

9. Test your portlets by directing your browser to your portal. Use your browser to display the first page of your custom portal, following the procedure described in "Setting Up the Portal Framework" on page 6-7. You should see your "User Index" and "User Index Details" portlets.

10. Now add some users to your portal. Do this by clicking on the key icon in the upper right-hand corner of your portal. This will bring up the signon page. Under the New User title bar, click Create. Follow the instructions to add new users.

11. Go back to main portal page. Notice that username ranges are displayed in the User Index portlet. Click on one of the ranges. Notice how the User Index Details portlet now displays detailed information on the users in the name range you selected.

You have now completed the advanced portlet creation projects.

Using the HTTP Request Method to Communicate Between Portlets

There are two issues that you should be aware of when using the HTTP request method to communicate between portlets.

Parameter Name Collisions Between Portlets

Because the HTTP request is broadcast to all portlets within the portal, you must be careful to avoid a parameter name collision between portlets. For example, suppose a portlet appends a URL parameter such as `src=/usr/local/src` to the anchor tags that it generates. If other portlets look for such a parameter, then they will all find it when the user clicks the link. Unless all portlets looking for that parameter are interpreting it the same way, confusing or bad parameters can be passed to the receiving portlets.

To avoid name collisions, remember that the URL parameter names that get encoded by using the URL string are global in nature.

Several Sets Of Portlets Using The Http Request Method At Once

When a set of portlets communicates, the code does the following:

```
<a href="<%response.encodeURL(createURL(request,
getHomePage(request),...and so on.
```

When the `getHomePage()`method is called, it strips off any parameters that were passed in from the last request. This is desirable when a single set of portlets communicates using this method. However, if an unrelated set of portlets is employing the same technique, then only one set can be “active” at any one time.

For example, one set of portlets (group A) includes a headline browser portlet and a news story display portlet. When a user clicks a headline, the headline browser uses the code above to generate a URL. The URL includes a parameter that tells the story display portlet where to find the story. When these are the only two portlets using this method, it works correctly.

Now lets add a second set of portlets to the example. Portlet group B includes a stock quote portlet and a stock detail portlet.When a user clicks a stock quote, the portlet uses the code above to generate a URL. The URL includes a parameter that tells its partner portlet where to find the stock details. Again, this method works correctly when just one set of portlets is communicating. However, what happens when both sets of portlets are using the `getHomePage()`method?

In this scenario, first the user clicks a story headline to see the story. The story portlet reads the parameter from the URL and displays the story. Then the user clicks a stock symbol to get detailed stock information, and the stock quote portlet generates a new anchor tag using a stock related parameter. The stock details display in the partner stock portlet just fine, but what happens in the story portlet?

When the user clicks the stock symbol, this second event notifies all the portlets to poll again. The URL generated by the stock quote portlet overwrote the news story parameter with its own stock related parameter, and so the new URL contains no parameters related to the story. If the story portlet does not find the parameter it expects, it may just display a blank portlet page. You can easily avoid this problem using the session cache. If the story portlet has already seen a parameter telling it where to find the story, it could use the session cache to default to the previous story location if there is no new information.

Other Customization Techniques

This tutorial has introduced you to the power of WebLogic Personalization Server and the Portal Framework. With the techniques described in the previous sections, you can now build a sophisticated portal.

However, there are more ways in which you can build your customized application. Although this chapter will not cover these additional techniques in detail, this section discusses each technique and provides pointers to other documentation that contains more information.

More Portlet Customization

In Projects 8 and 9, you created a maximized version of your portlet. In the same manner, you may create an editable version of your portlet. You may also specify a help page for your portlet, mandate that it appear for all users, allow/disallow it to be minimized, make it floatable, and make changes to the title bar appearance.

Editing the portlet's properties via the Associated Portlets link is only necessary if you wish to override the default values used to create the portlet.

For more information, about the Portal Management Administration Tools, see Creating and Managing Portals in the *WebLogic Personalization Server User's Guide*.

Database Interaction

In many cases you will want to persist data or retrieve persisted data from within your portal and portlets. In most cases, you will be using a database for this purpose. WebLogic Personalization Server provides simple connectivity with your database via Personalization Utility tags. Specifically, refer to the `<es:preparedStatement>` and `<es:simpleReport>` tags.

For more information, see "Personalization Utilities" on page 10-72.

Alternatively, enterprise applications will likely want to use Enterprise Java Beans for data persistence. WebLogic Commerce Server provides full support for the EJB specification.

For more information, consult the WebLogic Server documentation.

Java Beans Interaction

The Java Beans technology provides an easy way to remove the bulk of your code from your JavaServer Page files. This will free your JSP files from clutter, and make that code more maintainable and reusable. WebLogic Commerce Server provides full support for calling Java Beans from your JSP files.

For more information, consult a JavaServer Page handbook.

Personalization Advisor Functionality

The Advisor delivers content to a personalized application based on a set of rules and user profile information. It can retrieve any type of content from a Document Management system and display it in a JSP or use it in a servlet. The Advisor ties together all the services and components in the system to deliver personalized content. The Advisor component includes a JSP tag library and an Advisor EJB (stateless session bean) that access the WebLogic Personalization Server's core personalization services.

For more information, consult Chapter 2, "Creating Personalized Applications with the Advisor."

Internationalization

WebLogic Personalization Server provides a simple framework that allows access to localized text labels and messages. The internationalization (I18N) framework is accessible from JSP files through a small I18N tag library.

For more information, consult Chapter 8, "Localizing Applications with the Internationalization Tags."

Using Webflow

In WebLogic Personalization Server, Webflow is a feature that allows you to string together JSP files, input processors (IPs) and pipeline processors (PPs) without hard coding the linkage between them. Instead, the linkage is defined in an external Webflow properties file.

If you are considering using Webflow within the Portal Framework, see “Using Webflow Within a Portal” on page 7-6.

Commerce Functionality

The process customers go through when making a purchase from your Web site is one of the most common but complex aspects of an e-business. To help you get to market faster than your competitors, the WebLogic Commerce Server product provides you with an Order Processing package. This package contains default implementations for the most common e-business order-related services (such as shopping cart management, taxation, payment, and so on). Designed to be used out-of-the-box, the Order Processing package allows your site designers to customize the order process without the need for advanced programming skills. Additionally, it is easily extensible for those with advanced technical knowledge.

For more information, consult the *Order Processing Package* documentation.

Modifying the Portal Framework

In some cases, the Portal Framework may be generally suitable for your needs, but some aspects of it needs to be modified. A valid option in this case is to actually modify the JSP files in the repository folder. You are encouraged to use the Portal Framework files in any way to help you get to market quickly. Be aware that some changes you make may be inconsistent with the Administration Tool, therefore you will need to implement your own administration functionality in those cases.

To do this, you will need to read and understand the contents of the JSP files located in the repository folder. Therefore you will need to be comfortable with JSP. Also, you will need to be comfortable with the WebLogic Personalization Server custom JSP tags and provided classes such as `PortalJSPBase`.

For more information, see Chapter 10, "JSP Tag Library Reference."

Building Your Site Without the Portal Framework

You may not want a portal metaphor when creating your site. Or, you may find that your site design would require extensive changes to the Portal Framework. In both cases, you may choose to build your site from scratch. In this case, you have the full power of JSP and HTML at your disposal, as well as the commerce components (shopping cart management, taxation, payment) and the personalization components (rules, property sets, content management, user and group management) via the JSP tag libraries.

For more information, see Chapter 10, "JSP Tag Library Reference."

Framework Files

The following table displays the names and functions of the template JSP files provided with the WebLogic Personalization Server framework. Each of these files is located in the root directory of the portal which it serves, such as `/portals/repository`.

Table 6-1 Framework Templates

JSP File Name	Function
`_user_add_portlets.jsp`	The tool employed by the end user to add/remove portlets.
`_user_layout.jsp`	The tool employed by the end user to update portlet layout.
`_userlogin.jsp`	The user login page.
`_userreg.jsp`	The new user registration page.
`_userreg_summary.jsp`	The user profile summary page.
`alternatefooter.jsp`	The footer displayed when a portlet is maximized or detached.
`alternateheader.jsp`	The header displayed when a portlet is maximized or detached.

Table 6-1 Framework Templates

JSP File Name	Function
baseheader.jsp	A stripped version header.jsp, intended for general use beyond the portal home page.
color_picker.jsp	The color palette employed by the user color preferences tool.
error.jsp	A general-purpose page used for displaying run-time errors.
error_footer.jsp	The footer displayed with error.jsp.
error_header.jsp	The header displayed with error.jsp.
footer.jsp	The footer displayed with the main portal page.
fullscreenportlet.jsp	The page used to display a maximized or detached portlet.
gen_prefs.jsp	The tool employed by the end user to update general user profile information.
header.jsp	The header displayed with the main portal page.
help.jsp	The end user help page.
layout_script.jsp	The JavaScript used by the end user layout tool.
portal.jsp	The main portal page.
portalcontent.jsp	The page which prescribes portlet layout within the main portal page.
portalerror.jsp	The default error page displayed when an access attempt to a portal page fails.
portalnotexist.jsp	The page which displays a general message indicated that the requested portal does not exist.
portlet.jsp	The page which constructs a portlet, combining portlet title bar, banner, header, content, and footer.
privacy_policy.jsp	A placeholder for a company privacy policy statement.
status.jsp	The page used to display end user status messages.
suspended.jsp	The page which provides a message indicating that the requested portal is currently non-operational, typically for maintenance reasons.

Table 6-1 Framework Templates

JSP File Name	Function
`titlebar.jsp`	The portlet title bar. Contains appropriate portlet icons and portlet name.
`user_colors.jsp`	The end user color preferences tool.

7 Using the Catalog Application in a Portal

This chapter describes how to deploy your portal as a Web application, add e-commerce features such as site security and user authorization, use Webflow within a portal,and use pieces of the demo catalog application in a portal.

This topic includes the following sections:

- Deploying a Portal as a Web Application
- Using E-Commerce Functionality Within a Portal
- Using Webflow Within a Portal
- Reusing Pieces of the Demo Catalog Application in a Portal

Note: Throughout this chapter, the environment variable WL_COMMERCE_HOME is used to indicate the directory in which you installed the WebLogic Commerce Server 3.2 and WebLogic Personalization Server 3.2 software.

Deploying a Portal as a Web Application

To build a portal Web application using the Portal Framework, follow these steps:

1. In the %WL_COMMERCE_HOME%/server/webapps folder, create a new folder. Give it the name of your Web application. For this discussion, this new folder will be referred to as "Example" in the webapps folder.

2. Create a subfolder in your new webapps/Example folder called WEB-INF. In this folder put your Web application's web.xml file.

   Note: If you do not already have a web.xml file, you may copy the one in webapps/admin/WEB-INF. You must remove the security-constraint and security-role elements from the document.

   If you are using tag libraries, copy the tag library descriptor files (*.tld) into the WEB-INF directory. Also copy the portal framework .tld files from portal/WEB-INF/*.tld.

3. Create a new file in your Example folder called index.jsp. In this file, put a JavaServer Page (JSP) forward to your portal home page:

   ```
   <jsp:forward page="/application/Example" />
   ```

   where Example should be replaced by the name of your property set.

4. Open your weblogic.properties file found in %WL_COMMERCE_HOME% for editing. Search for the word "webapp." You will find a line beginning as follows:

   ```
   weblogic.httpd.webApp.wlcs=
   ```

   Copy this line and paste the copy below the original. Change the word "wlcs" to be the name of your Web application. There are two occurences of "wlcs", you must change both.

5. Start your WebLogic Commerce Server by executing StartCommerce.bat found in %WL_COMMERCE_HOME%.

6. Log on to the administration tool. Navigate to the Property Set Manager.

7. Create a new property set. Give it the same name as your Web application. Copy properties from `APPLICATION_INIT._DEFAULT_PORTAL_INIT`. The property set type is "Application Init." Click Create and then Back.

   Click on the property set name to edit it. You will need to change the following properties:

   ```
   defaultdest = your portal's main page
   homepage = your portal's main page
   workingdir = /
   PORTAL_NAME = Example
   TTL = set low during development, then restore when deploying for
   service.
   ```

8. After you finish your changes, you must wait up to 5 minutes for the server to recognize the new changes. This time period is dictated by the `TTL` (time to live) property. Alternatively, you may eliminate this wait time by adding the parameter `flowReset=true` to any URL for your Web application, which will cause the server to reload the property set.

9. Click Home, and then navigate to the Portal Manager.

10. On the Portal title bar, click the Create button. This displays a form that allows you to create a new portal.

11. For the portal name, enter the exact same text as you did for the `PORTAL_NAME` property in your property set. The rest of the form fields can be left as the default. Click Create to create your portal.

12. Test your site by opening a browser window and navigate to `http://localhost:7501/Example` where "Example" must match the name you gave to your Web application in the weblogic.properties file in step 4.

You have successfully deployed your portal as a Web application.

Using E-Commerce Functionality Within a Portal

This section discusses the scenerio in which you want to have a portal as the entry point for your users, but also wish to build e-commerce functionality into your application. An example of this case might be a portal with one portlet showing the user's shopping cart, another portlet allowing a search of the product catalog, and a third portlet showing the user's order history.

An immediate requirement of this type of site is that it be deployed as a Web application. This is a requirement of the Commerce components, and therefore this requirement also applies to a portal that includes Commerce components. Before you can begin adding e-commerce features to your portal, you must deploy your portal as a Web application. This involves a number of steps, including the creation of a `web.xml` file. For more information, see the previous section "Deploying a Portal as a Web Application" on page 7-2.

Once your portal is a Web application, a major issue to consider is site security. The Portal Framework includes its own security model. Portal page security constraints are defined by personalization rules, the Portal Administration Tools, and the Portal end user personalization tools. The declaritive security model of J2EE is much too simplistic for a portal deployment. For instance, when a user who belongs to more than one group logs onto a portal, the portal needs to be able to query the user on which group to use for personalization. This query action cannot be implemented when declaritive security is employed. Therefore, when working with portals, you should avoid creating security constraints in the Web application's `web.xml` file.

Another security issue is user authentication. When implementing your site, you will want single sign-on capabilities. To do this, the Portal Framework and Commerce features will share the login. To this end, the Portal Framework must perform the login request. Additionally, the Commerce features require the user's unified user type to be WLCS_Customer.

Finally, there are several configuration requirements when assembling your portal. This includes deploying your site as a Web application, and also the use of a special destination determiner.

The purpose of this section is to guide you step-by-step through the issues discussed previously. As you proceed through the steps, a brief explanation is given. For a more thorough explanation of an individual step, consult the relevent documentation.

Follow these steps before employing Commerce features in your portal:

1. **Deploy your portal as a Web application.** This process is described in the previous section.
2. **Construct a single unified Application_Init property set.** You must use the `_DEFAULT_PORTAL_INIT` template for your application initialization property set.
3. **Establish a single destination determiner.** The destination determiner to use is `com.beasys.commerce.webflow.WLCSPortalDestinationDeterminer`. It is capable of handling both WebLogic Personalization Server and WebLogic Commerce Server requests.
4. **Establish a single destination handler**. Establish this destination handler: `com.beasys.commerce.foundation.flow.ServletDestinationHandler`
5. **Configure your portal main page as the application home page.** The destination determiner is programmed to return the user to the home page if no destination information is included in the HTTP request.
6. **Use _userlogin.jsp for User Authentication.** The login page included in the Portal Framework is `_userlogin.jsp`. In order to satisfy the portal's security model, all logins must go through this JSP. This page performs the necessary method calls (`setUser`, `setLoggedIn`, and `setSucessor`) required by the portal security mechanism.
7. **Ensure that the user has a unified profile type of WLCS_Customer.** The WebLogic Commerce Server functionality requires an extension to the basic user type. This extension is named `WLCS_Customer`, and is a unified profile type. Any user of your portal with Commerce features must have a type of `WLCS_Customer`.
8. **Avoid declaritive security in your site.** To reduce the complexity of the portal security model, try to avoid declaritive security in your site. Declaritive security appears as security constraints in your Web application's `web.xml` file. This is not a necessary step, but it makes security management easier.

Using Webflow Within a Portal

In WebLogic Commerce Server, Webflow is a feature that allows you to string together JavaServer Page (JSP) files, input processors (IPs) and pipeline processors (PPs) without hard coding the linkage between them. Instead, the linkage is defined in an external Webflow properties file.

Something to consider is how Webflow works with portals. When using Webflow, your users are conducted through a number of complete JSP pages as they work through a process. The portal, on the other hand, generally keeps the user on a single portal page, while the contents of that page (the portlets) change state. Due to this difference, in the current implementation, you cannot use the Webflow feature of page transitions while you employ the Portal Framework. But you can utilize the power of input processors and pipeline processors from within a portlet. This section details how to do this.

The first requirement of Webflow is that your site must be deployed as a Web application. The first step of this process is to create and deploy your portal as a Web application, as described in the section "Using E-Commerce Functionality Within a Portal" on page 7-4.

Once you have deployed your portal as a Web application, follow these steps:

1. Start your server and log on to the administration client.
2. Navigate to the Property Set Manager. Click on your property set to edit it.
3. You need to use a special destination determiner. Edit your `destinationdeterminer` property, and set it to be: `com.beasys.commerce.Webflow.WLCSPortalDestinationDeterminer`.
4. In a text editor, open or create your Webflow.properties located in your `%WL_COMMERCE_HOME%` folder.

   In this folder, you need to create transitions from your portlet to input processors and pipeline processors. It is important to understand that the destination determiner will route the flow from your portlet to the input processors and pipeline processors. Once Webflow has traversed through the IPs and PPs, it will forward the request to the URL that you specify in the Webflow method call in your portlet JSP (this will be discussed later). An example Webflow property file is as follows:

```
destinmyportlet.jsp.link(mylink) = myportlet.inputprocessor
myportlet.inputprocessor.success = myportlet.pipeline
myportlet.pipeline.success = myportlet2.pipeline
```

In this example, once Webflow has traversed through the second pipeline processor, it will allow the default destination determiner to forward to the URL specified in the request.

In WebLogic Commerce Server, you may have only one Web application that is Webflow enabled. You must identify the property set to be used by Webflow. Do this by adding the following section to your `web.xml` file located in your `web-inf` folder:

```
<context-param>
   <param-name>WLCS_APPLICATION_URL</param-name>
   <param-value>/application/commercewf</param-value>
</context-param>
```

Replace `commercewf` with the name of your property set.

Finally, you need to connect to Webflow from your portlet. In your portlet JSP, you must make a call to createWebflowURL. In this call, you must specify two parameters. Specify a parameter called `portalized` as `true`, and a parameter called `dest` with the URL that Webflow should go to after it has finished. For example:

myportlet.jsp

```
<%@ page import="com.beasys.commerce.webflow.*" %>
<%@ page
extends="com.beasys.commerce.portal.admin.PortalJspBase"%>

<form method=”POST”
        action="<%= WebflowJSPHelper.createWebflowURL(
           pageContext,
           "portlet.jsp",
           "link(mylink)",
           “&portalized=true&dest=/portal.jsp”,
           true) %>">
...
</form>
```

In this example, when the user clicks the Submit button the request will be forwarded to the destination of the myportlet.jsp.link(mylink) transition. Once Webflow has finished with all input processors and pipeline processors it has found there, it will forward to `portal.jsp`. Your `dest` parameter should refer to your portal page and not your portlet.

5. Test your portal.

If you successfully completed all the steps in this exercise, you should now have Webflow working within your portal.

Reusing Pieces of the Demo Catalog Application in a Portal

If you start your commerce server and use a browser to navigate to `http://localhost:7501/application/wlcs`, you can experiment with the WebLogic Commerce Server sample catalog application. There are some useful pieces of functionality in this sample application that you may want to build into your portal site as a portlet. This section explains how to do this.

First, note that the WebLogic Commerce Server catalog application relies on Webflow. Therefore, you must integrate Webflow with your portal. See "Using Webflow Within a Portal" on page 7-6. Also, the Catalog application uses Commerce features, so you must prepare your portal as described in the section "Using E-Commerce Functionality Within a Portal" on page 7-4.

After you have followed those directions, complete the following steps.

1. Identify the JSP file in the catalog project that provides the functionality you wish to have in your application. There is no better way to do this than to open JSP files in the `%WL_COMMERCE_HOME%/server/webapps/wlcs` folder and track the code down. Copy this file into your application folder.

2. Remove from this file any functionality you do not want in your portlet. This includes both links to the WebLogic Commerce Server catalog site and also page-scoped items such as the WebLogic Commerce Server catalog header and footers.

   Make your portlet extend PortalJspBase by adding the following line to the top of the JSP file:

   ```
   <%@ page
   extends="com.beasys.commerce.portal.admin.PortalJspBase"%>
   ```

3. Identify the transitions in the default `webflow.properties` file that correspond to your portlet JSP. Update as necessary.

4. Update the Webflow calls in your JSP (as explained in the previous section).
5. Register your portlet JSP with the server through the administration tool.
6. Add your new portlet to your portal.
7. Test your new portlet.

8 Localizing Applications with the Internationalization Tags

This topic includes the following sections:

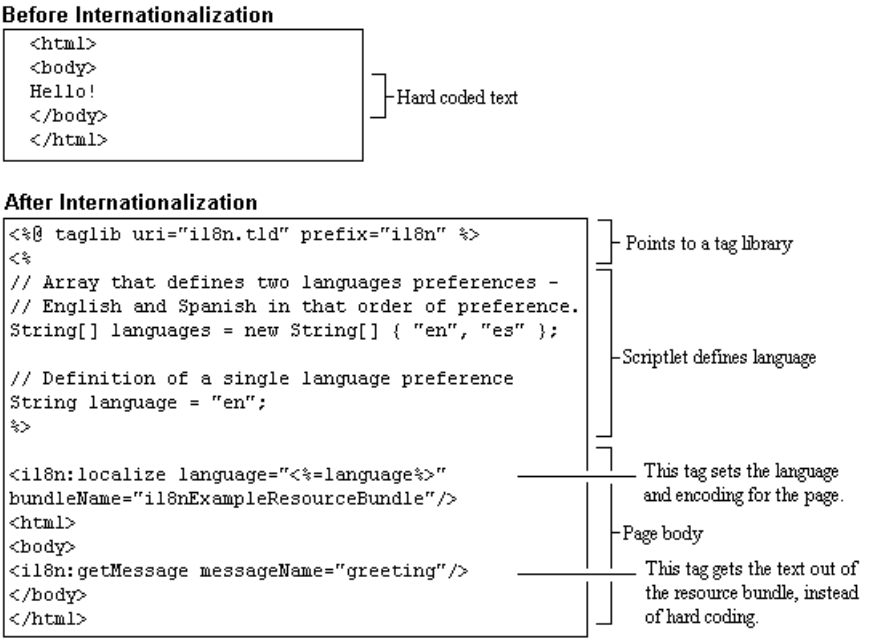
- What Is the I18N Framework?
- Localizing Your JSP
  - <i18n:getMessage>
  - <i18n:localize>
  - Character Encoding
  - Steps for Localizing Your Application
  - Code Examples
- Localizing the BEA WebLogic Personalization Server
  - Static Text
  - Constructed Messages
  - Resource Bundles Used in the WebLogic Personalization Server Tools

What Is the I18N Framework?

WebLogic Personalization Server provides a simple framework that allows access to localized text labels and messages. The internationalization (I18N) framework is accessible from JavaServer Pages (JSPs) through a small I18N tag library. An example is shown in Figure 8-1. The JSP extension tag library provides the following services:

1. Retrieves a static text label from a resource bundle (implemented as a properties file).
2. Retrieves a message from a resource bundle (implemented as a properties file).
3. Initializes a page context with a particular language, country, and variant for label and message retrieval throughout a page.
4. Properly sets the content type (text/html) and character encoding for a page.

Figure 8-1 An Example of Internationalization Code



Localizing Your JSP

The conventions used in the I18N tag library are based on the more general conventions used to internationalize Java applications. To understand the conceptual foundations for the `<i18n:getMessage>` tag, see the *Javadoc* for `java.text.MessageFormat` in the Sun Microsystem, Inc. *Java 2 SDK, Standard Edition* documentation. To better understand the ideas that served as the foundation for these tags, study the *Javadoc* for `java.util.ResourceBundle` and `java.util.Locale`.

The following tags are included in the I18N framework:

```
<i18n:getMessage>
<i18n:localize>
```

<i18n:getMessage>

This tag retrieves a localized label or message (based on the absence/presence of an `args` attribute). The tag optionally takes a bundle name, language, country, and variant to aid in locating the appropriate properties file for resource bundle loading.

This tag is used in the localization of JSP pages. All pages that have an internationalization requirement should use this tag.

For more information about the `<i18n:getMessage>` tag, see Chapter 10, "JSP Tag Library Reference."

<i18n:localize>

This tag allows you to specify a language, country, variant, and resource bundle name to use throughout a page when accessing resource bundles via the `<i18n:getMessage>` tag. This is a convenient way to specify these attributes once, so that you do not have to specify them again each time you use `<i18n:getMessage>` to retrieve localized static text or messages.

Note: Changes to the resource bundles will not be recognized until the server is restarted.

The <i18n:localize> tag also specifies a character encoding and content type to be specified for a JSP page. Because of this, the tag should be used as early in the page as possible—before anything is written to the output stream—so that the bytes are properly encoded. If you intend to display text in more than one language, pick a character set that encompasses all the languages on the page.

When an HTML page is included in a larger page (for example, as portlets are included in portal pages), only the larger page can use the <i18n:localize> tag. This is because the <i18n:localize> tag sets the encoding for the page, and the encoding must be set in the parent (including) page before any bytes are written to the response's output stream. Therefore, be careful that the encoding for the parent page is sufficient for all the content on that page as well as any included pages. The child (included) pages may continue to use the <i18n:getMessage> tag.

Note: Do not use the <i18n:localize> tag in conjunction with the <%@ page contentType="<something>" > page directive defined in the JSP specification. The directive is unnecessary if you are using this tag, and can result in inconsistent or wrong contentType declarations.

For more information about the <i18n:localize> tag, see Chapter 10, "JSP Tag Library Reference."

The JspMessageBundle

The <i18n:getMessage> tag uses the com.beasys.commerce.i18n.jsp.JspMessageBundle class. Unlike a ResourceBundle, the JspMessageBundle looks only for properties files (like the PropertyResourceBundle) within the ServletContext (on the doc path). This means that you can keep MessageBundle properties files relative to the associated JSP page, instead of having to have them on the CLASSPATH.

Another difference is that JspMessageBundles are specified using the "/" character instead of the ".". For instance, the path to a JspMessageBundle might look like this: /jsp/ordersystem/placeOrder.

If a bundle name is specified, then it can be specified *absolutely* or *relatively*. Absolute paths are treated as such if they begin with a "/". Paths not beginning with "/" are searched for relative to the JSP page's location.

If no bundle name is specified, then bundle name defaults to the name of the JSP page. For instance, if you have a JSP page called placeOrder.jsp, then JspMessageBundle would look in the same directory for a placeOrder.properties file to serve as the JspMessageBundle for the placeOrder.jsp page.

When searching for a JspMessageBundle, both the doc root and repository directories are searched, in that order. Repository directories are directories specified during servlet registration and serve as a place to store common files such as images. If no message bundle can be found, a MissingResourceException occurs. For a more in-depth description of the repository directory convention, see “Repository” on page 3-11.

How the Localization Tag Works

The `<i18n:localize>` tag first examines all provided attributes and default attributes**,** and then performs the following three steps:

1. **Determines the base bundle name.**

   If a base bundle name is not provided, the bundle name defaults to the name of the JSP page with .properties appended.

   For example, if the name of the JSP page is placeOrder.jsp, then the default bundle name would be placeOrder.properties.

2. **Determines the language to use.**

   The tag will first look for resource bundles that correspond to the language parameter passed in to the tag.

   If no match between bundle and language is found, then the tag will try to find a match between resource bundles and languages defined in the request header.

   If a match can be made, the first language that matches is the language that is used.

   If no language is specified, the default is US English (en_US).

   If no message bundle can be found, then language is set to nothing ("") and "UTF-8" encoding will be used unless otherwise specified.

3. **Determines which character encoding (charset) to use.**

   If character encoding is not specified, a charset appropriate for the language determined in step 2 is chosen.

If a character encoding is specified, then that will be the charset used by the page, regardless of what language was chosen in step 2.

Once the charset is determined, it is specified for the page by calling the `setContentType()` method on the servlet response. A call to `setContentType()` might look like this:

```
response.setContentType("text/html; charset=ISO-8859-1");
```

Character Encoding

When specifying the encoding, it is important to note that some encodings may not be supported for your particular operating system, virtual machine, or client browsers. To see what Sun Microsystem, Inc. supports in the J2SE package, see http://www.java.sun.com/products/jdk/1.2/docs/guide/internat/encoding.doc.html

If for any reason an encoding for a language cannot be determined and none is specified, UTF-8 encoding is used.

Displaying More than One Character Set on a Page

In general, it is best is to leave the charset parameters unspecified since this is more flexible and fault tolerant. An exception might be when two languages (such as Greek and Japanese) need to be displayed in the same page. In that case, you can set the charset to "UTF-8".

For a page with multiple charsets to display correctly, the end users must have the appropriate fonts installed on their machines. If a font cannot be found, non-printable characters will typically display in place of the missing characters. (Non-printable characters often look like rows of empty boxes.)

Default Character Encodings

Figure 8-1 shows how the `<i18n:localize>` tag maps languages to character encodings. These are the default settings.

You can override these defaults by providing any charset tag parameter you choose. For example, in the table below, the default charset for Japanese is Shift_JIS, but you could pass in x-sjis, EUC_JP, or iso-2022-jp instead. Or, as another example, to use Chinese Taiwan locale in place of Chinese, override GB2312 with Big5.

Table 8-1 Default Character Encodings

Language Code	Language Name	Character Encoding
ar	Arabic	ISO-8859-6
be	Byelorussian	ISO-8859-5
bg	Bulgarian	ISO-8859-5
ca	Catalan	ISO-8859-1
cs	Czech	ISO-8859-2
da	Danish	ISO-8859-1
de	German	ISO-8859-1
el	Greek	ISO-8859-7
en	English	ISO-8859-1
es	Spanish	ISO-8859-1
et	Estonian	ISO-8859-1
fi	Finnish	ISO-8859-1
fr	French	ISO-8859-1
hr	Croatian	ISO-8859-2
hu	Hungarian	ISO-8859-2
is	Icelandic	ISO-8859-1

it	Italian	ISO-8859-1
iw	Hebrew	ISO-8859-8
ja	Japanese	Shift_JIS
ko	Korean	EUC_KR
lt	Lithuanian	ISO-8859-2
lv	Latvian (Lettish)	ISO-8859-2
mk	Macedonian	ISO-8859-5
nl	Dutch	ISO-8859-1
no	Norweigan	ISO-8859-1
pl	Polish	ISO-8859-2
pt	Portuguese	ISO-8859-1
ro	Romanian	ISO-8859-2
ru	Russian	ISO-8859-5
sh	Serbo-Croatian	ISO-8859-5
sk	Slovak	ISO-8859-2
sl	Slovenian	ISO-8859-2
sq	Albanian	ISO-8859-2
sr	Serbian	ISO-8859-5
sv	Swedish	ISO-8859-1
th	Thai	TIS620
tr	Turkish	ISO-8859-9
uk	Ukrainian	ISO-8859-5
zh	Chinese	GB2312
other		UTF-8

Steps for Localizing Your Application

1. Familiarize yourself with the documentation for the Internationalization <i18n:*> tags in the Chapter 10, "JSP Tag Library Reference.". For sample code, see Figure 8-1 "An Example of Internationalization Code" on page 8-2.

2. Include the <i18n:localize> tag in all pages with an internationalization requirement. The tag should be used as early in the page as possible—before anything is written to the output stream—so that the bytes are properly encoded.

   For example: <%@ taglib uri="i18n.tld" prefix="i18n" %>

   For example: <i18n:localize language="<%=language%>"

Note: When HTML pages are being included inside a larger page, only the larger page can use the <i18n:localize> tag.

3. Move all text that must be localized (including image URLs that must be localized) to property files that serve as resource bundles. Provide a resource bundle (property file) for each language you plan to support. One resource bundle per JSP page per language is the recommended approach.

Note: Changes to the property files will not be recognized until the server is restarted.

   For example: Use <i18n:getMessaage messageName="greeting"/> instead of hardcoding "Welcome!"

4. Specify a directory path for the property files (resource bundles). The bundle location must be specified *relative* to the JSP location, or *absolutely*, under the document root.

5. Refer to all localized text in a JSP page by using the <i18n:getMessage> tag. Make sure the <i18n:getMessage> tag is referring to the correct resource bundle location (relative or absolute path).

   For example:
   If the JSP is in public_html\mypage.jsp, then the bundle location could be (absolute) "/mypage/text_us.properties" or
   (relative) "text_us.properties".

6. Test the page for all languages that you support. Make sure that the localized text and images display correctly and that the page layout is correct.

Code Examples

The following examples show how to use the JSP internationalization framework with JavaScript and Java scriptlets.

Using the JSP Internationalization Framework with JavaScript

This example displays a JavaScript dialog with a localized message in it.

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%
String language="en";
%>
<i18n:localize language="<%=language%>"
bundleName="i18nJavaScriptExampleResourceBundle"/>

<script language="JavaScript">
function popDialog() {
alert("<i18n:getMessage messageName="greeting"/>")
}
</script>

<html>
<body>
<a href="javascript:popDialog();">Click here to see localized
text!</a>
</body>
</html>
```

Using the JSP Internationalization Framework with Java Scriptlets

This example gets a localized message, and uses that message in two Java scriptlets. One scriptlet prints to system out, the other inlines it into the page.

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%
String language="en";
%>
<i18n:localize language="<%=language%>"
bundleName="i18nJavaScriptExampleResourceBundle"/>

<html>
<body>
<i18n:getMessage messageName="greeting" id="theGreeting"/>
<p>
```

```
<%="Localized text for 'greeting': " + theGreeting%>
<p>
<%
System.out.println("Localized text for 'greeting': " +
theGreeting);
%>

</body>
</html>
```

Localizing the BEA WebLogic Personalization Server

Up to this point, this chapter has discussed localizing the application that you are building with the BEA WebLogic Personalization Server.

In developing your application, you may be required to localize some of the portal tools in the WebLogic Personalization Server. This section provides information for developers who need to localize the administration tools that are provided with this product, or who are deriving their application from examples that ship with the WebLogic Personalization Server.

The WebLogic Personalization Server Administration Tool is supported by JSP bean objects which employ Java internationalization conventions in the practice of presenting error and status messages. These beans use a BEA utility object called `com.beasys.commerce.i18n.MessageBundle` in conjunction with text-based properties files to produce two types of locale-specific display text. The two types of text are as follows:

- Static Text
- Constructed Messages

Static Text

WebLogic Personalization Server uses the following convention when naming static text entries in the properties files:

```
propertyName.txt=propertyValue
```

For example: `error.txt=Error Occurred`

A static text property is acquired from a loaded MessageBundle using the following method:

```
public String getString(String propertyName)
```

For example:

```
System.out.printin(messageBundle.getString("error.txt"));
```

For more information, see the *Javadoc* for the Portal API documentation.

Constructed Messages

The localized display text generated at run time often depends on one or more variables, and the order of these variables in a text segment is locale-specific. In this case, the WebLogic Personalization Server provides a means for constructing message segments for display.

WebLogic Personalization Server uses the following convention when naming message entries in properties files:

```
propertyName.msg=propertyValue
```

For example:

```
fieldRequired.msg={0} is a required field.
```

A constructed message is acquired from a loaded MessageBundle using the following method:

```
public String getMessage(Object[] args, String propertyName)
```

For example:

```
Object[] args = new Object[] {“ContentURL”};
```

```
System.out.println(messageBundle.getMessage(args,
"fieldRequired.msg"));
```

For more information, see the *Javadoc* for the Portal API documentation..

Note: The MessageBundle's `getMessage()` method internally uses a `java.text.MessageFormat` object. To understand how the `getMessage()` method works, look at the *Javadoc* for `java.text.MessageFormat`.

Resource Bundles Used in the WebLogic Personalization Server Tools

Each properties file that supports a particular bean includes the bean name and a property extension. For example, the property file that supports the `com.beasys.portal.admin.jspbeans.PortalJspBean` bean resides in the `i18n` directory beneath `com/beasys/portal/admin/jspbeans`, and is called `PortalJspBean.properties`.

Localizing System Messages

You can localize the resource bundles that contain system messages related to the WebLogic Personalization Server Administration Tools and sample applications. Changes to the resource bundles will be recognized when the server is restarted.

Use the following properties files to localize system messages. These property files are found under `<WL_COMMERCE_HOME>/classes`:

`com/beasys/commerce/axiom/util.i18n/JSPBeanBase.properties`

`com/beasys/commerce/user/jsp/beans/i18n/LDAPConfigBean.properties`

`com/beasys/commerce/user/jsp/beans/i18n/ProfileTypeBean.properties`

`com/beasys/commerce/user/jsp/beans/i18n/PropertyBean.properties`

`com/beasys/commerce/user/jsp/beans/i18n/PropertySetBean.properties`

`com/beasys/commerce/user/jsp/beans/i18n/RealmConfigBean.properties`

`com/beasys/commerce/user/jsp/beans/i18n/UserBean.properties`

`com/beasys/commerce/portal/admin/jspbeans/i18n/PortalJspBean.properties`

```
com/beasys/commerce/portal/admin/jspbeans/i18n/PortletJspBean.properties
com/beasys/commerce/portal/admin/jspbeans/i18n/PortalPersonalization.properties
com/beasys/commerce/portal/admin/jspbeans/i18n/PortalRemoveJspBean.properties
com/beasys/commerce/portal.jspbeans/i18n/PortalAppearanceBean.properties
com/beasys/commerce/axiom.util/i18n/JspBeanBase.properties
```

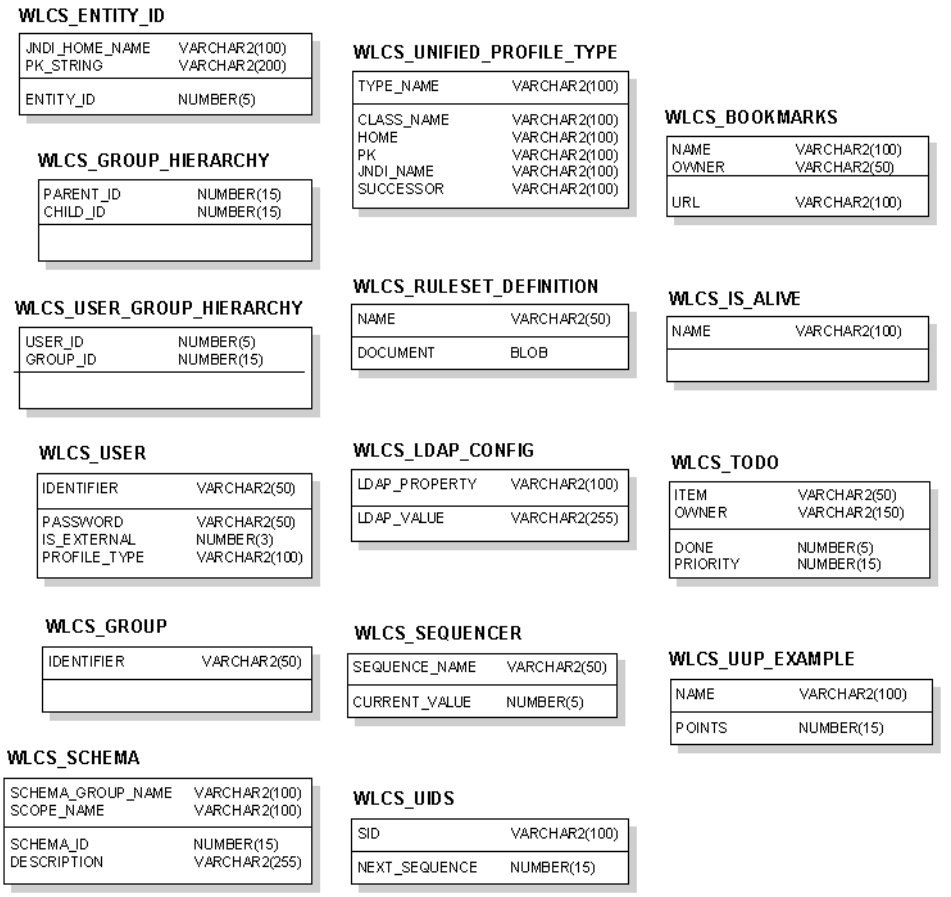
9 WebLogic Personalization Server Schema

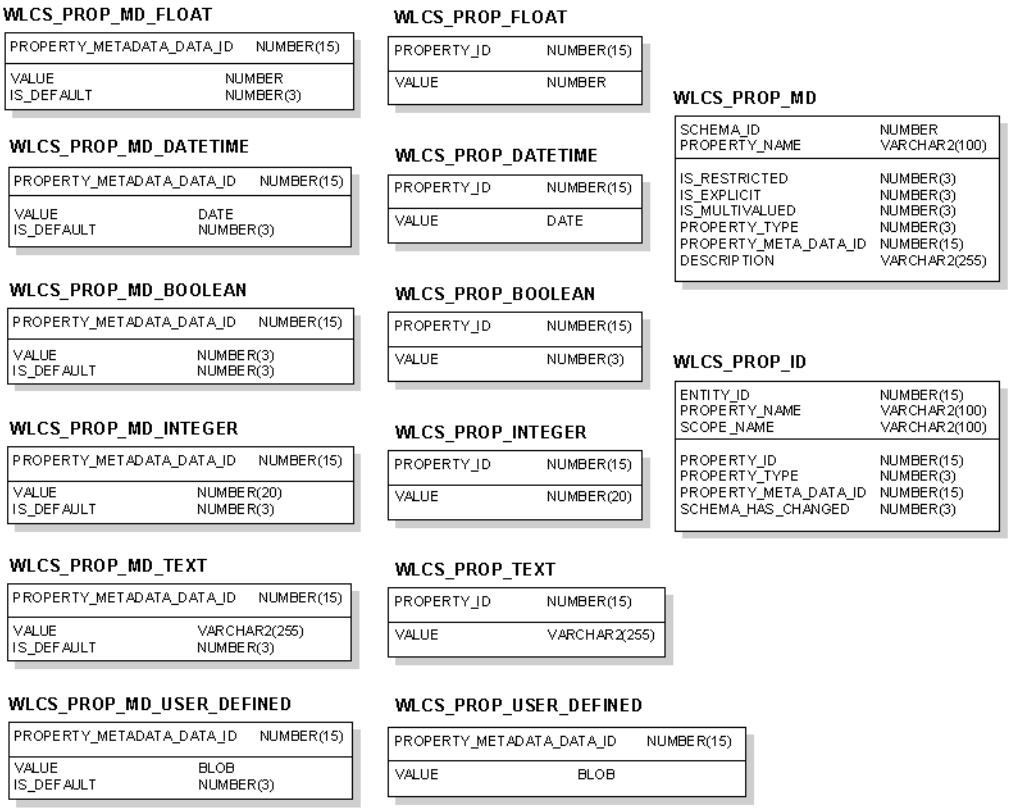
This chapter documents the database schema for the WebLogic Personalization Server and includes the following sections:

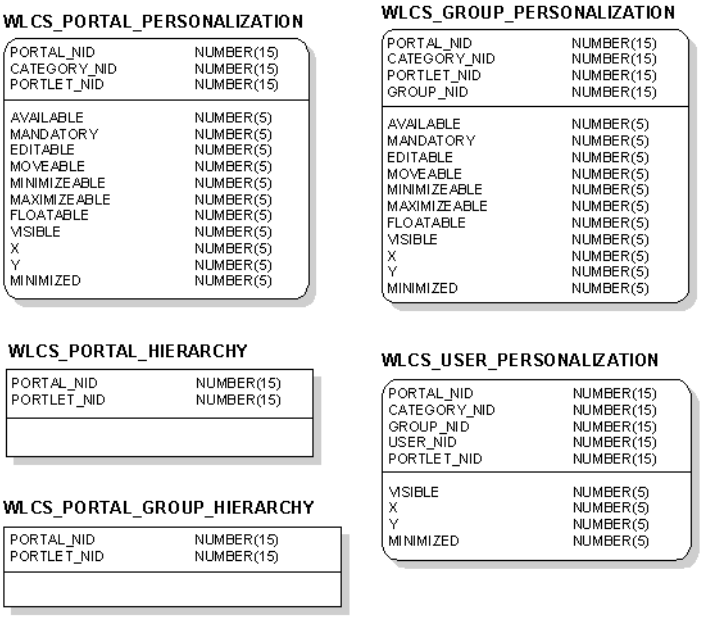
- The Entity-Relationship Diagram
  - The Schema Tables
- The Tables Comprising the WebLogic Personalization Server
- The SQL Scripts Used to Create the Database

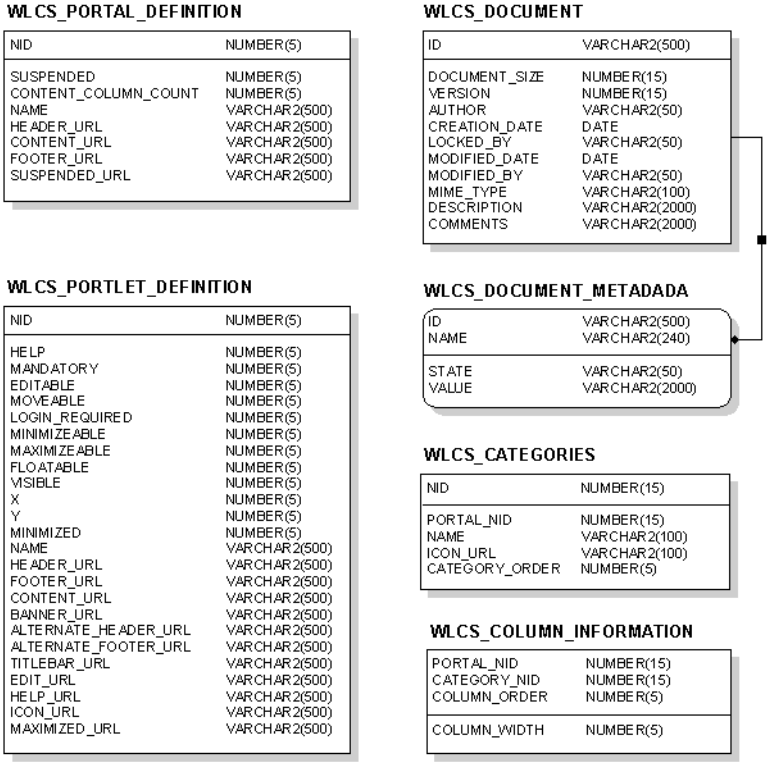
The Entity-Relationship Diagram

The following figures comprise the Entity-Relationship Diagram (ERD) for the WebLogic Personalization Server database.


WLCS_ENTITY_ID
JNDI_HOME_NAME VARCHAR2(100)
PK_STRING VARCHAR2(200)
ENTITY_ID NUMBER(5)
WLCS_UNIFIED_PROFILE_TYPE
TYPE_NAME VARCHAR2(100)
CLASS_NAME VARCHAR2(100)
HOME VARCHAR2(100)
PK VARCHAR2(100)
JNDI_NAME VARCHAR2(100)
SUCCESSOR VARCHAR2(100)
WLCS_BOOKMARKS
NAME VARCHAR2(100)
OWNER VARCHAR2(50)
URL VARCHAR2(100)
WLCS_GROUP_HIERARCHY
PARENT_ID NUMBER(15)
CHILD_ID NUMBER(15)
WLCS_RULESET_DEFINITION
NAME VARCHAR2(50)
DOCUMENT BLOB
WLCS_IS_ALIVE
NAME VARCHAR2(100)
WLCS_USER_GROUP_HIERARCHY
USER_ID NUMBER(5)
GROUP_ID NUMBER(15)
WLCS_USER
IDENTIFIER VARCHAR2(50)
PASSWORD VARCHAR2(50)
IS_EXTERNAL NUMBER(3)
PROFILE_TYPE VARCHAR2(100)
WLCS_LDAP_CONFIG
LDAP_PROPERTY VARCHAR2(100)
LDAP_VALUE VARCHAR2(255)
WLCS_TODO
ITEM VARCHAR2(50)
OWNER VARCHAR2(150)
DONE NUMBER(5)
PRIORITY NUMBER(15)
WLCS_GROUP
IDENTIFIER VARCHAR2(50)
WLCS_SEQUENCER
SEQUENCE_NAME VARCHAR2(50)
CURRENT_VALUE NUMBER(5)
WLCS_UUP_EXAMPLE
NAME VARCHAR2(100)
POINTS NUMBER(15)
WLCS_SCHEMA
SCHEMA_GROUP_NAME VARCHAR2(100)
SCOPE_NAME VARCHAR2(100)
SCHEMA_ID NUMBER(15)
DESCRIPTION VARCHAR2(255)
WLCS_UIDS
SID VARCHAR2(100)
NEXT_SEQUENCE NUMBER(15)


WLCS_PROP_MD_FLOAT
PROPERTY_METADATA_DATA_ID NUMBER(15)
VALUE NUMBER
IS_DEFAULT NUMBER(3)
WLCS_PROP_FLOAT
PROPERTY_ID NUMBER(15)
VALUE NUMBER
WLCS_PROP_MD
SCHEMA_ID NUMBER
PROPERTY_NAME VARCHAR2(100)
IS_RESTRICTED NUMBER(3)
IS_EXPLICIT NUMBER(3)
IS_MULTIVALUED NUMBER(3)
PROPERTY_TYPE NUMBER(3)
PROPERTY_META_DATA_ID NUMBER(15)
DESCRIPTION VARCHAR2(255)
WLCS_PROP_MD_DATETIME
PROPERTY_METADATA_DATA_ID NUMBER(15)
VALUE DATE
IS_DEFAULT NUMBER(3)
WLCS_PROP_DATETIME
PROPERTY_ID NUMBER(15)
VALUE DATE
WLCS_PROP_MD_BOOLEAN
PROPERTY_METADATA_DATA_ID NUMBER(15)
VALUE NUMBER(3)
IS_DEFAULT NUMBER(3)
WLCS_PROP_BOOLEAN
PROPERTY_ID NUMBER(15)
VALUE NUMBER(3)
WLCS_PROP_ID
ENTITY_ID NUMBER(15)
PROPERTY_NAME VARCHAR2(100)
SCOPE_NAME VARCHAR2(100)
PROPERTY_ID NUMBER(15)
PROPERTY_TYPE NUMBER(3)
PROPERTY_META_DATA_ID NUMBER(15)
SCHEMA_HAS_CHANGED NUMBER(3)
WLCS_PROP_MD_INTEGER
PROPERTY_METADATA_DATA_ID NUMBER(15)
VALUE NUMBER(20)
IS_DEFAULT NUMBER(3)
WLCS_PROP_INTEGER
PROPERTY_ID NUMBER(15)
VALUE NUMBER(20)
WLCS_PROP_MD_TEXT
PROPERTY_METADATA_DATA_ID NUMBER(15)
VALUE VARCHAR2(255)
IS_DEFAULT NUMBER(3)
WLCS_PROP_TEXT
PROPERTY_ID NUMBER(15)
VALUE VARCHAR2(255)
WLCS_PROP_MD_USER_DEFINED
PROPERTY_METADATA_DATA_ID NUMBER(15)
VALUE BLOB
IS_DEFAULT NUMBER(3)
WLCS_PROP_USER_DEFINED
PROPERTY_METADATA_DATA_ID NUMBER(15)
VALUE BLOB


WLCS_PORTAL_PERSONALIZATION
PORTAL_NID NUMBER(15)
CATEGORY_NID NUMBER(15)
PORTLET_NID NUMBER(15)
AVAILABLE NUMBER(5)
MANDATORY NUMBER(5)
EDITABLE NUMBER(5)
MOVEABLE NUMBER(5)
MINIMIZEABLE NUMBER(5)
MAXIMIZEABLE NUMBER(5)
FLOATABLE NUMBER(5)
VISIBLE NUMBER(5)
X NUMBER(5)
Y NUMBER(5)
MINIMIZED NUMBER(5)
WLCS_GROUP_PERSONALIZATION
PORTAL_NID NUMBER(15)
CATEGORY_NID NUMBER(15)
PORTLET_NID NUMBER(15)
GROUP_NID NUMBER(15)
AVAILABLE NUMBER(5)
MANDATORY NUMBER(5)
EDITABLE NUMBER(5)
MOVEABLE NUMBER(5)
MINIMIZEABLE NUMBER(5)
MAXIMIZEABLE NUMBER(5)
FLOATABLE NUMBER(5)
VISIBLE NUMBER(5)
X NUMBER(5)
Y NUMBER(5)
MINIMIZED NUMBER(5)
WLCS_PORTAL_HIERARCHY
PORTAL_NID NUMBER(15)
PORTLET_NID NUMBER(15)
WLCS_USER_PERSONALIZATION
PORTAL_NID NUMBER(15)
CATEGORY_NID NUMBER(15)
GROUP_NID NUMBER(15)
USER_NID NUMBER(15)
PORTLET_NID NUMBER(15)
VISIBLE NUMBER(5)
X NUMBER(5)
Y NUMBER(5)
MINIMIZED NUMBER(5)
WLCS_PORTAL_GROUP_HIERARCHY
PORTAL_NID NUMBER(15)
PORTLET_NID NUMBER(15)


WLCS_PORTAL_DEFINITION
NID NUMBER(5)
SUSPENDED NUMBER(5)
CONTENT_COLUMN_COUNT NUMBER(5)
NAME VARCHAR2(500)
HEADER_URL VARCHAR2(500)
CONTENT_URL VARCHAR2(500)
FOOTER_URL VARCHAR2(500)
SUSPENDED_URL VARCHAR2(500)
WLCS_DOCUMENT
ID VARCHAR2(500)
DOCUMENT_SIZE NUMBER(15)
VERSION NUMBER(15)
AUTHOR VARCHAR2(50)
CREATION_DATE DATE
LOCKED_BY VARCHAR2(50)
MODIFIED_DATE DATE
MODIFIED_BY VARCHAR2(50)
MIME_TYPE VARCHAR2(100)
DESCRIPTION VARCHAR2(2000)
COMMENTS VARCHAR2(2000)
WLCS_PORTLET_DEFINITION
NID NUMBER(5)
HELP NUMBER(5)
MANDATORY NUMBER(5)
EDITABLE NUMBER(5)
MOVEABLE NUMBER(5)
LOGIN_REQUIRED NUMBER(5)
MINIMIZEABLE NUMBER(5)
MAXIMIZEABLE NUMBER(5)
FLOATABLE NUMBER(5)
VISIBLE NUMBER(5)
X NUMBER(5)
Y NUMBER(5)
MINIMIZED NUMBER(5)
NAME VARCHAR2(500)
HEADER_URL VARCHAR2(500)
FOOTER_URL VARCHAR2(500)
CONTENT_URL VARCHAR2(500)
BANNER_URL VARCHAR2(500)
ALTERNATE_HEADER_URL VARCHAR2(500)
ALTERNATE_FOOTER_URL VARCHAR2(500)
TITLEBAR_URL VARCHAR2(500)
EDIT_URL VARCHAR2(500)
HELP_URL VARCHAR2(500)
ICON_URL VARCHAR2(500)
MAXIMIZED_URL VARCHAR2(500)
WLCS_DOCUMENT_METADADA
ID VARCHAR2(500)
NAME VARCHAR2(240)
STATE VARCHAR2(50)
VALUE VARCHAR2(2000)
WLCS_CATEGORIES
NID NUMBER(15)
PORTAL_NID NUMBER(15)
NAME VARCHAR2(100)
ICON_URL VARCHAR2(100)
CATEGORY_ORDER NUMBER(5)
WLCS_COLUMN_INFORMATION
PORTAL_NID NUMBER(15)
CATEGORY_NID NUMBER(15)
COLUMN_ORDER NUMBER(5)
COLUMN_WIDTH NUMBER(5)

The Tables Comprising the WebLogic Personalization Server

The WebLogic Personalization Server is comprised of the following tables. In this list, the tables are sorted by functionality:

Documentation Management tables:
- WLCS_DOCUMENT
- WLCS_DOCUMENT_METADATA

Portal Management tables:
- WLCS_PORTAL_DEFINITION
- WLCS_COLUMN_INFORMATION
- WLCS_PORTLET_DEFINITION
- WLCS_PORTAL_PERSONALIZATION
- WLCS_GROUP_PERSONALIZATION
- WLCS_USER_PERSONALIZATION
- WLCS_PORTAL_GROUP_HIERARCHY
- WLCS_PORTAL_HIERARCHY
- WLCS_CATEGORIES

Rule Editor table:
- WLCS_RULESET_DEFINITION

User Management tables:
- WLCS_USER
- WLCS_GROUP
- WLCS_GROUP_HIERARCHY
- WLCS_USER_GROUP_HIERARCHY
- WLCS_UIDS

Tables used in the Sample Portal Application:
- WLCS_BOOKMARKS
- WLCS_TODO
- WLCS_UUP_EXAMPLE

Common tables used by both WLPS and WLCS:

WLCS_IS_ALIVE
WLCS_SEQUENCER
WLCS_SCHEMA
WLCS_PROP_MD
WLCS_PROP_MD_BOOLEAN
WLCS_PROP_MD_INTEGER
WLCS_PROP_MD_TEXT
WLCS_PROP_MD_DATETIME
WLCS_PROP_MD_USER_DEFINED
WLCS_PROP_MD_FLOAT
WLCS_ENTITY_ID
WLCS_PROP_ID
WLCS_PROP_BOOLEAN
WLCS_PROP_INTEGER
WLCS_PROP_TEXT
WLCS_PROP_DATETIME
WLCS_PROP_USER_DEFINED
WLCS_PROP_FLOAT
WLCS_USER
WLCS_GROUP
WLCS_GROUP_HIERARCHY
WLCS_USER_GROUP_HIERARCHY
WLCS_UNIFIED_PROFILE_TYPE
WLCS_LDAP_CONFIG

The Schema Tables

In this section, the WebLogic Personalization Server schema tables are arranged alphabetically as a data dictionary.

Note: Even though the following documentation references “foreign keys” to various tables, these constraints do not currently exist in this release of WebLogic Personalization Server. However, they will be (available in future releases) in place in future versions of WebLogic Personalization Server and we want you to be aware of these relationships now.

Table 9-1 describes the WLCS_BOOKMARKS table. This table is used by the Example portal and is not used except for demonstration purposes. It contains information used in the Bookmark portlet.

The Primary Key is comprised of NAME and OWNER.

Table 9-1 WLCS_BOOKMARKS

Column Name	Data Type	Description and Recommendations
URL	VARCHAR2(50)	The URL of the bookmark.
NAME	VARCHAR2(150)	The name of the bookmark.
OWNER	VARCHAR2(150)	The owner of the bookmark.

Table 9-2 describes the WLCS_CATEGORIES table. This table is used to store category information for the portal portion of the WebLogic Personalization Server application.

Note: The CATEGORY feature has not been implemented at this time and, therefore, this table is not being used/populated.

The Primary Key is NID.

Table 9-2 WLCS_CATEGORIES

Column Name	Data Type	Description and Recommendations
NID	NUMBER(15)	Category identifier.
PORTAL_NID	NUMBER(15)	The Portal identifier. This column is a foreign key to the NID column of the WLCS_PORTAL_DEFINITION table.
NAME	VARCHAR2(100)	The name for the category.
ICON_URL	VARCHAR2(100)	The URL pointing to the icon associated with the category. This may be null.
CATEGORY_ORDER	NUMBER(5)	The sequence number identifying the order of display.

Table 9-3 describes the WLCS_COLUMN_INFORMATION table. This table is used to store column definition information for each portal and category.

The Primary Key is comprised of PORTAL_NID, CATEGORY_NID and COLUMN_ORDER.

Table 9-3 WLCS_COLUMN_INFORMATION

Column Name	Data Type	Description and Recommendations
PORTAL_NID	NUMBER(15)	The Portal identifier. This column is a foreign key to the NID column of the WLCS_PORTAL_DEFINITION table.
CATEGORY_NID	NUMBER(15)	The Category identifier.
COLUMN_WIDTH	NUMBER(5)	The value entered here is a percentage of the screen width. An example would be 30. This represents how wide this particular portal column is to be (30% of the screen).
COLUMN_ORDER	NUMBER(5)	A sequence number identifying the display sequence for this column. Starting at the left-most part of the screen the COLUMN_ORDER would be 1.

Table 9-4 describes the WLCS_DOCUMENT table. This table is used to store information pertinent to each document used within the WebLogic Personalization Server.

The Primary Key is ID.

Table 9-4 WLCS_DOCUMENT

Column Name	Data Type	Description and Recommendations
ID	VARCHAR2(500)	The identifier of the document. This specifies the relative path (case sensitive using forward slashes) to the actual file.
DOCUMENT_SIZE	NUMBER(15)	The size of the document in bytes.
VERSION	NUMBER(15)	The version of the document.

Table 9-4 WLCS_DOCUMENT (Continued)

Column Name	Data Type	Description and Recommendations
AUTHOR	VARCHAR2(50)	The author's name of this document.
CREATION_DATE	DATE	The date this document was created in the system.
LOCKED_BY	VARCHAR2(50)	This column identifies who has this document locked for edits or updates.
MODIFIED_DATE	DATE	This tells you when this document record was last modified.
MODIFIED_BY	VARCHAR2(50)	This column stores the name of the individual who last modified the document record.
DESCRIPTION	VARCHAR2(50)	A description of the document.
COMMENTS	VARCHAR2(50)	An area to store miscellaneous notes about the document.
MIME_TYPE	VARCHAR2(100)	This column identifies which MIME type (or file type) is associated with this document. This is supposed to be MIME 1.0.

Table 9-5 describes the WLCS_DOCUMENT_METADATA table. This table is used to store user-defined properties associated with each document.

The Primary Key is ID and NAME.

Table 9-5 WLCS_DOCUMENT_METADATA

Column Name	Data Type	Description and Recommendations
ID	VARCHAR2(500)	The document identifier. This is a foreign key to the ID column of the WLCS_DOCUMENT table.
NAME	VARCHAR2(240)	The metadata name.
VALUE	VARCHAR2(2000)	The value to be associated with the metadata name (NAME).

Table 9-5 WLCS_DOCUMENT_METADATA (Continued)

Column Name	Data Type	Description and Recommendations
STATE	VARCHAR2(50)	The current state of this metadata property. This is used by Interwoven and can be set to null.

Table 9-6 describes the WLCS_ENTITY_ID table. Any ConfigurableEntity within the system will have an entry in this table.

The Primary Key is comprised of JNDI_HOME_NAME and PK_STRING.

Table 9-6 WLCS_ENTITY_ID

Column Name	Data Type	Description and Recommendations
JNDI_HOME_NAME	VARCHAR2(100)	Defines what type of ConfigurableEntity this is.
PK_STRING	VARCHAR2(200)	Unique identifier within the ConfigurableEntity.
ENTITY_ID	NUMBER(15)	A sequence-generated number providing a unique identifier used throughout the system (in the Property tables and so on).

Table 9-7 describes the WLCS_GROUPS table. This table is used to maintain each of the various Group identifiers.

The Primary Key is comprised of IDENTIFIER.

Table 9-7 WLCS_GROUP

Column Name	Data Type	Description and Recommendations
IDENTIFIER	VARCHAR2(50)	The group name. This column is a foreign key to the PK_STRING column in the WLCS_ENTITY_ID table.

Table 9-8 describes the WLCS_GROUP_HIERARCHY table. This table stores relationship information between groups.

The Primary Key is comprised of PARENT_ID and CHILD_ID.

Table 9-8 WLCS_GROUP_HIERARCHY

Column Name	Data Type	Description and Recommendations
PARENT_ID	NUMBER(15)	The parent group identifier. This column is a foreign key to the ENTITY_ID column in the WLCS_ENTITY_ID table.
CHILD_ID	NUMBER(15)	The child group identifier. This column is a foreign key to the ENTITY_ID column in the WLCS_ENTITY_ID table.

Table 9-9 describes the WLCS_GROUP_PERSONALIZATION table. Portals can be associated to groups and this table helps establish those relationships and maintain specific information for the group.

The Primary Key is comprised of PORTAL_NID, CATEGORY_NID, PORTLET_NID and GROUP_NID.

Table 9-9 WLCS_GROUP_PERSONALIZATION

Column Name	Data Type	Description and Recommendations
PORTAL_NID	NUMBER(15)	The portal identifier. This column is a foreign key to the NID column of the WLCS_PORTAL_DEFINITION table.
CATEGORY_NID	NUMBER(15)	The category identifier. This column is a foreign key to the NID column of the WLCS_CATEGORIES table.
PORTLET_NID	NUMBER(15)	The portlet identifier. This column is a foreign key to the NID column of the WLCS_PORTLET_DEFINITION table.
GROUP_NID	NUMBER(15)	The group identifier. This column is a foreign key to the ENTITY_ID column of the WLCS_ENTITY_ID table.

Table 9-9 WLCS_GROUP_PERSONALIZATION (Continued)

Column Name	Data Type	Description and Recommendations
AVAILABLE	NUMBER(5)	A switch to identify whether or not this portlet is available.
MANDATORY	NUMBER(5)	This flag, when set, overrides the VISIBLE flag and requires the portlet be displayed.
EDITABLE	NUMBER(5)	This flag determines whether a user is allowed to edit any content.
MOVEABLE	NUMBER(5)	This column is not being used.
MINIMIZEABLE	NUMBER(5)	This flag determines whether or not the user will be allowed to minimize the portlet.
MAXIMIZEABLE	NUMBER(5)	This flag determines whether or not the user will be allowed to maximize the portlet.
FLOATABLE	NUMBER(5)	This flag determines whether the portlet can open up in its own browser window.
VISIBLE	NUMBER(5)	This flag determines whether or not the portlet is visible.
X	NUMBER(5)	The X coordinate determines the placement of the portlet on the screen. This is zero based and refers to the column placement (0=column 1, 1=column 2 and so on).
Y	NUMBER(5)	The Y coordinate determines placement of the portlet on the screen. Like the X coordinate, it is zero based. The Y coordinate refers to the row placement (0=row 1, 1=row 2 and so on).
MINIMIZED	NUMBER(5)	This flag determines whether or not the portlet should be displayed in a minimized format when initially displayed.

Table 9-10 describes the WLCS_IS_ALIVE table. This table is used by the JDBC connection pools to insure the connection to the database is still alive.

Table 9-10 WLCS_IS_ALIVE

Column Name	Data Type	Description and Recommendations
NAME	VARCHAR2(100)	Used by the JDBC connection pools to insure the connection to the database is still alive.

Table 9-11 describes the WLCS_LDAP_CONFIG table. This table holds configuration information for LDAP functionality within the User Management module.

The Primary Key is LDAP_PROPERTY.

Table 9-11 WLCS_LDAP_CONFIG

Column Name	Data Type	Description and Recommendations
LDAP_PROPERTY	VARCHAR2(100)	The property name.
LDAP_VALUE	VARCHAR2(255)	The property value.

Table 9-12 describes the WLCS_PORTAL_DEFINITION table.

The Primary Key is NID.

Table 9-12 WLCS_PORTAL_DEFINITION

Column Name	Data Type	Description and Recommendations
NID	NUMBER(15)	The identifier for the portal definition.
NAME	VARCHAR2(500)	The name of the portal definition. Any combination of numbers and letters will be accepted in this field.
HEADER_URL	VARCHAR2(500)	Enter a URL to display as the portal header. It can be a JSP or HTML fragment.

Table 9-12 WLCS_PORTAL_DEFINITION (Continued)

Column Name	Data Type	Description and Recommendations
CONTENT_URL	VARCHAR2(500)	Enter a URL relative to your portal working directory.
FOOTER_URL	VARCHAR2(500)	Enter a URL to display as the portal footer. It can be a JSP or HTML fragment.
CONTENT_COLUMN_COUNT	NUMBER(5)	Specifies the number of content columns. Valid values at this time would be 1, 2, or 3.
SUSPENDED	NUMBER(5)	Set this flag to suspend the portal application and replace the portal home page with an "under maintenance" screen until service resumes.
SUSPENDED_URL	VARCHAR2(500)	When the SUSPENDED flag is set this URL will point to the JSP page to be displayed while the application is in suspend mode.

Table 9-13 describes the WLCS_PORTAL_GROUP_HIERARCHY table. This table maintains records showing which groups are associated with each portal.

The Primary Key is comprised of PORTAL_NID and GROUP_NID.

Table 9-13 WLCS_PORTAL_GROUP_HIERARCHY

Column Name	Data Type	Description and Recommendations
PORTAL_NID	NUMBER(15)	The portal identifier. This column is a foreign key to the NID column of the WLCS_PORTAL_DEFINITION table.
GROUP_NID	NUMBER(15)	The group identifier. This column is a foreign key to the ENTITY_ID column of the WLCS_ENTITY_ID table.

Table 9-14 describes the WLCS_PORTAL_HIERARCHY table. This table contains records showing which portlets are associated with each portal.

The Primary Key is comprised of PORTAL_NID and PORTLET_NID.

Table 9-14 WLCS_PORTAL_HIERARCHY

Column Name	Data Type	Description and Recommendations
PORTAL_NID	NUMBER(15)	The portal identifier. This column is a foreign key to the NID column of the WLCS_PORTAL_DEFINITION table.
PORTLET_NID	NUMBER(15)	The portlet identifier. This column is a foreign key to the NID column of the WLCS_PORTLET_DEFINITION table.

Table 9-15 describes the WLCS_PORTAL_PERSONALIZATION table. This table maintains information pertinent to each personalized portal definition.

The Primary Key is comprised of PORTAL_NID, CATEGORY_NID and PORTLET_NID.

Table 9-15 WLCS_PORTAL_PERSONALIZATION

Column Name	Data Type	Description and Recommendations
PORTAL_NID	NUMBER(15)	The portal identifier. This column is a foreign key to the NID column of the WLCS_PORTAL_DEFINITION table.
CATEGORY_NID	NUMBER(15)	The category identifier. This column is a foreign key to the NID column of the WLCS_CATEGORIES table.
PORTLET_NID	NUMBER(15)	The portlet identifier. This column is a foreign key to the NID column of the WLCS_PORTLET_DEFINITION table.
AVAILABLE	NUMBER(5)	This flag, when set, overrides the VISIBLE flag and requires the portlet be displayed. 0 equates to FALSE and 1 equates to TRUE.

Table 9-15 WLCS_PORTAL_PERSONALIZATION (Continued)

Column Name	Data Type	Description and Recommendations
EDITABLE	NUMBER(5)	This flag determines whether a user is allowed to edit the content of the portal. 0 equates to FALSE and 1 equates to TRUE.
MOVEABLE	NUMBER(5)	This column is not being used. 0 equates to FALSE and 1 equates to TRUE.
MINIMIZEABLE	NUMBER(5)	This flag determines whether or not the user will be allowed to minimize the portlet. 0 equates to FALSE and 1 equates to TRUE.
MAXIMIZEABLE	NUMBER(5)	This flag determines whether or not the user will be allowed to maximize the portlet. 0 equates to FALSE and 1 equates to TRUE.
FLOATABLE	NUMBER(5)	This flag determines whether the portlet can open up in its own browser window. 0 equates to FALSE and 1 equates to TRUE.
VISIBLE	NUMBER(5)	This flag determines whether or not the portlet is visible. 0 equates to FALSE and 1 equates to TRUE.
X	NUMBER(5)	The X coordinate determines the placement of the portlet on the screen. This is zero based and refers to the column placement (0=column 1, 1=column 2 and so on).
Y	NUMBER(5)	The Y coordinate determines placement of the portlet on the screen. Like the X coordinate, it is zero based. The Y coordinate refers to the row placement (0=row 1, 1=row 2 and so on).
MINIMIZED	NUMBER(5)	This flag determines whether or not the portlet should be displayed in a minimized format when displayed initially. 0 equates to FALSE and 1 equates to TRUE.

Table 9-16 describes the WLCS_PORTLET_DEFINITION table. This table maintains information pertinent to each portlet definition.

The Primary Key is comprised of NID.

Table 9-16 WLCS_PORTLET_DEFINITION

Column Name	Data Type	Description and Recommendations
NID	NUMBER(15)	The portlet identifier.
NAME	VARCHAR2(500)	The name of your portlet. Any combination of numbers and letters will be accepted in this field.
HEADER_URL	VARCHAR2(500)	Enter a URL to display as the portlet header. It can be a JSP or HTML fragment.
FOOTER_URL	VARCHAR2(500)	Enter a URL to display as the portlet footer. It can be a JSP or HTML fragment.
CONTENT_URL	VARCHAR2(500)	Enter a URL relative to your portal working directory.
BANNER_URL	VARCHAR2(500)	Enter a URL to display as the portlet banner under the portlet titlebar. It can be a JSP or HTML fragment.
ALTERNATE_HEADER_URL	VARCHAR2(500)	Enter a URL to display as a Web page header when the portlet is floated or maximized. If this is null, the portal framework uses a default called alternateheader.jsp.
ALTERNATE_FOOTER_URL	VARCHAR2(500)	Enter a URL to display as a Web page footer when the portlet is floated or maximized. If this is null, the portal framework uses a default called alternatefooter.jsp.
TITLEBAR_URL	VARCHAR2(500)	Enter a URL to display as the portlet titlebar. It can be a JSP or HTML fragment.
EDIT_URL	VARCHAR2(500)	If the EDITABLE flag has been set then a URL will be stored here that enables the user to edit the portlet content.

Table 9-16 WLCS_PORTLET_DEFINITION (Continued)

Column Name	Data Type	Description and Recommendations
HELP_URL	VARCHAR2(500)	If the HELP flag has been set then a URL must be specified that opens a help topic related to the portlet.
ICON_URL	VARCHAR2(500)	A URL to display an icon (GIF) on the left side of the portlet title bar. This image should be 27 pixels wide by 20 pixels high with 2 pixels of transparency on the right.
HELP	NUMBER(5)	This flag determines whether users can access a help screen in the portlet. If set, a Help icon displays in the portlet titlebar.
MAXIMIZED_URL	VARCHAR2(500)	A URL for the content area of the maximized page. The default URL is your portlet content area URL.
MANDATORY	NUMBER(5)	This flag, when set, overrides the VISIBLE flag and requires the portlet be displayed. 0 equates to FALSE and 1 equates to TRUE.
EDITABLE	NUMBER(5)	This flag determines whether a user is allowed to edit the content of a portlet. 0 equates to FALSE and 1 equates to TRUE.
MOVEABLE	NUMBER(5)	This column is not being used. 0 equates to FALSE and 1 equates to TRUE.
LOGIN_REQUIRED	NUMBER(5)	This flag determines whether or not security is required for access to the portlet. 0 equates to FALSE and 1 equates to TRUE.
MINIMIZEABLE	NUMBER(5)	This flag determines whether or not the user will be allowed to minimize the portlet. 0 equates to FALSE and 1 equates to TRUE.
MAXIMIZEABLE	NUMBER(5)	This flag determines whether or not the user will be allowed to maximize the portlet. 0 equates to FALSE and 1 equates to TRUE.

Table 9-16 WLCS_PORTLET_DEFINITION (Continued)

Column Name	Data Type	Description and Recommendations
FLOATABLE	NUMBER(5)	This flag determines whether the portlet can open up in its own browser window. 0 equates to FALSE and 1 equates to TRUE.
VISIBLE	NUMBER(5)	This flag determines whether or not the portlet is visible. 0 equates to FALSE and 1 equates to TRUE.
X	NUMBER(5)	The X coordinate determines the placement of the portlet on the screen. This is zero based and refers to the column placement (0=column 1, 1=column 2 and so on).
Y	NUMBER(5)	The Y coordinate determines placement of the portlet on the screen. Like the X coordinate, it is zero based. The Y coordinate refers to the row placement (0=row 1, 1=row 2 and so on).
MINIMIZED	NUMBER(5)	This flag determines whether or not the portlet should be displayed in a minimized format when displayed initially. 0 equates to FALSE and 1 equates to TRUE.

Table 9-17 describes the WLCS_PROP_BOOLEAN table. This table stores property values for boolean properties.

The Primary Key is PROPERTY_ID.

Table 9-17 WLCS_PROP_BOOLEAN

Column Name	Data Type	Description and Recommendations
PROPERTY_ID	NUMBER(15)	The identifier for each boolean property.
VALUE	NUMBER(3)	The value for each boolean property identifier.

Table 9-18 describes the WLCS_PROP_DATETIME table. This table stores property values for date and time properties.

The Primary Key is PROPERTY_ID.

Table 9-18 WLCS_PROP_DATETIME

Column Name	Data Type	Description and Recommendations
PROPERTY_ID	NUMBER(15)	The identifier for each date and time property.
VALUE	DATE	The value for each data and time property identifier.

Table 9-19 describes the WLCS_PROP_FLOAT table. This table stores property values for float properties.

The Primary Key is PROPERTY_ID.

Table 9-19 WLCS_PROP_FLOAT

Column Name	Data Type	Description and Recommendations
PROPERTY_ID	NUMBER(15)	The identifier for each float property.
VALUE	NUMBER	The value associated with each float property identifier.

Table 9-20 describes the WLCS_PROP_ID table. Any property assigned to a ConfigurableEntity has a unique PROPERTY_ID. This identifier and associated information is stored here.

The Primary Key is ENTITY_ID, PROPERTY_NAME and SCOPE_NAME.

Table 9-20 WLCS_PROP_ID

Column Name	Data Type	Description and Recommendations
ENTITY_ID	NUMBER(15)	A system generated value and foreign key to the WLCS_ENTITY_ID column.
SCOPE_NAME	VARCHAR2(100)	This column may be null. If this property is defined in a property set, then the SCOPE_NAME will match the SCHEMA_NAME for that property set in the WLCS_SCHEMA table.
PROPERTY_NAME	VARCHAR2(100)	The name of the property.
PROPERTY_TYPE	NUMBER(3)	This column identifies the type of property we are dealing with (for example, boolean, integer, float, text, and so on).
PROPERTY_META_DATA_ID	NUMBER(15)	The identifier for the Property metadata information. Again, we use the PROPERTY_TYPE column to identify which type of Property metadata we are looking at (for example, boolean, integer, and so on).
SCHEMA_HAS_CHANGED	NUMBER(3)	A flag informing to identify whether anything in the WLCS_SCHEMA or WLCS_PROP_MD_xxx tables has changed. If so, then certain cleanup activities must be performed prior to using this property next time.
PROPERTY_ID	NUMBER(15)	The property identifier is a unique system generated number.

Table 9-21 describes the WLCS_PROP_INTEGER table. This table stores property values for integer properties.

The Primary Key is PROPERTY_ID.

Table 9-21 WLCS_PROP_INTEGER

Column Name	Data Type	Description and Recommendations
PROPERTY_ID	NUMBER(15)	The identifier of the integer property.
VALUE	NUMBER(20)	The value associated with the integer property.

Table 9-22 describes the WLCS_PROP_MD table. This table stores information about defined properties in a property set.

The Primary Key is SCHEMA_ID.

Table 9-22 WLCS_PROP_MD

Column Name	Data Type	Description and Recommendations
SCHEMA_ID	NUMBER(15)	A foreign key to the WLCS_SCHEMA table.
PROPERTY_NAME	VARCHAR2(100)	The name of a property.
DESCRIPTION	VARCHAR2(255)	A description of the property.
IS_RESTRICTED	NUMBER(3)	If set TRUE, the value of the property is constrained to a set of values. 0 equates to FALSE and 1 equates to TRUE.
IS_EXPLICIT	NUMBER(3)	If set TRUE, the property value may be coming from an external source. 0 equates to FALSE and 1 equates to TRUE.
IS_MULTIVALUED	NUMBER(3)	Some properties may have more than one value. 0 equates to FALSE and 1 equates to TRUE.
PROPERTY_TYPE	NUMBER(3)	Defines the property type (boolean, text and so on).
PROPERTY_META_DATA_ID	NUMBER(15)	The primary key is a unique, system-generated value.

Table 9-23 describes the WLCS_PROP_MD_BOOLEAN table. This table stores property set definitions for the boolean property type.

The Primary Key is PROPERTY_META_DATA_ID.

Table 9-23 WLCS_PROP_MD_BOOLEAN

Column Name	Data Type	Description and Recommendations
PROPERTY_META_DATA_ID	NUMBER(15)	A unique identifier for this Property metadata and foreign key to the WLCS_PROP_MD table.
VALUE	NUMBER(3)	The value associated with the Property metadata.
IS_DEFAULT	NUMBER(3)	This flag tells us whether or not the VALUE column is the default value for this piece of Property metadata. 0 equates to FALSE and 1 equates to TRUE.

Table 9-24 describes the WLCS_PROP_MD_DATETIME table. This table stores property set definitions for the date and time property type.

The Primary Key is PROPERTY_META_DATA_ID.

Table 9-24 WLCS_PROP_MD_DATETIME

Column Name	Data Type	Description and Recommendations
PROPERTY_META_DATA_ID	NUMBER(20)	A unique identifier for this Property metadata.
VALUE	DATE	The value associated with the Property metadata.
IS_DEFAULT	NUMBER(3)	This flag tells us whether or not the VALUE column is the default value for this piece of Property metadata. 0 equates to FALSE and 1 equates to TRUE.

Table 9-25 describes the WLCS_PROP_MD_FLOAT table. This table stores property set definitions for the float property type.

The Primary Key is PROPERTY_META_DATA_ID.

Table 9-25 WLCS_PROP_MD_FLOAT

Column Name	Data Type	Description and Recommendations
PROPERTY_META_DATA_ID	NUMBER(15)	A unique identifier for this Property metadata.
VALUE	NUMBER	The value associated with the Property metadata.
IS_DEFAULT	NUMBER(3)	This flag tells us whether or not the VALUE column is the default value for this piece of Property metadata. 0 equates to FALSE and 1 equates to TRUE.

Table 9-26 describes the WLCS_PROP_MD_INTEGER table. This table stores property set definitions for the Integer property type.

The Primary Key is PROPERTY_META_DATA_ID.

Table 9-26 WLCS_PROP_MD_INTEGER

Column Name	Data Type	Description and Recommendations
PROPERTY_META_DATA_ID	NUMBER(15)	A unique identifier for this Property metadata.
VALUE	NUMBER(20)	The value associated with the Property metadata.
IS_DEFAULT	NUMBER(3)	This flag tells us whether or not the VALUE column is the default value for this piece of Property metadata. 0 equates to FALSE and 1 equates to TRUE.

Table 9-27 describes the WLCS_PROP_MD_TEXT table. This table stores property set definitions for the text property type.

The Primary Key is PROPERTY_META_DATA_ID.

Table 9-27 WLCS_PROP_MD_TEXT

Column Name	Data Type	Description and Recommendations
PROPERTY_META_DATA_ID	NUMBER(15)	A unique identifier for this Property metadata.
VALUE	VARCHAR2(255)	The value associated with the Property metadata.
IS_DEFAULT	NUMBER(3)	This flag tells us whether or not the VALUE column is the default value for this piece of Property metadata. 0 equates to FALSE and 1 equates to TRUE.

Table 9-28 describes the WLCS_PROP_MD_USER_DEFINED table. This table stores property set definitions for any user defined property type.

The Primary Key is PROPERTY_META_DATA_ID.

Table 9-28 WLCS_PROP_MD_USER_DEFINED

Column Name	Data Type	Description and Recommendations
PROPERTY_META_DATA_ID	NUMBER(15)	A unique identifier for this Property metadata.
VALUE	BLOB	The value associated with the Property metadata.
IS_DEFAULT	NUMBER(3)	This flag tells us whether or not the VALUE column is the default value for this piece of Property metadata. 0 equates to FALSE and 1 equates to TRUE.

Table 9-29 describes the WLCS_PROP_TEXT table. This table stores property values for the text for the text property type.

The Primary Key is PROPERTY_ID.

Table 9-29 WLCS_PROP_TEXT

Column Name	Data Type	Description and Recommendations
PROPERTY_ID	NUMBER(15)	The identifier of the text property.
VALUE	VARCHAR2(255)	The value associated with the text property.

Table 9-30 describes the WLCS_PROP_USER_DEFINED table. This table stores property values for any user-defined property type.

The Primary Key is PROPERTY_ID.

Table 9-30 WLCS_PROP_USER_DEFINED

Column Name	Data Type	Description and Recommendations
PROPERTY_ID	NUMBER(15)	The identifier of the user-defined property.
VALUE	BLOB	The value associated with the user-defined property.

Table 9-31 describes the WLCS_RULESET_DEFINITION table. This table contains all rule sets.

The Primary Key is NAME.

Table 9-31 WLCS_RULESET_DEFINITION

Column Name	Data Type	Description and Recommendations
NAME	VARCHAR2(50)	The identifier, or name, of the rule set.
DOCUMENT	BLOB	The XML document containing the rule set definition.

Table 9-32 describes the WLCS_SCHEMA table. This table stores property set definitions.

The Primary Key is comprised of SCHEMA_GROUP_NAME and SCOPE_NAME.

Table 9-32 WLCS_SCHEMA

Column Name	Data Type	Description and Recommendations
SCHEMA_GROUP_NAME	VARCHAR2(100)	The type of object this schema is used for.
SCOPE_NAME	VARCHAR2(100)	The application name since it is defining names for the application.
DESCRIPTION	VARCHAR2(255)	A description of the schema.
SCHEMA_ID	NUMBER(15)	A system generated number used throughout the application.

Table 9-33 describes the WLCS_SEQUENCER table. The WLCS_SEQUENCER table is used to maintain all of the sequence identifiers (for example, property_meta_data_id_sequence, and so on) used in the application.

The Primary Key is SEQUENCE_NAME.

Table 9-33 WLCS_SEQUENCER

Column Name	Data Type	Description and Recommendations
SEQUENCE_NAME	VARCHAR2(50)	A unique name used to identify the sequence.
CURRENT_VALUE	NUMBER(15)	The current value of the sequence.

Table 9-34 describes the WLCS_TODO table. This table is used by the Example portal and is not used except for demonstration purposes. It contains information used in the To Do portlet.

The Primary Key is ITEM and OWNER.

Table 9-34 WLCS_TODO

Column Name	Data Type	Description and Recommendations
ITEM	VARCHAR2(50)	The activity to be accomplished.
DONE	NUMBER(5)	The status identifying whether this item has been completed.
PRIORITY	NUMBER(5)	The priority of the activity.
OWNER	VARCHAR2(150)	The individual who owns, or is responsible, for this activity.

Table 9-35 describes the WLCS_UIDS table. This table stores sequence information in a generic database independent format.

The Primary Key is SID.

Table 9-35 WLCS_UIDS

Column Name	Data Type	Description and Recommendations
SID	VARCHAR2(100)	The name of the sequence.
NEXT_SEQUENCE	NUMBER(15)	The next value available for use with the sequence.

Table 9-36 describes the WLCS_UNIFIED_PROFILE_TYPE table. This table allows registration of classes which extend the ProvidedUser class.

The Primary Key is TYPE_NAME.

Table 9-36 WLCS_UNIFIED_PROFILE_TYPE

Column Name	Data Type	Description and Recommendations
TYPE_NAME	VARCHAR2(100)	Any unique name used for easy lookup.
CLASS_NAME	VARCHAR2(100)	The name of the remote interface class.
HOME	VARCHAR2(100)	The name of the home class.
PK	VARCHAR2(100)	The name of the primary key class.
JNDI_NAME	VARCHAR2(100)	The name to look up in the JNDI tree.
SUCCESSOR	VARCHAR2(100)	This column allows you to define another class should the TYPE_NAME not exist. This column is a foreign key to TYPE_NAME of the WLCS_UNIFIED_PROFILE_TYPE table.

Table 9-37 describes the WLCS_USER table. This table stores all user login/password combinations.

The Primary Key is IDENTIFIER.

Table 9-37 WLCS_USER

Column Name	Data Type	Description and Recommendations
IDENTIFIER	VARCHAR2(50)	The user login. This column is a foreign key to the PK_STRING column of the WLCS_ENTITY_ID table.
PASSWORD	VARCHAR2(50)	The encrypted password.
IS_EXTERNAL	NUMBER(3)	This flag determines whether a user came from an external realm as opposed to the internal database realm.
PROFILE_TYPE	VARCHAR2(100)	A foreign key to the TYPE_NAME in the WLCS_UNIFIED_PROFILE_TYPE table.

Table 9-38 describes the WLCS_USER_GROUP_HIERARCHY table. This table allows you to store associated users and groups.

The Primary Key is comprised of USER_ID and GROUP_ID.

Table 9-38 WLCS_USER_GROUP_HIERARCHY

Column Name	Data Type	Description and Recommendations
USER_ID	NUMBER(15)	The ENTITY_ID of a user. This column is a foreign key to the USER_ID column of the WLCS_ENTITY_ID table.
GROUP_ID	NUMBER(15)	The ENTITY_ID of a group. This column is a foreign key to the USER_ID column of the WLCS_ENTITY_ID table.

Table 9-39 describes the WLCS_USER_PERSONALIZATION table.This table contains personalized portal information for the user.

The Primary Key is comprised of PORTAL_NID, CATEGORY_NID, GROUP_NID, USER_NID and PORTLET_NID.

Table 9-39 WLCS_USER_PERSONALIZATION

Column Name	Data Type	Description and Recommendations
PORTAL_NID	NUMBER(15)	The portal identifier. This column is a foreign key to the NID column of the WLCS_PORTAL_DEFINITION table.
CATEGORY_NID	NUMBER(15)	The category identifier. This column is a foreign key to the NID column of the WLCS_CATEGORIES table.
GROUP_NID	NUMBER(15)	The group identifier. This column is a foreign key to the ENTITY_ID column of the WLCS_ENTITY_ID table.
USER_NID	NUMBER(15)	The user identifier. This column is a foreign key to the ENTITY_ID column of the WLCS_ENTITY_ID table.

Table 9-39 WLCS_USER_PERSONALIZATION (Continued)

Column Name	Data Type	Description and Recommendations
PORTLET_NID	NUMBER(15)	The portlet identifier. This column is a foreign key to the NID column of the WLCS_PORTLET table.
VISIBLE	NUMBER(5)	This flag determines whether or not the portlet is visible. 0 equates to FALSE and 1 equates to TRUE.
X	NUMBER(5)	The X coordinate determines the placement of the portlet on the screen. This is zero based and refers to the column placement (0=column 1, 1=column 2 and so on).
Y	NUMBER(5)	The Y coordinate determines placement of the portlet on the screen. Like the X coordinate, it is zero based. The Y coordinate refers to the row placement (0=row 1, 1=row 2 and so on).
MINIMIZED	NUMBER(5)	This flag determines whether or not the portlet should be displayed in a minimized format when displayed initially. 0 equates to FALSE and 1 equates to TRUE.

Table 9-40 describes the WLCS_UUP_EXAMPLE table. This is an example of how to use the Unified Profile Types.

The Primary Key is NAME.

Table 9-40 WLCS_UUP_EXAMPLE

Column Name	Data Type	Description and Recommendations
NAME	VARCHAR2(100)	A username.
POINTS	NUMBER(15)	A point accumulator based on various actions taken by the user.

The SQL Scripts Used to Create the Database

WebLogic Personalization Server is installed as part of the WebLogic Commerce Server installation. If you install WebLogic Commerce Server into an NT environment and accept the defaults, you will have a directory path that looks like this:

```
C:\WebLogicCommerce\
```

Under this main directory you will find a database directory:

```
C:\WebLogicCommerce\db\
```

Under the database directory you will find several directories for each database currently supported:

- `C:\WebLogicCommerce\db\cloudscape`
- `C:\WebLogicCommerce\db\oracle`

At this particular level, you will find scripts which are common to both WebLogic Commerce Server and WebLogic Personalization Server.

Table 9-41 Scripts Common to Commerce Server and Personalization Server

Create-all-oracle.sql	This script executes all of the Oracle scripts. It will create tables, indexes, constraints and populate tables as well.
Create-common-oracle.sql	This script creates tables, indexes, and constraints common to both servers (WebLogic Commerce Server andWebLogic Personalization Server).
Create-wlcs-oracle.sql	This script calls all of the appropriate scripts used to build and populate theWebLogic Commerce Server database.
Create-wlps-oracle.sql	This script calls all of the appropriate scripts used to build and populate the WebLogic Personalization Server database.
Insert-common-oracle.sql	This script populates the common tables used by both servers (WebLogic Commerce Server and WebLogic Personalization Server).

Finally, under each database directory you will also find a WLCS and a WLPS directory:

For example, under the Oracle database directory you would find:

```
C:\WebLogicCommerce\db\oracle\wlcs
```

```
C:\WebLogicCommerce\db\oracle\wlps
```

These directories contain scripts which are specific to the individual server (WebLogic Commerce Server and WebLogic Personalization Server).

Under C:\WebLogicCommerce\db\oracle\wlcs you will find the scripts listed in Table 9-42.

Table 9-42 Scripts Specific to WebLogic Commerce Server

Create-catalog-oracle.sql	Creates tables, indexes and constraints related to the Catalog portion of the WebLogic Commerce Server.
Create-order-oracle.sql	Creates tables, indexes and constraints related to the Order Management portion of the WebLogic Commerce Server.
Insert-catalog-data-oracle.sql	Populates the Catalog portion of the WebLogic Commerce Server database with preliminary data.
Insert-order-data-oracle.sql	Populates the Order Management portion of the WebLogic Commerce Server database with preliminary data.
Insert-wlcs-common-oracle.sql	Populates the database with preliminary data in the tables shared by both the Catalog and Order Management pieces of the WebLogic Commerce Server.

Under C:\WebLogicCommerce\db\oracle\wlps you will find the scripts listed in Table 9-43:

Table 9-43 Scripts Specific to WebLogic Personalization Server

Create-app-oracle.sql	Creates tables associated with the example application.
Create-document-oracle.sql	Creates tables, indexes and constraints associated with the Document Management portion of the WebLogic Personalization Server.
Create-portal-oracle.sql	Creates tables, indexes and constraints associated with the Portal Management portion of the WebLogic Personalization Server.
Create-ruleeditor-oracle.sql	Creates the tables associated with the Rule Set Engine portion of the WebLogic Personalization Server.
Insert-pzsamples-oracle.sql	Populates the WebLogic Personalization Server database with preliminary data.

10 JSP Tag Library Reference

The JSP tags included with WebLogic Personalization Server allow developers to create personalized applications without having to program using Java.

Note: The `es:` prefix stands for e-commerce services.
The `esp:` prefix stands for e-commerce services portal.
The `pz:` prefix stands for personalization.

This topic includes the following sections:

- The Advisor
  <pz:contentQuery>
  <pz:contentSelector>
  <pz:div>
- Content Management
  <cm:getProperty>
  <cm:printDoc>
  <cm:printProperty>
  <cm:select>
  <cm:selectById>
- Flow Manager
  <fm:getApplicationURI>
  <fm:getCachedAttribute>
  <fm:getSessionAttribute>
  <fm:setCachedAttribute>
  <fm:setSessionAttribute>
  <fm:removeCachedAttribute>
  <fm:removeSessionAttribute>

- Internationalization
  <i18n:localize>
  <i18n:getMessage>

- Portal Management
  <esp:eval>
  <esp:get>
  <esp:getGroupsForPortal>
  <esp:monitorSession>
  <esp:portalManager>
  <esp:portletManager>
  <esp:props>

- Property Sets
  <ps:getPropertyNames>
  <ps:getPropertySetNames>

- User Management
  Profile Management Tags
    <um:getProfile>
    <um:getProperty>
    <um:getPropertyAsString>
    <um:removeProperty>
    <um:setProperty>
  Group-User Management Tags
    <um:addGroupToGroup>
    <um:addUserToGroup>
    <um:changeGroupName>
    <um:createGroup>
    <um:createUser>
    <um:getChildGroupNames>
    <um:getChildGroups>
    <um:getGroupNamesForUser>
    <um:getParentGroupName>
    <um:getTopLevelGroups>
    <um:getUsernames>
    <um:getUsernamesForGroup>
    <um:removeGroup>
    <um:removeGroupFromGroup>
    <um:removeUser>
    <um:removeUserFromGroup>
  Security Tags

<um:login>
<um:logout>
<um:setPassword>

- Utility Tags: Personalization Utilities
  <es:counter>
  <es:date>
  <es:forEachInArray>
  <es:isNull>
  <es:monitorSession>
  <es:notNull>
  <es:preparedStatement>
  <es:simpleReport>
  <es:transposeArray>
  <es:uriContent>

- Utility Tags: WebLogic Utilities
  <wl:process>
  <wl:repeat>

The Advisor

By matching content to information contained in the user profile, the Advisor ties together all the other services and components in the system to deliver personalized content.

To import the Advisor JSP tags, use the following code:

```
<%@ taglib uri="pz.tld" prefix="pz" %>
```

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<pz:contentQuery>

The <pz:contentQuery> tag (Table 10-1) performs a content attribute search for content in a content manager. If the useCache attribute is set to true, the results of a content management query will be cached. The tag only has a begin tag and does not have a body or end tag. It returns an array of Content objects as determined by the Advisor.

Personalization content tags required for JSP developers to access the Content object returned might include:

An object array iterator tag. This tag provides a way to iterate over the Content objects in the array. Use the <es:forEachInArray> tag to iterate over an array of Objects. (See "<es:forEachInArray>" on page 10-73 for more information.)

- Content access tags. Content tags access metadata attributes in the content, retrieve content, and put it into the HTTP response output stream. (See "Content Management" on page 10-11 for more information.)

Table 10-1 <pz:contentQuery>

Tag Attribute	Req'd	Type	Description	R/C
max	No	String, long	Limits the maximum number of content items returned. If not present, it returns all of the content items found.	R
sortBy	No	String	A list of document attributes by which to sort the content. The syntax follows the SQL *order by* clause. The sort specification is limited to a list of the metadata attribute names and the keywords ASC and DESC. Examples: sortBy="creationDate" sortBy="creationDate ASC, title DESC"	R
query	Yes	String	A content query string used to search for content. Example: query= "mimetype contains 'text' && author='Proulx'"	R

Table 10-1 <pz:contentQuery> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
contentHome	Yes	String	The JNDI name of the ContentManager EJB Home to use to find content. The object in JNDI at this name must implement a `create` method which returns an object which implements the ContentManager interface. If not specified, the system searches the default content home.	R
id	Yes	String	The array variable name that contains the content objects found. If it finds no objects, it returns an empty array (not null) of Content objects.	C
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the `cacheId`, `cacheScope` and `cacheTimeout` settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R

Table 10-1 <pz:contentQuery> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application` ■ `session` (the default) ■ `page` ■ `request`	R

Example:

```
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ page input="com.beasys.commerce.content.ContentHelper" %>
.
.
.
<pz:contentQuery id="docs"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME%>"
   query="author = 'Hemingway'" />

<ul>


 <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.beasys.commerce.axiom.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
  </es:forEachInArray>
</ul>
```

<pz:contentSelector>

The `<pz:contentSelector>` tag allows arbitrary personalized content to be recommended based on a content selector rule. These rules are created and defined using the WebLogic Personalization Server rules editor. A content selector rule first

determines whether a user fits the specified classification (for example, high income), and then selects content based on another qualifier (such as productType = diamond jewelry.)

To cache the results of the content selector rule, set the `useCache` attribute to `true`. If the cache has not timed out, subsequent calls to the contentSelector tag will return the cached results without re-executing the rule.

The ruleSet URI protocol is as follows:

```
protocol://RuleSetDefinition-home-JNDI-name/RuleSet-name
```

- Where `protocol` is JDBC,
- `RuleSetDefinition-home-JNDI-name` is `com.beasys.commerce.axiom.reasoning.rules.RuleSheetDefinitionHome` which is the EJB home name of the RuleSet definition home, and
- `ruleset-name` is the unique identifier for the rule set, that is, the name given in the rules editor.

Example:

```
jdbc://com.beasys.commerce.axiom.reasoning.rules.RuleSheetDefinit
ionHome/AcmeRules
```

The `<pz:contentSelector>` tag (Table 10-2) only has a begin tag and does not have a body or end tag. It returns an array of `Content` objects as determined by the Advisor.

Tags possibly required for JSP developers to access the `Content` objects returned might include:

- An object array iterator tag. This tag provides a way to iterate over the `Content` objects in the array. Use the `<es:forEachInArray>` tag to iterate over an array of `Objects`.
- Content access tags. Content tags access metadata attributes in the content and retrieve content and put it into the HTTP response output stream. (See “Content Management” on page 10-11 for more information.)

Table 10-2 <pz:contentSelector>

Tag Attribute	Req'd	Type	Description	R/C
ruleSet	Yes	String	The URI for the rule set that contains the ContentSelector rule.	R
rule	Yes	String	The rule is the name of the classifier rule in the ruleSet that the rules advislet uses to classify the user.	R
max	No	String, long	Limits the maximum number of content items returned. If not present, or if equal to -1L, it returns all of the content items found.	R
sortBy	No	String	A list of document attributes by which to sort the content. The syntax follows the SQL *order by* clause. The sort specification is limited to a list of the metadata attribute names and the keywords ASC and DESC. Examples: sortBy="creationDate" sortBy="creationDate ASC, title DESC"	R
query	No	String	A content query string used to search for content. Example: query="'mimetype contains 'text' && author='Salinger'"	R
contentHome	Yes	String	The JNDI name of the ContentManager EJB Home to use to find content. The object in JNDI at this name must implement a `create` method which returns an object which implements the ContentManager interface. If not specified, the system searches the default content home.	R

Table 10-2 <pz:contentSelector> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	The array variable name that contains the content objects found. If it finds no objects, it returns an empty array (not null) of Content objects.	C
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the cacheId, cacheScope and cacheTimeout settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application` ■ `session` (the default) ■ `page` ■ `request`	R

Example:

```
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ page input="com.beasys.commerce.content.ContentHelper" %>
.
.
.
<pz:contentSelector id="docs" ruleSet="jdbc://com.beasys.
    commerce.axiom.reasoning.rules.
    RuleSheetDefinitionHome/AcmeRules"
    rule="PremierCustomerSpotlight"
    contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME%>"
<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.beasys.commerce.axiom.content.Content">
      <li>The document title is: <cm:printproperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

Note: The `sortBy` attribute, when used in conjunction with the `max` attribute, works differently for explicit (system-defined) and implicit (user-defined) attributes. If you sort on explicit attributes (`identifier, mimeType, size, version, author, creationDate, modifiedBy, modifiedDate, lockedBy, description,` or `comments`) the sort is done on the database; therefore if you combine `max="10"` and `sortBy`, the system will perform the sort and then get the first 10 items. If you sort on implicit attributes, the sort is done *after* the max have been selected.

<pz:div>

The `<pz:div>` tag (Table 10-3) allows a user-provided piece of content to be turned on or off as a result of a classifier rule being executed by a rules advislet. If the result is `true`, the content is turned on; if `false` it is turned off. This tag has a begin tag, a body, and an end tag. If it evaluates `true`, the tag returns the `Classification` object determined by the rules engine.

Table 10-3 <pz:div>

Tag Attribute	Req'd	Type	Description	R/C
ruleSet	Yes	String	The URI for the rule set that contains the Classifier rule.	R
rule	Yes	String	The rule is the name of the classifier rule in the ruleSet that the rules advislet uses to classify the user.	R
id	No	String	The variable name that is a handle for the returned Classification object.	C

Example:

```
<%@ taglib uri="pz.tld" prefix="pz" %>
.
.
.
<pz:div ruleSet="jdbc://com.beasys.commerce.axiom.reasoning.rules.
RuleSheetDefinitionHome/AcmeRules" rule="PremierCustomer">
   <p>Please check out our new Premier Customer bonus program.<p>
</pz:div>
```

Content Management

The Content Management component includes four JSP tags. These tags allow a JSP developer to include non-personalized content in a HTML-based page. The `cm:select` and `cm:selectbyid` tags support content caching for content searches. Note that none of the tags support or use a body.

To import the Content Management JSP tags, use the following code:

```
<%@ taglib uri="cm.tld" prefix="cm" %>
```

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<cm:getProperty>

The `<cm:get Property>` tag (Table 10-4) retrives the value of the specified content metadata property into a variable specified by `resultId`. It does not have a body. If `resultId` is not specified, the value will be inlined into the page, similar to the `<cm:printProperty>` tag. This tag operates on any ConfigurableEntity, not just the Content object. However, it does not support ConfigurableEntity successors.

Table 10-4 <cm:getProperty>

Tag Attribute	Req'd	Type	Description	R/C
id	No	String	The JSP script variable name which contains the Content instance from which to get the properties.	R
entity	No	ConfigurableEntity	Specifies the com.beasys.commerce.foundation. ConfigurableEntity object from which to get the property. If this is specified and non-null, `id` is ignored. Otherwise, `id` will be used.	R
name	Yes	String	The name of the property to print.	R
scope	No	String	The scope name for the property to get. If not specified, null is passed in, which is what Document objects expect.	R
resultId	no	String	The name of the JSP script variable which will be populated with the value of the property. If this is not specified, then the value of the property will be inlined into the body of the JSP. If this is specified, then `encode`, `default`, `maxLength`, `dateFormat`, and `numFormat` are ignored.	C

Table 10-4 <cm:getProperty> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
resultType	no	String	The Java type of the property. If this is not specified, then `java.lang.Object` is used.	C
encode	No	String	Either html, url, or none: ■ If html, then the value will be html encoded so that it appears in HTML as expected (& becomes *&*, < becomes *<*, > becomes *>*, and " becomes *"*). ■ If url, then it is encoded to x-www-form-urlencoded format via the java.net.URLEncoder. ■ If none or unspecified, no encoding is performed.	R
default	No	String	The value to print if the property is not found or has a null value. If this is not specified and the property value is null, nothing is printed.	R
maxLength	No	String, int	The maximum length of the property's value to print. If specified, values longer than this will be truncated.	R
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and prints nothing if an error occurs. `True`: Throws an exception. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
dateFormat	No	String	The java.text.SimpleDateFormat string to use to print the property, if it is a java.util.Date. If the property is not a Date, this is ignored. If this is not set, the Date's default `toString` method is used.	R

Table 10-4 <cm:getProperty> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
numFormat	No	String	The java.text.DecimalFormat string to use to print the property, if it is a java.lang.Number. If the property is not a Number, this is ignored. If this is not set, the Number's default `toString` method is used.	R

Example:

Get the String value of the `name` property from the Content object stored at `doc` and place it in the `contentName` variable:

```
<cm:getProperty resultId="contentName" resultType="String"
 id="content" name="name" />
<es:notNull item="<%=contentName%>">
The name is not null.
</es:notNull>
```

<cm:printDoc>

The `<cm:printDoc>` tag (Table 10-5) inlines the raw bytes of a `Document` object into the JSP output stream. This tag does not support a body and only supports `Document` objects. It does not differentiate between text and binary data.

Table 10-5 <cm:printDoc>

Tag Attribute	Req'd	Type	Description	R/C
id	No	String	The JSP script variable name which contains the Content instance from which to get the properties.	R
blockSize	No	String, int	The size of the blocks of data to read. The default is 8K. Use 0 or less to read the entire block of bytes in one operation.	R

Table 10-5 <cm:printDoc> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
start	No	String, int	Specifies the index in the bytes where to start reading. Defaults to 0.	R
end	No	String, int	Specifies the index in the bytes where to stop reading. The default is to read to the end of the bytes.	R
encode	No	String	Either html, url, or none: ■ If html, then the value will be html encoded so that it appears in HTML as expected (& becomes &, < becomes <, > becomes >, and " becomes "). ■ If url, then it is encoded to x-www-form-urlencoded format via the java.net.URLEncoder. ■ If none or unspecified, no encoding is performed.	R
document	No	Document	Specifies the com.beasys.commerce.axiom.document.Document to use. If this is specified and non-null, `id` will be ignored. Otherwise, `id` will be used.	R
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and prints nothing if an error occurs. `True`: Throws an exception. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
baseHref	No	String	The URL of the document's BASE HREF. This can be either an absolute URL or a relative URL.	R

Note: If `baseHref` is provided, then the `<cm:printDoc>` tag will output a starting <BASE HREF> using the value of the `baseHref` parameter. If `baseHref` is not a fully complete URL, the missing parts will be filled in based upon the URL of the outermost page. Additionally, the `<cm:printDoc>` will use the `FlowManagerHelper.getAppliactionFlowManager()` method to determine if the tag is operating under a FlowManager instance (a personalized application, a WebFlow'ed application, a portal).

Additionally, if `baseHref` is provided, then, after printing the document, the `<cm:printDoc>` tag will output a <BASE HREF> based upon the URL of the outermost page.

Example:

To get a Document object from an `id` in the `request` attributes and inline the Document's text (which might contain relative links):

```
<% String contentId = request.getParameter("contentId"); %>
<cm:selectById contentId="<%=contentId%>" id="doc" />
<cm:printDoc id="doc" blockSize="1000" baseHref="/ShowDocServlet"
/>
```

<cm:printProperty>

The `<cm:printProperty>` tag (Table 10-6) inlines the value of the specified content metadata property as a string. It does not have a body. This tag operates on any `ConfigurableEntity`, not just the `Content` object. However, it does not support `ConfigurableEntity` successors.

Table 10-6 <cm:printProperty>

Tag Attribute	Req'd	Type	Description	R/C
id	No	String	The JSP script variable name which contains the Content instance from which to get the properties.	R
name	Yes	String	The name of the property to print.	R

Table 10-6 <cm:printProperty> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
entity	No	ConfigurableEntity	Specifies the com.beasys.commerce.foundation. ConfigurableEntity object from which to get the property. If this is specified and non-null, `id` is ignored. Otherwise, `id` will be used.	R
scope	No	String	The scope name for the property to get. If not specified, null is passed in, which is what Document objects expect.	R
encode	No	String	Either html, url, or none: ■ If html, then the value will be html encoded so that it appears in HTML as expected (& becomes *&*, < becomes *<*, > becomes *>*, and " becomes *"*). ■ If url, then it is encoded to x-www-form-urlencoded format via the java.net.URLEncoder. ■ If none or unspecified, no encoding is performed.	R
default	No	String	The value to print if the property is not found or has a null value. If this is not specified and the property value is null, nothing is printed.	R
maxLength	No	String, int	The maximum length of the property's value to print. If specified, values longer than this will be truncated.	R

Table 10-6 <cm:printProperty> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and prints nothing if an error occurs. `True`: Throws an exception. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
dateFormat	No	String	The java.text.SimpleDateFormat string to use to print the property, if it is a java.util.Date. If the property is not a Date, this is ignored. If this is not set, the Date's default `toString` method is used.	R
numFormat	No	String	The java.text.DecimalFormat string to use to print the property, if it is a java.lang.Number. If the property is not a Number, this is ignored. If this is not set, the Number's default `toString` method is used.	R

Example:

To have a text input field's default value be the first 75 characters of the subject of a `Content` object stored at `doc`:

```
<form action="javascript:void(0)">
   Subject: <input type="text" size="75" name="subject"
   value="<cm:printProperty id="doc" name="Subject" maxLength="75"
   encode="html"/>" >
</form>
```

<cm:select>

This tag uses only the search expression query syntax to select content. It does not support or use a body. After this tag has returned the `<es:forEachInArray>` tag (see "<es:forEachInArray>" on page 10-73), zero can be used to iterate over the array of `Content` objects. This tag (Table 10-7) supports generic `Content` via a `ContentManager` interface.

Table 10-7 <cm:select>

Tag Attribute	Req'd	Type	Description	R/C
contentHome	No	String	The JNDI name of the ContentManager EJB Home to use to find content. The object in JNDI at this name must implement a `create` method which returns an object which implements the ContentManager interface. If not specified, the system searches the default content home.	R
max	No	String, long	Limits the maximum number of content items returned. If not present, or zero or less, it returns all of the content items found.	R
sortBy	No	String	A list of document attributes by which to sort the content. The syntax follows the SQL *order by* clause. The sort specification is limited to a list of the metadata attribute names and the keywords ASC and DESC. Examples: sortBy="creationDate" sortBy="creationDate ASC, title DESC"	R

Table 10-7 <cm:select> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and returns an empty array if an error occurs. `True`: Throws an exception that causes the JSP page to stop. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
id	Yes	String	The JSP script variable name that will contain the array of Content objects after this tag finishes.	C
query	No	String	A content query string used to search for content. Example: query="mimetype contains 'text' && author='Proulx'"	R
expr	No	Expression	The com.beasys.commerce.foundation.expression.Expression object to use to search for content. If this is null or not specified, then `query` must be specified. Otherwise, `query` is ignored.	R
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the `cacheId`, `cacheScope` and `cacheTimeout` settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R

Table 10-7 <cm:select> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application` ■ `session` (the default) ■ `page` ■ `request`	R
readOnly	No	String, Boolean	This attribute can have one of two values: If `true`, the ContentManager (specified via the `ContentHome` attribute) will try to return only lightweight (non-EJB) objects where possible. If `false` (not specified), the default value is used. Default= `ContentHelper.DEF_CONTENT_READONLY` (which is loaded from the `commerce.content.defaultReadOnly` property in the weblogiccommerce.properties file).	R

Example:

To find the first five text `Content` objects that are marked as news items for the evening using the ContentCache, and print out the titles in a list:

```
<cm:select
contentHome="<%=ContentHelper.DEF_CONTENT_MANAGER_HOME%>" max="5"
```

```
useCache="true" cacheTimeout="300000" cacheId="Evening News"
sortBy="creationDate ASC, title ASC" query="
    type = 'News' && timeOfDay = 'Evening' && mimetype like
    'text/*' " id="newsList"/>

<ul>
   <es:forEachInArray array="<%=newsList%>" id="newsItem"
   type="com.beasys.commerce.axiom.content.Content">
       <li><cm:printProperty id="newsItem" name="Title"
       encode="html" />
   </es:forEachInArray>
</ul>
```

<cm:selectById>

The `<cm:selectById>` tag (Table 10-8) retrieves content using the `Content's` unique identifier. This tag does not have a body. This tag is basically a wrapper around the `select` tag. It works against any `Content` object which has a string-capable primary key.

Table 10-8 <cm:selectById>

Tag Attribute	Req'd	Type	Description	R/C
contentHome	No	String	The JNDI name of the ContentManager EJB Home to use to find content. The object in JNDI at this name must implement a `create` method which returns an object that implements the ContentManager interface. If not specified, the system searches the default content home.	R
contentId	Yes	String	The string identifier of the piece of content.	R

Table 10-8 <cm:selectById> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and returns null if an error occurs. `True`: Throws an exception that causes the JSP page to stop. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
id	Yes	String	The JSP script variable name that contains the Content object after this tag finishes. If the Content object with the specified identifier does not exist, it contains null.	C
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the `cacheId`, `cacheScope` and `cacheTimeout` settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R

Table 10-8 <cm:selectById> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application` ■ `session` (the default) ■ `page` ■ `request`	R
readOnly	No	String, Boolean	This attribute can have one of two values: If `true`, the ContentManager (specified via the `ContentHome` attribute) will try to return only lightweight (non-EJB) objects where possible. If `false` (not specified), the default value is used. Default= `ContentHelper.DEF_CONTENT_READONLY` (which is loaded from the `commerce.content.defaultReadOnly` property in the weblogiccommerce.properties file).	R

Example:

To fetch the `Document` (using ContentCaching) with an identifier of `1234` and inline its content:

```
<cm:selectById
contentHome="<%=ContentHelper.DEF_CONTENT_MANAGER_HOME%>"
contentId="contentportlet/sports1.htm"
id="doc" useCache="true" cacheTimeout="300000" cacheId="1234" />
<cm:printDoc id="doc" />
```

Flow Manager

Thr Flow Manager tags are used for accessing the session, session cache, or the global cache. For scalability reasons, it is best to limit what gets placed into the session. For large sessions, session replication across servers is very costly. This tag library will give the user the ability to write to data that can be scoped to the application or across applications.

<fm:getApplicationURI>

The `<fm:getApplicationURI>` tag (Table 10-9) gets the application from the URL: http://localhost:7001/portals/application/exampleportal

When `includeContext="true"`, the tag returns `/context/path/pathinfo`, for example: `/portals/application/exampleportal`. This is required when a client browser needs to address the Web application context, for example, when using a form.

When `includeContext="false"`, the tag returns `/path/pathinfo`, for example `/application/exampleportal`. This is required when using Web applications and server side processing.

Table 10-9 <fm:getApplicationURI>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	The application as referenced by the Flow Manager. It can either get the value with the context or without. When used within a Web application, you must get the value without the context when using `<jsp:forward>`.	C
includeContext	No	boolean	Determines whether or not to include the servlet context with the application name. Defaults to `true`.	R

Example:

```
<%@ taglib uri="fm.tld" prefix="fm" %>
<%@ taglib uri="weblogic.tld" prefix="wl" %>

<wl:process name="formSubmit">
       <fm:getApplicationURI id="uri" includeContext="false"/>
       <jsp:forward page="<%=uri%>"/>
</wl:process>
```

<fm:getCachedAttribute>

The `<fm:getCachedAttribute>` tag (Table 10-10) gets an attribute out of the session/global cache. This value can be scoped to the application or not.

Table 10-10 <fm:getCachedAttribute>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	Object	The variable to store the retrieved value.	C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R
global	No	boolean	The value scoped to the session or the global scope. Defaults to `false`.	R

Example:

```
<%@ taglib uri="fm.tld" prefix="fm" %>

    <%Portal portal = null;%>
    <fm:getCachedAttribute id="tportal"
           name="<%=PortalTagConstants.CACHED_PORTAL%>"
 global="true" />
    <es:isNull item="<%=tportal%>" >
        <esp:portalManager action="get" id="myPortal"
                portalName="<%=portalName%>"/>
        <%tportal=myPortal;%>
```

```
        <fm:setCachedAttribute
                name="<%=PortalTagConstants.CACHED_PORTAL%>"
value="<%=myPortal%>" global="true" />
    </es:isNull>
    <%portal=(Portal)tportal;%>
```

<fm:getSessionAttribute>

The `<fm:getSessionAttribute>` tag (Table 10-11) gets an attribute out of the HttpSession. The attribute may be scoped to the application (by default).

Table 10-11 <fm:getSessionAttribute>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	Object	The variable to store the retrieved value.	C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R

Example:

```
<%@ taglib uri="fm.tld" prefix="fm" %>

<fm:getSessionAttribute id="username" name="portal.username"
     scoped="true" />
```

The name is: `<%=username%>`

<fm:removeCachedAttribute>

The `<fm:removeCachedAttribute>` tag (Table 10-12) removes an attribute from the session/global cache. This value can be scoped to the application or not.

Table 10-12 <fm:removeCachedAttribute>

Tag Attribute	Req'd	Type	Description	R/C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R
global	No	boolean	The value scoped to the session or the global scope. Defaults to `false`.	R

Example:

```
<%@ taglib uri="fm.tld" prefix="fm" %>

<fm:removeCachedAttribute
     name="<%=PortalTagConstants.CACHED_PORTAL%>" global="true" />
```

<fm:removeSessionAttribute>

The `<fm:removeSessionAttribute>` tag (Table 10-13) removes an attribute from the HttpSession. The attribute may be scoped to the application (by default).

Table 10-13 <fm:removeSessionAttribute>

Tag Attribute	Req'd	Type	Description	R/C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R

Example:

```
<%@ taglib uri="fm.tld" prefix="fm" %>
```

```
<fm:removeSessionAttribute name="portal.username" scoped="true" />
```

<fm:setCachedAttribute>

The `<fm:setCachedAttribute>` tag (Table 10-14) sets an attribute in the session/global cache. This value can be scoped to the application or not.

Table 10-14 <fm:setCachedAttribute>

Tag Attribute	Req'd	Type	Description	R/C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to `true`.	R
global	No	boolean	The value scoped to the session or the global scope. Defaults to `false`.	R
value	Yes	Object	The value to set.	R

Example:

```
<%@ taglib uri="fm.tld" prefix="fm" %>

  <%Portal portal = null;%>
  <fm:getCachedAttribute id="tportal"
        name="<%=PortalTagConstants.CACHED_PORTAL%>"
global="true" />
  <es:isNull item="<%=tportal%>" >
     <esp:portalManager action="get" id="myPortal"
        portalName="<%=portalName%>"/>
     <%tportal=myPortal;%>
     <fm:setCachedAttribute
        name="<%=PortalTagConstants.CACHED_PORTAL%>"
value="<%=myPortal%>" global="true" />
  </es:isNull>
  <%portal=(Portal)tportal;%>
```

<fm:setSessionAttribute>

The <fm:setSessionAttribute> tag (Table 10-15) sets an attribute in the HttpSession. The attribute may be scoped to the application (by default).

Table 10-15 <fm:setSessionAttribute>

Tag Attribute	Req'd	Type	Description	R/C
name	Yes	String	The name of the name/value pair.	R
scoped	No	boolean	The name/value pair scoped to the application. Defaults to true.	R
value	Yes	Object	The value to set.	R

Example:

```
<%@ taglib uri="fm.tld" prefix="fm" %>

<fm:setSessionAttribute name="portal.username"
     value="joe developer" scoped="true" />
```

Internationalization

These tags are used in the localization of JSP pages that have an internationalization requirement.

Use the following code to import the utility tag library:
```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
```

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<i18n:localize>

This tag allows you to define the language, country, variant, and base bundle name to be used throughout a page when accessing resource bundles via the `<i18n:getmessage>` tag.

This tag (Table 10-16) also specifies a character encoding and content type to be specified for a JSP page. Because of this, the tag should be used as early in the page as possible—before anything is written to the output stream—so that the bytes are properly encoded.

Note: When an HTML page is included in a larger page, only the larger page can use the `<i18n:localize>` tag. This is because the `<i18n:localize>` tag sets the encoding for the page, and the encoding must be set in the parent (including) page before any bytes are written to the response's output stream. The parent page must set an encoding that is sufficient for all the content on that page as well as any included pages.

Note: Do not use the `<i18n:localize>` tag in conjunction with the `<%@ page contentType="<something>" >` page directive defined in the JSP specification. The directive is unnecessary if you are using this tag, and can result in inconsistent or wrong `contentType` declarations.

Table 10-16 <i18n:localize>

Tag Attribute	Req'd	Type	Description	R/C
bundleName	No	String	The base name of the MessageBundle is used to retrieve localized text for a JSP page.	R
language	No	String or String []	A String—two character ISO Language Code—denoting the user's preferred language, or a String[] containing a list of preferred language codes for a user, with stronger preferences indexed lower (earlier) in the array.	R
country	No	String	The two character ISO Country Code for a country. For example, this code would be used to look for a MessageBundle containing text localized to English speaking users in the U.S. as opposed to English speaking users in the U.K.	R
variant	No	String	A String representing a locale's variant. The variant is used when localization demands a more specific locale than can be denoted by having just language and a country.	R
locale	No	java.util.Locale	Instead of specifying language, country, and variant as Strings, a `java.util.Locale` object can be provided. If provided, the values in the Locale's language, country, and variant fields will negate any of the other language, country, and variant values passed to the tag as Strings.	R
charset	No	String	The name of the character encoding set to use for this page. Defaults to "UTF-8" if no encoding can be determined for the chosen language, otherwise an encoding approprite for the chosen language is used.	R

Table 10-16 <i18n:localize> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
contentType	No	String	The type of content contained in the page, defaults to "text/html".	R

Example:

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>

<%
// Array that defines two languages preferences - English and
// Spanish in that order of preference.
String[] languages = new String[] { "en", "es" };

// Definition of a single language preference
String language = "en";
%>

<i18n:localize language="<%=language%>"
bundleName="i18nExampleResourceBundle"/>
<html>
<body>
<i18n:getMessage messageName="greeting"/>
</body>
</html>
```

<i18n:getMessage>

This tag (Table 10-17) is used in conjunction with the `<i18:localize>` tag to retrieve localized static text or messages from a JspMessageBundle.

Table 10-17 <i18n:getMessage>

Tag Attribute	Req'd	Type	Description	R/C
id	No	String	Holds the value of the label (or message) in the JSP page.	C
messageName	Yes	String	The key for the message bundle.	R

Table 10-17 <i18n:getMessage> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
messageArgs	No	Object []	The arguments to the message bundle. If no args are provided, it is assumed that static text (not a message) is to be returned. For example, {"Wednesday", "78"}; might be used to construct the message "Today is Wednesday, and the temperature is 78 degrees Fahrenheit."	R
bundleName	No	String	If properly initialized in the <i18n:localize> tag, there is no need to pass this tag attribute unless it is desired to use a different bundle for a particular tag invocation	R
language	No	String	If properly initialized in the <i18n:localize> tag, there is no need to pass this tag attribute, unless it is desired to use a different language for a particular tag invocation.	R
country	No	String	If properly initialized in the <i18n:localize> tag, there is no need to pass this tag attribute, unless it is desired to use a different country for a particular tag invocation.	R
variant	No	String	If properly initialized in the <i18n:localize> tag, there is no need to pass this tag attribute, unless it is desired to use a different variant for a particular tag invocation.	R
locale	No	java.util.Locale	If properly initialized in the <i18n:localize> tag, there is no need to pass this tag attribute, unless it is desired to use a different locale (language, country, and variant) for a particular tag invocation.	R

Example:

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>

<%
// Definition of a single language preference
String language = "en";

// Creation of message arguments
Object[] args = new Object[]
{
new Integer(14),
new Integer(100)
};
%>

<i18n:localize language="<%=language%>"
bundleName="i18nExampleResourceBundle"/>
<html>
<body>
<i18n:getMessage messageName="greeting"/>
<i18n:getMessage messageName="message" messageArgs="<%=args%>"/>
</body>
</html>
```

This code would produce this output:

```
Welcome To This Page! 14 out of 100 files have been saved.
```

Portal Management

The Portal Management component includes JSP tags for access to the fundamental data comprising a portal, such as portal and portlet properties.

To import the Portal Management JSP tags, use the following code:

```
<%@ taglib uri="esp.tld" prefix="esp" %>
```

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<esp:eval>

The `<esp:eval>` tag (Table 10-18) is used to evaluate a conditional attribute of a portlet, for example, `isMinimizeable`. The tag expects a `com.beasys.portal.Portlet` to be accessible in the session with the key `PortalTagConstants.PORTLET`. If the conditional attribute evaluates to `true`, the body of the `<esp:eval>` tag is processed. Otherwise, it is not.

Table 10-18 <esp:eval>

Tag Attribute	Req'd	Type	Description	R/C
tag	Yes	String	The name of the portlet attribute to evaluate. The following attributes can be retrieved: ■ `isEditable` ■ `isVisible` ■ `hasHelp` ■ `isMandatory` ■ `isMoveable` ■ `isMinimizeable` ■ `isMaximizeable` ■ `isFloatable` ■ `isMinimized`	R

Table 10-18 <esp:eval> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
target	No	Portlet	The com.beasys.portal.Portlet to be evaluated.	R

Example:

```
<esp:eval tag="isMinimizable">
     <% titleBar.include(minimizeButton); %>
</esp:eval>
```

<esp:get>

The `<esp:get>` tag (Table 10-19) retrieves a `String` attribute of a portlet. This tag expects a `com.beasys.portal.Portlet` to be accessible in the session with the key `PortalTagConstants.PORTLET`.

Table 10-19 <esp:get>

Tag Attribute	Req'd	Type	Description	R/C
tag	Yes	String	The name of the portlet attribute to retrieve. The following attributes can be retrieved: ■ `editURL` ■ `maximizedURL` ■ `headerURL` ■ `footerURL` ■ `contentURL` ■ `title`	R
target	No	Portlet	The com.beasys.portal.Portlet to be evaluated.	R

Example:

```
<tr>
  <td>
    <esp:get tag="title"/>
  </td>
</tr>
```

<esp:getGroupsForPortal>

The `<esp:getGroupsForPortal>` tag (Table 10-20) retrieves the names of the groups associated with a Portal. The results are put into the variable declared in the `id` attribute of the tag, which is a `String` array.

Table 10-20 <esp:getGroupsForPortal>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	A resulting string array containing the names of the groups associated with the given Portal.	R
portalName	Yes	String	The name of the Portal to be checked for associated groups.	R

Example:

```
<esp:getGroupsForPortal id ="groups" portalName="<%=portalName%>">
for (i=0;i<groups.length;i++)
{

String groupName = groups[i];
}
</esp:getGroupsForPortal>
```

<esp:monitorSession>

The <esp:monitorSession> tag (Table 10-21) can be added to the beginning of any JSP page to disallow access to the page if the session is not valid or if the user is not logged in.

Table 10-21 <esp:monitorSession>

Tag Attribute	Req'd	Type	Description	R/C
goToPage	No	String	The error page that you want displayed if the page is not accessible. The default value is `portalerror.jsp`.	R
loginRequired	No	String	Indicates whether the user is required to be logged in to access the JSP page including the tag. The default value is `false`.	R

Example:

```
<esp:monitorSession loginRequired="true" />
```

<esp:portalManager>

The <esp:portalManager> tag (Table 10-22) is used to perform `create`, `get`, `getColumnInfo`, `update`, and `remove` actions on `com.beasys.portal.Portal` objects. This tag is an empty tag.

Table 10-22 <esp:portalManager>

Tag Attribute	Req'd	Type	Description	R/C
id	When action equals `get` or `getColumnInfo`	String	The name to which resultant information is assigned for subsequent use in the JSP page.	R

Table 10-22 <esp:portalManager> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
action	No	String	The action to perform. Allowed values include: `create`: Creates a new portal. `get`: (default value) Retrieves an object of type com.beasys.portal.Portal. `getColumnInfo`: Retrieves a com.beasys.portal.PortalColumnInformation`[]` `update`: Updates the provided target com.beasys.portal.Portal. `remove`: Removes the provided target com.beasys.portal.Portal.	R
portalName	No	String	The name of the portal to retrieve, or whose column information is to be retrieved. The default value is `session.getValue(com.beasys.commerce.portal.admin.PortalAdminHelper.qualifiedName(PortalTagConstants.PORTAL_NAME, request))`	R
target	When action equals `create`, `update`, or `remove`	Portal	The com.beasys.portal.Portal to be created, updated, or removed.	R

Example:

```
<esp:portalManager id="portal" action="get"
portalName="BEAPortal"/>
```

<esp:portletManager>

The `<esp:portletManager>` tag (Table 10-23) is used to perform `create`, `get`, `getArranged`, `update`, and `remove` actions on `com.beasys.portal.Portlet` objects. This tag is an empty tag.

Table 10-23 <esp:portletManager>

Tag Attribute	Req'd	Type	Description	R/C
id	When action equals `get` or `getArranged`	String	The name to which resultant information is assigned for subsequent use in the JSP page.	R
action	No	String	The action to perform. Allowed values include: `create`: Creates a new portlet. `get`: Retrieves an object of type com.beasys.portal.Portlet. `getArranged`: Retrieves a com.beasys.portal.Portlet`[][]` that prescribes the row-column layout of portlets for the provided portal-user-group combination. `update` (default value): Updates the provided target com.beasys.portal.Portlet. `remove`: Removes the provided target com.beasys.portal.Portlet.	R
portalName	No	String	The name of the portal corresponding to the target portlet or to the portlet(s) to be retrieved. The default value is `session.getValue(com.beasys.commerce.portal.admin.PortalAdminHelper.qualifiedName(PortalTagConstants.PORTAL_NAME, request))`.	R
portletName	No	String	The name of the portlet corresponding to the target portlet or to the portlet(s) to be retrieved. There is no default value.	R

Table 10-23 <esp:portletManager> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
groupId	No	Long	The name of the group corresponding to the target portlet or to the portlet(s) to be retrieved. The default value is `com.beasys.commerce.axiom.jsp.JspHelper.getSessionValue(com.beasys.commerce.user.tags.UserManagerTagConstants.PROFILE_SUCCESSOR_UID, request)`	R
userId	No	Long	The name of the user corresponding to the target portlet or to the portlet(s) to be retrieved. The default value is `com.beasys.commerce.axiom.jsp.JspHelper.getSessionValue(com.beasys.commerce.user.tags.UserManagerTagConstants.PROFILE_USER_UID, request)`	R
target	When action equals `create`, `update`, or `remove`	Portlet	The com.beasys.portal.Portlet to be created, updated, or removed.	R
scope	No	String	The scope to be applied to the provided action. Allowed values include: `global` (default value): Specifies that portlet creation, removal, retrieval, or update should apply across all portals, groups, and users. `portal`: Specifies that portlet creation, removal, retrieval, or update applies to the provided portal. `group`: Specifies that portlet creation, removal, retrieval, or update applies to the provided portal-group combination. `user`: Specifies that portlet creation, removal, retrieval, or update applies to the provided portal-group-user combination.	R

Example:

```
<esp:portletManager id="arrangedPortlets" action="getArranged"
userId="myUser" portalName="myPortal"/>
```

<esp:props>

The <esp:props> tag (Table 10-24) is used to get a property from the Portal Properties bean. The Portal Properties bean's deployment descriptor contains default values used by the Portal Administration Tool.

Table 10-24 <esp:props>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	A java.lang.String variable name for the property value.	R
propertyName	Yes	String	The name of the property to get in the Portal Property Bean.	R

Example:

```
<esp:props id="headerURL"
propertyName="commerce.default.portal.headerURL" />
```

Property Sets

The Property Set tags allow access to the list of available properties and property sets. Manipulation of property sets can be done either programatically or through the administration tools.

Use the following code to import the utility tag library:

```
<%@ taglib uri="ps.tld" prefix="ps" %>
```

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<ps:getPropertyNames>

The `<ps:getPropertyNames>` tag (Table 10-25) is used to get a list of property names given a property set.

Table 10-25 <ps:getPropertyNames>

Tag Attribute	Req'd	Type	Description	R/C
propertySet	Yes	String	The name of the property set to add the new search.	R
schemaGroupName	Yes	String	Type of property set to search (as defined in com.beasys.commerce.foundation.property. SchemaManagerConstants).	R
id	Yes	String	The `id` of the variable to hold the list of property names, as a String array.	C

Table 10-25 <ps:getPropertyNames> (Continued) (Continued)

Tag Attribute	Req'd	Type	Description	R/C
result	Yes	String	The identifier of an Integer variable that will be created and initialized with the result of the operation. Possible values: *Query is successful*: `PropertySetTagConstants.PROPERTY_SEARCH_OK` *Problem getting the list of property names*: `PropertySetTagConstants.PROPERTY_SEARCH_FAILED` *Property set named by* `propertySetName` *and* `schemaGroupName` *could not be found*: `PropertySetTagConstants.INVALID_PROPERTY_SET`	C

Example:

```
<ps:getPropertyNames propertySet="<%myPropertySet%>"
     schemaGroupName="<%SchemaManagerConstants.USER_TYPE%>"
     id="propertyNames" result="myResult"/>
```

<ps:getPropertySetNames>

The `<ps:getPropertySetNames>` tag (Table 10-26) is used to get a list of property sets given a property set type.

Table 10-26 <ps:getPropertySetNames>

Tag Attribute	Req'd	Type	Description	R/C
schemaGroupName	Yes	String	The type of the property set to search (as defined in com.beasys.commerce.foundation.property.SchemaManagerConstants).	R

Table 10-26 <ps:getPropertySetNames> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	The identifier of the variable to hold the list of property names, as a String array.	C
result	Yes	String	The identifier of an Integer variable that will be created and initialized with the result of the operation. Possible values: *Query is successful*: `PropertySetTagConstants.PROPERTY_SET_SEARCH_OK` *Problem getting the list of property names*: `PropertySetTagConstants.PROPERTY_SET_SEARCH_FAILED` *Property set named by* `propertySetName` *and* `schemaGroupName` *could not be found*: `PropertySetTagConstants.INVALID_PROPERTY_SET`	C

User Management

User Management tags allow access to user and group profile information, as well as operations such as creating and deleting users and groups, and managing user-group relationships.

To import the User Management JSP tags, use the following code:

```
<%@ taglib uri="um.tld" prefix="um" %>
```

Profile Management Tags

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<um:getProfile>

The `<um:getProfile>` tag (Table 10-27) retrieves the profile corresponding to the provided profile key and profile type. The tag has no enclosed body. The retrieved profile can be treated simply as a `com.beasys.commerce.foundation.ConfigurableEntity`, or can be cast to the particular implementation of `ConfigurableEntity` that it is. Along with the profile key and profile, an explicit successor key and successor type can be specified, as specified by the `profileType` attribute. This successor will then be used, along with the retrieved profile, in subsequent invocations of the `<um:getProperty>` tag to ensure property inheritance from the successor. If no successor is retrieved, standard `ConfigurableEntity` successor search patterns will apply to retrieved properties.

Table 10-27 <um:getProfile>

Tag Attribute	Req'd	Type	Description	R/C
profileKey	Yes	String	A unique identifier that can be used to retrieve the profile which is sought. Example: "<%=username%>"	R

Table 10-27 <um:getProfile> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
successorKey	No	String	A unique identifier that can be used to retrieve the profile successor. Example: "<%=defaultGroup%>"	R
successorType	No	String	The profile successor type to be retrieved. If specified, this profile type *must* correspond to a profile type registered via the Unified Profile Type tool in the User Management suite of administration tools, and its bean must conform to the rules of Unified User Profile creation. By default, the tag retrieves a profile of type `com.beasys.commerce.axiom.contact.Group`, unless otherwise specified. Example: "AcmeGroup"	C
scope	No	String	The HTTP scope of the retreived profile. Pass `"request"` or `"session"` as the values. Defaults to `session`.	C
groupOnly	No	String	Specifies to retrieve a `com.beasys.commerce.axiom.contact.Group`, rather than `com.beasys.commerce.axiom.contact.User`, for the default profile type. No successor will be retrieved when this value is `true`. Defaults to `false`.	C
profileId	No	String	A variable name from which the retrieved profile is available for the duration of the JSP's page scope.	C

Table 10-27 <um:getProfile> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
profileType	No	String	Allows theJSP developer to specify what type of User profile object to return. If the given profileKey refers to a baseUser object, this attribute should be left blank. Otherwise, if it returns to an extended User object defined by a Unified Profile Type, the name of the Unified Profile Type should be supplied in this field.	C
successorId	No	String	A variable name from which the retrieved successor is available for the duration of the JSP's page scope.	C
result	No	String	A variable name from which the result of the operation is available. Possible values: *Success:* userManagerTagConstants.GET_PROFILE_OK *Error encountered:* userManagerTagConstants.GET_PROFILE_FAILED userManagerTagConstants.NO_SUCH_PROFILE userManagerTagConstants.NO_SUCH_SUCCESSOR	C

Example 1:

This example shows a profile of type `AcmeUser` being retrieved with no successor specified, and an explicitly-supplied `session` scope.

```
<um:getProfile profileKey="bob" profileType="AcmeUser"
profileId="myProfile" scope="session"/>
```

Example 2:

This example shows a default profile type (`com.beasys.commerce.axiom.contact.User`) being retrieved with a default successor type (`com.beasys.commerce.axiom.contact.Group`), and an explicitly-supplied `request` scope.

```
<um:getProfile profileKey="bob" successorKey="engineering"
scope="request"/>
```

Example 3:

This example shows a profile type of `AcmeUser` being retrieved with a successor type of `AcmeGroup`, and an implicitly-supplied `session` scope.

```
<um:getProfile profileKey="bob" profileType="AcmeUser"
    successorKey="engineering" successorType="AcmeGroup"
    profileId="myProfile"/>
```

<um:getProperty>

The `<um:getProperty>` tag (Table 10-28) retrieves the property value for a specified property set-property name pair. The tag has no enclosed body. The value returned is an `Object`. In typical cases, this tag is used after the `<um:getProfile>` tag is invoked to retrieve a profile for session use. The property to be retrieved is retrieved from the session profile. If the `<um:getProfile>` tag has not been used upon invoking the `<um:getProperty>` tag, the specified property value is retrieved from the Anonymous User Profile. See the chapter Creating and Managing Users in the *WebLogic Personalization Server User's Guide* for more information.

Table 10-28 <um:getProperty>

Tag Attribute	Req'd	Type	Description	R/C
propertySet	No	String	The Property Set from which the property's value is to be retrieved. Example: "Demo Portal" **Note:** If no property set is provided, the property is retrieved from the profile's default (unscoped) properties.	R

Table 10-28 <um:getProperty> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
propertyName	Yes	String	The name of the property to be retrieved. Example: "background_color"	R
id	No	String	If the `id` attribute is supplied, the value of the retrieved property will be available in the variable name to which `id` is assigned. Otherwise, the value of the property is inlined.	C

Example 1:

```
<um:getProperty id="myTitlebarBGColor" propertySet="exampleportal"
propertyName="titlebar_bgcolor"/>
My titlebar bg color is <%=myTitlebarBGColor%>.
```

Example 2:

```
My titlebar bg color is <um:getProperty propertySet="exampleportal"
propertyName="titlebar_bgcolor"/>.
```

<um:getPropertyAsString>

The `<um:getPropertyAsString>` tag (Table 10-29) works exactly like the `<um:getProperty>` tag above, but ensures that the retrieved property value is a `String`. The following example shows a multi-valued property which returns a `Collection`, but presents a list of favorite colors.

Table 10-29 <um:getPropertyAsString>

Tag Attribute	Req'd	Type	Description	R/C
propertySet	No	String	The Property Set from which the property's value is to be retrieved. Example: "Demo Portal" **Note:** If no property set is provided, the property is retrieved from the profile's default (unscoped) properties.	R
propertyName	Yes	String	The name of the property to be retrieved. Example: "background_color"	R
id	No	String	If the `id` attribute is supplied, the value of the retrieved property will be available in the variable name to which `id` is assigned. Otherwise, the value of the property is inlined.	C

Example:

```
<um:getPropertyAsString id="myFaveColors"
propertySet="exampleportal" propertyName="fave_colors"/>
My favorite colors are <%=myFaveColors%>.
```

<um:removeProperty>

The `<um:removeProperty>` tag (Table 10-30) removes the specified property from the current session's profile or from the Anonymous User Profile. The tag has no enclosed body. Subsequent calls to `<um:getProperty>` for a removed property would result in the default value for the property as prescribed by the property set, or from the Profile's successor.

Table 10-30 <um:removeProperty>

Tag Attribute	Req'd	Type	Description	R/C
propertySet	No	String	The Property Set from which the property's value is to be retrieved. Example: "Demo Portal" **Note:** The property is removed from the profile's default (unscoped) properties if no property set is provided.	R
propertyName	Yes	String	The name of the property to be removed. Example: "background_color"	R

Example:

```
<um:removeProperty propertySet="<%=thePropertySet%>"
propertyName="<%=thePropertyName%>"/>
```

<um:setProperty>

The `<um:setProperty>` tag (Table 10-31) updates a property value for either the session's current profile, or for the Anonymous User Profile. This tag has no enclosed body.

Table 10-31 <um:setProperty>

Tag Attribute	Req'd	Type	Description	R/C
propertySet	No	String	The Property Set in which the property's value is to be set. Example: "Demo Portal" **Note:** The property is set for the profile's default (unscoped) properties if no property set is provided.	R

Table 10-31 <um:setProperty>

Tag Attribute	Req'd	Type	Description	R/C
propertyName	Yes	String	The name of the property to be set. Example: "background_color"	R
value	Yes	Object	The new property value.	C

Example:

```
<% String myName = request.getParameter("name"); %>
<um:setProperty propertySet="exampleportal" propertyName="name"
value="<%=myName%>"/>
```

Group-User Management Tags

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<um:addGroupToGroup>

The `<um:addGroupToGroup>` tag (Table 10-32) adds the group corresponding to the provided `childGroupName` to the group corresponding to the provided `groupName`. Since a group can only have one parent, any previous database records which reflect the group belonging to another parent will be destroyed. Both the parent group and the child group must previously exist for proper tag behavior. The tag has no enclosed body.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-32 <um:addGroupToGroup>

Tag Attribute	Req'd	Type	Description	R/C
childGroupName	Yes	String	The name of the child group. Example: "<%=childGroupName%>"	R
parentGroupName	Yes	String	The name of the parent group. Example: "<%=parentGroupName%>"	R
result	Yes	String	The name of an Integer variable to which the result of the add group to group operation is assigned. Possible values: *Success*: UserManagerTagConstants.ADD_GROUP_OK *Error encountered*: UserManagerTagConstants.ADD_GROUP_FAILED	C

Example:

```
<um:addGroupToGroup childGroupName="<%=childGroupName%>"
parentGroupName="<%=parentGroupName%>" result="result"/>
```

<um:addUserToGroup>

The `<um:addUserToGroup>` tag (Table 10-33) adds the user corresponding to the provided `username` to the group corresponding to the provided `groupName`. Both the specified user and the specified group must previously exist for proper tag behavior. The tag has no enclosed body.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-33 <um:addUserToGroup>

Tag Attribute	Req'd	Type	Description	R/C
username	Yes	String	The name of the user to be added to the group. Example: "<%=username%>"	R
groupName	Yes	String	The name of the group to which the user is being added. Example: "<%=groupName%>"	R
result	Yes	String	The name of an Integer variable to which the result of the add user to group operation is assigned. Possible values: *Success:* UserManagerTagConstants.ADD_USER_OK *Error encountered:* UserManagerTagConstants.ADD_USER_FAILED	C

Example:

```
<um:addUserToGroup userName="<%=userName%>"
groupName="<%=groupName%>" result="result"/>
```

<um:changeGroupName>

The `<um:changeGroupName>` tag (Table 10-34) changes the name of the group corresponding to the specified `oldGroupName` to the specified `newGroupName`. This tag has no enclosed body.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-34 <um:changeGroupName>

Tag Attribute	Req'd	Type	Description	R/C
oldGroupName	Yes	String	The old group name. Example: "<%=oldGroupName%>"	R
newGroupName	Yes	String	The new group name. Example: "<%=newGroupName%>"	R
result	Yes	String	The name of an Integer variable to which the result of the change group name operation is assigned. Possible values: *Success:* UserManagerTagConstants.GROUP_CHANGE_OK *Error encountered:* UserManagerTagConstants.GROUP_CHANGE_FAILED	C

Example:

```
<um:changeGroupname oldGroupName="<%=oldGroupName%>"
newGroupName="<%=changeGroupName%>" result="result"/>
```

<um:createGroup>

The `<um:createGroup>` tag (Table 10-35) creates a new `com.beasys.commerce.axiom.contact.Group` object. This tag has no enclosed body.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-35 <um:createGroup>

Tag Attribute	Req'd	Type	Description	R/C
groupName	Yes	String	The name of the new group. Example: "<%=groupName%>"	R
id	No	String	A variable name to which the created Group object is available for the duration of the page's scope.	C
parentName	No	String	The name of the group to set as the parent of the new group.	R
result	Yes	String	The name of an Integer variable to which the result of the create group operation is assigned. Possible Values: *Success:* UserManagerTagConstants.CREATE_GROUP_OK *Error encountered:* UserManagerTagConstants.CREATE_GROUP_FAILED *A group with the specified group name already exists*: UserManagerTagConstants.GROUP_EXISTS	C

Example:

```
<um:creategroup groupName="<%=groupName%>" result="result"/>
```

<um:createUser>

The `<um:createUser>` tag (Table 10-36) creates a new `com.beasys.commerce.axiom.contact.User` object. This tag has no enclosed body. Although classified as a Group-User management tag, this tag can be used in conjunction with run-time activities, in that it will persist any properties associated with a current Anonymous User Profile if specified.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-36 <um:createUser>

Tag Attribute	Req'd	Type	Description	R/C
username	Yes	String	The name of the new user. Example: "<%=username%>"	R
password	Yes	String	The password for the new user. Example: "<%=password%>"	R
profileType	No	String	Specifies the extended type of user (for example, WLCS_Customer) to create a user of that type.	R
saveAnonymous	No	String	Whether to persist current anonymous user profile attributes in the newly-created user's profile. Defaults to `false`. Example: "`false`"	R
id	No	String	A variable name to which the created User object is available for the duration of the page's scope.	C
result	Yes	String	The name of an Integer variable to which the result of the create user operation is assigned. Possible values: *Success:* UserManagerTagConstants.CREATE_USER_OK *Error encountered*: UserManagerTagConstants.CREATE_USER_FAILED *A user with the specified username already exists*: UserManagerTagConstants.USER_EXISTS	C

Example:

```
<um:createUser userName="<%=username%>" password="<%=password"%>
result="result"/>
```

<um:getChildGroupNames>

The <um:getChildGroupNames> tag (Table 10-37) returns the names of any groups that are children of the given group.

Table 10-37 <um:getChildGroupNames>

Tag Attribute	Req'd	Type	Description	R/C
groupName	Yes	String	The name of the group to search for child groups.	R
id	Yes	String	The name of the identfier which will be assigned the String array of child group names.	C

<um:getChildGroups>

The <um:getChildGroups> tag (Table 10-38) retrieves an array of com.beasys.commerce.axiom.contact.Group objects that are children of the Group corresponding to the provided groupName. The information is taken from the personalization database tables, and reflects the group hierarchy information as set up from the Group Administration and Realm Configuration Administration Tools. This tag has no enclosed body.

Table 10-38 <um:getChildGroups>

Tag Attribute	Req'd	Type	Description	R/C
groupName	Yes	String	The name of the group whose children are sought. Example: "<%=groupName%>"	R

Table 10-38 <um:getChildGroups> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	A variable name to which the child Group objects are available for the duration of the page's scope. Example: "childGroups"	C

Example:

```
<um:getchildgroups groupName="<%=groupName%>" id="childGroups"/>
```

<um:getGroupNamesForUser>

The `<um:getGroupNamesForUser>` tag (Table 10-39) retrieves a `String` array that contains the group names corresponding to groups to which the provided user immediately belongs. This tag has no enclosed body.

Table 10-39 <um:getGroupNamesForUser>

Tag Attribute	Req'd	Type	Description	R/C
username	Yes	String	The name of the user whose matching groups are sought. Example: "<%=username%>"	R
id	Yes	String	A variable name to which the resultant group names are assigned. Example: "myGroups"	C

Example:

```
<um:getGroupNamesForUser userName="<%=username%>" id="myGroups"/>
```

<um:getParentGroupName>

The `<um:getParentGroupName>` tag (Table 10-40) retrieves the name of the parent of the `com.beasys.commerce.axiom.contact.Group` object associated with the provided `groupName`. The information is taken from the personalization database

tables, and reflects the group hierarchy information as set up from the Group Administration and Realm Configuration Administration Tools. This tag has no enclosed body.

Table 10-40 <um:getParentGroupName>

Tag Attribute	Req'd	Type	Description	R/C
groupName	Yes	String	The name of the group whose parent group name is sought. Example: "<%=groupName%>"	R
id	Yes	String	A variable name to which the name of the parent is available for the duration of the page's scope. Example: "parentGroupName"	C

Example:

```
<um:getParentGroupName groupName="<%=groupName%>"
id="parentGroupName"/>
```

<um:getTopLevelGroups>

The `<um:getTopLevelGroups>` tag (Table 10-41) retrieves an array of `com.beasys.commerce.axiom.contact.Group` objects, each of which has no parent group. The information is taken from the personalization database tables, and reflects the group hierarchy information as set up from the Group Administration and Realm Configuration Administration Tools. This tag has no enclosed body.

Table 10-41 <um:getTopLevelGroups>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	A variable name to which the top-level `Group` objects are available for the duration of the page's scope. Example: "topLevelGroups"	C

Example:

```
<um:getTopLevelGroups id="topLevelGroups"/>
```

<um:getUsernames>

The `<um:getUsernames>` tag (Table 10-42) retrieves a `String` array that contains the usernames matching the provided search expression. The search expression supports only the asterisk (*) wildcard character, and is case insensitive. As many asterisks as desired may be used in the search expression. This tag has no enclosed body.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-42 <um:getUsernames>

Tag Attribute	Req'd	Type	Description	R/C
searchExp	No	String	The search expression to apply to the user name search. Defaults to '*' Example: "t*"	R
userLimit	No	String (representing an Integer)	The maximum number of users to be returned from the search. (String which has a particular `Integer.valueOf.`) Defaults to 100. If user count exceeds userLimit, the length of the array in id is truncated to the length of userLimit. Example: "500"	R
id	Yes	String	A variable name to which the resultant user names are assigned. Example: "myUsers"	C

Table 10-42 <um:getUsernames> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
result	No	String	The name of an Integer variable to which the result of the `getUsernames` operation is assigned. Possible values: *Success:* UserManagerTagConstants.USER_SEARCH_OK *General error*: UserManagerTagConstants.USER_SEARCH_FAILED	C

Note: The `USER_SEARCH_FAILED` value is returned only when a general error occurs while searching for the user, such as a database connection failure. If no user matches the search criteria, the result will not be equal to `UserManagerTagConstants.USER_SEARCH_FAILED`, but the length returned by the array in `id` will be zero.

Example:

```
<um:getUsernames userLimit="500" searchExp="t*" id="myUsers"/>
<%System.out.println("I found " + myUsers.length + " users.");%>
```

<um:getUsernamesForGroup>

The `<um:getUsernamesForGroup>` tag (Table 10-43) retrieves a `String` array that contains the usernames matching the provided search expression and correspond to members of the provided group. The search expression supports only the asterisk (*) wildcard character, and is case insensitive. As many asterisks as desired may be used in the search expression. This tag has no enclosed body.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-43 <um:getUsernamesForGroup>

Tag Attribute	Req'd	Type	Description	R/C
searchExp	No	String	The search expression to apply to the user name search. Defaults to" *". Example: `"t*"`	R
groupName	Yes	String	The name of the group whose matching members are sought. Example: "engineering"	R
userLimit	No	String (representing an Integer)	The maximum number of users to be returned from the search. (String which has a particular `Integer.valueOf.`) Defaults to 100. If user count exceeds userLimit, the length of the array in id is truncated to the length of userLimit. Example: "500"	R
id	Yes	String	A variable name to which the resultant user names are assigned. Example: "myUsers"	C
result	No	String	The name of an Integer variable to which the result of the get usernames for group operation is assigned. Possible values: *Success:* UserManagerTagConstants.USER_SEARCH_OK *General error*: UserManagerTagConstants.USER_SEARCH_FAILED	C

Note: The `USER_SEARCH_FAILED` value is returned only when a general error occurs while searching for the user, such as a database connection failure. If no user matches the search criteria, the result will not be equal to `UserManagerTagConstants.USER_SEARCH_FAILED`, but the length returned by the array in `id` will be zero.

Example:

```
<um:getUsernamesForGroup groupName="engineering" userLimit="500"
searchExp="t*" id="myUsers"/>
<%System.out.println("I found " + myUsers.length + " users in my
group.");%>
```

<um:removeGroup>

The `<um:removeGroup>` tag (Table 10-44) removes the `com.beasys.commerce.axiom.contact.Group` object corresponding to the provided `groupName`. This tag has no enclosed body.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-44 <um:removeGroup>

Tag Attribute	Req'd	Type	Description	R/C
groupName	Yes	String	The name of the user to be removed. Example: "<%=groupName%>"	R
result	Yes	String	The name of an Integer variable to which the result of the remove group operation is assigned. Possible values: *Success:* UserManagerTagConstants.REMOVE_GROUP_OK *Error encountered:* UserManagerTagConstants.REMOVE_GROUP_FAILED	C

Example:

```
<um:removeGroup groupName="<%=groupName%>" result="result"/>
```

<um:removeGroupFromGroup>

The <um:removeGroupFromGroup> tag (Table 10-45) removes a child group from a parent group.

Table 10-45 <um:removeGroupFromGroup>

Tag Attribute	Req'd	Type	Description	R/C
childGroupName	Yes	String	The name of the child group to remove from its parent.	R
parentGroupName	Yes	String	The name of the parent group from which the child group will be removed.	R
result	Yes	String	The name of an Integer variable to which the result of the remove group from group operation is assigned. Possible values: *Success:* UserManagerTagConstants.REMOVE_GROUP_OK *Failure:* UserManagerTagConstants.REMOVE_GROUP_FAILED	C

<um:removeUser>

The <um:removeUser> tag (Table 10-46) removes the com.beasys.commerce.axiom.contact.User object corresponding to the provided username. It can remove any type of extended user that has its profileType set in the database. This tag has no enclosed body.

Note: This tag should only be invoked when the weblogic.security.realmClass property in the weblogic.properties file is com.beasys.commerce.axiom.contact.security.RDBMSRealm.

Table 10-46 <um:removeUser>

Tag Attribute	Req'd	Type	Description	R/C
username	Yes	String	The username of the user to be removed. Example: "<%=username%>"	R
result	Yes	String	The name of an Integer variable to which the result of the remove user operation is assigned. Possible values: *Success:* UserManagerTagConstants.REMOVE_USER_OK *Error encountered*: UserManagerTagConstants.REMOVE_USER_FAILED	C

Example:

```
<um:removeUser userName="<%=username%>" result="result"/>
```

<um:removeUserFromGroup>

The `<um:removeUserFromGroup>` tag (Table 10-47) removes a user from a group.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-47 <um:removeUserFromGroup>

Tag Attribute	Req'd	Type	Description	R/C
username	Yes	String	The username of the user to remove from the given group.	R

Table 10-47 <um:removeUserFromGroup> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
groupName	Yes	String	The name of the group from which the given user will be removed.	R
result	Yes	String	The name of an Integer variable to which the result of the remove user from group operation is assigned. Possible values: *Success:* UserManagerTagConstants.REMOVE_USER_OK *Failure:* UserManagerTagConstants.REMOVE_USER_FAILED	C

Security Tags

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<um:login>

The `<um:login>` tag (Table 10-48) provides weak authentication (username, password) against the current security realm, and sets the authenticated user as the current WebLogic user. This tag has no enclosed body.

Note: The login tag requires a `username` attribute and a `password` attribute to be present in the HTTP request.

Table 10-48 <um:login>

Tag Attribute	Req'd	Type	Description	R/C
result	Yes	String	The name of an Integer variable to which the result of the login operation is assigned. Possible values: *Success:* UserManagerTagConstants.LOGIN_OK *General error when performing authentication*: UserManagerTagConstants.LOGIN_ERROR *Authentication failed because of invalid username/password combination*: UserManagerTagConstants.LOGIN_FAILED	C

<um:logout>

The `<um:logout>` tag (Table 10-49) ends the current user's WebLogic Server session. This is independent of the FlowManager's user session tracking, and should be used in combination with the `<um:login>` tag.

Table 10-49 <um:logout>

Tag Attribute	Req'd	Type	Description	R/C
No attributes				

<um:setPassword>

The `<um:setPassword>` tag (Table 10-50) updates the password for the user corresponding to the provided username.

Note: This tag should only be invoked when the `weblogic.security.realmClass` property in the `weblogic.properties` file is `com.beasys.commerce.axiom.contact.security.RDBMSRealm`.

Table 10-50 <um:setPassword>

Tag Attribute	Req'd	Type	Description	R/C
username	Yes	String	The username of the user whose password is to be changed.	R
password	Yes	String	The new user password.	R
result	No	String	The name of an Integer variable to which the result of the set password operation is assigned. Possible values: *Success*: UserManagerTagConstants.SET_PASSWORD_OK *Failure*: UserManagerTagConstants.SET_PASSWORD_FAILED	C

Personalization Utilities

The <es:jsptaglib> tag contains generic tags you can use to create JSP pages.

Use the following code to import the utility tag library:

```
<%@ taglib uri="es.tld" prefix="es" %>
```

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<es:counter>

The <es:counter> tag (Table 10-51) is used to create a for loop.

Table 10-51 <es:counter>

Tag Attribute	Req'd	Type	Description	R/C
type	No	String	The type of the counter. Possible values are int or long. Default is int.	R
id	Yes	String	A unique name for the variable.	R
minCount	Yes	Int	The start position for the loop.	R
maxCount	Yes	Int	The end position for the loop.	R

Example:

```
<es:counter id="iterator" minCount="0" maxCount="10">
   <% System.out.println(iterator);%>
</es:counter>
```

<es:date>

The <es:date> tag (Table 10-52) is used to get a date- and time-formatted String based on the user's time zone preference.

Table 10-52 <es:date>

Tag Attribute	Req'd	Type	Description	R/C
timeZoneId	No	String	Defaults to the time zone on the server.	R
formatStr	No	String	A date and time format string that adheres to the java.text.SimpleDateFormat. The default value is MM/dd/yyyy HH:mmss:z.	R

Example:

```
<es:date formatStr="MMMM dd yyyy" timeZoneId="MST" />
```

<es:forEachInArray>

The <es:forEachInArray> tag (Table 10-53) is used to iterate over an array.

Table 10-53 <es:forEachInArray>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	The variable for each value in the array.	R
type	Yes	String	The type of each value in the array.	R
array	Yes	Object []	The array to iterate over.	R
counterId	No	String	The position in the array.	R

Example:

```
<es:forEachInArray id="item" array="<%=items%>" type="String"
counterId="i">
```

```
     <% System.out.println("items[" + i + "]: " + item);%>
</es:forEachInArray>
```

<es:isNull>

The <es:isNull> tag (Table 10-54) is used to check if a value is null. In the case of a String, the <es:isNull> tag is used to check if the String is null or has a value. An empty string will cause isNull to be false. (An empty string is not null.)

Table 10-54 <es:isNull>

Tag Attribute	Req'd	Type	Description	R/C
item	Yes	Object	The variable to evaluate.	R

Example:

```
<es:isNull item="<%=value%>">
     Error: the value is null.
</es:isNull>
```

<es:monitorSession>

The <es:monitorSession> tag (Table 10-55) can be added to the beginning of any JSP page to disallow access to the page if the session is not valid or if the user is not logged in.

Table 10-55 <es:monitorSession>

Tag Attribute	Req'd	Type	Description	R/C
goToPage	No	String	The error page that you want displayed if the page is not accessible. The default value is portalerror.jsp.	R

Table 10-55 <es:monitorSession>

Tag Attribute	Req'd	Type	Description	R/C
loginRequired	No	String	Indicates whether the user is required to be logged in to access the JSP page including the tag. The default value is `false`.	R

Example:

```
<es:monitorSession loginRequired="true" />
```

<es:notNull>

The `<es:notNull>` tag (Table 10-56) is used to check if a value is not null. In the case of a `String`, the `<es:notNull>` tag is used to check if the `String` is not null or has a value. An empty string will cause `notNull` to be `true`. (An empty string is treated as a value.)

Table 10-56 <es:notNull>

Tag Attribute	Req'd	Type	Description	R/C
item	Yes	Object	The variable to evaluate.	R

Example:

```
<es:notNull item="<%=value%>">
     The value is not null.
</es:notNull>
```

<es:preparedStatement>

The <es:preparedStatement> tag (Table 10-57) is used to create a JDBC prepared statement.

Table 10-57 <es:preparedStatement>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	The variable in which the PreparedStatement is returned.	R
sql	Yes	String	The SQL query statement.	R
transactionIsolationLevel	No	Integer	Used to define isolation levels. Possible values: 0 = TRANSACTION_NONE 1 = TRANSACTION_READ_UNCOMMITTED 2 = TRANSACTION_READ_COMMITTED 4 = TRANSACTION_REPEATABLE_READ 8 = TRANSACTION_SERIALIZABLE Defaults to 2.	R

Example:

```
<es:preparedStatement id="ps" sql="<%=bookmarkBean.QUERY%>">
<%@ include file="startPreparedStatement.inc" %>
<%
bookmarkBean.createQuery(ps, owner);
java.sql.ResultSet resultSet = ps.executeQuery();
bookmarkBean.load(resultSet);
%>
<%@ include file="endPreparedStatement.inc" %>
</es:preparedStatement>
```

<es:simpleReport>

The <es:simpleReport> tag (Table 10-58) is used to create two-dimensional array out of a simple query.

Table 10-58 <es:simpleReport>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	The variable that holds the resultant two-dimensional array converted from the java.sql.ResultSet specified by the resultSet tag attribute.	R
resultSet	Yes	java.sql.ResultSet	The result set that holds the java.sql.ResultSet.	R

Example:

```
<es:simpleReport id="report" resultSet="<%=resultSet%>">
         <%
            for (int i=0; i<report.length; i++ )
            {
               for (int j=0; j<report[i].length; j++ )
               {
                  ...
               }
            }
         %>
   </es:simpleReport>
```

<es:transposeArray>

The <es:transposeArray> tag (Table 10-59) is used to transpose a standard [row][column] array to a [column][row] array.

Table 10-59 <es:transposeArray>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	The variable that holds the [c][r] array.	R
type	Yes	String	The type of variable in the [r][c] array, such as String.	R
array	Yes	Object[][]	The variable that holds the [r][c] array.	R

Example:

```
<es:transposeArray id="byColumnRow" array="<%=byRowColumn%>"
type="String">
   ...
</es:transposeArray>
```

<es:uriContent>

The <es:uriContent> tag (Table 10-60) is used to pull content from a URL. It is best used for grabbing text-heavy pages.

Table 10-60 <es:uriContent>

Tag Attribute	Req'd	Type	Description	R/C
id	Yes	String	The variable that holds the downloaded content of the URI.	R
uri	Yes	String	The fully qualified URI from which to get the content.	R

Example:

```
<es:uriContent id="uriContent"
uri="http://www.beasys.com/index.html">
<%
   out.print(uriContent);
```

```
%>
</es:uriContent>
```

Note: If you combine HTML pages with relative URL's, you must fully qualify them to the correct host in each URL, or else images (on other resources) may not be retrieved properly by the browser.

WebLogic Utilities

The <wl:jsptaglib> tag library contains custom JSP extension tags which are supplied as a part of the WebLogic server platform.

To import the WebLogic Utilities JSP tags, use the following code:

```
<%@ taglib uri="weblogic.tld" prefix="wl" %>
```

Note: In the following tables, Req'd specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

Note: Refer to the *Javadoc* for further descriptions of the wl tags.

<wl:process>

The <wl:process> tag (Table 10-61) is used for query attribute-based flow control. By using a combination of the four attributes, you can selectively execute the statements between the <wl:process> and </wl:process> tags.

Table 10-61 <wl:process>

Tag Attribute	Req'd	Type	Description	R/C
name	No	String	The name of a query attribute.	R
notName	No	String	The name of a query attribute.	R
value	No	String	The value of a query attribute.	R
notValue	No	String	The value of a query attribute.	R

Statements between the <wl:process> tags will be executed according to the matrix below:

	Value	notValue	Neither "value" nor "notValue"
name	Named attribute is equal to the value.	Named attribute does not equal the value.	Name attribute's value is not null.
not Name			notName attribute's value is null.

Example:

```
<wl:process name="lastBookRead" value="A Man in Full">
<!-- This section of code will be executed
   if lastBookRead exists and the value of lastBookRead is
   "A Man in Full" -->
</wl:process>
```

<wl:repeat>

The `<wl:repeat>` tag (Table 10-62) is used to iterate over a variety of Java objects, as specified in the set attribute.

Table 10-62 <wl:repeat>

Tag Attribute	Req'd	Type	Description	R/C
set	No	Object	The set of objects that includes: ■ Enumerations ■ Iterators ■ Collections ■ Arrays ■ Vectors ■ Result Sets ■ Result Set MetaData ■ Hashtable keys	R
counter	No	Int	Iterate over first "count" entries in the set.	R
id	No	String	Variable to contain current object being iterated over.	C
type	No	String	Type of object that results from iterating over the set you passed in. Defaults to Object. This type must be fully qualified.	C

Index

D

E

F

G

H

I

J

L

Q

R

S

T

U

W