BEA WebLogic Personalization Server Developer's Guide

BEA WebLogic Personalization Server 2.0.1
Document Edition 2.0.3 Service Pack 3
May 2001

Copyright

Copyright © 2000 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

BEA, ObjectBroker, TOP END, and Tuxedo are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Connect, BEA Manager, BEA MessageQ, BEA Jolt, M3, eSolutions, eLink, WebLogic, WebLogic Enterprise, WebLogic Commerce Server, and WebLogic Personalization Server are trademarks of BEA Systems, Inc.

All other company names may be trademarks of the respective companies with which they are associated.

WebLogic Personalization Server Developer's Guide

Document Edition	Date	Software Version
1.0	January 2000	BEA WebLogic Personalization Server 1.7
1.1	February 2000	BEA WebLogic Personalization Server 1.7.1
2.0	April 2000	BEA WebLogic Personalization Server 2.0
2.0.1 Service Pack 1	July 2000	BEA WebLogic Commerce Servers 2.0.1 Service Pack 1
2.0.3 Service Pack 3	May 2001	BEA WebLogic Commerce Servers 2.0.1 Service Pack 3

Contents

3. Foundation Classes and Utilities

4. JSP Tag 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 covers the following topics:

- **Overview of Personalization Development**: WebLogic Personalization Server 2.0 provides developer components and utilities that enable developers to create personalized applications. The pieces documented in in this guide include the Personalization Advisor, Foundation classes and utilities, and JSP tag reference.
- **Creating Personalized Applications with the Personalization Advisor**: Personalization 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.
- **Foundation Classes and Utilities**: The Foundation is a set of miscellaneous utilites 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 Personalization Advisor services.
- **JSP Tag Reference**: The JSP tags included with WebLogic Personalization Server 2.0 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 eCommerce site using BEA WebLogic Personalization Server. It assumes a familiarity with the WebLogic Personalization Server platform and related Web technologies as described below. The topics in this document are organized primarily around development goals and the tasks needed to accomplish them. Generally, a set of topics also speaks to a particular development role and requires the basic knowledge with regard to the technology focus of that role:

- *Java Server Page (JSP) developer* creates JSPs using the tags provided or by creating custom tags as needed.
- *Application assembler*, *system analyst*, or *systems integrator* writes rules, writes, schemas, 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 Personalization Server engine, if that level of customization is needed.

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 that you are using the documentation for the BEA WebLogic Personalization Server 2.0 release.

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.

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

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*

Convention	Item
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.
`...`	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

The following topics are included:

Personalization Server Runtime Architecture
- Personalization Advisor
- Portal Management
- Foundation Classes and Utilities
- User Management
- Content Management
- Rules Management

Foundation Classes and Utilities

JSP Tags

Integration of External Components

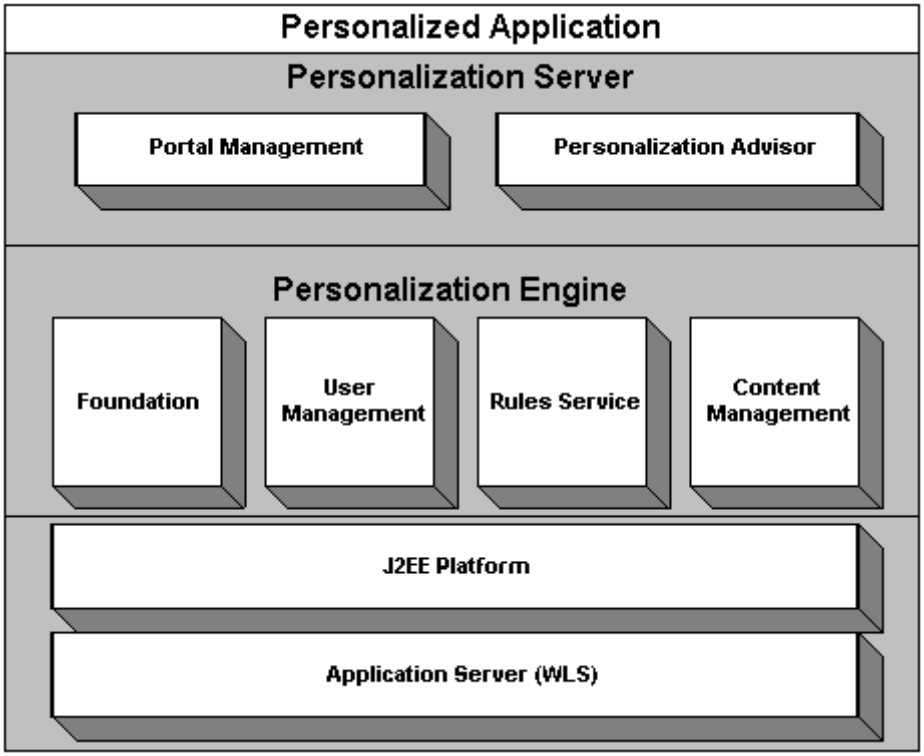
Support for Native Types

WebLogic Personalization Server provides developers with the ability to create personalized applications for eCommerce web sites. This developer's guide provides information about the Personalization Advisor, Foundation Classes and Utilities, and JSP Tags.

Personalization Server Runtime Architecture

The WebLogic Personalization Server (WLPS) runtime 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 personalization 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.

Personalization Advisor

The personalization 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 personalization advisor includes classifying users, selecting content based on user properties, and querying content management directly. The personalization 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.

Foundation Classes and Utilities

The foundation component provides a set objects and utilities to support personalization activities. This component provides EJB objects to support HTTP handling, including object definition for request and session objects, as well as object-based utilities specifically designed to support portals.

User Management

The user management component supports the runtime 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 runtime 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 component is the runtime service that runs the rulesheets that are built in the rules management administration tool. This component is accessible only via the functionality of the personalization advisor tags. This component uses the JRules runtime 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 Personalization Advisor services.

JSP Tags

The JSP tags included with WebLogic Personalization Server 2.0 allow developers to create personalized applications without having to program using Java.

Table 1-1 Java Server Page (JSP) Tags Overview

Library	Tag	Description
Personalization Advisor	`<pz:div>`	Turns a user-provided piece of content on or off based on the results of a classifier rule.
	`<pz:contentquery>`	Provides content based on search expression query syntax.
	`<pz:contentselector>`	Provides content based on results of a contentselector rule and subsequent content query.
Content Management	`<cm:select>`	Selects content based on a search expression query syntax.
	`<cm:selectbyid>`	Retrieves content using the content's unique identifier.
	`<cm:printproperty>`	Inlines the value of the specified content metadata property as a string.
	`<cm:printdoc>`	Inlines the raw bytes of a document object in to the JSP out put stream.
Portal Management	`<pt:portalmanager>`	Provides the ability to do `create`, `get`, `getColumnInfo`, `update`, and `remove` actions on a Portal object.

Table 1-1 Java Server Page (JSP) Tags Overview

	`<pt:portletmanager>`	Provides the ability to do `create`, `get`, `getArranged`, `update`, and `remove` actions on a Portlet object.
	`<pt:eval>`	Evaluates a conditional attribute of a portlet. An example of a conditional attribute is `isMinimizeable`.
	`<pt:get>`	Retrieves a String attribute of a portlet.
	`<pt:monitorsession>`	Disallows access to a page if the session is not valid or if the user has not logged in.
User Management	`<um:login>`	Authenticates a user/password combination.
	`<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:setproperty>`	Sets a new value for the specified property for the current user profile in the session.
	`<um:removeproperty>`	Removes the property from the current user profile in the session.
	`<um:createuser>`	Creates a new persisted User object with the specified user name and password.
Personalization Utilities	`<es:condition>`	Evaluates a Boolean expression and executes the body if true.
	`<es:counter>`	Creates a `for loop` construct.
	`<es:foreachinarray>`	Iterates over an array.

Table 1-1 Java Server Page (JSP) Tags Overview

	`<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: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.
	`<es:uricontent>`	Pulls content from a URL.
	`<es:date>`	Gets a date and time formatted string based on the user's time zone preference.
	`<es:usertransaction>`	Wraps database code within a single transaction.
WebLogic Utilities	`<wl:process>`	Provides a parameter-based flow control construct.

Integration of External Components

A range of external components either come already integrated into the 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 the following table. Other custom component integrations are possible given the JSP and EJB basis for the Personalization Server, but the entire range of possibilities is not addressed here.

Table 1-2 Useful External Components for 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.	Standard WebLogic Server JDBC connection pools used.
LDAP authentication	Can be set up automatically using administration tools and property files.	Uses WebLogic Server security realms.
LDAP retrieval of user and group info	Can be set up automatically using administration tools.	Built into EJB persistence for User entity bean.
Legacy database of users	None.	Requires Unified User Profile extension of User entity bean.
Content management engine	Reference implementation provided.	API/SPI support from Documentum provided.
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

WLPS supports the native types shown in the following table.

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
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.

CHAPTER

2 Creating Personalized Applications with the Personalization Advisor

The following topics are included:

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

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

What is the Personalization Advisor?

The Personalization Advisor delivers content to a personalized application based on a set of rules and user profile information. The Personalization Advisor, a stateless session bean that gives personalized advice and recommendations, can retrieve any type of content in a Document Management system and display it with a JSP page or use it in a servlet. In the WLPS 2.0 release, it gives advice on user classifications and recommends content.

The Personalization Advisor provides access to dynamic personalization functionality through JSP tags and a stateless session bean. It is a key component of the WebLogic Personalization Server because it ties together the core personalization server services that include:

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

The Personalization Advisor component includes a JSP tag library and a Personalization Advisor EJB stateless session bean that accesses the core personalization services. 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 Personalization Advisor JSP tags. In addition, a Java developer can access the underlying Personalization EJB and its features via the public Personalization Advisor bean interface (see the API documentation for more information). Think of the Personalization Advisor as sitting on top of the core services to provide a unified personalization API.

Personalization 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 developer.

In addition, Personalization Advisor provides information about user classifications. For example, an application can ask Personalization Advisor if, based on pre-defined 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.

For WebLogic Personalization Server 2.0, Personalization Advisor recommends document content for the following items:

- `Documents` returned by content selectors using rules-based matching against user profile information. See "Matching content to users with the <pz:contentselector> JSP tag" on page 2-6 for more information about rules-based matching.
- Web content included or excluded as determined by a user's classification using rules-based matching against user profile information. See "Classifying users with the JSP <pz:div> tag" on page 2-5 for more information about classifying users.
- `Documents` returned by document attribute searches. See "Selecting content with the <pz:contentquery> JSP tag" on page 2-5 for more information about searching for content.

You get advice from Personalization 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 on and off content 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:contentquery>`, and `<pz:contentselector>`
- **Using the Personalization Advisor session bean**. The Personalization Advisor session bean recommends content to personalized applications by matching advice requests with registered personalization agents that perform recommendations. The page or application developer may use the PersonalizationAdvisor Session bean directly in place of the tags, if desired.

Creating Personalized Applications with JSP Tags

Personalization Advisor provides three JSP tags to help developers create personalized applications. These tags provide a JSP view to the Personalization 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 Personalization Advisor's `<pz:div>` and `<pz:contentselector>` tags. The `<pz:contentquery>` tag does not require you to 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>` 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: See Creating and Managing Content for more information about how WLPS manages content.

- The `<pz:contentselector>` 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: See Creating a content selector rule for information about defining a content selector rule.

In addition to using JSP tags to create personalized applications, you can work directly with the advisor bean. See "Creating Personalized Applications with the Personalization Advisor Session Bean" on page 2-7 for more information about using the bean.

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: See Creating a classifier rule for information about creating classifier rules.

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="ejb://com.beasys.commerce.axiom.reasoning.rules.Ruleshee
tDefinitionHome/AcmeRules" rule="PremierCustomer">
    <p>Please check out our new Premier Customer bonus program…<p>
</pz:div>
```

You can also use the advisor bean to classify users. See "Classifying users with the Personalization Advisor Session Bean" on page 2-10 for more information about using the bean to classify users.

Note: The terms *rulesheet* and *ruleset* refer to the same object and are used interchangeably throughout this documentation.

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: See“<pz:contentquery>” on page 4-4 for information about using the <cm:select> JSP tag. This tag currently provides similar functionality to the <cm:select> tag.

This 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: See “<cm:printproperty>” on page 4-12 and “<es:foreachinarray>” on page 4-39 for more information about the <cm:printproperty> and <es:foreachinarray> JSP tags.

You can also use the advisor bean to select content. See “Selecting content with the Personalization Advisor Session Bean” on page 2-12 for more information about using the bean to select content.

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 Personalization Advisor executes the query defined in the rule to retrieve content. See “<pz:contentselector>” on page 4-5 for more information about the <pz:contentselector> tag.

Note: See Creating a content selector rule for more information about using content selector rules.

This example asks the Personalization 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="ejb://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 to match content to users. See “Matching content to users with the Personalization Advisor Session Bean” on page 2-14 for more information about using the bean to match content to users.

Creating Personalized Applications with the Personalization 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. Refer to the API documentation for more information about using the session bean to create personalized applications.

The Personalization Advisor’s agent implementation encapsulates all of the logic and operations required for the request types it handles. It makes recommendations and returns a type of `AdviceResults` to the Personalization Advisor. In general, the process includes the following steps:

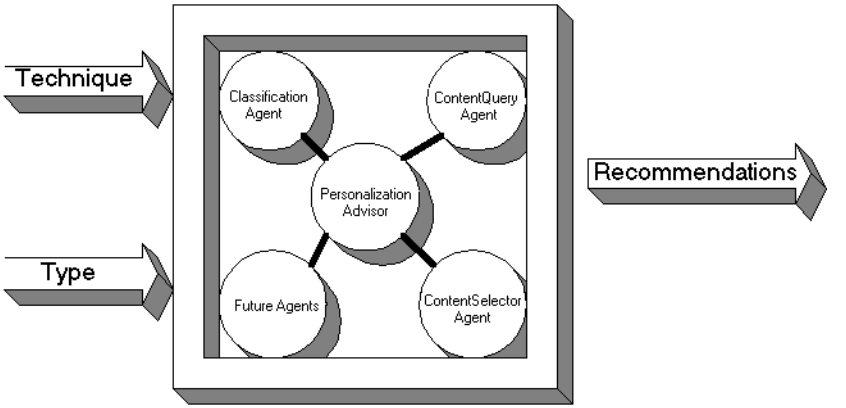
1. Create an instance of the Personalization Advisor Session bean.
2. Use the Personalization Advisor's `createTemplate` factory method to create a Request object.

   This method also determines the best Advice Agent to use for the request by mapping the `AdviceRequestClassName` and technique to the best fit Agent. The technique name uses the technique name parameter in the JSP tags `<pz:bea.rules>` and `<pz:bea.query>`. See "Specifying a personalization technique URL" on page 2-9 for more information about specifying the request technique.
3. Set the required and optional inputs for the `Request` object.
4. Call the `advise` method.

   The Personalization Advisor calls the best agent to make the recommendation. The agent determines the recommendations and the Personalization 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 Personalization Advisor, the Personalization Advisor bean delegates the request to a registered personalization agent that can handle the request. The Personalization Advisor's job is to determine which registered personalization agent is best suited for making recommendations for the request, based on the advice request type and the personalization technique.

The Personalization Advisor uses the advice request type and a personalization technique identifier to determine which registered personalization agent to delegate the advice request to. The agent then makes the recommendations and returns the advice results back to the Personalization Advisor. This design encapsulates all of the advice logic into the agent and allows agents to be specialized.



Specifying a personalization technique URL

To execute a recommendation request, the Personalization Advisor requires that you specify a personalization technique URL. Pass in a personalization technique identifier that specifies the personalization namespace, vendor, and technique. The personalization technique identifier follows the JDBC driver specification:

```
namespace:vendor.technique
```

- The namespace represents the personalization namespace. The Personalization Advisor only accepts the *pz* namespace in WLPS 2.0.
- The vender is the name of the vendor supplying the personalization technique. For the 2.0 release, *bea* is the vendor name.
- The technique is the name given to the Personalization Advisor to determine which Personalization Agent can best perform the recommendation request.

Note: For the 2.0 release, the Personalization Advisor accepts the following personalization technique names:

- *rules*
- *query*

The table shows the logic the Personalization Advisor uses to determine how to map a recommendation request to a Personalization Agent. Note that some combinations are not valid. For example, you cannot send a <pz:bea.rules> technique request with a `ContentQueryAdviceRequest`.

Personalization Technique	Advice Request Type	Inferred Personalization Agent
pz:bea.rules	ClassificationAdviceRequest	ClassificationAgent Uses rules-based matching with an inference engine that classifies a user.
pz:bea.rules	ContentSelectorAdviceRequest	ContentSelectorAgent ■ 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.
pz:bea.query	ContentQueryAdviceRequest	ContentQueryAgent Performs a content attribute search with a content management system.

Classifying users with the Personalization Advisor Session Bean

For classification requirements beyond what the JSP tags provide, or to use classification in a servlet, developers can use the Personalization Advisor EJB directly. The following sequence describes the process of asking the Personalization Advisor for a classification (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 Personalization Advisor Session bean.
2. Call the Personalization 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 `P13NJspBase.createP13NSession(HttpServletSession))`
   - `User` object (retrieved from `P13NJspBase.createP13NProfile(HttpServletSession))`
   - `Request` object (retrieved from `P13NJspBase.createP13NRequest(HttpServletSession))`
   - `java.sql.Timestamp` object representing *now*
   - rulesheet name (see What are Rulesheets? for more information)
   - rule name (see Creating and Managing Rules for more information)
   - optional `Successor` object (for example, the user's group).
4. Call the `advise` method on the Personalization Advisor.
5. The Personalization 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/p13nadvisor/ClassificationExample.java`

```
try
{
    ClassificationAdviceRequest request = null;
    AdviceRequest arequest = anAdvisor.createRequestTemplate
        ("com.beasys.commerce.axiom.p13n.agents.ClassificationAdviceRequest”,
        "pz:bea.rules");
```

```
    request = (ClassificationAdviceRequest)arequest;
    HttpServletRequest someRequest = (HttpServletRequest)
        pageContext.getRequest();
    P13NJspBase page = (P13NJspBase)pageContext.getPage();
    ConfigurableEntity user = page.createP13NProfile(httpRequest);
    request.setUser(user);

    request.setHttpRequest(page.createP13NRequest(httpRequest));
    request.setHttpSession(page.createP13NSession(httpRequest));
    request.setNow(new Timestamp(System.currentTimeMillis()));
    request.setRuleSheet(rulesheet);
    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. See "Classifying users with the JSP <pz:div> tag" on page 2-5 for more information about using the tag to classify users.

Selecting content with the Personalization Advisor Session Bean

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

1. Create an instance of the Personalization Advisor Session bean.
2. Call the Personalization 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 Personalization Advisor.
5. The Personalization 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(
        "com.beasys.commerce.axiom.p13n.agents.ContentQueryAdviceRequest",
        "pz:bea.query");

    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. See "Selecting content with the <pz:contentquery> JSP tag" on page 2-5 for more information about using the tag to select content.

Matching content to users with the Personalization Advisor Session Bean

For content matching requirements beyond what the JSP tags provide, or to use classification in a servlet, developers can use the Personalization Advisor EJB directly. The following sequence describes the process of asking the Personalization 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 Personalization Advisor Session bean.
2. Call the Personalization 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 P13NJspBase.createP13NSession(HttpServletSession))
   - `User` object (retrieved from P13NJspBase.createP13NProfile(HttpServletSession))
   - `Request` object (retrieved from P13NJspBase.createP13NRequest(HttpServletSession))
   - `java.sql.Timestamp` object representing *now*
   - rulesheet name (see What are Rulesheets? for more information)
   - rule name (see Creating and Managing Rules for more information)
   - optional `Successor` object (e.g., 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 Personalization Advisor.

5. The Personalization 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 that this code is just a model and not complete):

Note: This code is just a model and is not complete. The complete example resides in the following files:
`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/p13nadvisor/ContentSelectorExample.java`

```
try
{
    AdviceRequest arequest = anAdvisor.createRequestTemplate
        ("com.beasys.commerce.axiom.p13n.agents.ContentSelectorAdviceRequest",
        "pz:bea.rules");

    request = (ContentSelectorAdviceRequest)arequest;
    HttpServletRequest someRequest =
        (HttpServletRequest)pageContext.getRequest();
    P13NJspBase page = (P13NJspBase)pageContext.getPage();
    ConfigurableEntity user = page.createP13NProfile(someRequest);

    request.setUser(user);
    request.setHttpRequest(page.createP13NRequest(someRequest));
    request.setHttpSession(page.createP13NSession(someRequest));
    request.setNow(new Timestamp(System.currentTimeMillis()));
    request.setRuleSheet(rulesheet);
    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. See "Matching content to users with the <pz:contentselector> JSP tag" on page 2-6 for more information about using the tag to match content to users.

CHAPTER

3 Foundation Classes and Utilities

The following topics are included:

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 Personalization Advisor services.

JSP Service Manager

The JSP Service Manager provides several benefits to JSP developers of non-portal pages. The JSP Service Manager servlet, much like the Portal Service Manager, primarily acts as a traffic cop for the JSP code, including verifying and updating relevant HTTP session information, providing access to useful methods, and providing security services for requests.

Note: Use of the JSP Service Manager in a personalized application is optional, but highly recommended.

The JSP Service Manager works in conjunction with the `P13nJspBase` class, initializing session data accessed via the `P13nJspBase` class methods. JSP pages will typically extend `P13nJspBase`.

The three main benefits of the JSP Service Manager include:

1. providing a common set of useful methods to JSP files so you do not have to replicate code from one page to another. For example, the Service Manager monitors sessions and handles time-outs.
2. validating the destination of requested JSP files. This security mechanism provides error handling for invalid page probing.
3. using a common file repository for JSP and/or image files. The Service Manager uses the repository specified in the weblogic.properties file.

Note: For example usage of the JSP Service Manager and the Base class methods, see the `mpbb servlet` and `buybeans/home.jsp`.

Configuring the JSP Service Manager

Register an instance of this servlet in the `weblogic.properties` file for each Service Manager you deploy. The following is a sample JSP Service Manager servlet registration for a servlet named *myServlet*.

```
weblogic.httpd.register.myServlet=com.beasys.commerce.axiom.
        jsp.JspServiceManager
    weblogic.httpd.initArgs.myServlet=\
    homepage=/mydir/home.jsp,\
    defaultdest=/mydir/home.jsp,\
    workingdir=/mydir/,\
    repositorydir=/repository/,\
    timeout=-1,\
    sessioncomparator=com.beasys.commerce.axiom.jsp.
        DefaultSessionComparator,\
    groupname=everyone,\
    allowautologin=true
```

Parameter Name	Required	Description
homepage	yes	The home page JSP returned by the system in auto-login. (This page is qualified from *`yourDocumentRoot`* as defined in the `weblogic.properties` file.) Example: homepage=/servlets/myServlet/home.jsp
groupname	no	The default group name for this servlet instance.
defaultdest	yes	The default destination page JSP if there is not a valid session for the user.

timeout	no	Time-out for the cookies or session valued in seconds and defaulting to -1. If set to -1, the cookies expire upon exiting the browser. If cookies are disabled, the session invalidates upon browser exit. To retain user login information between browser sessions, set the time-out to a large positive number, such as *999999*, and *set autologin=true.*
workingdir	yes	The working directory for the implementation that is the path to your JSP pages. Example: `workingdir=/servlets/myPortal/`
allowautologin	no	Determines whether a client with valid cookies can automatically login. The default is false.
repositorydir	yes	Location of default files, (gifs, JSP, etc.)
sessioncomparator	yes	How to determine if the session is valid.

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.

HTTP Handling

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

The Personalization 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 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 API documentation for more information.

The tag implementations for the `<pz:div>` and `<pz:contentselector>` tags create the Personalization `Request` surrogate for the `HttpRequest` before calling the Personalization 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, WLPS 2.0 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:

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()
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 API documentation)). While the `HttpSession` object can be wrapped by directly calling the Personalization `Session` constructor, it is recommend that developers use the `createP13NSession` helper method on `P13NJspBase` (`com.beasys.commerce.axiom.p13n.jsp.P13NJspBase`)See the 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 Personalization 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. WLPS 2.0 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 “Configuring the JSP Service Manager” on page 3-3) 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 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

P13NJspBase

`P13NJspBase` provides convenience methods to developers writing JSP pages (including but not limited to portals and portlets) that included 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, etc.) for an application.

Note that the Personalization `<pz:div>` and `<pz:contentselector>` tags require that the pages they are included in be subclasses of `P13NJspBase`.

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 JSP Tag Reference

The following topics are included:

<um:changegroupname>
<um:creategroup>
<um:createuser>
<um:getchildgroups>
<um:getgroupnamesforuser>
<um:getparentgroupname>
<um:gettoplevelgroups>
<um:getusernames>
<um:getusernamesforgroup>
<um:removegroup>
<um:removeuser>

Security tags
<um:login>
<um:setpassword>

Personalization Utilities
<es:condition>
<es:counter>
<es:foreachinarray>
<es:isnull>
<es:notnull>
<es:preparedstatement>
<es:simplereport>
<es:transposearray>
<es:uricontent>
<es:date>
<es:usertransaction>

WebLogic Utilities
<wl:process>

The JSP tags included with WebLogic Personalization Server 2.0 allow developers to create personalized applications without having to program using Java.

Personalization Advisor

To import the Personalization Advisor JSP tags, use the following code:

```
<%@ taglib uri="pz.tld" prefix="pz" %>
```

<pz:div>

The `<pz:div>` tag 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 agent. 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.

Tag Attribute	Required	Description
ruleset	yes	The URI for the rule set that contains the Classifier rule. For release 2.0, support exists only for the EJB protocol for locating a ruleset. Example: `ruleset= "ejb://RuleSetDefinitionHome/AcmePortal/rulesets/ruleset1.xml"`
rule	yes	The rule is the name of the classifier rule in the ruleset that the rules agent uses to classify the user.
id	no	The variable name that will be placed in the classification object.

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>
```

<pz:contentquery>

The <pz:contentquery> tag performs a content attribute search for content in a content manager. 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 Personalization 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 (see "<es:foreachinarray>" for more information) 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" for more information.)

Tag Attribute	Required	Description
max	no	Limits the maximum number of content items returned. If not present, it returns all of the content items found.
sortby	no	A list of document attribute to sort the content by. 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"
query	yes	A content query string used to search for content. Example: query="mimetype contains 'text' && author='Proulx'"
contenthome	yes	The name of the content home bean. This maps to a JNDI name for the content home that handles a specific type of content (e.g., documents) and its provider (Documentum vs. Interwoven).

id	yes	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.

Example:

```
<%@ 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>
```

<pz:contentselector>

The `<pz:contentselector>` tag allows arbitrary personalized content to be recommended based on a content selector rule. A content selector rule determines if the user matches the classification part of content selector rule. If a match, then a content query is performed based on the query in the rule.

Note: The terms *rulesheet* and *ruleset* refer to the same object and are used interchangeably throughout this documentation.

The ruleset URI protocol is as follows:

```
protocol://RuleSetDefinition-home-JNDI-name/RuleSheet-Name
```

- where *`protocol`* is *ejb*,
- *`RuleSetDefinition-home-JNDI-name`* is `com.beasys.commerce.axiom.reasoning.rules.RuleSheetDefinitionHome`, which is the EJB home name of the RuleSheet definition home, and
- *`ruleset-primary-key`* is the unique identifier for the rule set.

Example:

```
ejb://com.beasys.commerce.axiom.reasoning.rules.RuleSheetDefinitionHome/AcmeRules
```

The `<pz:contentselector>` tag only has a begin tag and does not have a body or end tag. It returns an array of `Document` objects as determined by the Personalization 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 (see "<es:foreachinarray>" for more information) 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" for more information.)

Tag Attribute	Required	Description
ruleset	yes	The URI for the rule set that contains the ContentSelector rule. For release 2.0, support exists only for the EJB protocol for locating a rule set. Example: `ruleset= "ejb://RuleSetDefinitionHome/AcmePortal/ruleset/ruleset1.xml"`
rule	yes	The rule is the name of the contentSelector rule in the rule set to be used by the rules agent to recommend content based on the rule.
max	no	Limits the maximum number of content items returned. If not present, it returns all of the content items found.

sortby	no	A list of document attribute to sort the content by. 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"
query	no	A content query string that can be appended as an and phrase to the content query in the contentSelector rule. This allows the JSP developer to add a runtime query to the content selector rule. Example: query="mimetype contains like 'text/*'"
contenthome	yes	The name of the content home bean. This maps to a JNDI name for the content home that handles a specific type of content (documents vs. catalog items) and its provider (Documentum vs. Interwoven).
id	yes	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.

Example:

```
<%@ taglib uri="pz.tld" prefix="pz" %>
.
.
.
<pz:contentselector id="docs" ruleset="ejb://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>
```

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 a per-user, HTTP session-based Content cache 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" %>
```

<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 4-39)can be used to iterate over the array of `Content` objects. This tag supports generic `Content` via a `ContentManager` interface.

Tag Attribute	Required	Description
contentHome	no	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.
max	no	Limits the maximum number of content items returned. If not present, it returns all of the content items found.
sortBy	no	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"

failOnError	no	This parameter 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.
id	yes	The JSP script variable name that will contain the array of Content objects after this tag finishes.
query	yes	A content query string used to search for content. Example: query="mimetype contains 'text' && author='Proulx'"
useCache	no	Determines whether Content is cached. This parameter can have one of two values: *false* (default value): ContentCache is not used. If false (not specified), `cacheId` and `cacheTimeout` settings are ignored. *true*: ContentCache is used. Cached Content is stored in the user's HttpSession.
cacheId	no	The id name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the id parameter of the tag is used.
cacheTimeout	no	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.

readOnly	no	If *true*, the ContentManager (specified via the `ContentHome` parameter) 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.

Example:

To find the first five text `Content` objects at *bea.eDocs.CMgr* that are marked as news items for the evening using the ContentCache, and print out the titles in a list:

```
<cm:select contentHome="bea.eDocs.CMgr" 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 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.

Tag Attribute	Required	Description
contentHome	no	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.
contentId	yes	The string identifier of the piece of content.
failOnError	no	This parameter 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.
id	yes	The JSP script variable name that contains the Content object after this tag finishes. If the Content object with the specified id does not exist, it contains null.
useCache	no	Determines whether Content is cached. This parameter can have one of two values: *false* (default value): ContentCache is not used. If false (not specified), `cacheId` and `cacheTimeout` settings are ignored. *true*: ContentCache is used. Cached Content is stored in the user's HttpSession.
cacheId	no	The id name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the id parameter of the tag is used.
cacheTimeout	no	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.

readOnly	no	If *true*, the ContentManager (specified via the `ContentHome` parameter) 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.

Example:

To fetch the `Document` from *bea.eDocs.CMgr* (using ContentCaching) with id of *1234* and inline its content:

```
<cm:selectbyid contentHome="bea.eDocs.CMgr" contentId="1234"
id="doc" useCache="true" cacheTimeout="300000" cacheId="1234" />
<cm:printdoc id="doc" />
```

<cm:printproperty>

The `<cm:printproperty>` tag 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.

Tag Attribute	Required	Description
id	yes	The JSP script variable name which contains the Content instance from which to get the properties.
name	yes	The name of the property to print.
scope	no	The scope name for the property to get. If not specified, null is passed in, which is what Document objects expect.

encode	no	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.
default	no	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.
maxLength	no	The maximum length of the property's value to print. If specified, values longer than this will be truncated.
failOnError	no	This parameter 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.
dateFormat	no	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.
numFormat	no	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.

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:printdoc>

The <cm:printdoc> tag 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.

Tag Attribute	Required	Description
id	yes	The JSP script variable name which contains the Content instance from which to get the properties.
blockSize	no	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.
start	no	Specifies the index in the bytes where to start reading. Defaults to 0.
end	no	Specifies the index in the bytes where to stop reading. The default is to read to the end of the bytes.
failOnError	no	This parameter 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.

Example:

To get a Document object from an id in the request parameters and inline the Document's text:

```
<cm:selectbyid contentHome="<%=contentHome%>"
contentId="<%=contentId%>" id="doc"/>
<cm:printdoc id="doc" blockSize="1000" />
```

Portal Management

To import the Portal Management JSP tags, use the following code:

```
<%@ taglib uri="lib/esportal.jar" prefix="pt" %>
```

<pt:portalmanager>

The `<pt:portalmanager>` tag is used to perform `create`, `get`, `getColumnInfo`, `update`, and `remove` actions on `com.beasys.portal.Portal` objects. This tag is an empty tag.

Tag Attribute	Required	Description
id	when action equals *get* or *getColumnInfo*	The name to which resultant information is assigned for subsequent use in the JSP page.
action	no	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.
portalName	no	The name of the portal to retrieve, or whose column information is to be retrieved. The default value is the value returned by `PortalJspBase.getSessionValue(PortalJsp.Base.`PORTAL_NAME).

target	when action equals *create*, *update*, or *remove*	The com.beasys.portal.Portal to be created, updated, or removed.

Example:

```
<pt:portalmanager id="portal" action="get"
portalName="BEAPortal"/>
```

<pt:portletmanager>

The `<pt:portletmanager>` tag is used to perform `create`, `get`, `getArranged`, `update`, and `remove` actions on `com.beasys.portal.Portlet` objects. This tag is an empty tag.

Tag Attribute	Required	Description
id	when action equals *get* or *getArranged*	The name to which resultant information is assigned for subsequent use in the JSP page.
action	no	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.
portalName	no	The name of the portal corresponding to the target portlet or to the portlet(s) to be retrieved. The default value is the value returned by `PortalJspBase.getSessionValue(PortalJsp.Base.`PORTAL_NAME).

portletName	no	The name of the portlet corresponding to the target portlet or to the portlet(s) to be retrieved. The default value is the value returned by `PortalJspBase.getSessionValue(PortalJsp.Base. PORTLET_NAME)`.
groupId	no	The name of the group corresponding to the target portlet or to the portlet(s) to be retrieved. The default value is `GROUP_ID`.
userId	no	The name of the user corresponding to the target portlet or to the portlet(s) to be retrieved. The default value is `USER_ID`.
target	when action equals *create*, *update*, or *remove*	The com.beasys.portal.Portlet to be created, updated, or removed.
scope	no	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.

Example:

```
<pt:portletmanager id="arrangedPortlets" action="getArranged"
userName="myUser" portalName="myPortal"/>
```

This is how `<pt:portletmanager>` is used in `portalcontent.jsp`:

```
<pt:portletmanager action="getArranged" id ="allPortlets"
userId="<%=portalUserUID.longValue()%>"
groupId="<%=portalGroupUID.longValue()%>"
portalName="<%=portalName%>"/>
```

<pt:eval>

The <pt:eval> tag 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 <pt:eval> tag is processed. Otherwise, it is not.

Tag Attribute	Required	Description
tag	yes	The name of the portlet attribute to evaluate.

Example:

```
<pt:eval tag="isMinimizeable">
     <% titleBar.include(minimizeButton); %>
</pt:eval>
```

<pt:get>

The <pt:get> tag 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.

Tag Attribute	Required	Description
tag	yes	The name of the portlet attribute to retrieve.

Example:

```
<tr>
   <td>
      <pt:get tag="title"/>
   </td>
</tr>
```

<pt:monitorsession>

The `<pt:monitorsession>` tag 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.

Tag Attribute	Required	Description
goToPage	no	The error page that you want displayed if the page is not accessible. The default value is *portalerror.jsp*.
loginRequired	no	Indicates whether the user is required to be logged in to access the JSP page including the tag. The default value is **FALSE**.

Example:

```
<pt:monitorsession loginRequired="true" />
```

<pt:props>

The <pt:props> tag 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.

Tag Attribute	Required	Description
id	yes	A java.lang.String variable name for the property value.
portalName	yes	The name of the property to get in the Portal Property Bean.

Example:
<pt:props id="headerURL" propertyName="default.portal.headerURL" />

<pt: getgroupsforportal>

The `<pt:getgroupsforportal>` tag retrieves the names of the groups associated with a Portal. The results are put into the variable declared in the *`id`* parameter of the tag, which is a `String` array.

Tag Attribute	Required	Description
id	yes	A resulting string array containing the names of the groups associated with the given Portal.
portalName	yes	The name of the Portal to be checked for associated groups.

User Management

To import the User Management JSP tags, use the following code:

```
<%@ taglib uri="lib/um_tags.jar" prefix="um" %>
```

Profile management tags

<um:getprofile>

The `<um:getprofile>` tag 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 as 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.

Tag Attribute	Required	Description
profileKey	yes	A unique identifier that can be used to retrieve the profile which is sought. Example: "<%=username%>"
profileType	no	The profile 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.User`, unless otherwise specified. Example: "AcmeUser"
successorKey	no	A unique identifier that can be used to retrieve the profile successor. Example: "<%=defaultGroup%>"
successorType	no	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"
scope	no – defaults to *session*	The HTTP scope of the retreived profile. Pass `Request` or `Session` as the values.
groupOnly	no – defaults to *false*	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*.

profileId	no	A variable name from which the retrieved profile is available for the duration of the JSP's page scope.
successorId	no	A variable name from which the retrieved successor is available for the duration of the JSP's page scope.

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 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 User Management documentation for more information.

Tag Attribute	Required	Description
propertySet	no – the property is retrieved from the profile's default (unscoped) properties if no property set is provided.	The Property Set from which the property's value is to be retrieved. Example: "Demo Portal"
propertyName	yes	The name of the property to be retrieved. Example: "background_color".
useCache	no – defaults to *false*.	Prescribes whether to first search the local cache of property values for the property, and whether to cache the value locally once retrieved remotely. This attribute can be set to `true` to significantly enhance performance.
id	no	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.

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 works exactly as the `<um:getproperty>` tag, 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.

Example:

```
<um:getpropertyasstring id="myFaveColors"
propertySet="exampleportal" propertyName="fave_colors"/>
My favorite colors are <%=myFaveColors%>.
```

<um:removeproperty>

The <um:removeproperty> tag removes the specified property from the current session 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 Session's successor.

Tag Attribute	Required	Description
propertySet	no	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.
propertyName	yes	The name of the property to be removed. Example: "background_color".

Example:

```
<um:removeproperty propertySet="<%=thePropertySet%>"
propertyName="<%=thePropertyName%>"/>
```

<um:setproperty>

The <um:setproperty> tag updates a property value for either the current session profile, or for the Anonymous User Profile. The tag has no enclosed body.

Tag Attribute	Required	Description
propertySet	no	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.
propertyName	yes	The name of the property to be set. Example: "background_color".
useCache	no – defaults to *false*	Prescribes whether to set the value for the property in the local cache after setting the property for the current profile.
value	yes	The new property value. This attribute ***must*** be a previously-declared variable, as shown in the example.

Example:

```
<% String myName = request.getParameter("name"); %>
<um:setproperty propertySet="exampleportal" propertyName="name"
value="<%=myName%>"/>
```

Group-user management tags

<um:addgrouptogroup>

The `<um:addgrouptogroup>` tag adds the group corresponding to the provided `childGroupName` to the group corresponding to the provided `parentGroupName`. 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`.

Tag Attribute	Required	Description
childGroupName	yes	The name of the child group. Example: "<%=childGroupName%>"
parentGroupName	yes	The name of the parent group. Example: "<%=parentGroupName%>"
resultId	yes	The possible results of the add group to group operation. Possible values include: ■ *success*: `UserManagerTagConstants.ADD_GROUP_OK` ■ *error encountered*: `UserManagerTagConstants.ADD_GROUP_FAILED`

Example:

```
<um:addgrouptogroup childGroupName="<%=childGroupName%>"
parentGroupName="<%=parentGroupName%>" resultId="result"/>
```

<um:addusertogroup>

The `<um:addusertogroup>` tag adds the user corresponding to the provided `userName` to the group corresponding to the provided `parentGroupName`. 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`.

Tag Attribute	Required	Description
userName	yes	The name of the user to be added to the group. Example: "<%=username%>"
groupName	yes	The name of the group to which the user is being added. Example: "<%=groupName%>"
resultId	yes	The possible results of the add user to group operation. Possible values include: ■ *success*: `UserManagerTagConstants.ADD_USER_OK` ■ *error encountered*: `UserManagerTagConstants.ADD_USER_FAILED`

Example:

```
<um:addusertogroup userName="<%=userName%>"
groupName="<%=groupName%>" resultId="result"/>
```

<um:changegroupname>

The `<um:changegroupname>` tag changes the a name of the group corresponding to the specified `oldGroupName` to the specified `newGroupName`. 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`.

Tag Attribute	Required	Description
oldGroupName	yes	The old group name. Example: "<%=oldGroupName%>"

newGroupName	yes	The new group name. Example: "<%=newGroupName%>"
resultId	yes	The name of an Integer variable to which the result of the change group name operation is assigned. Possible values include: ■ *success*: `UserManagerTagConstants.GROUP_CHANGE_OK` ■ *error encountered*: `UserManagerTagConstants.GROUP_CHANGE_FAILED`

Example:

```
<um:changeGroupname oldGroupName="<%=oldGroupName%>"
newGroupName="<%=changeGroupName%>" resultId="result"/>
```

<um:creategroup>

The `<um:creategroup>` tag creates a new `com.beasys.commerce.axiom.contact.Group` object. 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`.

Tag Attribute	Required	Description
groupName	yes	The name of the new group. Example: "<%=groupName%>"
id	no	A variable name to which the created Group object is available for the duration of the page's scope.

resultId	yes	The possible results of the create user operation. Possible Values: `UserManagerTagConstants.CREATE_GROUP_OK:` Success `UserManagerTagConstants.CREATE_GROUP_FAILED:` Error encountered `UserManagerTagConstants.GROUP_EXISTS:` A user with the specified username already exists

Example:

```
<um:creategroup groupName="<%=groupName%>" resultId="result"/>
```

<um:createuser>

The `<um:createuser>` tag creates a new `com.beasys.commerce.axiom.contact.User` object. The 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`.

Tag Attribute	Required	Description
userName	yes	The name of the new user. Example: "<%=username%>"
password	no	The password for the new user. Example: "<%=password%>"
saveAnonymous	no – defaults to *false*	Whether to persist current anonymous user profile attributes in the newly-created user's profile. Example: "false"
id	no	A variable name to which the created User object is available for the duration of the page's scope.

resultId	yes	The possible results of the create user operation. Possible Values: `UserManagerTagConstants.CREATE_USER_OK`: Success `UserManagerTagConstants.CREATE_USER_FAILED`: Error encountered `UserManagerTagConstants.USER_EXISTS`: A user with the specified username already exists

Example:

```
<um:createuser userName="<%=username%>" password="<%=password"%>
resultId="result"/>
```

<um:getchildgroups>

The `<um:getchildgroups>` tag 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. The tag has no enclosed body.

Tag Attribute	Required	Description
groupName	yes	The name of the group whose children are sought. Example: "<%=groupName%>"
id	yes	A variable name to which the child Group objects are available for the duration of the page's scope. Example: "childGroups"

Example:

```
<um:getchildgroups groupName="<%=groupName%>" id="childGroups"/>
```

<um:getgroupnamesforuser>

The <um:getgroupnamesforuser> tag retrieves a String array that contains the group names matching the provided search expression and corresponding to groups to which the provided user belongs. The tag has no enclosed body.

Tag Attribute	Required	Description
userName	yes	The name of the user whose matching groups are sought. Example: "<%=username%>"
id	yes	A variable name to which the resultant group names are assigned. Example: "myGroups"

Example:

```
<um:getgroupnamesforuser userName="<%=username%>" id="myGroups"/>
```

<um:getparentgroupname>

The <um:getparentgroupname> tag 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. The tag has no enclosed body.

Tag Attribute	Required	Description
groupName	yes	The name of the group whose parent group name is sought. Example: "<%=groupName%>"
id	yes	A variable name to which the to which the name of the parent is available for the duration of the page's scope. Example: "parentGroupName"

Example:

```
<um:getparentgroupname groupName="<%=groupName%>"
id="parentGroupName"/>
```

<um:gettoplevelgroups>

The <um:gettoplevelgroups> tag retrieves and 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. The tag has no enclosed body.

Tag Attribute	Required	Description
id	yes	A variable name to which the top-level Group objects are available for the duration of the page's scope. Example: "topLevelGroups"

Example:

```
<um:gettoplevelgroups id="topLevelGroups"/>
```

<um:getusernames>

The <um:getusernames> tag 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. 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.

Tag Attribute	Required	Description
searchExp	no – defaults to '*'	The search expression to apply to the user name search. Example: "t*"

userLimit	no – defaults to 100	The maximum number of users to be returned from the search. (String which has a particular `Integer.valueOf`) Example: “500”
id	yes	A variable name to which the resultant user names are assigned. Example: “myUsers”
resultId	yes	The possible results of the get user names operation. Possible Values: `UserManagerTagConstants.USER_SEARCH_OK`: success UserManagerTagConstants.`USER_SEARCH_FAILED`: general error `UserManagerTagConstants.USER_LIMIT_EXCEEDED`: matching user count exceeds limit

Example:

```
<um:getusernames userLimit=”500” searchExp=”t*” id=”myUsers”/>
<%System.out.println(“I found “ + myUsers.length + “ users.”);%>
```

<um:getusernamesforgroup>

The `<um:getusernamesforgroup>` tag 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. 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`.

Tag Attribute	Required	Description
searchExp	no – defaults to *	The search expression to apply to the user name search. Example: “t*”
groupName	yes	The name of the group whose matching members are sought. Example: “engineering”
userLimit	no – defaults to 100	The maximum number of users to be returned from the search. (String which has a particular `Integer.valueOf`) Example: “500”
id	yes	A variable name to which the resultant user names are assigned. Example: “myUsers”
resultId	yes	The possible results of the get user names for group operation. Possible Values: `UserManagerTagConstants.USER_SEARCH_OK`: success UserManagerTagConstants.`USER_SEARCH_FAILED`: general error `UserManagerTagConstants.USER_LIMIT_EXCEEDED`: matching user count exceeds limit

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 removes the `com.beasys.commerce.axiom.contact.Group` object corresponding to the provided `groupName`. 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`.

Tag Attribute	Required	Description
groupName	yes	The name of the user to be removed. Example: "<%=groupName%>"
resultId	yes	The possible results of the remove group operation. Possible values include: ■ *success*: UserManagerTagConstants. REMOVE_GROUP_OK ■ *error encountered*: UserManagerTagConstants. REMOVE_GROUP_FAILED

Example:

```
<um:removegroup groupName="<%=groupNamename%>" resultId="result"/>
```

<um:removeuser>

The `<um:removeuser>` tag removes the `com.beasys.commerce.axiom.contact.User` object corresponding to the provided `userName`. 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`.

Tag Attribute	Required	Description
userName	yes	The name of the user to be removed. Example: "<%=username%>"

resultId	yes	The possible results of the remove user operation. Possible values include: ■ *success*: UserManagerTagConstants.REMOVE_USER_OK ■ *error encountered*: UserManagerTagConstants.REMOVE_USER_FAILED

Example:

```
<um:removeuser userName="<%=username%>" resultId="result"/>
```

Security tags

<um:login>

The `<um:login>` tag provides weak authentication (username, password) against the current security realm, and sets the authenticated user as the current WebLogic user. The tag has no enclosed body.

Note: The login tag requires a *`username`* parameter and a *`password`* parameter to be present in the HTTP request.

Tag Attribute	Required	Description
resultId	yes	The possible results of the login operation. UserManagerTagConstants.LOGIN_FAILED - username/password combo invalid UserManagerTagConstants.LOGIN_ERROR - an error occurred when performing authentication. Possible values: ■ `UserManagerTagConstants.LOGIN_OK`: Success ■ `UserManagerTagConstants.LOGIN_FAILED`: Authentication failed because of invalid username/password combination ■ `UserManagerTagConstants.LOGIN_ERROR`: General error when performing authentication.

<um:setpassword>

The `<um:setpassword>` tag 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`.

Tag Attribute	Required	Description
userName	yes	The name of the user whose password is to be changed.
password	no	The new user password.
resultId	no	The possible results of the set password operation. ■ Success: User Management tag contstants. SET_PASSWORD_OK ■ Failure: User Management tag contstants. SET_PASSWORD_FAILED

Personalization Utilities

The <es:jsptaglib> contains generic tags you can use to create JSP pages. Use the following code to import the utility tag library:

```
<%@ taglib uri="lib/esjsp.jar" prefix="es" %>
```

<es:condition>

The <es:condition> tag is used to evaluate a Boolean expression. The tag can either be an empty tag or a tag whose body is executed if the condition evaluates to *true*.

Tag Attribute	Required	Description
test	no	The expression to test.
id	When the tag is empty.	The variable name to assign. The default value is *id*.

Example:

```
<es:condition id="isYes" test="a.equals(b)"/>
```

<es:counter>

The <es:counter> is used to create a for loop.

Tag Attribute	Required	Description
type	no	The type of the counter. This can either be an `int` or a `long`. Default is `int`.
id	yes	A unique name for the variable.
minCount	yes	The start position for the loop.

maxCount	yes	The end position for the loop.

Example:

```
<es:counter id="iterator" minCount="0" maxCount="10">
   <% System.out.println(iterator);%>
</es:counter>
```

<es:foreachinarray>

The `<es:foreachinarray>` tag is used to iterate over an array.

Tag Attribute	Required	Description
id	yes	The variable for each value in the array.
type	yes	The type of each value in the array.
array	yes	The array to iterate over.
counterId	no	The position in the array.

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 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 empty.

Tag Attribute	Required	Description
id	yes	The variable to evaluate.

Example:

```
<es:isnull id="value">
     Error: the value is null.
</es:isnull>
```

<es:notnull>

The <es:notnull> tag 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 empty.

Tag Attribute	Required	Description
id	yes	The variable to evaluate.

Example:

```
<es:notnull id="value">
     The value is not null.
</es:notnull>
```

<es:preparedstatement>

The <es:preparedstatement> tag is used to create a JDBC prepared statement.

Tag Attribute	Required	Description
id	yes	The variable in which the PreparedStatement is returned.
sql	yes	The SQL with <es:preparedstatement> tags.
pool	yes	The JDBC connection pool from which to get a connection.

Example:

```
<es:preparedstatement id="ps" sql="select last_name from user where
id=?" pool="jdbcPool">
<%
```

```
   ps.setInt(1, 1234);
   ResultSet rs = ps.execute();
   if (rs.next())
   {
     System.out.println(rs.getString(1));
   }
%>
</es:preparedstatement>
```

<es:simplereport>

The <es:simplereport> tag is used to create two-dimensional array out of a simple query.

Tag Attribute	Required	Description
id	yes	The variable that holds the [ ] array.
resultSet	yes	The result set that turns two-dimensional.

Example:

```
<es:simplereport id="report" resultSet="resultSet">
<%
   for (int i=0; ireport[i].length; j++ )
   {
     ...
   }
 }
%>
</es:simplereport>
```

<es:transposearray>

The <es:transposearray> tag is used to transpose a standard [row][column] array to a [column][row] array.

Tag Attribute	Required	Description
id	yes	The variable that holds the [c][r] array.
type	yes	The type of variable in the [r][c] array, such as String.
array	yes	The variable that holds the [r][c] array.

Example:

```
<es:transposearray id="byColumnRow" array="byRowColumn"
type="String">
   ...
</es:transposearray>
```

<es:uricontent>

The `<es:uricontent>` tag is used to pull content from a URL. It is best used for grabbing text-heavy pages.

Tag Attribute	Required	Description
id	yes	The variable that holds the downloaded content of the URI.
uri	yes	The fully-qualified URI from which to get the content.

Example:

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

<es:date>

The <es:date> tag is used to get a date- and time-formatted String based on the user's time zone preference.

Tag Attribute	Required	Description
timeZoneId	no	Defaults to the time zone on the server.
formatStr	no	A date and time format string that adheres to the java.text.SimpleDateFormat. The default value is *MM/dd/yyyy HH:mmss:z*.

Example:

```
<es:date formatStr="MMMM dd yyyy" timeZoneId="MST" />
```

<es:usertransaction>

The <es:usertransaction> tag is used to wrap database-intensive code within one efficient transaction.

Note: Do not nest these calls. The system does not support nested transactions.

Tag Attribute	Required	Description
timeout	no	The user transaction timeout in seconds. The default value is *600*.

Example:

```
<es:usertransaction>
   <% //database inserts %>
</es:usertransaction>
```

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="lib/wljsp.jar" prefix="wl" %>
```

<wl:process>

The <wl:process> tag is used for query parameter-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.

Tag Attribute	Required	Description
name	no	The name of a query parameter.
notname	no	The name of a query parameter.
value	no	The value of a query parameter.
notvalue	no	The value of a query parameter.

Statements between the <wl:process> tags will be executed according to the matrix:

	value	**notvalue**	**(none)**
name	named parameter is equal to the value	named parameter does not equal the value	named parameter is empty
notname			named parameter is not empty

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>
```

Index

R

S

T

U

W