BEA WebLogic Personalization Server™

Guide to Building Personalized Applications

Version 4.0
Document Date: May 10, 2002

Guide to Building Personalized Applications

Document Edition	Date	Software Version
4.5	May 10, 2002	BEA WebLogic Portal 4.0 BEA Weblogic Personalization Server 4.0

Contents

4. Working with Content Selectors

5. Using the Expression Package

About This Document

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

This document includes the following topics:

- Chapter 1, "Overview of Personalization Development," provides developer components and utilities that enable developers to create personalized applications. The pieces documented in this guide include the Advisor, Foundation classes and utilities, and JSP tag reference.
- Chapter 2, "Creating Personalized Applications with the Advisor," recommends content and performs several important functions in creating a personalized application, including searching for content, tying the other core personalization services together, and matching content to user profiles.
- Chapter 3, "Introducing the Rules Framework," discusses how the Rules Management component allows developers to create business rules that turn on and off content and match content to users according to their profile information.
- Chapter 4, "Working with Content Selectors," shows how a Business Analyst (BA) can use content selectors to specify conditions under which WebLogic Personalization Server retrieves one or more documents.
- Chapter 5, "Using the Expression Package," illustrates how to use the services of the Expression Package. Any arithmetic, boolean, relational or conditional statement can be represented in the Java object model by using an Expression Package. You can use the Expression Package to dynamically assemble and evaluate your own business logic.
- Chapter 6, "Foundation Classes and Utilities," describes the Foundation, a set of miscellaneous utilities to aid JSP and Java developers in the development of personalized applications using the WebLogic Personalization Server. Its utilities include JSP files and Java classes that can be used by JSP developers to gain

access to functions provided by the server and helpers for gaining access to Advisor services.

- Chapter 7, “Creating and Managing Property Sets,”discusses how Property Set Management allows you to create property sets, the schema of personalization attributes, and the properties that make up property sets.
- Chapter 8, “Creating and Managing Users,” discusses how User Management joins enterprise data about users with profile data that is used to personalize the user’s view of the application.
- Chapter 9, “Creating and Managing Content,” documents how the Content Manager provides content and document management capabilities for use in personalization services. The Content Manager works with files or with content managed by third-party vendor tools
- Chapter 10, “Working with Ad Placeholders,” shows how ad placeholders display documents that advertise products or services (ads) and record customer reactions to them.
- Chapter 11, “Creating Localized Applications with the Internationalization Tags,” provides a simple framework that allows access to localized text and messages. The internationalization (I18N) framework is accessible from JSP through a small I18N tag library.
- Chapter 12, “The WebLogic Personalization Server Database Schema,” documents the database schema for the WebLogic Personalization Server.
- Chapter 13, “Personalization Server JSP Tag Library Reference,” describes the JSP tags included with WebLogic Personalization Server that allow developers to create personalized applications without having to program using Java.

What You Need to Know

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

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

e-docs Web Site

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

How to Print the Document

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

A PDF version of this document is available on the WebLogic Portal documentation Home page on the e-docs Web site. A PDF version of this document is also available in the documentation kit on the product CD. Or you can download the documentation kit from the WebLogic Portal portion of the BEA Download site.

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@bea.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 WebLogic Personalization Server release 4.0.

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.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

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

Convention	Item
italics	Indicates emphasis or book titles.

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

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

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

Convention	Item
|	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

WebLogic Personalization Server provides developers with the ability to create personalized applications for e-commerce Web sites. This topic provides a broad overview of personalization development for Java and JSP developers.

This topic includes the following sections:

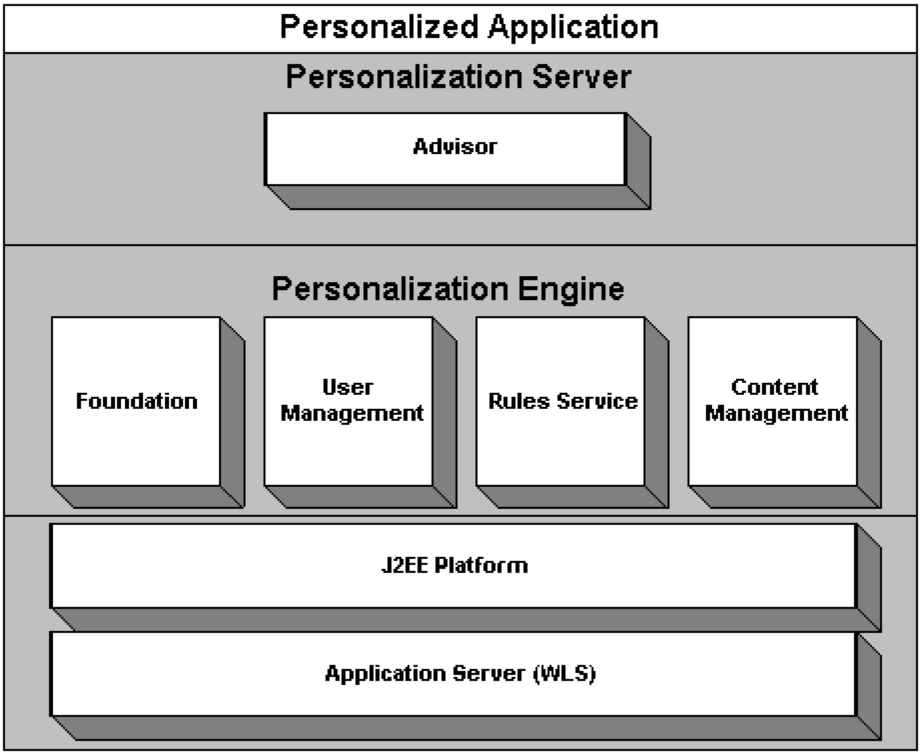
- Personalization Server Run-Time Architecture
  - Advisor
  - User Management
  - Content Management
  - Rules Management
  - Foundation Classes and Utilities
- JSP Tags
- Integration of External Components
- Support for Native Types

Personalization Server Run-Time Architecture

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

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

Figure 1-1 WebLogic Personalization Server High Level Architecture



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

Advisor

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

User Management

The User Management component supports the run-time access of users, groups, and the relationships between them. The notion of property sets is used by 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 base user profile, to provide custom data source access to user property values.

Content Management

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

Rules Management

The Rules Management component is the run-time service that runs the rules that are built in the E-Business Control Center.

Foundation Classes and Utilities

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

JSP Tags

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

Table 1-1 JavaServer Page JSP Tags Overview

Library	Tag	Description
Ads	`<ad:adTarget>`	Queries the content management system and displays ads.
Content Management	`<cm:getProperty>`	Retrieves the value of the specified content metadata property.
	`<cm:printDoc>`	Inlines the raw bytes of a document object into the JSP output stream.
	`<cm:printProperty>`	Inlines the value of the specified content metadata property as a string.
	`<cm:select>`	Selects content based on a search expression query syntax.
	`<cm:selectById>`	Retrieves content using the content's unique identifier.

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

Library	Tag	Description
Internationalization	`<i18n:localize>`	Defines the language, country, variant, and base bundle name to be used throughout a page when accessing resource bundles via the `<i18n:getmessage>` tag. Also allows a character encoding and content type to be specified for a JSP.
	`<i18n:getMessage>`	Used in conjunction with the `<i18:localize>` tag to retrieve localized static text or messages.
Personalization	`<pz:contentQuery>`	Provides content based on search expression query syntax.
	`<pz:contentSelector>`	Provides content based on results of a content selector rule and subsequent content query.
	`<pz:div>`	Turns a user-provided piece of content on or off based on the results of a classifier rule.
Placeholders	`<ph:placeholder>`	Implements a placeholder, which describes the behavior for a location on a JSP page.
Property Sets	`<ps:getPropertyNames>`	Used to get a list of property names given a property set.
	`<ps:getPropertySetNames>`	Used to get a list of property sets given a property set type.
	`<ps: getRestrictedPropertyValues >`	Used to get a list of restricted property values given a property set type and property name.
User Management (Profile)	`<um:getProfile>`	Retrieves the Unified User Profile object.
	`<um:getProperty>`	Gets the value for the specified property from the current user profile in the session.

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

Library	Tag	Description
	`<um:getPropertyAsString>`	Works exactly like the `<um:getProperty>` tag above, but ensures that the retrieved property value is a `String`.
	`<um:removeProperty>`	Removes the property from the current user profile in the session.
	`<um:setProperty>`	Sets a new value for the specified property for the current user profile in the session.
(Group-User Management)	`<um:addGroupToGroup>`	Adds the group corresponding to the provided `childGroupName` to the group corresponding to the provided `parentGroupName`.
	`<um:addUserToGroup>`	Adds the user corresponding to the provided `userName` to the group corresponding to the provided `parentGroupName`.
	`<um:createGroup>`	Creates a new group in the realm, and a corresponding group profile in the personalization database.
	`<um:createUser>`	Creates a new persisted User object with the specified username and password.
	`<um:getChildGroupNames>`	Returns the names of any groups that are children of the given group.
	`<um:getGroupNamesForUser>`	Retrieves a `String` array that contains the group names matching the provided search expression and corresponding to groups to which the provided user belongs.
	`<um:getParentGroupName>`	Retrieves the name of the parent of the group associated with the provided `groupName`. The information is taken from the realm.

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

Library	Tag	Description
	`<um:getTopLevelGroups>`	Retrieves an array of group names, each of which has no parent group. The information is taken from the realm.
	`<um:getUsernames>`	Retrieves a String array that contains the usernames matching the provided search expression.
	`<um:getUsernamesForGroup>`	Retrieves a `String` array that contains the usernames matching the provided search expression and correspond to members of the provided group.
	`<um:removeGroup>`	Removes the group corresponding to the provided `groupName`.
	`<um:removeGroupFromGroup>`	Removes a child group from a parent group.
	`<um:removeUser>`	Removes the user corresponding to the provided `username`. It can remove any type of extended user that has its profileType set in the database.
	`<um:removeUserFromGroup>`	Removes a user from a group.
(Security)	`<um:login>`	Authenticates a user/password combination.
	`<um:logout>`	Ends the current user's WebLogic Server session. This tag should be used in combination with the `<um:login>` tag.
	`<um:setPassword>`	Updates the password for the user corresponding to the provided username.

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

Library	Tag	Description
Personalization Utilities	`<es:convertSpecialChars>`	Converts characters which would normally signify special meaning to an HTML browser into characters which can be displayed as intended.
	`<es:counter>`	Creates a `for loop` construct.
	`<es:date>`	Gets a date and time formatted string based on the user's time zone preference.
	`<es:forEachInArray>`	Iterates over an array.
	`<es:isNull>`	Checks to see if a value is null. If the value type is a `String`, also checks to see if the `String` is empty.
	`<es: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:transposeArray>`	Transposes a standard [row][column] array to a [column][row] array.
	`<es:uriContent>`	Pulls content from a URL.
WebLogic Utilities	`<wl:cache>`	Specifies that its contents do not necessarily need to be updated every time it is displayed.
	`<wl:process>`	Provides a attribute-based flow control construct.
	`<wl:repeat>`	Used to iterate over a variety of Java objects, as specified in the set attribute.

Integration of External Components

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

Table 1-2 Useful External Components the Personalization Server

External Component	Out-of-the-Box Support	Methods and Notes
DBMS	Integrated and tested with Cloudscape, Oracle 8.1.6, and 8.1.7.	Uses standard WebLogic Server JDBC connection pools.
LDAP authentication	Can be set up automatically using administration tools and property files.	Uses WebLogic Server security realms.
LDAP retrieval of user and group information	Can be set up automatically using administration tools.	Built into EJB persistence for User entity bean.
Legacy database of users	None.	Requires Unified User Profile extension of User entity bean.
Content Management engine	Reference implementation provided.	Provides API/SPI support from third-party vendors.
Legacy content database	None.	Requires either extension of Document entity bean or custom implementation of content management SPI.

Support for Native Types

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

Table 1-3 Native Types

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

The WLPS Advisor is an easy-to-use and flexible access point for personalization services—including personalized content, user segmentation and the underlying rules engine.

This topic includes the following sections:

- What Is the Advisor?
  - The Advisor Delivers Content to a Personalized Application
  - The Advisor Provides Information About User Classifications
  - You Can Use the Advisor in One of Two Ways
- The WLPS Advisor Architecture
  - Writing a Custom Advislet
  - Understanding the Advislet Registry
  - Registering a Single Advislet
  - Advislet Chaining
  - Registering a Compound Advislet
- Creating Personalized Applications with the Advisor JSP Tags
  - Classifying Users with the JSP <pz:div> Tag

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

■ Creating Personalized Applications with the Advisor Session Bean

- Classifying Users with the Advisor Session Bean
- Querying a Content Management System with the Advisor Session Bean
- Matching Content to Users with the Advisor Session Bean

What Is the Advisor?

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

The Advisor Delivers Content to a Personalized Application

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

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

- User Profile Management
- Rules Manager
- Content Management
- Personalization Platform

The tag library and session bean contain personalization logic to access these services, sequence personalization actions, and return personalized content to the application. It is also possible to write your own advisor plug-ins and access them with JSP tags you create.

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

The Advisor recommends content for the following items:

- Web content included or excluded as determined by a user's classification using rules-based matching against user profile information. For more information about classifying users, see "Classifying Users with the JSP <pz:div> Tag" on page 2-10 and "Classifying Users with the Advisor Session Bean" on page 2-15.
- Documents returned by document attribute searches. For more information about searching for content, see "Selecting Content with the <pz:contentQuery> JSP Tag" on page 2-11 and "Querying a Content Management System with the Advisor Session Bean" on page 2-16.
- Documents returned by content selectors using rules-based matching against user profile information or user's classification. For more information about rules-based matching, see "Matching Content to Users with the <pz:contentSelector> JSP Tag" on page 2-12 and "Matching Content to Users with the Advisor Session Bean" on page 2-17.

Note: User classification is done in the E-Business Control Center. You will see the term "customer segmentation" used in the GUI tool to refer to user classification and classifier rules.

The Advisor Provides Information About User Classifications

In addition to supplying content to personalized applications, the Advisor can also provide information about user classifications. For example, an application can ask the Advisor if, based on predefined rules, the current user is classified as a *Premier*

Customer or an *Aggressive Investor*, and take action accordingly. The Advisor accomplishes this classification by gathering relevant user profile information, submitting it to the Rules Manager, and returning the classification to the caller.

For more information about classifying users, see "Classifying Users with the JSP <pz:div> Tag" on page 2-10 and "Classifying Users with the Advisor Session Bean" on page 2-15.

You Can Use the Advisor in One of Two Ways

- **Using the JSP tags**. Developers will probably find it easiest to use the JSP tags when building typical pages. The tags provide ways to switch content on and off based on user classification, return content based on a static query, and match content to users based on rules that execute a content query. The JSP tags that perform these tasks are: `<pz:div>`, `<pz:contentSelector>`, and `<pz:contentQuery>`.
- **Using the Advisor session bean**. The page or application developer may use the Advisor session bean directly in place of the tags, if desired. The Advisor session beans provide ways to switch content on and off based on user classification, return content based on a static query, and match content to users based on rules that execute a content query.

The WLPS Advisor Architecture

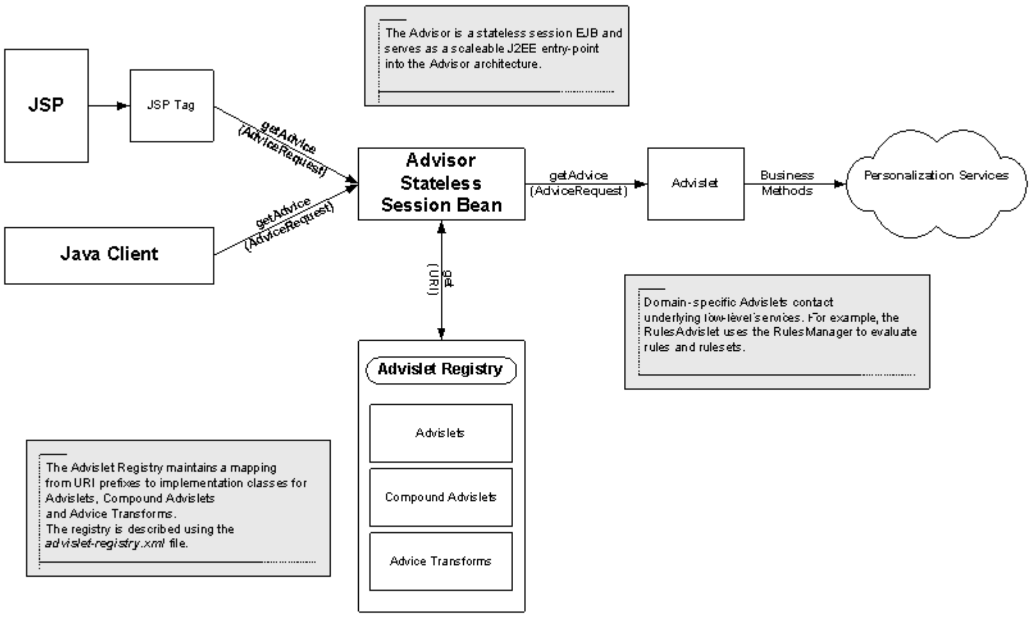
The Advisor is a stateless session EJB and has a simple interface with a `getAdvice` method on it. The `getAdvice` method returns `Advice` objects that contain the detailed result information that was returned from the personalization services.

The argument to the `getAdvice` method is an `AdviceRequest` object that contains a number of name-value pairs that define the inputs to the Advisor. The `AdviceRequest` has an interface very similar to the `HttpRequest` object and allows predefined as well as custom input parameters to be stored.

Each incoming `AdviceRequest` has a URI associated with it. The Advisor uses the URI prefix (the part before the colon) to look up an Advislet using the AdvisletRegistry. Advislets are typically simple Java classes that implement a personalization function such as user segmentation or content retrieval. The AdvisletRegistry maintains the deployment mappings from URI prefixes to Advislet instances.

Note: The relationship between the Advisor and an Advislet is similar to the relationship between the Server and a servlet (though an Advislet is independent of HTTP). An Advislet is registered with a prefix with the Advisor and will be invoked for all incoming `AdviceRequests` with that prefix.

Figure 2-1 The Advisor Architecture


The Advisor is a stateless session EJB and serves as a scaleable J2EE entry-point into the Advisor architecture.
JSP
JSP Tag
getAdvice (AdviceRequest)
Java Client
getAdvice (AdviceRequest)
Advisor Stateless Session Bean
getAdvice (AdviceRequest)
Advislet
Business Methods
Personalization Services
get (URI)
Domain-specific Advislets contact underlying low-level services. For example, the RulesAdvislet uses the RulesManager to evaluate rules and rulesets.
Advislet Registry
Advislets
Compound Advislets
Advice Transforms
The Advislet Registry maintains a mapping from URI prefixes to implementation classes for Advislets, Compound Advislets and Advice Transforms. The registry is described using the advislet-registry.xml file.

Writing Custom Advislets and Registering Them Using the Advislet Registry

At the core of the Advisor framework is the Advislet Registry. The Advisor uses the Advislet Registry to determine which Advislets to invoke in the processing of an advice request.

The WebLogic Personalization Server provides a number of Advislets which support the three personalization JSP tags: `<pz:classifier>`, `<pz:contentselector>` and `<pz:contentquery>`. To extend this functionality or to interface with third-party systems, you can write a custom Advislet and register it with the Advislet Registry.

Writing a Custom Advislet

To write a custom Advislet a developer simply has to implement the Advisor interface, providing implementations of these three methods: `getAdvice`, `getRequiredAttributes` and `validateAdviceRequest`.

When the Advisor receives an `AdviceRequest` object, it calls `validateAdviceRequest` before passing it to the registered Advislet's `getAdvice` method. The `validateAdviceRequest` method should throw an `IllegalArgumentException` if some necessary attributes are missing or malformed.

In addition to the Advislet interface, an Advislet implementation must have a public constructor with two parameters. The Advisor will use these parameters when it creates instances of the Advislet.

- The first parameter is of type `Advisor.` It contains a reference to the Advisor creating this Advislet.
- The second parameter is an implementation of the Metadata interface. It contains the Advislet's name, description, and version information as specified in the Advislet Registry.

Note: Unless otherwise indicated, all classes referenced here reside in the `com.bea.p13n.advisor` package.

A default implementation of Advislet is provided in the `AbstractAdvislet` abstract class. Simply extend this class, override the `getAdvice` method and provide the required constructor to create your own Advislet.

Understanding the Advislet Registry

We have already discussed how the Advislet Registry associates uri prefixes with Advislet implementations. Once we look inside the Advislet Registry however, the story becomes a bit more complicated.

In the case of the `contentquery://` URI prefix, all of the work is done by one Advislet — `com.bea.p13n.content.advislets.ContentQueryAdvisletImpl` However, other prefixes (such as `contentselector://`) require a sequence of Advislets to be chained together to produce the required advice. In these cases a CompoundAdvislet is registered against the uri prefix to shield this complexity from the user. The specification of which Advislets to register against which uri prefixes is contained in the `advislet-registry.xml` file which can be found in the `ejbadvisor.jar` EJB JAR file within a WebLogic Personalization Server application. This file is scoped to the containing application, and therefore may have different contexts for different applications. An understanding of the contents of this file is essential to any customization of the Advislet framework.

Registering a Single Advislet

The following is an extract from the `advislet-registry.xml` file:

```
<!-- run a content query -->
<advislet>
<registration-key>contentquery</registration-key>
<metadata>
   <name>ContentQuery</name>
   <description>
   Advislet that can retrieve content from the Content Management
   System based on a content query.
   </description>
   <author>BEA Systems</author>
        …
</metadata>
<implementation-class>com.bea.commerce.platform.content.advislets
.ContentQueryAdvisletImpl</implementation-class>
```

```
</advislet>
```

The most important tags are `<registration-key>` and `<implementation-class>`. In the case of an Advislet, `<registration-key>` should specify the uri prefix that this Advislet is to be registered against and `<implementation-class>` should specify the fully qualified class name of the implementing Advislet class. The metadata information is useful for versioning Advislets and should be included.

Advislet Chaining

`AdviceTransform` objects are used to chain two Advislets together using a CompoundAdvislet. An `AdviceTransform` object provides the mapping between the outputs of one Advislet and the inputs of the next. The AdviceTransform interface simply specifies one method `transform` (`Advice` input, `AdviceRequest` output). which should be implemented to create the mapping required. `AdviceTransforms` are also registered in the AdvisletRegistry.

Registering a Compound Advislet

The following is an extract from the `advislet-registry.xml` file:

```
<!-- compound advislet that calls the rules engine and passes
results to the content management system -->

<compound-advislet>
   <registration-key>contentselector</registration-key>
   <metadata>
      <name>ContentSelector</name>
      <description>
      Advislet that retrieves Content from the Content Management
      system based on the evaluation of a rule set.
      </description>
      <author>BEA Systems</author>
      …
   </metadata>
   <sequence>
      <advice-transform>RulesInputTransform</advice-transform>
      <advislet>unmappedrulesClassifierIgnoreRuleName</advislet>
      <advice-transform>
          ClassifierToContentSelectorTransform
```

```
        </advice-transform>
        <advislet>unmappedrulesContentSelector</advislet>
        <advice-transform>
            RulesToContentTransform
        </advice-transform>
        <advislet>contentquery</advislet>
    </sequence>
</compound-advislet>
```

The `<sequence>` tag specifies the start of the sequence that makes up the compound. Entries can be either Advislets or AdviceTransforms which can occur in any order. The Advisor will invoke each element of the sequence in turn before proceeding to the next. The final Advice object generated will be returned to the user. In this way the implementation of the Advislet is hidden from the user who does not need to know whether a simple Advislet or a compound Advislet was used to generate the advice.

Creating Personalized Applications with the Advisor JSP Tags

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

- 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 content, including HTML tags, other JSP tags, and scriptlets. If the classifier rule returns `false`, the system skips the content between the start and end `<pz:div>` tags.

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

Note: For more information about how WebLogic Personalization Server manages content, see Chapter 9, "Creating and Managing Content," in this guide.

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

For information about defining a content selector rule, see the topic "Retrieving Documents with Content Selectors" in the *Guide to Using the E-Business Control Center*.

In addition to using JSP tags to create personalized applications, you can work directly with the Advisor bean. For more information about using the bean, see "Creating Personalized Applications with the Advisor Session Bean" on page 2-13.

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: Rules are created in the E-Business Control Center. This GUI tool is designed to allow Business Analysts (BAs) to develop their own classifier rules. Because the Business Analysts are not exposed to the concept of rules, you will see classifer rules referred to as "customer segmentation."

For information about creating classifier rules with the E-Business Control Center, see the section "Creating a New Customer Segment" in the topic "Using Customer Segments to Target High-Value Markets" in the *Guide to Using the E-Business Control Center*.

You can also use the Advisor bean directly to classify users. For more information, see "Classifying Users with the Advisor Session Bean" on page 2-15.

Example

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
rule="PremierCustomer">
    <p>Please check out our new Premier Customer bonus program…<p>
</pz:div>
```

Selecting Content with the <pz:contentQuery> JSP Tag

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

Note: For information about using the `<pz:contentQuery>` JSP tag, see "<pz:contentQuery>" on page 13-26. This tag provides similar functionality to the `<cm:select>` tag.

Example

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

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib uri="pz.tld" prefix="pz" %>
.
.
.
<pz:contentQuery id="docs"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
query="author = 'Hemingway'" />

<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

Note: For more information about these JSP tags, see "<cm:printProperty>" on page 13-11 and "<es:forEachInArray>" on page 13-71.

You can also use the Advisor bean directly to select content. For more information, see "Querying a Content Management System with the Advisor Session Bean" on page 2-16.

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

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

Notes: For more information about this tag, see "<pz:contentSelector>" on page 13-29.

For information about creating classifier rules, see the topic "Using Customer Segments to Target High-Value Markets" in the *Guide to Using the E-Business Control Center*.

Example

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

```
<%@ page import="com.bea.p13n.content.ContentHelper" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="es.tld" prefix="es" %>
.
.
.
<pz:contentSelector id="docs"
    rule="PremierCustomerSpotlight"
    contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>" />
<ul>
```

```
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

You can also use the Advisor bean directly to match content to users. For more information, see "Matching Content to Users with the Advisor Session Bean" on page 2-17.

Creating Personalized Applications with the Advisor Session Bean

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

Note: See the WebLogic Personalization Server *Javadoc* API documentation for more information about using the session bean to create personalized applications.

The following steps provide a general overview of the process involved for an application to get content recommendations from the Advisor.

1. Look up an instance of the Advisor session bean.
2. Use the AdvisorFactory's static `createAdviceRequest` method to create an AdviceRequest object.

Note: You must provide this method with the uri representing the request. The Advisor uses the uri prefix to determine which Advislet to invoke.

3. Set the required and optional attributes for the AdviceRequest object.
4. Call the Advisor's `getAdvice` method.

The Advisor calls the best Advislet to make the recommendation. The Advislet determines the recommendations and the Advisor then passes the resultant `Advice` object back to the application.

The Advisor uses the Advislet Registry to choose the Advislet to invoke.

5. The personalized application extracts the recommendation from the `Advice` object and uses it in the application.

When a personalized application requests advice from the Advisor, the Advisor bean delegates the request to a registered Advislet that can handle the request. The Advisor uses the uri prefix to determine which registered Advislet will receive the advice request. The Advislet then makes the recommendations and returns the `Advice` object back to the Advisor. This design encapsulates all of the advice logic into the Advislet and allows developers to create custom Advislets for more specialized purposes.

Attributes objects act as parameters for the request. Attributes objects can be set on the `AdviceRequest` object and are associated with a `String` object representing the name of the attribute.

Three Advislets are supplied with the system: Classifier Advislet, ContentQuery Advislet and ContentSelector Advislet. Names for the attributes that need to be set on the supplied Advislets are defined as static Strings in the `AdviceRequestConstants` interface.

Table 2-4 shows the logic the Advisor uses to determine how to map a recommendation request to an Advislet.

Table 2-4 Mapping a URI Prefix to an Advislet

Uri Prefix	Inferred Advislet
`classifier`	Uses a rules-based inference engine to classify a user based on rules written using the E-Business Control Center.
`contentselector`	■ Uses a rules-based inference engine to classify a user. ■ Determines if the user matches the classification. ■ Uses a rules-based inference engine to obtain a content query for the classification. ■ Selects content based on the content query obtained.

Table 2-4 Mapping a URI Prefix to an Advislet (Continued)

`contentquery`	Performs a content attribute search on a specified content management system.

The following sections demonstrate how to directly access the Advisor to provide the same functionality as that provided by the JSP tags.

Classifying Users with the Advisor Session Bean

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

Note: Unless otherwise indicated, all classes used here reside in the `com.bea.p13n.advisor` package.

1. Look up and create an instance of the Advisor session bean. The `EJB_REF_NAME` constant found in the EJB Advisor Home interface may be used as the JNDI name of the Advisor EJB Home.

2. Use the AdvisorFactory's static `createAdviceRequest` method to create an `AdviceRequest` object. In this case, the URI argument should be "`classifier://`".

3. Set the required attributes on the `AdviceRequest` object (see `AdviceRequestConstants`). These include:

   - `HTTP_REQUEST` – the request object (retrieved from `com.bea.p13n.httpRequest.createP13NRequest(HttpServletRequest)`).

   - `HTTP_SESSION` – the session object (retrieved from `com.bea.p13n.httpSession.createP13NSession(HttpServletRequest)`).

   - `USER` – the user object (retrieved from `com.bea.p13n.usermgmt.SessionHelper.getProfile(HttpServletRequest)`).

   - `TIME_INSTANT` – a `java.sql.Timestamp` object representing *now.*

- `RULES_RULENAME_TO_FIRE` – (optional) the name of the segmentation rule to fire. (For more information about customer segment rules, see Chapter 3, “Introducing the Rules Framework,” in this guide.)

4. Call the `getAdvise` method on the Advisor, supplying the newly created `AdviceRequest`.
5. The Advisor returns an instance of `Advice`. The `getResult` method is called to obtain the classification object. If a classification object is returned, then the classification is considered to be `true`. If the return value is `null`, the classification is considered to be `false`.

Note: If the optional Advise Request parameter `RULES_RULENAME_TO_FIRE` is not supplied, there may be multiple classifications returned for the user.

Querying a Content Management System with the Advisor Session Bean

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

Note: Unless otherwise indicated, all classes used here reside in the `com.bea.p13n.advisor` package.

1. Look up and create an instance of the Advisor session bean. The `EJB_REF_NAME` constant found in the EJB Advisor Home interface may be used as the JNDI name of the Advisor EJB Home.
2. Use the AdvisorFactory’s static `createAdviceRequest` method to create an `AdviceRequest` object. In this case, the URI argument should be “`contentquery://`”
3. Set the required attributes on the `AdviceRequest` object (see `AdviceRequestConstants`). These include:
   - `CONTENT_MANAGER_HOME` (required) – the JNDI name to find a content manager home interface.
   - `CONTENT_QUERY_STRING` (required) – the query to run against the system.

- `CONTENT_QUERY_SORT_BY` (optional) – the order in which to sort the returned results.
- `CONTENT_QUERY_MAX_ITEMS` (optional) – the maximum instances to return.

4. Call the `getAdvise` method on the Advisor, supplying the newly created `AdviceRequest`.
5. The Advisor returns an instance of `Advice`. The `getResult` method is called to obtain the array of `Content` objects representing the results of the content query.

Matching Content to Users with the Advisor Session Bean

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

Note: Unless otherwise indicated, all classes used here reside in the `com.bea.p13n.advisor` package.

1. Look up and create an instance of the Advisor session bean. The `EJB_REF_NAME` constant found in the EJB Advisor Home interface may be used as the JNDI name of the Advisor EJB Home.
2. Use the AdvisorFactory's static `createAdviceRequest` method to create an `AdviceRequest` object. In this case the uri argument should be "`contentselector://`"
3. Set the required attributes on the `AdviceRequest` object (see `AdviceRequestConstants`). These include:
   - `HTTP_REQUEST` – the request object (retrieved from `com.bea.p13n.httpRequest.createP13NRequest(HttpServletRequest)`).
   - `HTTP_SESSION` – the session object (retrieved from `com.bea.p13n.httpSession.createP13NSession(HttpServletRequest)`).

- `USER` – the user object (retrieved from `com.bea.p13n.usermgmt.SessionHelper.getProfile(HttpServletRequest)`).
- `TIME_INSTANT` – a `java.sql.Timestamp` object representing *now.*
- `RULES_RULENAME_TO_FIRE` – (optional) the name of the segmentation rule to fire. (For more information about customer segment rules, see Chapter 3, "Introducing the Rules Framework," in this guide.)
- `CONTENT_MANAGER_HOME` (required) – the JNDI name to find a content manager home interface.
- `CONTENT_QUERY_STRING` (required) – the query to run against the system.
- `CONTENT_QUERY_SORT_BY` (optional) – the order in which to sort the returned results.
- `CONTENT_QUERY_MAX_ITEMS` (optional) – the maximum instances to return.

4. Call the `getAdvise` method on the Advisor, which supplies the newly created `AdviceRequest`.
5. The Advisor returns an instance of `Advice`. The getResult method is called to obtain the array of `Content` objects representing the recommendation.

CHAPTER

3 Introducing the Rules Framework

Rules Management forms a key part of the personalization process by prescribing a flexible and powerful mechanism for expressing business rules. The business logic encompassed by these rules allows robust delivery of personalized content marketed specifically to each end user type.

This topic includes the following sections:

What Is the Rules Manager?

WebLogic Personalization Server offers a robust personalization solution through a set of components that provide edit-time and run-time services for delivering personalized content to end users while browsing a Web site. These personalization components use business rules to match users and groups with appropriate content. The logic encompassed by the rules forms a critical piece of the personalization process.

1. The Rules Management Framework in WebLogic Personalization Server provides editing, deploying, and run-time capabilities for providing personalized content based on externalized rules. This component includes two major parts: an edit-time GUI that allows Business Analysts to define and deploy business rules, and a run-time service for evaluating defined business rules.
2. The Rules Manager EJB is a scalable, stateless J2EE entry point into the underlying BEA Rules Engine. It provides the run-time services necessary to execute the business rules defined in the E-Business Control Center.

Rules are created in the E-Business Control Center. This GUI tool is designed to allow Business Analysts to develop their own content selector rules and classifier rules. Because the Business Analysts are not exposed to the concept of rules, you will see content selector rules called simply “content selectors” and classifier rules referred to as “customer segments.”

Business rules can by dynamically loaded or modified from the E-Business Control Center into a running server. This allows the Business Analyst to modify the site’s flow and logic, and reduces their reliance on programmers.

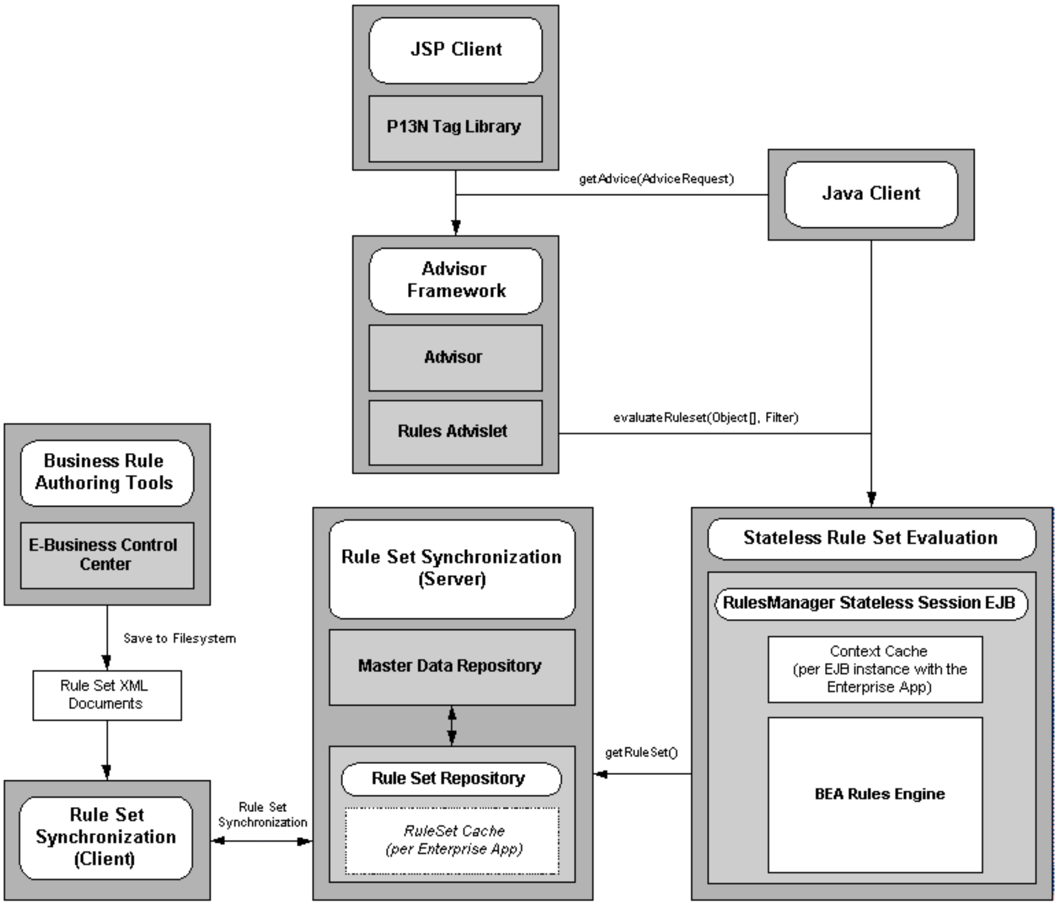
Well-known Objects

The Rules Management component uses several well-known objects during the rule evaluation process:

- `TIME_INSTANT`: A well-known object in the rule editor, of type `com.bea.p13n.xml.schema TimeInstant` that corresponds to the instant of a user request.

- `User`: For each call to the rules engine, a single `User` object will be provided for use by the rules. `User` has a fixed schema, determined dynamically at edit time by calling the User Management component. Given that the `User` might have a `Numeric` schema attribute called *`age`*, a valid rule condition might be: `User.age > 35.`
- `Request`: This object is used in the same way as the `User` object. The `Request` properties are defined in a default property set. This object encapsulates the information that is available from the `HttpServletRequest`. (For more information, see "Default Request Property Set" on page 6-3 in the "Foundation Classes and Utilities" topic of this guide.)
- `Session`: This object is used in the same way as the `User` object. The `Session` properties are defined in a default property set. This object encapsulates the information that is available from the `HttpSession.` (For more information, see the "Default Session Property Set" on page 6-5 in the "Foundation Classes and Utilities" topic of this guide.)

Figure 3-1 The Rules Framework


JSP Client
P13N Tag Library
getAdvice(AdviceRequest)
Java Client
Advisor Framework
Advisor
Rules Advislet
evaluateRuleset(Object[], Filter)
Business Rule Authoring Tools
E-Business Control Center
Save to Filesystem
Rule Set XML Documents
Rule Set Synchronization (Client)
Rule Set Synchronization
Rule Set Synchronization (Server)
Master Data Repository
Rule Set Repository
RuleSet Cache (per Enterprise App)
getRuleSet()
Stateless Rule Set Evaluation
RulesManager Stateless Session EJB
Context Cache (per EJB instance with the Enterprise App)
BEA Rules Engine

How the Rules Engine Works

The Rules Engine functions by applying a set of rules to objects in working memory. This working memory is first populated with input from the calling objects, such as the user profile request session, among other things. In this way, a representation of the user's profile exists in working memory before any rules are actually fired.

Rules can be executed only within a context. The context associates a rule set with working memory. The context provides an interface to the Rules Engine that controls the relationship between the rule part of the application and the working memory.

This working memory is operated on by the production rules, which are contained in rule sets. The left-hand sides (LHS) of these rules are evaluated against the objects in the working memory. If the patterns on the LHS are matched, then the actions contained in the right-hand side (RHS) of the rules are performed. Some of these actions may add new objects into the working memory. For example, if our Classifier rule tests for `USER.age > 45`, then we might add a new `Classification` object into working memory.

The production system is executed by performing the following operations:

1. Match: Evaluates the LHSs of the rules to determine which are satisfied given the current contents of working memory.
2. Conflict resolution: Selects one rule with a satisfied LHS. If no rules have satisfied the LHSs, halts the interpreter.
3. Act: Performs the actions in the RHS of the selected rule.
4. Go to step 1.

Rules will continue to operate on the working memory until the conflict resolution set is zero (that is, no more rules can fire).

After the Rules Engine has halted, the rules manager component returns a list of objects remaining in working memory. A likely scenario will have an object remaining of the type “Classification” or “ContentQuery.”

The Rules Manager will then iterate over these remaining objects and filter them using an optional Object filter. The filter can selectively ignore objects or mutate them.

What Are Rule Sets?

The BEA WebLogic Personalization Server provides two rule sets that act as containers for the rules created in the E-Business Control Center: the global classifications rule set and the global content selectors rule set.

Rules within a rule set may refer to any properties. In general, you should not change or delete properties if a rule refers to it. Adding properties does not affect existing rules.

Deploying Rule Sets

Rules sets are represented as XML files that can be read and edited by the E-Business Control Center. In order to propagate Rule Set changes that have been made in the E-Business Control Center, the corresponding Rule Set XML documents must be synchronized with the target application. In order to do so, the Rules Framework takes advantage of the WebLogic Personalization Server's Data Synchronization Framework. For more information about the data synchronization frameset, see the topic "Synchronizing Application Data" in the*Deployment Guide*.

When a user decides to deploy Rule Set changes from the E-Business Control Center to an application, the Data Synchronization Framework propagates Rule Set XML document updates to a Rules Manager instance within the target application. The Rules Manager parses the new Rule Set XML document into a binary representation understood by the BEA Rules Engine, and thereafter, application calls to the Rules Manager will use the new Rule Sets during rule execution.

It is important to note that the Rules Set XML documents that are managed by a particular instance of the Rules Manager are scoped to that application. That is, changes to Rule Sets are only seen by the target application and not by any other applications.

It is also important to note that changes to Rule Sets are immediate, and unlike previous WebLogic Personalization Server release, the Rules Manager no longer relies upon a time-to-live in order to propagate Rule Set changes throughout a cluster.

Classifier Rules

Classifier rules are created in the E-Business Control Center. For information and instruction on creating classifier rules (called "customer segments" in the E-Business Control Center), see the topic "Using Customer Segments to Target High-Value Markets" in the *Guide to Using the E-Business Control Center*.

Classifier rules dynamically categorize users into groups (user segments). A classifier rule determines if a user profile meets a set of conditions and places the user in a category based upon the result. Essentially, if the user profile meets the conditions, it is classified according to the classifier rule; if it does not meet the classification conditions, the user profile is not included in the classification group.

The following examples use pseudo-code to illustrate the logic involved in processing a classifier rule. Note the implicit *and* between the rule phrases.

This rule classifies a user who is male and is accessing the site between December 1 and December 26:

```
Classifier MaleChristmasShopper
If User has the following characteristics:
    User.gender == “male” or “M”
    and Date > 12/1 AND Date < 12/26
```

This rule classifies a user whose annual income is over one hundred thousand dollars:

```
Classifier HighEarner
If User has the following characteristics:
    User.income > 100000
```

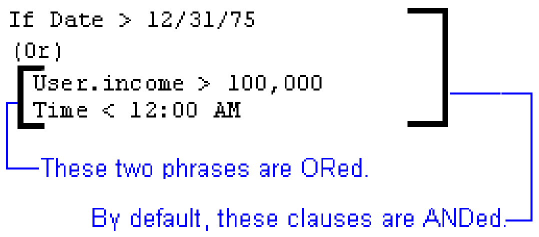
Classifier rules are the building blocks of more complicated rules. Content selector rules can use classifier rules as they select personalized content to match a user or group profile. (See “Content Selector Rules” below.)

Use the `<pz:div>` JSP tag to execute a classifier rule in a JSP page. For a complete listing, see “<pz:div>” on page 13-34 in the “JSP Tag Library Reference” topic of this guide.

The AND and OR operators

Figure 3-2 shows an example of clauses ANDed and ORed together. By default, all clauses in a rule are ANDed together.The OR operator is applied to nested (indented) child clauses below the OR operator. In that case, the nested statements are ORed, and ANDed to clauses not nested around them. This simple illustration uses pseudo-code, and does not represent the actual XML used to represent these rules.

Figure 3-2 AND and OR Example


If Date > 12/31/75
(Or)
User.income > 100,000
Time < 12:00 AM
These two phrases are ORed.
By default, these clauses are ANDed.

Content Selector Rules

Content selectors are created in the E-Business Control Center. For instructions on using the GUI tool to create content selectors, see the E-Business Control Center online help. A copy of the information presented in online help is available on the e-docs Web site—see the topic "Retrieving Documents with Content Selectors" in the *Guide to Using the E-Business Control Center*.

Content selector rules invoke rule-based content queries that return content based on the user profile. This type of rule may use references to classifier rules to define it. It also produces dynamic queries at runtime to select content from a document collection.

The power of producing dynamic queries that match content with user profiles allows content selectors to deliver highly customized content to end users. Since content selector rules can use queries to select content based on run-time parameters, they allow the system to match personalized content to user profiles.

Note: Although a profile may meet the criteria of a content selector rule, the rule may not return any content objects. Why? If no content matches the query's criteria, the query cannot return a content object.

You can use the `<pz:contentSelector>` JSP tag to invoke content selector rules in JSP pages. (For a complete listing, see "<pz:contentSelector>" on page 13-29 in the topic "JSP Tag Library Reference" in this guide.)

For an in-depth look at using content selectors, see Chapter 4, "Working with Content Selectors," in this guide.

Debugging Rule Sets

Note: The underlying structure of the Rules Engine has been enhanced for WebLogic Personalization Server release 4.0. If you have created rules and rule sets in previous versions of this product, please see the *Migration Guide* for additional information.

What Is the Relationship Between Property Sets and Rules?

You might notice that a rule set you have used in the past begins to function incorrectly. This behavior is probably due to a change in the property set with which the rule set has a relationship.

Rules rely on property sets to provide the properties they use to evaluate user and group profiles. If a property is modified after a rule that uses it has been created, rules may contain dangling references to properties that no longer exist or that have been changed.

As much as possible, you should avoid modifying properties after defining rules that rely upon them. Since the property set defines the schema for the properties the rules act upon, any change to the properties the rules use will affect the schema and may alter the validity of the rules. In general, be careful when modifying or deleting existing properties.

Note: You can add properties without affecting existing rules.

Content Type and Content Selector Rules

Another problem can occur when you change a content's metadata types after creating a content selector rule based on that content type's metadata. Remember that the content selector rule relies upon metadata to locate content. If you change content metadata and a content selector rule references the previous metadata, the rule will not work correctly.

Configuring the Rules Framework

The various components of the Rules Framework are configured with an external configuration file called `rules.properties`. This file resides in the `p13n_util.jar` JAR file (within the `com/bea/p13n/rules` directory) that can be found in the root directory of any WebLogic Personalization Server application. This section explains each of the configuration properties that can be set in this file.

Note: Changes to the `rules.properties` file are only seen by the application in which the file resides. That is, this configuration file is scoped to the application. This makes it possible to configure the Rules Framework differently for different applications.

Rules Engine Expression Validation

If this property is set to `true`, the BEA Rules Engine will validate all Rules expressions (both conditions and actions) exactly one time. This property may be set to `true` during development and testing for additional expression validation.

```
##
# Rules engine expression validation:
#
# If this property is set to true, the rules engine
# will validate expressions the first time they are
# executed.
##

rules.engine.expression.validation=false
```

Rules Engine Error Handling and Reporting

The following two properties determine the type of exceptions that will be propagated to the user during Rules Engine execution. If the `rules.engine.throw.expression.exceptions` parameter is set to `false`, no exceptions will be propagated, and any condition expression that generates an exception will evaluate to false. Otherwise, all exceptions, except those listed with the `rules.engine.ignorable.exceptions` parameter, will be propagated to the user.

```
##
# Rules engine pattern expression execution error handling:
#
# rules.engine.throw.expression.exceptions

#
# If this property is set to true, pattern expression
# execution exceptions will be thrown. Otherwise, a pattern
# expression exception will cause the pattern condition to
# evaluate to false.
#
# Defaults to true.
#
# rules.engine.throwable.exceptions (list of class names)
#
# If the previous property is set to true, expression exceptions
# with embedded exceptions of a type other than the listed classes
# will be thrown. If no class types are specified, all expression
# exceptions will be thrown.
#
# Defaults to all exception class types.
##

rules.engine.throw.expression.exceptions=true
rules.engine.ignorable.exceptions=java.lang.NullPointerException
```

Rules Engine Listeners

This is an internal property and should not be modified.

```
##
# Rules engine startup rule event listeners (list of class names).
##

rules.engine.startup.listeners=
```

Rules Engine Expression Caching Optimizations

This is an internal property and should not be modified.

```
##
# Rules engine expression optimizations:
#
# 0 => No expression optimizations.
# 1 => Local expression optimizations.
# 2 => Global expression optimizations.
#
# Defaults to 0.
##

rules.engine.expression.optimizations=2
```

Rules Parser

The following are internal properties and should not be modified.

```
##
# Rule Set Parser Node Support Classes
#
# This property supports a comma-delimited list of classes
# extending the base AST NodeSupport class. Such classes
# provide node creation support for rules-schema namespaces
# required for constructing the intermediate AST representing a
# given RuleSet instance.
#
# All NodeSupport subclasses must co-exist peacefully with the
# required CoreNodeSupport instance.
##

parser.node.support.list=\
    com.bea.p13n.expression.internal.parser.expression.
    ExpressionNodeSupport,\
    com.bea.p13n.rules.internal.parser.wlcs.WlcsNodeSupport,\
    com.bea.p13n.content.query.ContentQueryNodeSupport

##
# Rule Set Parser Transform Visitor Class
#
# This property specifies the ExpressionTranformVisitor or
# subclass to be used for intermediate AST-to-RuleSet
# transformations.
```

```
#
##

parser.transform=\
    com.bea.p13n.rules.internal.parser.wlcs.WlcsTransformVisitor
```

CHAPTER

4 Working with Content Selectors

A content selector is one of several mechanisms that WebLogic Portal provides for retrieving documents from a content management system. A *document* is a graphic, a segment of HTML or plain text, or a file that must be viewed with a plug-in. (We recommend that you store most of your Web site's dynamic documents in a content management system because it offers an effective way to store and manage information.)

Using content selectors, a Business Analyst (BA) can specify conditions under which WebLogic Portal retrieves one or more documents. For example, a BA can use a content selector to encapsulate the following conditions: between May 1 and May 10, if a Gold Customer views this page, find and retrieve any documents that describe sailing along the Maine coast.

A Business Analyst uses the BEA E-Business Control Center to define the conditions that activate a content selector and to define the query the content selector uses to find and retrieve documents. Then, a Business Engineer (BE) uses content selector JSP tags and a set of other JSP tags to retrieve and display the content targeted by the content selector.

This topic includes the following sections:

- What Are Content Selectors?
- Using Content-Selector Tags and Associated JSP Tags
- How Content Selectors Select Documents

For a comparison of content retrieval methods available with WebLogic Personalization Server, see "Methods for Retrieving and Displaying Documents" on page 9-5.

For a technical discussion of content management in the WebLogic Personalization Server, see the topic "Creating and Managing Content" in this guide.

What Are Content Selectors?

Content selectors consist of the following elements:

- A set of conditions that determine when the content selector queries the content management system. The conditions can use the profile of the customer who is currently viewing a JSP page, properties from the user or session objects, or the current date/time. For a complete list of conditions, see the section "Conditions That Activate Content-Selector Queries," in the topic "Retrieving Documents with Content Selectors" in the *Guide to Using the E-Business Control Center*.

  BAs create and modify the set of conditions in the E-Business Control Center.

- A query that searches the content management system for one or more documents.

  BAs create and modify the query in the E-Business Control Center.

- A JSP tag that triggers the content selector to evaluate its conditions. The content selector JSP tag includes attributes that BEs can use to tune the performance of the content selection process. BEs use the JSP tags.

- A data object that WebLogic Personalization Server creates to contain the results of the query. Within the data object, WebLogic Personalization Server creates a list of individual data items (an array); the contents of each document in the data object is a separate item in the array. You can access the array only from the current JSP page, and only for the customer request that created it.

  To extend the availability of the data in the array, BEs can add attributes to the content selector JSP tag that cause WebLogic Portal to store the array in a cache. BEs can specify whether the scope of the cache applies to the application, session, page, or request.

To display the documents that are in the array (or the cache), a BE must use the `<es:forEachInArray>` tag. Depending on the scope of the cache, a `<es:forEachInArray>` can access a content-selector cache that WebLogic Portal created for another page and/or for another user.

Using Content-Selector Tags and Associated JSP Tags

To use the content selector features on a given JSP, a BE must add calls to the content selector JSP tag and a set of associated tags.

This section contains the following subsections:

- Attributes of the <pz:contentSelector> Tag
- Associated Tags That Support Content Selectors
- Common Uses of Content-Selector Tags and Associated Tags

For more information about the tags discussed here, see the topic “Personalization Server JSP Tag Library Reference” in this guide.

Attributes of the <pz:contentSelector> Tag

While BAs use the E-Business Control Center to configure the dynamic properties of a content selector, a BE uses attributes of the content selector tag to do the following:

- Identify the Content Selector Definition
- Identify the JNDI Home for the Content Management System
- Define the Array That Contains Query Results
- Create and Configure the Cache to Improve Performance

For a complete list and description of all content-selector attributes, see “<pz:contentSelector>” on page 13-29 in the topic “Personalization Server JSP Tag Library Reference” in this guide.

Identify the Content Selector Definition

The content selector definition that a BA creates in the E-Business Control Center determines the conditions that activate a content selector and the query that the active content selector runs.

To refer to this definition, you use the `rule` attribute:

```
<pz:contentSelector rule= { definition-name | scriptlet } >
```

You can use a scriptlet to determine the value of the `rule` attribute based on additional criteria. For example, you use a content selector in a heading JSP (`heading.inc`), which is included in other JSPs. A BA creates different content selectors for each page that includes `heading.inc`.

The BE uses a scriptlet in `heading.inc` to provide a value based on the page that currently displays the included JSP file. For example,

```
<%

   String banner = (String)pageContext.getAttribute("bannerPh");
   banner = (banner == null) ? "cs_top_generic" : banner;

%>

<!-- ------------------------------------------------------------- -->

<table width="100%" border="0" cellspacing="0" cellpadding="0" height="108">

  <tr><td rowspan="2" width="147" height="108">
  <pz:contentSelector rule="<%= banner %>" ... />

</td>
```

Identify the JNDI Home for the Content Management System

The content selector tag must use the `contentHome` attribute to specify the JNDI home of the content management system. If you use the reference content management system or a third-party integration, you can use a scriptlet to refer to the default content home. Because the scriptlet uses the `ContentHelper` class, you must first use the following tag to import the class into the JSP:

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
```

Then, when you use the content selector tag, specify the `contentHome` as follows:

```
<pz:contentSelector
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
... />
```

If you create your own content management system, you must specify the JNDI home for your system instead of using the ContentHelper scriptlet. In addition, if your content management system provides a JNDI home, you can specify that one instead of using the ContentHelper scriptlet.

Define the Array That Contains Query Results

You can use the following attributes to configure the array that contains the results of the content-selector query:

- `id`, which specifies a name for the array. This attribute is required.

  For example, `<pz:contentSelector id="docs" .../>` places documents in an array named `docs`.

- `max`, which limits the number of documents the content selector places in its array.

  For example, `<pz:contentSelector max="10" .../>` causes the content selector to stop retrieving documents when the array contains 10 documents.

  This attribute is optional and defaults to `-1`, which means no maximum.

- `sortBy`, which uses one or more document attribute to sort the documents in the array. The syntax for `sortBy` follows the SQL *order by clause* syntax.

  This attribute is optional. If you do not specify this attribute, the content selector returns the query results in the order that the content management system returns them.

  For example, `<pz:contentSelector sortBy="creationDate" .../>` places the documents that were created first at the beginning of the array.

  The tag
  `<pz:contentSelector sortBy="creationDate ASC, title DESC" .../>` places older documents at the beginning of the array. If any documents were created on the same day, it sorts those documents counter-alphabetically by title.

Create and Configure the Cache to Improve Performance

To extend accessibility of retrieved content, and to improve performance, you can optionally use content-selector attributes to create and configure a cache that contains the array contents. Without the cache, you can access the content-selector array only from the current JSP page, and only for the customer request that created it. In addition, each time a customer requests a JSP that contains the content selector tag, the content selector must run the query, potentially slowing the overall performance of WebLogic Personalization Server. To cache the contents of the array, use the following attributes:

- `useCache`, which determines whether the content selector places the array in a cache. To activate the cache, set this attribute to `true`. For example, `<pz:contentSelector cache="true" ...>`.

  To deactivate the cache, set the attribute to `false` or do not include it. For example, the following statements are equivalent:
  `<pz:contentSelector cache="false" .../>` or
  `<pz:contentSelector .../>`

- `cacheId`, which assigns a name to the cache. If you do not specify this attribute, the cache uses the name of the array (which you must specify with the `id` attribute). If you want to access the cache from a JSP or user session other than the one that created the array, you must specify a `cacheId`.

- `cacheTimeout`, which specifies the number of milliseconds that WebLogic Personalization Server maintains the cache. The content selector does not re-run the query until the number of seconds expires.

  For example, you create the following tag:
  `<pz:contentSelector cache="true" cacheTimeout=”300000” .../>`

  A customer requests the page that contains this content selector tag. The user leaves the page but, 2 minutes (120000 milliseconds) later, requests it again. The content selector evaluates its conditions, but because only 120000 milliseconds have expired since the content selector created the cache, it does not re-run the query. Instead, it displays the documents in the cache.

- `cacheScope`, which determines from where the cache can be accessed. You can provide the following values for this attribute:

  - `application`. Any JSP page in the Web application that any customer requests can access the cache.
  - `session` (the default). Any JSP in the Web application that the current customer requests can access the cache.

- `page`. Only the current JSP that any customer requests can access the cache.
- `request`. Only the current user request can access the cache. If a customer re-requests the page, the content selector re-runs the query and recreates the cache.

Associated Tags That Support Content Selectors

The following JSP tags support content-selector functions:

- `<um:getProfile>`, which retrieves the profile of the customer who is currently viewing the page. A content selector uses the customer profile to evaluate any conditions that involve customer properties.

  For example, if you create a content selector that runs a query for all customers in the Gold Customer customer segment, the content selector must access the customer profile to determine if it matches the customer segment.

  Even if a content selector does not currently use the customer profile for its conditions, we recommend that you include the `<um:getProfile>` tag; its affect on performance is minimal and with the tag, a BA can add customer-profile conditions to the content selector without requiring a BE to modify JSPs.

  The tag must be located closer to the beginning of the JSP than the content selector tag.

- `<es:forEachInArray>`, which iterates through the array that contains the results of a content-selector query. With this tag, you can use the following to work with the documents in the array:

  - The `System.out.println` method to print each item in the array.
  - The `<cm:getProperty>` tag to retrieve one or more attribute of the documents in the array. You can use the attributes to construct the HTML that a browser requires to display the documents. For example, you use the `<cm:getProperty>` tag to determine the value of a `MIME-type` attribute. If the MIME-type of a document in the array is an image, you print the HTML `<img>` tag with the appropriate attributes.

    You can also use attributes of the `<pz:contentSelector>` tag, such as `sortBy`, to work with the attributes of documents in the array. For more information, see "Attributes of the <pz:contentSelector> Tag" on page 4-3.

- The `<cm:printProperty>` to print one or more attribute of the documents in the array. For example, you can use this tag to print a list of document titles that the content selector retrieves.

Common Uses of Content-Selector Tags and Associated Tags

The combination of content selector definitions, tag attributes, and associated JSP tags creates a powerful set of tools for matching documents to customers in specific contexts. The following tasks are the most common uses of content selectors and associated tags:

- To Retrieve and Display Text-Type Documents
- To Retrieve and Display Image-Type Documents
- To Retrieve and Display a List of Documents
- To Access a Content-Selector Cache on a Different JSP

To Retrieve and Display Text-Type Documents

Note: This section assumes that the content selector query that the BA created in E-Business Control Center includes a filter to retrieve only text documents.

1. Open a JSP in a text editor.
2. Near the beginning of the JSP, add the following lines to import classes and tag libraries if they are not already in the JSP:

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
```

3. Add the following tag to get the customer profile, if the tag is not already in the JSP:

```
<um:getProfile>
```

If the JSP already uses this tag for some other purpose, it probably includes other attributes. Make sure that the tag is closer to the beginning of the JSP than the `<pz:contentSelector>` tag, which you use in the next step.

4. Add the following tags, where `SpringSailing` is the name of the content selector that a BA created in the E-Business Control Center:

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="textDocs"/>
<es:forEachInArray array="<%=textDocs%>" id="aTextDoc"
type="com.bea.p13n.content.Content">

  <p><cm:printDoc id="aTextDoc"/></p>

</es:forEachInArray>
```

Note: The above tags assume that the content selector query that the BA created in the E-Business Control Center includes a filter to retrieve only text documents. To verify the content type before you display it, you can surround the `<% "<P>" + aTextDoc + "</P>" %>` scriptlet with another scriptlet. For example:

```
<% if (aTextDoc.getMimeType().contains("text") != -1)
{
  %>
    <p><cm:printDoc id="aTextDoc"/></p>
<%
}
%>
```

5. Save the JSP. If you deploy the Web application as a `WAR` file, re-jar the Web application and deploy it.

   WebLogic Portal deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Portal waits for the page-check interval to expire before deploying any changes. For more information about setting the page-check interval, see the *Performance Tuning Guide*.

To Retrieve and Display Image-Type Documents

1. Open a JSP in a text editor.

2. Near the beginning of the JSP, add the following lines to import classes and tag libraries if they are not already in the JSP:

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
```

3. Add the following tag to get the customer profile, if the tag is not already in the JSP:

   ```
   <um:getProfile>
   ```

   If the JSP already uses this tag for some other purpose, it probably includes other attributes. Make sure that the tag is closer to the beginning of the JSP than the `<pz:contentSelector>` tag, which you create in the next step.

4. Add the following tags, where `SpringSailing` is the name of the content selector that a BA created in the E-Business Control Center:

   ```
   <pz:contentSelector rule="SpringSailing"
   contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
   id="ImageDocs"/>

   <es:forEachInArray array="<%=ImageDocs%>" id="anImageDoc"
   type="com.bea.p13n.content.Content">

     <img src="ShowDoc/<cm:printProperty

     id="anImageDoc" name="identifier" encode="url"/>"

   </es:forEachInArray>
   ```

   Note: The above tags assume that the content selector query that the BA created in E-Business Control Center includes a filter to retrieve only image documents. To verify the content type before you display it, you can surround the `<img>` tag with a scriptlet. For example:

   ```
   <% if (anImageDoc .getMimeType().contains("image"))
   {
   %>
     <img src="ShowDoc/<cm:printProperty
     id="anImageDoc" name="identifier" encode="url"/>">
   }
   %>
   ```

5. Save the JSP. If you deploy the Web application as a `WAR` file, re-jar the Web application and deploy it.

   WebLogic Portal deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Portal waits for the page-check interval to expire before deploying any changes. For more information about setting the page-check interval, see the *Performance Tuning Guide*.

To Retrieve and Display a List of Documents

1. Open a JSP in a text editor.
2. Near the beginning of the JSP, add the following lines to import classes and tag libraries if they are not already in the JSP:

   ```
   <%@ page import="com.bea.p13n.content.ContentHelper"%> <%@
   taglib uri="es.tld" prefix="es" %>
   <%@ taglib uri="pz.tld" prefix="pz" %>
   <%@ taglib uri="um.tld" prefix="um" %>
   ```

3. Add the following tag to get the customer profile, if the tag is not already in the JSP:

   ```
   <um:getProfile>
   ```

   If the JSP already uses this tag for some other purpose, it probably includes other attributes. Make sure that the tag is closer to the beginning of the JSP than the `<pz:contentSelector>` tag, which you create in the next step.

4. Add the following tags, where `SpringSailing` is the name of the content selector that a BA created in the E-Business Control Center:

   ```
   <pz:contentSelector rule="SpringSailing"
   contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
   id="docs"/>

   <ul>
      <es:forEachInArray array="<%=docs%>" id="aDoc"
      type="com.bea.p13n.content.Content">

      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />

      </es:forEachInArray>

   </ul>
   ```

5. Save the JSP. If you deploy the Web application as a `WAR` file, re-jar the Web application and deploy it.

   WebLogic Portal deploys the modifications. If you specified a page-check rate for your Web application, WebLogic Portal waits for the page-check interval to expire before deploying any changes. For more information about setting the page-check interval, see the *Performance Tuning Guide*.

To Access a Content-Selector Cache on a Different JSP

1. In a text editor, open the JSP page that contains the content selector tag. For example, you want to cache the results of the following tag:
   `<pz:contentSelector rule="SpringSailing" id="docs".../>`
2. Add attributes to the content selector tag as follows:

   ```
   <pz:contentSelector rule="SpringSailing"
   contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
   id="docs"
   useCache="true" cacheId="SpringSailingDocs"
   cacheTimeout="120000"
   cacheScope="application" />
   ```

   These attributes create a cache that WebLogic Portal maintains for 2 minutes (120000 milliseconds) and that can be accessed using the name `SpringSailingDocs` by any user from any page in the Web application. For more information about possible values for cacheScope, see "Create and Configure the Cache to Improve Performance" on page 4-6.
3. Save and deploy the JSP.
4. In a text editor, open the JSP from which you want to access the cache.
5. Use a content-selector tag that is identical to the tag you created in step 2. For example, on the current JSP, add the following tag:

   ```
   <pz:contentSelector rule="SpringSailing"
   contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
   id="docs"
   useCache="true" cacheId="SpringSailingDocs"
   cacheTimeout="120000"
   cacheScope="application" />
   ```
6. Save and deploy the JSP.

How Content Selectors Select Documents

When a user requests a JSP that contains a content selector tag, the following process occurs:

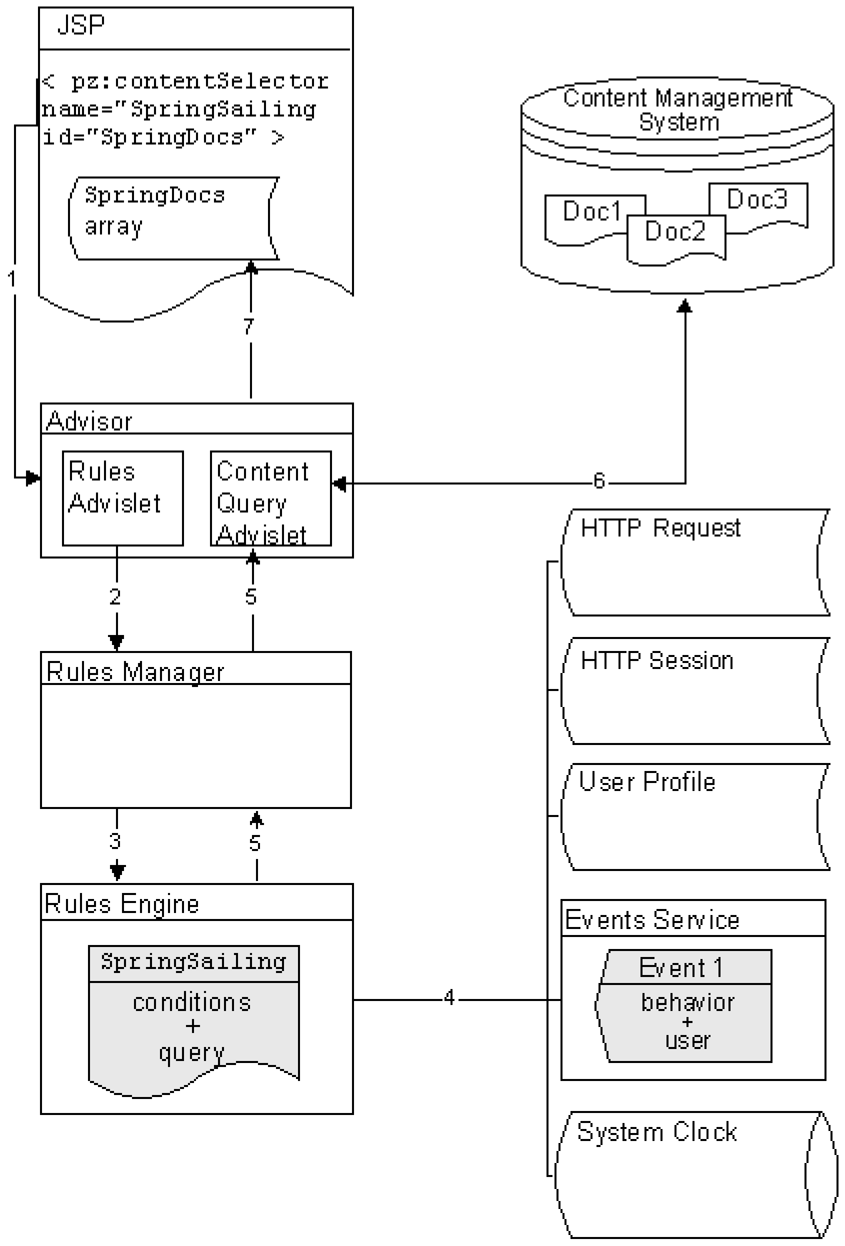
1. The content selector tag contacts the Advisor.

Note: For information about the Advisor, see Chapter 2, "Creating Personalized Applications with the Advisor."
For information about the Rules Engine, see Chapter 3, "Introducing the Rules Framework."

2. The Advisor forwards the content-selector request to the Rules Manager via the Rules Advislet.
3. The Rules Manager finds the corresponding content-selector definition and invokes the Rules Engine to evaluate the content selector's conditions.
4. Depending on the conditions that are defined for the content selector, the Rules Engine refers to any of the following:
   - The profile of the user who requested the JSP to determine if the user matches a customer segment or some other attribute that conditions in the content selector specify.
   - The HTTP Request and/or the HTTP Session attributes
   - The system clock to determine if the current time or date matches any time or date that conditions in the content selector specify.
5. If any of the conditions are met, the Rules Engine returns the content selector's query to the Advisor via the Rules Manager.
6. The Advisor forwards the query to the content management system via the Content Query Advislet.
7. The Advisor stores any query results in an array that only the current JSP can access. You can specify that the Advisor stores the results in a cache and that the cache is accessible beyond the current JSP invocation. For more information, see "Create and Configure the Cache to Improve Performance" on page 4-6.

Note that you must use other tags to display the documents that are in the array.

Figure 4-1 How Content Selectors Select Documents



CHAPTER

5 Using the Expression Package

This topic illustrates how to use the services of the Expression Package. Any arithmetic, boolean, relational or conditional statement can be represented in the Java object model by using an Expression Package. You can use the Expression Package to dynamically assemble and evaluate your own business logic.

This topic includes the following sections:

- Working with Expressions
    - The Expression Factory
    - Expression Package Services
- Code Examples
    - Stateful Evaluation of a Simple Expression
    - Stateful Evaluation of an Expression Containing Variables
    - Stateless Validation and Evaluation of an Expression Containing Variables
    - Stateful Validation and Evaluation of an Expression Containing Variables
- Expression Package Configuration Settings

Introducing the Expression Package

The Expression Package allows users to dynamically assemble and execute Java-based expressions. The package defines a set of Java classes that represent various types of expression operators, and contains services for evaluating expressions consisting of instances of these operators.

What Is the Expression Package?

The Expression Package includes a base Expression class, a Variable class, and the following operator classes for operating on Expressions and Variables:

- Basic language operators (object creation, method call, etc.)
- Logical operators
- Comparative operators
- Collection operators
- Mathematical operators
- String operators

The Expression Package also includes the following services for operating on expressions:

- `Unifier`—prepares an expression for evaluation.
- `Validator`—validates that an expression is well-formed before evaluation.
- `Optimizer`—optimizes the structure of an expression before evaluation.
- `Evaluator`—evaluates an expression and returns the result of evaluation.
- `Executor`—an aggregate service that combines the unification, validation, and evaluation processes.

Unlike an expression written directly in Java and executed from within a Java program, the Expression Package allows you to dynamically assemble and modify expressions from within your Java programs. An expression may be modified any number of times both before and after evaluation. When you assemble expressions using the Expression Package you can also take advantage the advanced features of the Expression Package, such as expression caching, validation, and optimization.

The Expression Package serves as the foundation of the BEA Rules Engine. The Rules Engine leverages the package in order to represent and evaluate rule condition and action expressions. Likewise, you can use the Expression Package to dynamically assemble and evaluate your own business logic.

The Package Structure for the Expression Package

The Expression Package interfaces and abstract classes can be found in the following package: `com.bea.p13n.expression`

The Expression Package operators are organized in the following packages:

Basic language operators—`com.bea.p13n.expression.operators`

Logical operators—`com.bea.p13n.expression.operators.logical`

String operators—`com.bea.p13n.expression.operators.string`

Mathematical operators—`com.bea.p13n.expression.operators.math`

Comparative operators—`com.bea.p13n.expression.operators.comparative`

Collection operators—`com.bea.p13n.expression.operators.collection`

The Expression Package related classes are packaged in the `p13n_util.jar` archive.

Assembling and Managing Expressions

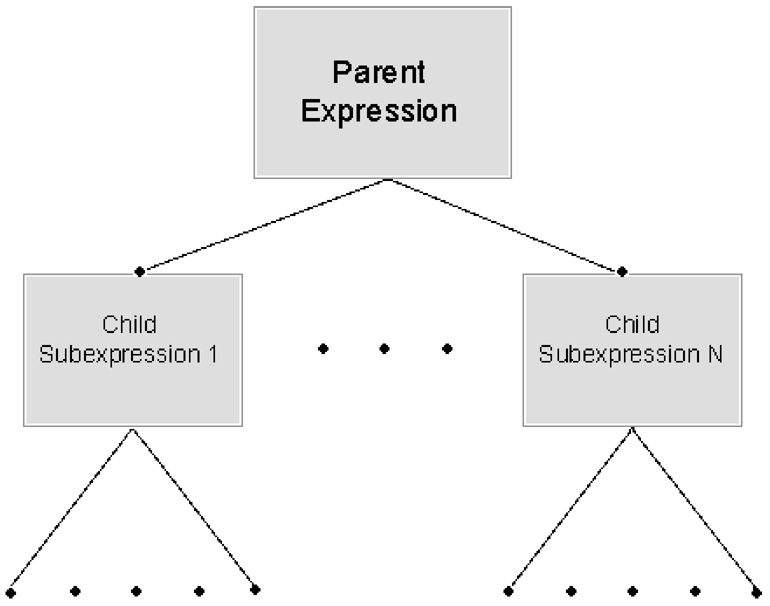
Before you can begin using expressions, you must first learn how to programmatically assemble them using the various operator classes provided in the Expression Package.

An expression is represented as a tree, where each node is another expression itself or a plain Java object. Expression trees are assembled in a bottom-up manner; a child expression or Java object is first created, and then added to a parent expression.

Figure 5-1 illustrates the steps required to build an expression tree.

- The first step in the expression assembly process is to create one or more child operators or Java objects.
- Next, a parent operator is created by supplying the child operators or Java objects to the parent operator's constructor.
- This process of creating subexpressions continues until the entire expression is assembled.

Figure 5-1 Building an Expression Tree



Maintaining Parent-child Relationships

Each of the operator classes defined in the Expression Package extends a common base class that contains the necessary logic for maintaining parent-child relationships; therefore, you do not have to worry about maintaining these relationships while assembling expressions. However, it is possible to modify the structure of an expression after it has been created.

Table 5-5 shows the operators provided in the `Expression` interface for adding, modifying, or removing subexpressions in an expression.

Table 5-5 Methods for Building an Expression Tree

Java Method	Description
`addSubExpression`	Adds a child (can be a subexpression) to an expression object.

Table 5-5 Methods for Building an Expression Tree

Java Method	Description
`removeSubExpression`	Removes an object (can be a subexpression) of the expression object.
`setSubExpression`	Replaces existing object (of an expression) by the given object (can be a subexpression).
`getSubExpression`	Can be used to access the children of an expression object.
`getParent`	Can be used to access the parent expression of an expression object.

See the *Javadoc* API documentation for more information about the `Expression` interface.

Managing the Expression Cache

The expression interface also includes methods to manage the caching of results. The result of evaluating an expression may be cached in each expression object. When the cache is enabled for an expression, trying to evaluate the same expression a second time will return the cached value.

Note: By default, caching is turned off. You may want to keep the cache turned off for some operators, such as `MethodCall`.

Table 5-6 shows the methods provided in the `Expression` interface to manage the caching of results.

Table 5-6 Methods to Manage Caching of Results

Java Method	Description
`setCacheEnabled`	Can be used to enable or disable the cache for an expression.
`isCacheEnabled`	Can be used to check if the cache is enabled for an expression.
`isCached`	Can be used to check if a result is currently cached for an expression.
`getCachedValue`	Can be used to get the current cached result of evaluating the expression.

See the *Javadoc* API documentation for more information about the `Expression` interface.

Expression Package Operators

The following section describes the inheritance hierarchy of the Expression Package operators and contains detailed information on each operator class:

- Operator Inheritance Hierarchy
- Basic Language Operators
- Logical Operators
- String Operators
- Mathematical Operators
- Comparative Operators
- Collection Operators

Operator Inheritance Hierarchy

Before you start using operator classes, it is important to first understand the operator inheritance hierarchy.

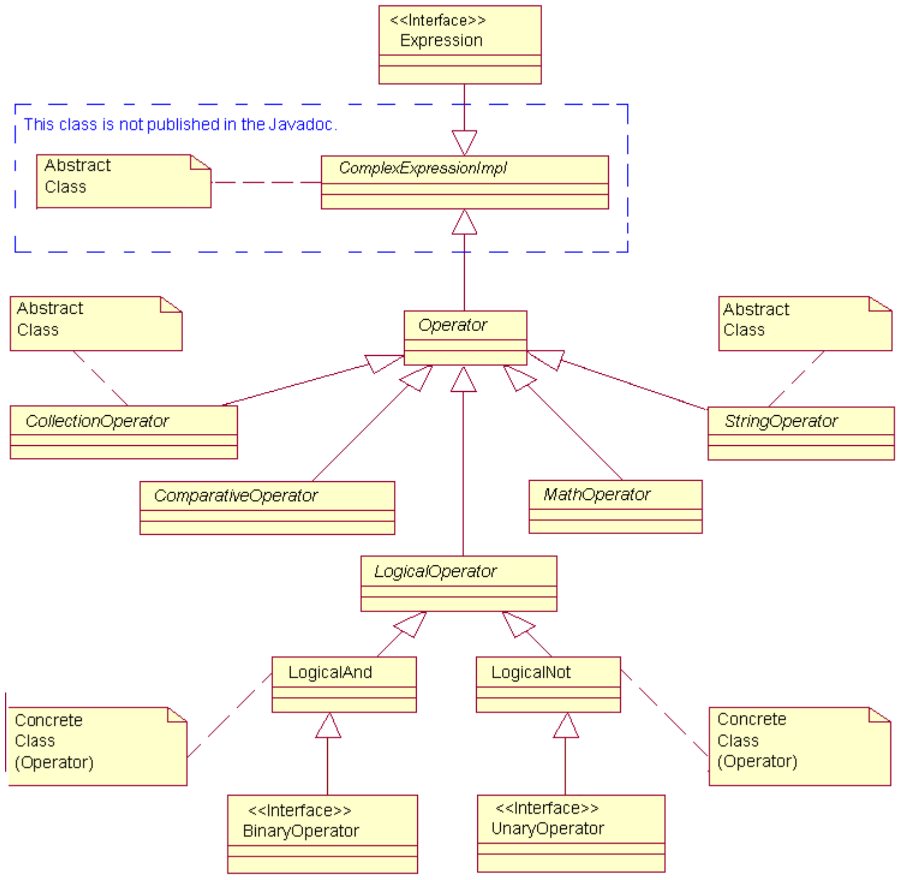
All operator classes must implement the `com.bea.p13n.expression.Expression` interface. As explained previously, the `Expression` interface contains methods for maintaining result caching and parent-child relationships.   Rather than having each operator implement the methods on the `Expression` interface directly, each operator class eventually extends from an abstract base class `(ComplexExpressionImpl)` that contains implementations for all `Expression` methods.

Each operator sub-package contains an abstract base expression class that all operators within the package extend. For example, the `com.bea.p13n.expression.operator.logical` package contains an abstract operator base class `LogicalOperator` from which all logical operators extend.

Finally, in order to differentiate between unary and binary operators, two interfaces, `com.bea.p13n.expression.operator.UnaryOperator` and `BinaryOperator`, are implemented by the corresponding operator types.

The Operator Inheritance Hierarchy is illustrated in Figure 5-2.

Figure 5-2 Operator Inheritance Hierarchy Diagram



Basic Language Operators

The `com.bea.p13n.expression.operator` package contains basic Java language operators for branching, dynamic class loading, object creation and introspection, class and object method calls, and system information retrieval.

An important class in this category is `Variable`. This class is used to represent Java variables, and as such, has an associated name and class type. It can be used to build expressions that contain variables that are bound to values (of an appropriate class) during the evaluation process (via the `Unifier` service).

Table 5-7 shows the complete list of available operators in this category.

Table 5-7 Miscellaneous Operators

Java Class	Description
ClassForName	Dynamically loads the class with the given name.
ClassGetName	Returns the name of a given class.
If	IF operator (`if-then-else` in Java)
Instanceof	Creates a new instance of a given class.
MethodCall	Invokes a method on a given object or class.
ObjectGetClass	Returns the class of a given object.
ObjectHashCode	Returns the hash code of a given object.
ObjectToString	Returns the string representation of a given object.
SystemCurrentTimeMillis	Return the current time in milliseconds according to the system clock.
SystemGetProperty	Returns the value of the system property with a given name.
Variable	Used to represent expression variables. A variable has an associated name and type and is bound to a value during evaluation.

Examples of Basic Language Operators

The if-then-else statement:

```
if (9 > 8) then true else false;
```

can be represented as:

```
new If(new GreaterThan(new Integer(9), new Integer(8)),
Boolean.TRUE, new Boolean.FALSE);
```

Logical Operators

The `com.bea.p13n.expression.operator.logical` package contains all the operators necessary to construct logical (boolean) expressions.

Table 5-8 contains a list of operators in this package.

Table 5-8 Logical Operators

Java Class	Description
LogicalAnd	Binary Logical AND operator (&& in Java)
LogicalOr	Binary Logical OR operator (`||` in Java)
LogicalMultiAnd	Logical AND operator (multiple && in Java)
LogicalMultiOr	Logical OR operator (multiple `||` in Java)
LogicalNot	Logical NOT operator (`!` in Java)

Examples of Logical Expressions

- The boolean condition `(true && false)` can be represented as:

  `new LogicalAnd(Boolean.TRUE, Boolean.FALSE);`

- The boolean condition `(true || false)` can be represented as:

  `new LogicalOr(Boolean.TRUE, Boolean.FALSE);`

- The boolean condition with more than one '&&' operator and more than two operands `(true && true && true && false)`can be represented as:

```
new LogicalMultiAnd(new Object[]{Boolean.TRUE, Boolean.TRUE,
Boolean.TRUE, Boolean.FALSE});
```

String Operators

The `com.bea.p13n.expression.operator.string` package contains operators that perform string operations.

Table 5-9 contains a list of operators in this package.

Table 5-9 String Operators

Java Class	Description
StringCharAt	Returns the character at the specified index within a string.
StringCompareToIgnoreCase	Compares two strings lexicographically, ignoring case considerations.
StringConcat	Concatenates a specified string to the end of another string.
StringContains	Tests if a string contains a given sub-string.
StringEndsWith	Tests if a string ends with a specified suffix.
StringEqualsIgnoreCase	Compares two strings, ignoring case considerations.
StringLength	Returns the length of a string.
StringLike	Tests if a string contains a given sub-string, ignoring case considerations.
StringReplace	Returns a new string resulting from replacing all occurrences of a given character with another given character in a target string.
StringStartsWith	Tests if a string starts with the specified prefix.
StringSubString	Returns a new string that is a substring of a given string.
StringToLowerCase	Converts all of the characters in a string to lower case.

Table 5-9 String Operators (Continued)

Java Class	Description
StringToUpperCase	Converts all of the characters in a string to upper case.
StringTrim	Removes white space from both ends of a string.

Examples of String Operators

The following statement:

```
String string = new String(FooBar”);
string.contains(“Foo”);
```

can be represented as:

```
new StringContains(new String (“FooBar”), “Foo”);
```

Mathematical Operators

The `com.bea.p13n.expression.operator.math` package contains mathematical operators that operate on `java.lang.Number` instances. Analogs to all the operations exposed by the `java.lang.Math` class are provided in this package as well as operators for addition, subtraction, multiplication, and division.

Table 5-10 contains a list of operators in this package.

Table 5-10 Math Operators

Java Class	Description
MathAbs	Operator analogous to the `java.lang.Math.abs()` method.
MathAcos	Operator analogous to the `java.lang.Math.acos()` method.
MathAdd	Mathematical addition.
MathAsin	Operator analogous to the `java.lang.Math.asin()` method.
MathAtan	Operator analogous to the `java.lang.Math.atan()` method.
MathAtan2	Operator analogous to the `java.lang.Math.atan2()` method.

Table 5-10 Math Operators (Continued)

Java Class	Description
MathCeil	Operator analogous to the `java.lang.Math.ceil()` method.
MathCos	Operator analogous to the `java.lang.Math.cos()` method.
MathDivide	Mathematical division.
MathExp	Operator analogous to the `java.lang.Math.exp()` method.
MathFloor	Operator analogous to the `java.lang.Math.floor()` method.
MathIeeeRemainder	Operator analogous to the `java.lang.Math.IEEEremainder()` method.
MathLog	Operator analogous to the `java.lang.Math.log()` method.
MathMax	Operator analogous to the `java.lang.Math.max()` method.
MathMin	Operator analogous to the `java.lang.Math.min()` method.
MathMultiply	Mathematical multiplication.
MathPow	Operator analogous to the `java.lang.Math.pow()` method.
MathRandom	Operator analogous to the `java.lang.Math.random()` method.
MathRint	Operator analogous to the `java.lang.Math.rint()` method.
MathRound	Operator analogous to the `java.lang.Math.round()` method.
MathSin	Operator analogous to the `java.lang.Math.sin()` method.
MathSqrt	Operator analogous to the `java.lang.Math.sqrt()` method.
MathSubtract	Mathematical subtraction.
MathTan	Operator analogous to the `java.lang.Math.tan()` method.
MathToDegrees	Operator analogous to the `java.lang.Math.toDegrees()` method.
MathToRadians	Operator analogous to the `java.lang.Math.toRadians()` method.

Examples of Mathematical Operators

The arithmetic statement:

```
int a = 1;
int b = 2;
int c = 3;
a + (b - c);
```

can be represented as:

```
Integer a = new Integer(1);
Integer b = new Integer(2);
Integer c = new Integer(3);
new MathAdd(a, new MathSubtract(b, c));
```

Comparative Operators

The `com.bea.p13n.expression.operator.comparative` package contains operators for performing comparative operations.

Table 5-11 contains a list of operators in this package.

Table 5-11 Comparative Operators

Java Class	Description
Equals	Tests two objects for equality.
GreaterOrEquals	Tests if one object is greater or equal to another object.
GreaterThan	Tests if one object is greater than another object.
LessOrEquals	Tests if one object is less than or equal to another object.
LessThan	Tests if one object is less than another object.
NotEquals	Tests two objects for inequality.

Example of Comparative Operators

The comparative statement:

```
(9 < 10)
```

can be represented as:

```
new LessThan(new Integer(9), new Integer(10));
```

Collection Operators

The `com.bea.p13n.expression.operator.collection` package contains operators that operate on `java.util.Collection` instances.

Table 5-12 contains a list of operators in this package.

Table 5-12 Collection Operators

Java Class	Description
CollectionContains	Returns true if a given collection contains a specified element.
CollectionContainsAll	Returns true if a given collection contains all of the elements in another specified collection.

Example of Collection Operators

The following statement:

```
Collection container = new LinkedList();
Object element = new Object();
container.contains(element);
```

can be represented as:

```
new CollectionContains(new LinkedList(), new Object());
```

Working with Expressions

After you have assembled an expression, you are ready to work with it using the various Expression Package services. These services allow you to prepare an assembled expression for evaluation, validate that the expression is well-formed, optimize its structure, and finally, evaluate the expression.

The following information is presented in this section:

- The Expression Factory
- Expression Package Services
  - Unification Service
  - Optimization Service
  - Validation Service
  - Evaluation Service
  - Execution Service

The Expression Factory

The ExpressionFactory provides methods to create the various Expression Package services and data structures used by these services.

For example, the following method will create an instance of the `Validator` service:

```
ExpressionFactory.createValidator(null);
```

For more detail on how to construct the various Expression Package services, see the *Javadoc* API documentation.

Expression Package Services

The Expression Package offers services which can be used on any expression that is built using the operators in the Expression Package.

Unification Service

The `Unifier` is used to unify variables (assign values to variables) present in an expression. The `Unifier` uses a data structure known as a `UnificationList` that stores the variable name and the corresponding value of the variable. Like the `Unifier`, the `UnificationList` instances are created via the `ExpressionFactory`. The `Unifier` gets the value from the list for a particular variable using the variable name as a key to search the `UnificationList`, and binds the retrieved value to the variable.

See the *Javadoc* API documentation for more information about the `Unifier` interface and the `ExpressionFactory` class.

Optimization Service

The `Optimizer` is used to optimize an expression. The default optimization algorithm used by the `Optimizer` is shown below.

- Traverse an expression tree and add each unique subexpression to a list.
- If a subexpression is equal to an expression present in the list, then replace it with a proxy expression. The proxy expression delegates to the original expression.

See the *Javadoc* API documentation for more information about the `Optimizer` interface and the `ExpressionFactory` class.

Validation Service

The `Validator` is used to validate an expression. The default validation algorithm used by the `Validator` is as follows:

For each operand of an operator:

- Get the required type of the operand.
- If the operand is an expression, evaluate the expression and compare the type of the result with the required type; otherwise, assert that the operand is of the required type.
- If the type does not match or an error occurs during the evaluation of an operand expression, the `Validator` throws an `InvalidExpressionException` . An

`UnboundVariableException` is thrown if any variables in an expression are not bound to a value.

The `Validator` can be used in a stateless or stateful mode. In stateless mode, any expression evaluations necessary to perform validation will be executed in stateless mode.

For more information about stateless and stateful evaluation modes, see the "Evaluation Service" section below.

See the *Javadoc* API documentation for more information about the `Validator` interface and the `ExpressionFactory` class.

Evaluation Service

The `Evaluator` is used to evaluate an expression. An expression can be evaluated in stateful or stateless mode:

Stateful mode

In this mode, the value of each variable that appears in the expression is determined by retrieving the value set within the variable.

In other words, stateful mode relies upon the expression having been previously unified by a `Unifier`.

When an expression is evaluated in stateful mode and results caching is turned on, the results of evaluation will be cached within the expression.

Stateless mode

In this mode, the value of each variable that appears in the expression is determined by looking up a value that is bound to the name of the variable in an external data structure.

In other words, the evaluation process does not rely upon state associated with the expression, and as such, does not require the expression to be unified before evaluation.

The data structure that contains the name-value mappings for variables is known as a `UnificationList` and is associated with the `Evaluator`. Like the `Evaluator`, the `UnificationList` instances can be created using the `ExpressionFactory`.

A side effect of stateless mode is that expression evaluation cannot take advantage of results caching.

You can use of stateful mode in a situation where an expression need only be evaluated within a single thread of execution. In the case of multithreaded evaluation of a single expression, you *must* use stateless mode.

Note: If an expression does *not* contain variables, then there is no difference between the two evaluation modes.

See the *Javadoc* API documentation for more information about the `Evaluator` interface and the `ExpressionFactory` class.

Execution Service

The `Executor` aggregates the Unification Service, Validation Service and Evaluation Service. The `execute` method on an `Executor` takes a `Unifier`, a `Validator` and an `Evaluator` to execute a cycle of unification-validation-evaluation operations.

The algorithm used by the `Executor` is shown below:

Unification

- If the `Unifier` is not null, unify the expression.
- If the `Unifier` is null, do not unify the expression.

Note: The `Unifier` should be null in the case where the expression passed to the `Executor` is already unified, or the expression is to be evaluated in stateless mode.

Validation

- If the `Validator` is not null, validate the given expression.
- If the `Validator` is null, ignore validation.

Evaluation

- If the `Evaluator` is not null, evaluate the expression in stateful or stateless mode. (depending on the type of evaluator passed.)

- If the `Evaluator` is null, the `Executor` throws an `IllegalArgumentException`.
- Return the result.

Note: If the `Evaluator` passed is stateless, then the `Unifier` should be null.

See the *Javadoc* API documentation for more information about the `Executor` interface and the `ExpressionFactory` class.

Code Examples

This section contains examples that illustrate how to construct expressions programmatically and use the Expression Package services.

This section contains the following four code examples:

- Stateful Evaluation of a Simple Expression
- Stateful Evaluation of an Expression Containing Variables
- Stateless Validation and Evaluation of an Expression Containing Variables
- Stateful Validation and Evaluation of an Expression Containing Variables

Stateful Evaluation of a Simple Expression

A logical expression is constructed and executed in stateful mode. The expression does not contain any variables.

Example

The source code for creating and executing the expression is shown below:

```
Expression expression = new LogicalAnd(Boolean.TRUE,
Boolean.FALSE);
```

```
// Prepare for creating an executor by creating a stateful
// evaluator. Since the expression does not contain variables,
// we are not using a validator or a unifier in this example,
// so we will not create them.

// null is passed for the environment Map.

Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// null is passed for the environment Map.

Executor executor = ExpressionFactory.createExecutor(null);

// Execute the above expression by passing null for both the unifier
// and validator parameters.

Object result = executor.execute(expression, null, null,
evaluator);

// The result should be Boolean.FALSE.
```

Stateful Evaluation of an Expression Containing Variables

An expression containing variables is constructed and evaluated in stateful mode.

Example

The source code for creating and executing the expression in stateful mode is shown below.

```
// Create a variable that can store an object of type Boolean
// and whose name is “?booleanVariable”.

Variable booleanVariable = new Variable(“?booleanVariable”,
Boolean.class);

// Now, we will use the variable that we created in the above step.

Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);

// Next, we’ll unify the expression by binding any variables
// present in the expression. In the above case, there is one
// variable in the expression so the variable needs to be assigned a
// value. This is shown below.

// Create a UnificationList to store the variable name and value as
// key-value pairs.

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject(“?booleanVariable”, Boolean.FALSE);

// Create a unifier.

Unifier unifier = ExpressionFactory.createUnifier(null,
unificationList);

// Prepare for creating an executor by creating a stateful
// evaluator. We are not using a validator in this example,
// so we will not create one.

// null is passed for the environment Map.

Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// null is passed for environment Map.

Executor executor = ExpressionFactory.createExecutor(null);

// Execute the above expression by passing a unifier and a null
// validator.

Object result = executor.execute(expression, unifier, null,
evaluator);

// The result should be Boolean.FALSE.
```

Note: The expression can be unified before calling the `execute` method by calling the `unify` method on the `Unifier`. Once the expression is unified there is no need to pass a unifier to the `execute` method of the executor.

Stateless Validation and Evaluation of an Expression Containing Variables

An expression containing variables is constructed and evaluated in stateless mode. The `Validator` service is also used to validate the expression.

Example

The source code for creating and executing the expression in stateless mode is shown below.

```
// Create a variable that can store an object of type Boolean
// and whose name is “?booleanVariable”.

Variable booleanVariable = new Variable(“?booleanVariable”,
Boolean.class);

// Now we will use the variable that we created in the above step.

Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);

// Next, we’ll unify the expression by binding any variables
// present in the expression. In the above case there is one
// variable in the expression, so the variable needs to be assigned
// a value. This is shown below.

// Create a UnificationList to store the variable name and value as
// key-value pairs.

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject(“?booleanVariable”, Boolean.FALSE);

// Prepare for creating an executor by creating a stateless
// evaluator. We are not using a unifier in this example,
// so we will not create one.

// Creating a stateless evaluator by passing null for the
// environment Map and the UnificationList.

Evaluator evaluator = ExpressionFactory.createEvaluator(null,
unificationList);

// Creating a stateless validator.

Validator validator = ExpressionFactory.createValidator(null,
evaluator);

// Creating an executor.

Executor executor = ExpressionFactory.createExecutor(null);

// Execute the above expression by passing null for the unifier and
// a non-null validator.
```

```
Object result = executor.execute(expression, null, validator,
evaluator)

// The result should be Boolean.FALSE.

// After calling execute method, the given expression will not be
// modified by any services that were used above.

// The stateless execution mode is useful if an expression is shared
// between multiple threads.
```

Stateful Validation and Evaluation of an Expression Containing Variables

An expression containing variables is constructed and evaluated in stateful mode. The `Validator` service is also used to validate the expression.

Example

The source code for creating and executing the expression in a stateful mode is shown below.

```
// Create a variable that can store an object of type Boolean
// and whose name is “?booleanVariable”.

Variable booleanVariable = new Variable(“?booleanVariable”,
Boolean.class);

// Now we will use the variable that we created in the above step.

Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);

// Next, we’ll unify the expression by binding any variables
// present in the expression. In the above case, there is one
// variable in the expression, so the variable needs to be assigned
// a value. This is shown below.

// Create a UnificationList to store the variable name and value
// as key-value pairs.

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject(“?booleanVariable”, Boolean.FALSE);
```

```
// Create a unifier.

Unifier unifier = ExpressionFactory.createUnifier(null,
unificationList);

// Prepare for creating an executor by creating a stateful
// evaluator and validator.

// null is passed for the environment Map.

Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// null is passed for the environment Map.

// Creating a validator.

Validator validator = ExpressionFactory.createValidator(null);

// Creating an executor.

Executor executor = ExpressionFactory.createExecutor(null);

// Execute the above expression by passing a unifier and a non-null
// validator.

Object result = executor.execute(expression, unifier, validator,
evaluator);

// The result should be Boolean.FALSE.
```

Note: The expression can be unified before calling the `execute` method by calling the `unify` method on the `Unifier`. Once the expression is unified there is no need to pass a unifier to the execute method of the `Executor`. The validation service can be used directly by calling the `validate` method. The `validate` method throws an `InvalidExpressionException` if the given expression is invalid.

Expression Package Configuration Settings

The `expression.properties` file contains configuration settings for the Expression Package and should be modified with care.

This file is archived in `p13n_util.jar` under the package `com.bea.p13n.expression`.

```
##
# Expression Comparator null handling
#
# If the following property is set to true the Expression
# Comparator will return false as the result of comparing
# any non-null value to a null, regardless of the
# comparison being performed.
#
# Defaults to true.
##

expression.comparator.nullcheck=true

##

# Expression Comparator equality epsilon.
#
# The following property determines the epsilon value for
# numeric equality comparisons.
#

# Defaults to 0.
##

expression.comparator.epsilon=0.00001

##
# Expression Introspector Method Array Caching
#
# If the following property is set to true the Expression
# Introspector will cache the array of Methods implemented by a
# Java Class.
#

# Defaults to true.
##

expression.introspector.method.array.cache=true
```

```
##
# Expression Introspector Method Caching
#
# If the following property is set to true the Expression
# Introspector will cache Methods by signature.
#
# Defaults to true.
##

expression.introspector.method.cache=true

##
# Expression Parser Node Support Classes
#
# This property supports a comma-delimited list of classes
# extending the base AST NodeSupport class. Such classes
# provide node creation support for expression-schema namespaces
# required for constructing the intermediate AST representing a
# given Expression instance.
#
# All NodeSupport subclasses must co-exist peacefully with the
# required CoreNodeSupport instance.
##

parser.node.support.list=\

com.bea.p13n.expression.internal.parser.expression.ExpressionNode
Support

##
# Expression Parser Transform Visitor Class
#
# This property specifies the ExpressionTranformVisitor or
# subclass to be used for intermediate AST-to-Expression
# transformations.
#
##

parser.transform=\

com.bea.p13n.expression.internal.parser.expression.ExpressionTran
sformVisitor
```

CHAPTER

6 Foundation Classes and Utilities

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

This topic includes the following sections:

- Webflow
- HTTP Handling
- Personalization Request Object
  - Default Request Property Set
- Personalization Session Object
  - Default Session Property Sett
- Utilities
  - ContentHelper
  - TagSupportHelper
  - ProfileFactory
  - SessionHelper

Webflow

The Webflow mechanism externalizes a site's page flow and separates back-end processing activities from presentation. Both the flow of pages and the underlying business processing are configured in centralized XML files, making it easier than ever to maintain or modify the behavior of your Web site. Out-of-the-box, the Webflow mechanism comes with a number of components to get you started, but the Webflow can also be customized or extended to meet your specific objectives. For more information about the Webflow implementation in the WebLogic Portal and WebLogic Personalization Server, see the *Guide to Managing Presentation and Business Logic: Using Webflow and Pipeline* documentation.

HTTP Handling

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

The tags utilize helper classes that transform an `HttpRequest` and `Session` into serializable personalization surrogates for their HTTP counterparts. These surrogates are compatible with the Rules Engine 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.bea.p13n.http.Request`) before they can be set on the appropriate `AdviceRequest.` (See the *Javadoc* API documentation.) While the `HttpRequest` object can be wrapped by directly calling the Personalization `Request` constructor, it is recommend that developers use the `createP13NRequest` helper method on `Request` for this purpose. See the *Javadoc* API documentation for more information.

Caution: The tag implementations for the `<pz:div>` and `<pz:contentSelector>` tags create the Personalization `Request` surrogate for the `HttpRequest` before calling the Advisor bean, so JSP developers need not worry about the details of the `Request` object. Only developers accessing the `Advisor` 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 `propertySet` parameter to find a property set for a `Request` property set name and, if the property set is found, uses the default value in the schema.
6. If not found, `getProperty` uses the default value passed into the method call.

Default Request Property Set

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

The default `Request` properties include the following:

Request Property Name	Associated Request Method
Request Method	request.getMethod()
Request URI	request.getRequestURI()
Request Protocol	request.getProtocol()
Servlet Path	request.getServletPath()
Path Info	request.getPathInfo()
Path Translated	request.getPathTranslated()
Locale	request.getLocale()
Query String	request.getQueryString()
Content Length	request.getContentLength()
Content Type	request.getContentType()
Server Name	request.getServerName()
Server Port	request.getServerPort()
Remote User	request.getRemoteUser()
Remote Address	request.getRemoteAddr()
Remote Host	request.getRemoteHost()
Scheme	request.getAuthType()
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.bea.p13n.http.Session`) before they can be set on the appropriate `AdviceRequest.` (See the *Javadoc* API documentation). While the `HttpSession` object can be wrapped by directly calling the Personalization `Session` constructor, using the `createP13NSession` helper method on `Session` is recommended. See the *Javadoc* API documentation for more information.

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

Default Session Property Set

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

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 property set for the Personalization Session for the `propertySet` method parameter.
3. If it still does not find the property, it uses the `propertySet` parameter to find a property set for the `Session` property set name and, if the property set is found, uses the default value in the property set.

4. If it still does not find the property, it uses the default value passed into the `getProperty` method call.

Utilities

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

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.

`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.

TagSupportHelper

`TagSupportHelper(com.bea.p13n.servlets.jsp)` provides utility methods for working with the Advisor. It includes methods for creating `AdviceRequests` and for retrieving instances of the Advisor Service.

ProfileFactory

The `ProfileFactory (com.bea.p13n.usermgmt.profile.ProfileFactory)` utility is used to retrieve user and group profile information. The method `getProfile(String username, String groupname)` will return a `ProfileWrapper` object, which can then be queried for profile properties. If `getProfile` is called with only a username, a `ProfileWrapper` representing that user's profile will be returned. If it is called with only a groupname, a `ProfileWrapper` representing that group's profile will be returned. If it is called with both a user and group name, the returned ProfileWrapper will represent the user's profile, with the group's profile acting as an explicit successor. Each call to `getProperty` on the `ProfileWrapper` will use the group as an explicit successor unless a different successor is provided.

SessionHelper

`SessionHelper(com.bea.p13n.usermgmt.SessionHelper)` is used as a single access point to store and retrieve profiles in an HTTP `Request` or `Session`. It provides methods to put a `ProfileWrapper` in either a request or session using a well-known variable name, and a method to retrieve a `ProfileWrapper` from a request or session.

When retrieving the `ProfileWrapper` it will first look in the request, and then in the session. This utility can be used by different parts of an application to make sure that each component is dealing with the same profile. For example, a login component can use it to retrieve a profile and put it in the session; and later on a pipeline component can use it to access that same profile.

CHAPTER

7 Creating and Managing Property Sets

Property sets are the schemas for personalization attributes. Using the E-Business Control Center, you can create property sets and define the properties that make up these property sets.

This topic includes the following sections:

- Overview of Property Sets
  - Property Sets Serve as Namespaces for Properties
  - Where Property Sets Are Used
  - Property Definition Attributes
- Using the E-Business Control Center
  - Starting the Property Set Editors
  - Using the Property Set Editors

Overview of Property Sets

A property set is a convenient way to give a name to a group of properties for a specific purpose. For example, in the sample application you will find a User Profile property set named Customer Properties. This property set defines around thirty properties for an e-commerce customer, such as First Name, Last Name, Home Phone, Email, and Customer Type.

Property sets and property definitions are created in the E-Business Control Center, on the Site Infrastructure tab. A Property Editor allows you to give a new a property a name and a description, assign a data type, a selection mode, and a value range, and create a list of possible values for the property.

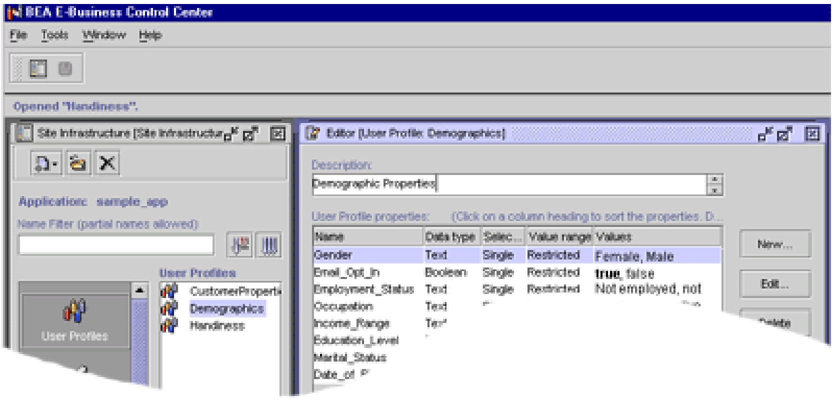
Although properties are designed in the EBCC, the value assigned to a property is created in the application. Properties are generally represented in an application as fields on a page. To use the Customer Properties property set to collect information about an e-customer, the application will typically present the thirty properties in a list or a table, with text boxes provided to fill in specific values for the customer.

Property Sets Serve as Namespaces for Properties

In the most general sense, a property can be considered a name/value pair. Property sets serve as namespaces for properties so that properties can be conveniently grouped and so that multiple properties with the same name can be defined.

For example, you might create a property set called Demographics to describe user profile properties. The Demographics property set contains properties called Age, Gender, Income, and so forth. Because property sets create unique namespaces for properties, another property set can also have a property called Gender, and the two values will be kept separate.

Figure 7-1 The Demographics Property Set uses the Gender attribute

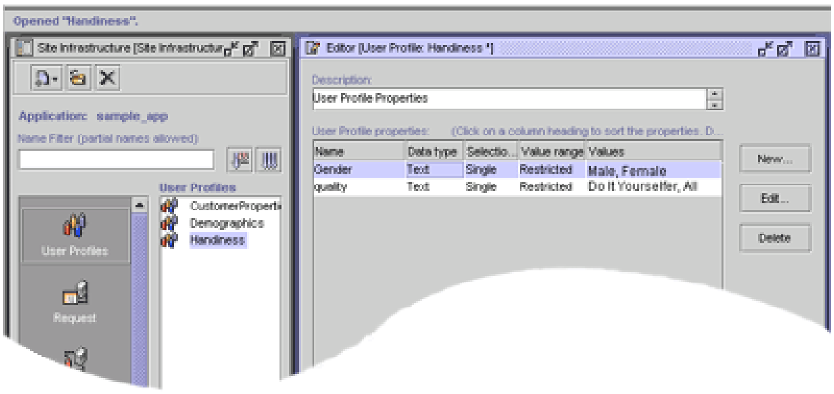


Description:
Demographic Properties

User Profile properties: (Click on a column heading to sort the properties. D...

Name	Data type	Selec...	Value range	Values
Gender	Text	Single	Restricted	Female, Male
Email_Opt_In	Boolean	Single	Restricted	true, false
Employment_Status	Text	Single	Restricted	Not employed, not
Occupation	Text			
Income_Range	Te			
Education_Level				
Marital_Status				
Date_of				

Figure 7-2 The Handiness Property Set also uses the Gender attribute



Description:
User Profile Properties

User Profile properties: (Click on a column heading to sort the properties. D...

Name	Data type	Selectio...	Value range	Values
Gender	Text	Single	Restricted	Male, Female
quality	Text	Single	Restricted	Do It Yourselfer, All

Where Property Sets Are Used

For Portal and Personalization Server purposes, property sets are applied to six major areas. Of these, five are configured in the E-Business Control Center.

Table 7-13 The Property Set Buttons on the Site Infrastructure Tab

User Profiles	**User Profiles** The User Profiles property set type is used for defining the property sets and properties that apply to user and group profiles. For example, a property set of this type might be created called “CustomerProperties.” Subsequent property retrieval for a particular user or group can then be scoped with this property set name to retrieve the user’s email address. For an in-depth discussion of how property retrieval works for users and groups, see “Creating and Managing Users” in the *Guide to Building Personalized Applications*.
Request	**HTTP Requests** An HTTP Request is the information that your browser sends to the server; the server sends back an HTTP response. The Request property set type is used for defining the property sets and properties that apply to HTTP requests. A Request property set type might be called “myApplicationRequest.” Properties available through this property set can then be accessed via the Advisor.
Session	**HTTP Sessions** A session contains short-lived, stateful information for the time that a browser is interacting with a server. The Session property set type is used for defining the property sets and properties that apply to HTTP sessions. A “Session” property set type might be called “myApplicationSession.” Properties available through this property set can then be accessed via the Advisor.
Catalog Structure	**Catalog Structure** The Catalog Structure property set type is used to define custom attributes for product items and product categories in the Commerce services catalog. These custom attributes can be used in addition to the default attributes provided by Commerce services in the catalog database tables. For more information, see the topic “Catalog Administration Tasks” in the *Guide to Building a Product Catalog*.

Table 7-13 The Property Set Buttons on the Site Infrastructure Tab

	Events The Events property set type is used to register a custom event. For the purpose of registering an event, you can consider an event property as a name-value pair. During the registration of a custom event, you specify the event's name, description, and one or more properties. Each property has a range, type of permissible value, and default value. The information you need to register an event should be available from your Business Engineer (BE) or Java developer.
Use the tools associated with your document management system.	**Content Management** The Content Management property set type is used for defining the configuration and run-time use of the content management system. Content Management property sets cannot be created or manipulated with the Personalization Server Administration Tools or the E-Business Control Center. Instead, use the tools associated with your document management system. For more complete information on this subject, see "Creating and Managing Content" in the *Guide to Building Personalized Applications*.

Property Definition Attributes

All properties includes the following information:

- **Property name—**the name of the property, such as `Gender`. Within a single property set, all properties must have a unique name. However, property names can be re-used in different property sets.
- **Data type—**specifies the data type of the property value. The possible values are Text, Numeric, Floating-Point number (equivalent to Double in Java), Boolean, and Date/Time.
- **Selection mode—**specifies whether a property is single-valued (has a single default value) or multi-valued (has a collection of default values).
- **Value range—**specifies whether the defaults are restricted to one specific value, one or more specific values, or any value.

Optionally, property definitions can also include the following:

- **Description—**a textual description of the property, perhaps describing the purpose of the property.

- **Values—**you can assign a list of values from which the user will pick, and you can designate which of the values is the default.

The following table lists the property definition attributes and values.

Table 7-14 Property Definition Attributes and Values

Property Definition Attribute	Attribute Value
Name	Text (100 character length maximum)
Description	Text (255 character length maximum)
Type	■ Text ■ Numeric – (equivalent to Long in Java) ■ Float – Floating-Point Number (equivalent to Double in Java) ■ Boolean ■ Date/Time (equivalent to `java.sql.Timestamp`)
Selection Mode	Single-valued or multi-valued
Value Range	Restricted or unrestricted
Default Value	Up to the user — can be `null`
Restricted Values	Allowable values if the property is restricted

Possible Combinations of Properties

As the previous list suggests, a combination of property values are possible. The possible combinations of properties are listed here:

- **Boolean**: The values for this type of property are either True or False. You can choose the default. The default value is displayed only in the Enter Property Values Window, not in the Edit Event Property window. When this data type is selected, the Selection mode and Value range are unavailable.
- **Single, Unrestricted**: This type of property has only one value, which is also the default value.

- **Single, Restricted**: This type of property has multiple available values and a single default value. You can select which value is the default.

  An instance of this property can have only one value assigned. For example, the property Favorite_Day_of_the_Week would have seven available values, but an instance of the property can have only one value (Saturday).

- **Multiple, Restricted**: This type of property has multiple available values. You can select one or more values as default values.

  An instance of this property can have multiple values assigned. For example, the property Favorite_Days_of_the_Week would have seven available values, and an instance of the property can have any number of values (Friday, Saturday, Sunday).

- **Multiple, Unrestricted**: This type of property has multiple values. You cannot select any defaults; all values are defaults.

Synchronizing Property Sets

Property sets are meant to be used as application code. Just as you would not normally deploy new code onto a production server without first testing and staging it, we recommend that you do not synchronize updates to the property sets onto a live server once the application itself is deployed into production.

Why would synchronizing to a live server be a problem? If you create a property set containing a property of a specific type, synchronize it and assign values to that property for users, then change the type and re-synchronize it, there will be inconsistent property value data in the database.

Let's look at an example. Suppose the "Date_of_Birth" property in the Demographics property set was originally a String. Users might have put in values in many different formats, such as "8/9/1974", "August 9th, 1974", "Aug 9 1974", and so forth. Then you decide these are too hard to parse and validate, so you make the "Date_of_Birth" property a Date/Time type. You redeploy the property set, but now there are values of the wrong type in the database.

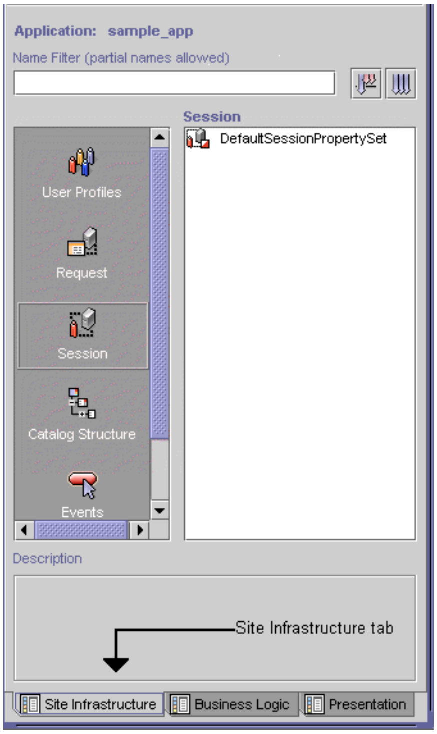
It's up to you, the developer, to do the data conversion to make these properties into the correct type. Or, if your data is just test or sample data, you can delete the data altogether and start over.

Using the E-Business Control Center

The E-Business Control Center tools allow you to create and manage sets of typed properties. Using the Site Infrastructure tab, property sets can be defined to describe properties for User Profiles, Requests, Sessions, Catalog Structures, and Events.

Note: The Site Infrastructure tab also contains the Webflows/Pipelines tool icon, which is used to create and edit new pipelines and webflows. Since webflows and pipelines do not use property sets, these topics will not be covered here. See the topic "Using the Webflow and Pipeline Editors" in the *Guide to Using the E-Business Control Center* documentation.

Figure 7-3 The Site Infrastructure Tab on the E-Business Control Center



Creating a property set is a simple task via the E-Business Control Center. A name for the set must be provided as well as description, and the type of property set (“User/Group”, “Session”, or “Request”) must be chosen.

Once a User/Group property set is created and deployed, property values can be edited for a particular user or group via the User Management user and group tools. For “Session” and “Request” properties, the only editable values are the default values set in the property definitions —run-time values are determined by values in the HTTP session or HTTP request, respectively.

Starting the Property Set Editors

To begin using the Editors, follow these steps:

1. Start the E-Business Control Center (EBCC). For detailed instructions on starting the EBCC, see the topic “Starting the E-Business Control Center” in the *Guide to Using the E-Business Control Center* documentation.
2. Create a new Web application or open an existing Web application for which you will be creating or editing a property set.

   For detailed instructions on performing these tasks, see “Creating an Application Structure for E-Business Control Center Data” or “Opening Application Data” in the *Guide to Using the E-Business Control Center* documentation.
3. Select the Site Infrastructure tab in the EBCC’s Explorer window, then click any of the tool icons in the Explorer window (except the Webflows/Pipelines icon.)

Using the Property Set Editors

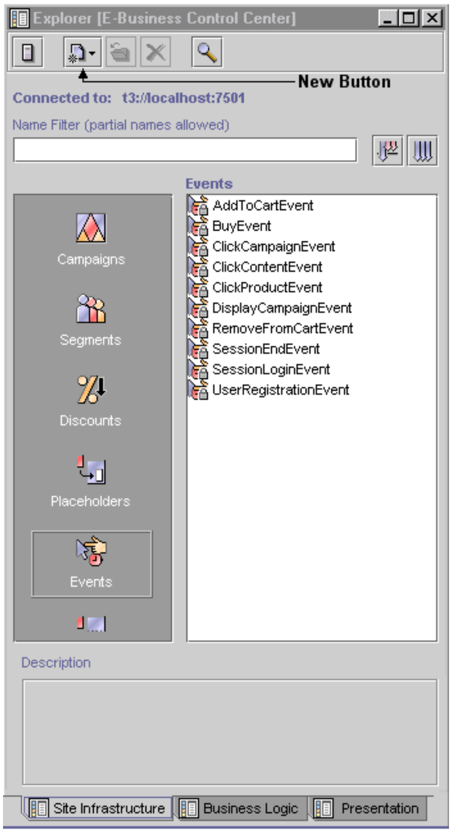
The property set editors works the same way for all property sets. For these examples, we will be using the E-Business Control Center to create and modify Event properties. The examples used here can be used to register a custom event. You can follow the same procedures to create and modify property sets for Users and Groups, HTTP Requests, HTTP Sessions, and the Catalog Structure.

To register a custom event, complete the following steps

1. Start the E-Business Control Center and connect it to a server. The Explorer window opens as shown in Figure 7-4.

   Note: For more information on connecting the E-Business Control Center to a server, see the topic “Connecting to the Server” in the *Guide to Using the E-Business Control Center*.

Figure 7-4 E-Business Control Center Window



AddToCartEvent
BuyEvent
ClickCampaignEvent
ClickContentEvent
ClickProductEvent
DisplayCampaignEvent
RemoveFromCartEvent
SessionEndEvent
SessionLoginEvent
UserRegistrationEvent

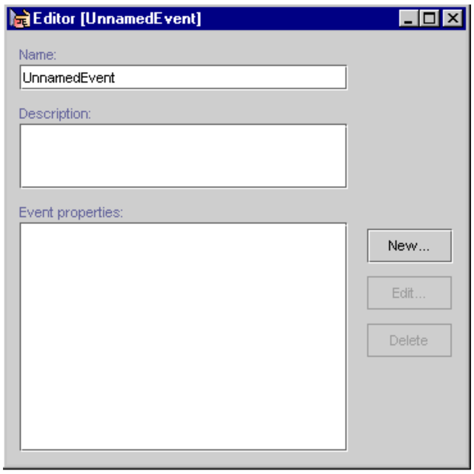
2. Open the Event Editor as follows:

   Note: You cannot edit the standard events.

   a. In the Explorer window, select the Event icon. A list of events appears in the Events field.

   b. Click the New Button, and then select Event. The Event Editor window appears as shown in Figure 7-5.

Figure 7-5 Event Editor Window



3. In the Edit Event Editor window, complete these steps:

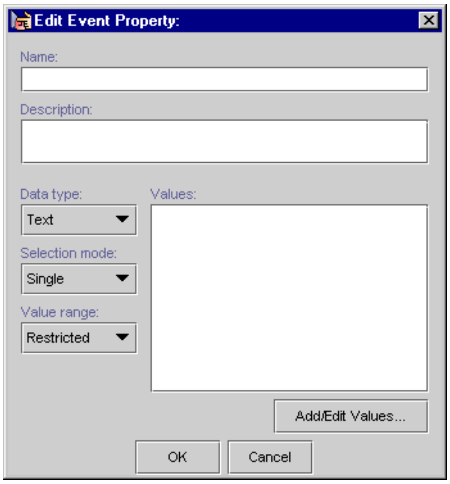
   a. In the Name field, enter a unique name for the event no longer than 100 characters (required).

   b. In the Description field, enter a description for the event no longer than 254 characters (required).

   c. Click the Save button in the E-Business Control Center system toolbar.

   d. To create properties for the event, click the New button. The Edit Event Property window opens, as shown in Figure 7-6.

Figure 7-6 Edit Event Property Window



4. In the Edit Event Property window, complete these steps:

   a. In the Name field, enter a unique name for the property no longer than 100 characters (required).

   b. In the Description field, enter a description of the property no longer than 254 characters (required).

   c. In the Data type list, select the data type. If you select Boolean as the data type, the Selection mode and Value range are no longer available. The default for Boolean is Single, Restricted.

   d. In the Selection mode list, select either Single or Multiple.

   e. In the Value range list, select whether the value is Restricted or Unrestricted.

   f. Click the Add/Edit Values button.

The type of window that appears depends on the values selected.

Property Values and Setting the Default Value

Depending on the data type, different steps are required for entering values and setting default values. The following property categories are available:

- Properties with Boolean or a Single Value and Single Default.

- Properties with Multiple Values and Single, Multiple, or All Defaults
- Properties with Date and Time Values

Properties with Boolean or a Single Value and Single Default

To enter the default value for Boolean property or a property with a single value and a single default (unrestricted), complete the following steps:

1. In the applicable Enter Property Value window (Figure 7-7 or Figure 7-8), perform one of the following:
   - For a Boolean property, select either True or False.
   - For a Single Value, Single Default property, enter a value.

Figure 7-7 Enter Property Values Window—Boolean

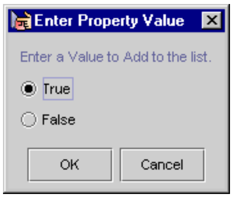
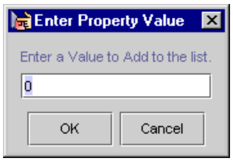


Figure 7-8 Enter Property Values Window—Single Value, Single Default



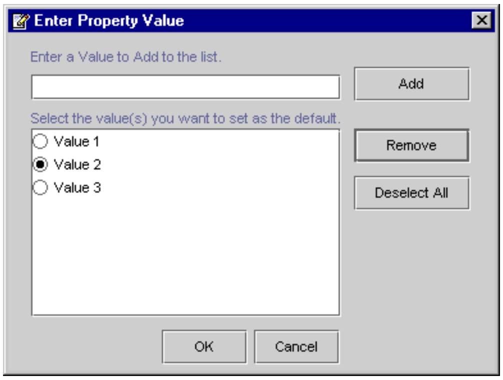
2. Click the OK button.
3. In the Edit Event Property window, click the OK button.

Properties with Multiple Values and Single, Multiple, or All Defaults

To enter multiple property values and set one or more defaults (unrestricted), complete the following steps:

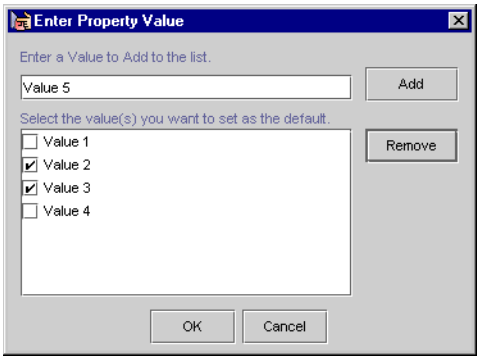
1. In the applicable Enter Property Values window (Figure 7-9 or Figure 7-10), enter a value, and then click the Add button.

Figure 7-9 Enter Property Values—Multiple Values, Single Default


Enter Property Value
Enter a Value to Add to the list.
Add
Select the value(s) you want to set as the default.
Remove
Deselect All
OK
Cancel

- ○ Value 1
- ● Value 2
- ○ Value 3

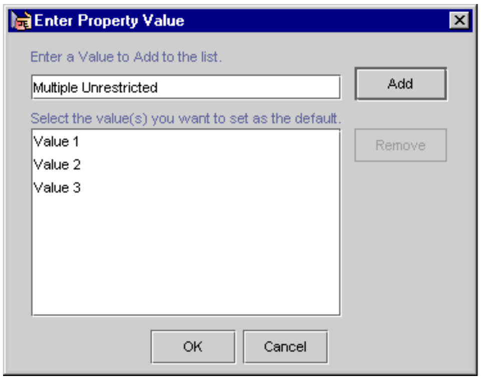
Figure 7-10 Enter Property Values—Multiple Values, Multiple Restricted Defaults


Enter Property Value
Enter a Value to Add to the list.
Add
Select the value(s) you want to set as the default.
Remove
OK
Cancel

Value 5

- [ ] Value 1
- [✓] Value 2
- [✓] Value 3
- [ ] Value 4

Figure 7-11 Enter Property Values—Multiple Values, Multiple Unrestricted Defaults


Enter Property Value
Enter a Value to Add to the list.
Multiple Unrestricted
Add
Select the value(s) you want to set as the default.
Value 1
Value 2
Value 3
Remove
OK
Cancel

2. Repeat the previous step until you have entered all values.
3. To select one or more default values, complete one of the following:
   - If you do not want to select a default, go to next numbered step.
   - For multiple values with a single default, select the value (radio button) that you want to set as the default, and then click the OK button.

   Note: To remove the default value for a property with multiple values and a single default, click the Deselect All button.

   - For multiple values with multiple restricted defaults, select the value (check boxes) that you want to set as defaults, and then click the OK button.

   Note: For multiple values without restrictions (that is, the Value range is Unrestricted), you do not need to select any defaults.
4. In the Edit Event Property window, click the OK button.

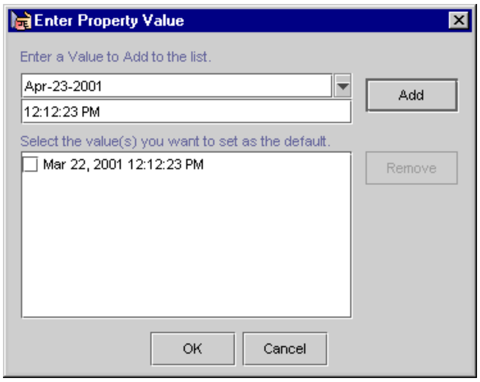
Properties with Date and Time Values

Properties with date and time values can use all Selection mode and Value range settings. For more information about these settings, see "Properties with Boolean or a Single Value and Single Default" and "Properties with Multiple Values and Single, Multiple, or All Defaults" on page 7-15.

To enter date and time values and set one or more defaults, complete the following steps:

1. In the Enter Property Values window shown in Figure 7-12, click the drop-down arrow in the Date list. A calendar appears.

Figure 7-12 Enter Date/Time Values


Enter Property Value

Enter a Value to Add to the list.

Apr-23-2001

12:12:23 PM

Select the value(s) you want to set as the default.

☐ Mar 22, 2001 12:12:23 PM

2. Select a date from the calendar.

3. In the Time field, enter a time.

4. Click the Add button.

5. To add more dates and times, repeat the first four steps until you have entered all the values.

6. To select one or more default values, complete one of the following:

   - If the event has a single date and time with a single default (restricted), click the OK button.

   - If the event has multiple dates and times with a single default (restricted), select the value (radio button) that you want to set as the default, and then click the OK button.

- If the event has multiple dates and times with multiple defaults (unrestricted), select the values (check boxes) that you want to set as the default, and then click the OK button.

7. In the Edit Event Property window, click the OK button.

Updating a Registered Custom Event

Whenever you make changes to a custom event's code, you should update that event's registration. Updating the registration lets the E-Business Control Center know about the changes in the custom event and aids campaign developers using the E-Business Control Center to modify any scenario actions that refer to the event.

To update a custom event, complete the following steps.

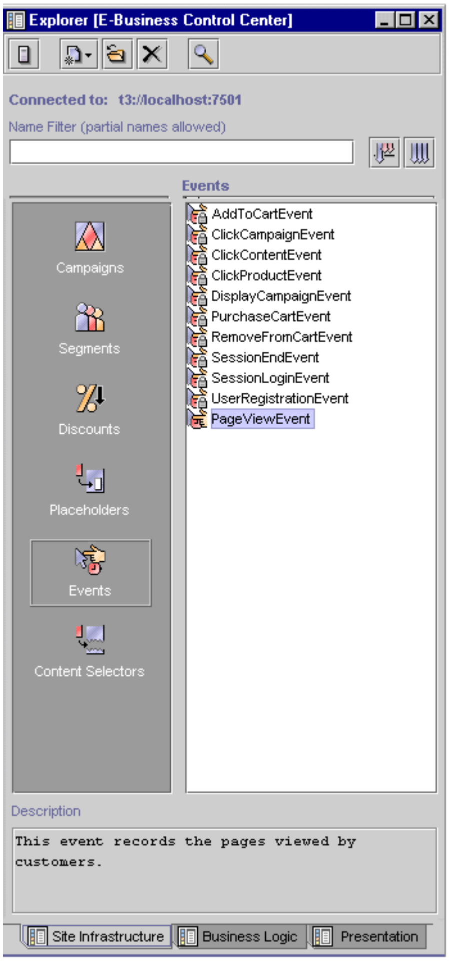
1. Start the E-Business Control Center and connect it to a Web server. The Explorer window opens.

   Note: For more information on connecting the E-Business Control Center to a server, see "Connecting to the Server" in the *Guide to Using the E-Business Control Center* documentation.

2. In the Explorer window, select the Event icon. A list of events appears in the Events field as shown in Figure 7-13.

   Note: You cannot edit standard events.

Figure 7-13 Explorer Window

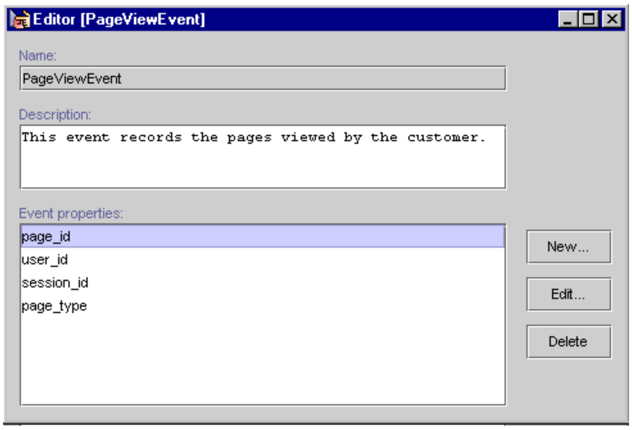


AddToCartEvent
ClickCampaignEvent
ClickContentEvent
ClickProductEvent
DisplayCampaignEvent
PurchaseCartEvent
RemoveFromCartEvent
SessionEndEvent
SessionLoginEvent
UserRegistrationEvent
PageViewEvent

```
This event records the pages viewed by
customers.
```

3. Double-click the custom event that you wish to edit. The Event Editor window opens as shown in Figure 7-14. The Event properties field displays a list of existing properties.

Figure 7-14 Event Editor Window



Name:
PageViewEvent

Description:
This event records the pages viewed by the customer.

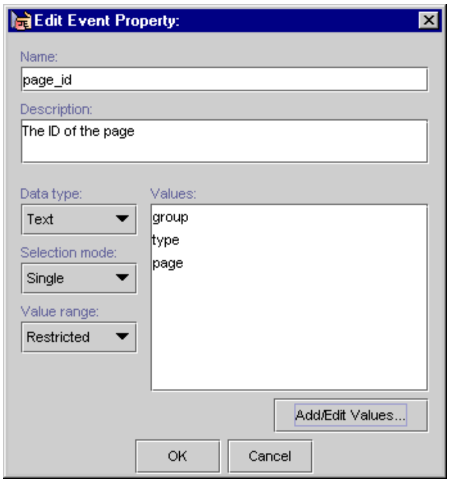
Event properties:
page_id
user_id
session_id
page_type

4. In the Event properties field, select the property that you want to edit.

   Note: For more information about setting custom event properties, see "Property Values and Setting the Default Value" on page 7-13.

5. Click the Edit button. The Edit Event Editor window opens as shown in Figure 7-15.

Figure 7-15 Edit Event Property Window



Name:
page_id

Description:
The ID of the page

Data type: Text

Selection mode: Single

Value range: Restricted

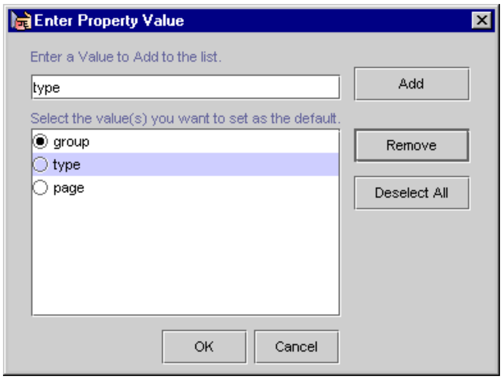
Values:
group
type
page

6. To change the Data type, Selection mode, or Value range, select a setting from the appropriate list box.

   Note: If you change the property setting Data type, Selection mode, or Value range, the associated values will be erased.

7. To add or change values, click the Add/Edit values button. The Enter Property Value window opens as shown in Figure 7-16.

Figure 7-16 Enter Property Value Window



Enter a Value to Add to the list.

type

Select the value(s) you want to set as the default.

- ● group
- ○ type
- ○ page

   a. To remove a value, select the value, and then click the Remove button.

   b. To add a value, enter the value, and then click the Add button.

   c. To change a value, select the value, remove it, and then add the new value.

   d. If required, select the default value or values.

   e. To remove the default value for a property with multiple values and a single default, click the Deselect All button.

   f. Click the OK button. The Enter Property Value window closes.

8. After you have finished updating the properties or values for the event, click the OK button in the Edit Event Property window.

CHAPTER

8 Creating and Managing Users

BEA WebLogic Personalization Server applications can store personal information about customers and display Web site content or offer other services based on customers' identities. For example, if you store information about the types of mutual funds in which your customer invests (conservative, moderate, aggressive), you can present content, advertisements, and additional fund recommendations that reflect each customer's preference.

The first sections of topic provide background information for the following WebLogic Personalization Server features:

- User and Group Profiles
- Security Realms and User Profiles
- Unified User Profiles
- Anonymous User Profiles
- Platform for Privacy Preferences Project (P3P)

The remaining sections provide instructions for creating and persisting data about your customers:

- Creating and Modifying Groups
- Creating and Modifying Users

- Accessing User and Group Data
- Setting Global Values for a Profile
- Accessing Properties from an LDAP Server
- Incorporating Data from Other External Sources

User and Group Profiles

A *customer profile* (or user profile) is a schema that determines which data you collect and store about a customer. Each piece of data in a customer profile is called a *customer property*. Customer properties can range from statically-defined properties, such as a user's social security number, to dynamically-created and persisted properties, such as Web-site tracking information for a particular user, or user preferences entered from a standard input screen.

Property Inheritance

User and group profiles can both inherit property values through the concept of a successor profile. If a property is requested from a profile, and no value is found for that property, the profile's successor will be queried for the profile. This process will be repeated recursively until either a value is found, or no more successors are found.

There are two types of successors: implicit and explicit.

An implicit successor is one that is set for a profile. For example, you can retrieve a user profile's `ProfileWrapper` object and call `setSuccessor()` on it, passing in a group name and a property set name. From that point on, that group will be used as a successor for the user profile when a property belonging to that property set is requested. A profile can have one implicit successor per property set, and one for the default, or null, property set. Properties that are set using a null property set name are are members of the default property set. To se an implicit successor for the default property set, you can call `ProfileWrapper.setSuccessor()` with null for the property set name argument.

Explicit successors are provided at the time of the property query, as an extra parameter to the `getProperty()` call. For example, calling `getProperty("FooPropertySet", "Bar", "SomeGroup")` on a user profile will look for the property called `Bar` in the `FooPropertySet`, then in the default property set, and if it is not found in either place it will retrieve the profile for the group called `SomeGroup` and repeat the search with that profile, and any of its implicit successors. Explicit successors will always be queried before implicit successors.

Users can only have groups as successors, and groups can only have other groups as successors. By default, when you add a group to a group, the parent becomes the child's implicit successor for the default property set. However, since users can belong to multiple groups, it is up to your application to decide which group to set as a user's successor.

The overall property search order, when `getProperty( "Foo", "Bar", "SomeGroup" )` is called, is as follows:

1. Look for a property named "`Bar`" in the "`Foo`" property set in the current profile.
2. Look for a property named "`Bar`" in the default property set, in the current profile.
3. Look for a property named "`Bar`" in the "`Foo`" property set in the "`SomeGroup`" profile. (This will start the search recursively in the "`SomeGroup`" profile.)
4. If there is an implicit successor for the "`Foo`" property set, look for a property named "`Bar`" in the "`Foo`" property set in that successor's profile. (Again, this will start the search recursively in that profile.)
5. If there is an implicit successor for the default property set, look for a property named "`Bar`" in the "`Foo`" property set in that successor's profile. (Again, this will start the search recursively in that profile.)
6. If the property is still not found, return the default as defined in the "`Foo`" property set.

Property Sets and Profiles

User profiles use property sets to organize the properties that they contain. A *property set* is a convenient way to give a name to a group of properties for a specific purpose; a *property set type* establishes a set of expectations for how a property set is used.

The sample applications define a property set named CustomerProperties which belongs to the User Profile property set type. CustomerProperties contains around thirty properties for an e-commerce customer, such as First Name, Last Name, Home Phone, Email, and Customer Type.

You create property sets and property definitions in the E-Business Control Center. You use JSP tags, APIs, or other WebLogic Personalization Server services to set the value of a property. WebLogic Personalization Server does not support creating a new property set type.

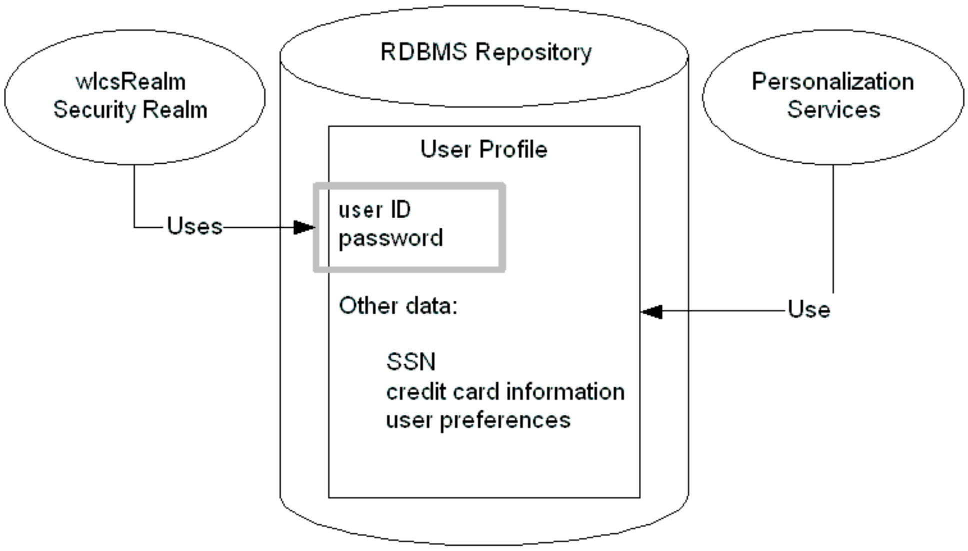
For more information about property sets, see Chapter 7, “Creating and Managing Property Sets,” in this guide.

Security Realms and User Profiles

A *security realm* determines how a user is authenticated and retrieves access control lists for given names. WebLogic Server supports several types of security realms for different environments and security needs.

The default WebLogic Personalization Server realm, wlcsRealm, stores and retrieves user IDs and passwords from the user profiles that are stored in the WebLogic Personalization Server RDMBS repository. (See Figure 8-1.)

Figure 8-1 wlcsRealm and User Properties



Alternate Security Realms and User Profiles

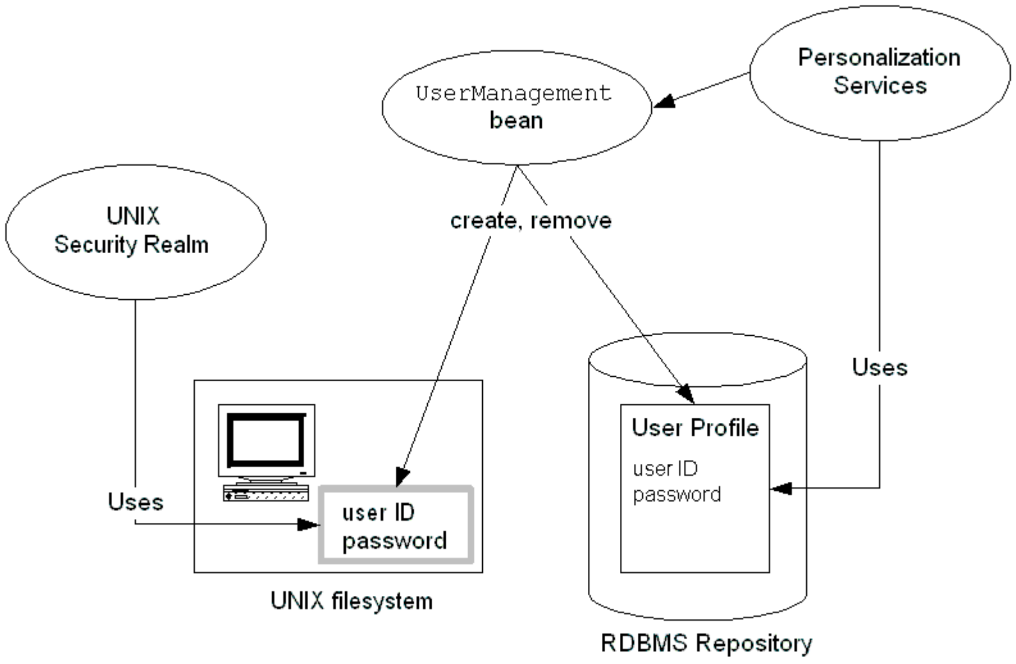
You can use other types of security realms to store and retrieve authentication data (user IDs, passwords, ACLs) in sources other than the RDBMS repository. For example, if you already have users and groups defined on a UNIX network, you can use the UNIX Security Realm to authenticate users and groups using UNIX login IDs and passwords.

For a list of the types of security realms that WebLogic Server supports, refer to "Security Fundamentals" in the *WebLogic Server Programming WebLogic Security* guide. Also refer to the WebLogic Portal *Security Guide*.

WebLogic Personalization Server provides two session beans, the `UserManager` and `GroupManager`, as a single point of entry for creating, editing, and removing user and group data. If you use an alternate security realm, methods in these beans act upon both the realm and the profile manager components to ensure that they stay synchronized.

For example, Figure 8-2 illustrates using the `UserManager` to create or remove users. A Personalization service uses `UserManager` to create user IDs and passwords in both the UNIX Realm and the user profile in the RDBMS data repository.

Figure 8-2 Synchronizing Security Realms and User Profiles



In general, if you are using APIs to manage user data, use the `UserManager` and `GroupManager` methods as opposed to manipulating the security realm directly.

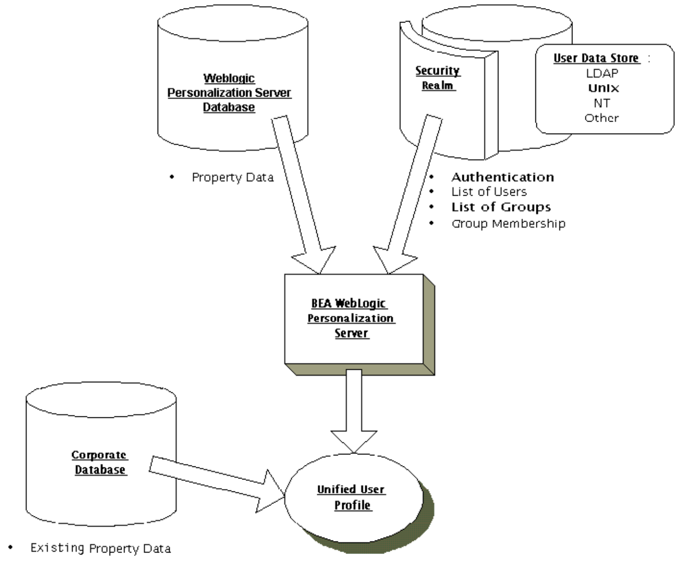
If you use a read-only realm such as LDAP Realm, `UserManager` and `GroupManager` cannot create users and groups in the realm. Instead, WebLogic Personalization Server creates user profiles for users and groups from the read-only realm on-the-fly. For example, if a user "joe" exists in your LDAP server and you are running the LDAP realm, the first time Joe's user profile is requested, the `UserManager` will see that the profile does not exist. It will then verify that Joe exists as a user in the LDAP realm, and will create a new profile record at that point and return it.

If you use a third-party tool to manage user records in an alternate security realm (such as the administration tool that comes with an LDAP server), the `UserManager` has no way to know when users and groups are removed from the realm. This can result in profile records that belong to users and groups that no longer exist. The User Management Administration Tools provide a means to clean up these leftover records. See the section "Deleting User Records That Do Not Exist in the Realm from the Personalization Database" on page 8-31.

Unified User Profiles

WebLogic Personalization Server provides the ability to combine user properties from the WebLogic Personalization Server RDBMS repository with user properties from other data sources, such as an LDAP server into a single Unified User Profile (UUP). With the UUP, developers and system users need not worry about the different underlying data sources of user data. To them there is just one place to go for user information—the user profile. (See Figure 8-3.)

Figure 8-3 UUP That Combines Several Data Sources



For information on setting up a Unified User Profile that gathers user properties from an LDAP server, refer to “Accessing Properties from an LDAP Server” on page 8-34. For information on gathering user properties from other data sources, refer to “Incorporating Data from Other External Sources” on page 8-36.

Anonymous User Profiles

Certain scenarios require an unidentified user to be able to use a system. While the unidentified user is using the system, you may need to have a profile for that user in order to set and get properties. For instance, a portal Web site might want to let new users tour the Web site and configure a few things before they actually have an official login name and password. The anonymous user profile allows for a user profile to be created for such a user.

An anonymous user profile can be treated just like a user profile for a known user, but the anonymous user profile only lives for the life of the user session. If the session is terminated without capturing an identity for the user, any profile information accumulated during the life of the anonymous user profile is lost. An anonymous user profile has no successor and will not retrieve default property values from a Property Set.

The anonymous user profile is available only through JSP tags. An anonymous profile is created when a `<um:setProperty>` or `<um:getProperty>` JSP tag is used before a `<um:getProfile>` tag has been called. If during a session a persistent user profile is created for the anonymous user, the `<um:createUser>` tag can be told to store the values from the anonymous profile into the new user profile. This is done with the `saveAnonymous` tag parameter set to `true`, as in:
`<um:createUser saveAnonymous="true">`.

Note: Campaigns cannot be used with anonymous users. Campaigns require a user ID that has two characteristics: the ID must be associated with a user profile, and that user profile must be saved (persisted). However, the anonymous profile for a user who is not logged in is a runtime profile (not saved), and not associated with a user ID.

Personalization features such as <pz:div> and <pz:contentSelector> JSP tags do work for anonymous users. This is because these features can use a runtime profile without a user ID.

For more information on these tags, see the section “User Management JSP tags” in Chapter 13, “Personalization Server JSP Tag Library Reference,” in this guide.

Platform for Privacy Preferences Project (P3P)

The Platform for Privacy Preferences Project (P3P) is an emerging industry standard that is designed to provide an automated way to compare consumers' privacy preferences with the privacy practices of the Web sites they visit. It lets Web sites express their privacy practices in a format that can be retrieved automatically and interpreted easily.

The P3P is a work-in-progress by the World Wide Web Consortium (W3C), a global group drawn from industry, academia, and privacy groups as well as public policy organizations. For more information about the World Wide Web Consortium's ongoing P3P effort, visit the P3P site at http://www.w3.org/P3P.

Essentially, P3P compliance means that your Web site presents a privacy policy to the user. As put forth in the P3P specification, a privacy policy is a set of one or more privacy statements that describe what personal user data a Web site will retrieve, and how the data is to be used. The P3P specification currently defines three mechanisms by which a Web site's privacy policy information can be presented to the end user:

- By publishing the policy reference file at a well-known URL.
  For complete information, see the P3P specification, section 2.2.1.
  http://www.w3.org/TR/P3P/#mechanism_ref
- By injecting a special header in each HTTP response served up by the Web server. For complete information, see the P3P specification, section 2.2.2.
  http://www.w3.org/TR/P3P/#syntax_ext
- By using an embedded <link> tag in the body of an HTML page.
  For complete information, see the P3P specification, section 2.2.3.
  http://www.w3.org/TR/P3P/#syntax_link

BEA Systems applauds the efforts of the World Wide Web Consortium and other organizations around the world working to empower users to control the use of their personal information on the Web sites they visit. However, it is important to note that WebLogic Personalization Server does not in any way enforce P3P compliance—that option is left up to the Web site developer.

Creating and Modifying Groups

This section describes using the WebLogic Portal Administration Tools to complete the following tasks:

- Creating Groups
- Adding Users to Groups
- Removing Users from Groups
- Editing Group Property Values
- Deleting Groups

For information on using JSP tags or APIs to create and modify groups, refer to Chapter 13, "Personalization Server JSP Tag Library Reference," and to the *Javadoc* API documentation for the `UserManager` and `GroupManager` EJBs.

Creating Groups

Note: The User Management tools do not allow the creation of a group called "everyone," because this is a reserved WebLogic Server group name.

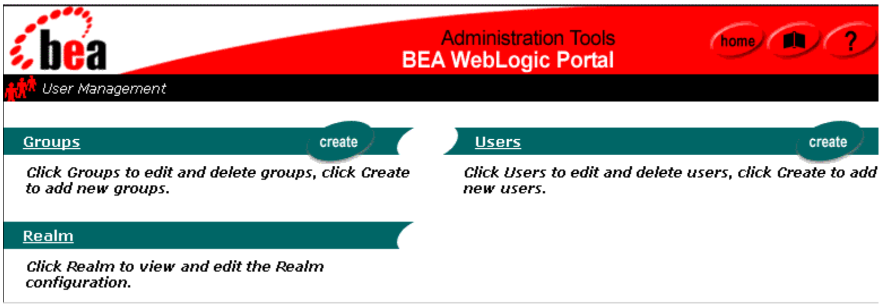
To create groups:

1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon.

   For more information, see "WebLogic Portal Administration Tools" in the *WebLogic Portal Architectural Overview*.

2. On the WebLogic Portal Administration Tools page, click the User Management icon. The User Management Home page appears. (See Figure 8-4.)

Figure 8-4 User Management Administration Tools Home Page



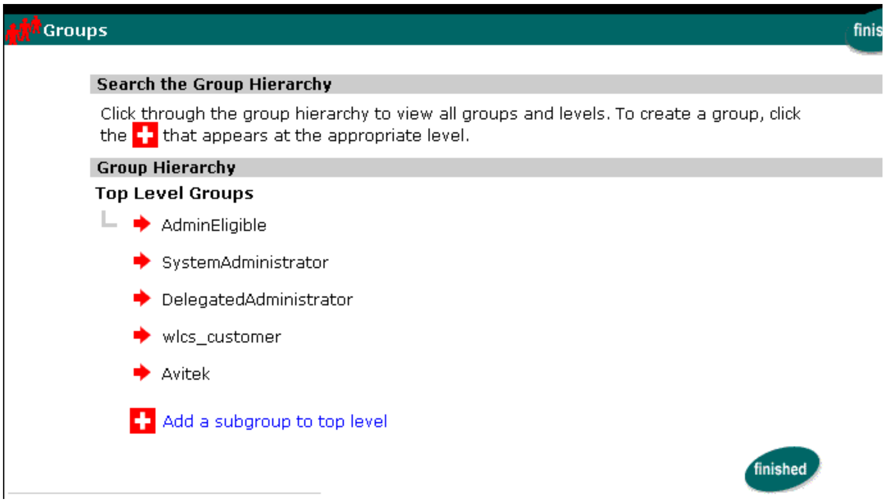
Click Groups to edit and delete groups, click Create to add new groups.

Click Users to edit and delete users, click Create to add new users.

Click Realm to view and edit the Realm configuration.

3. On the User Management Home page, click Create in the Groups banner. The Groups page appears. (Figure 8-5)

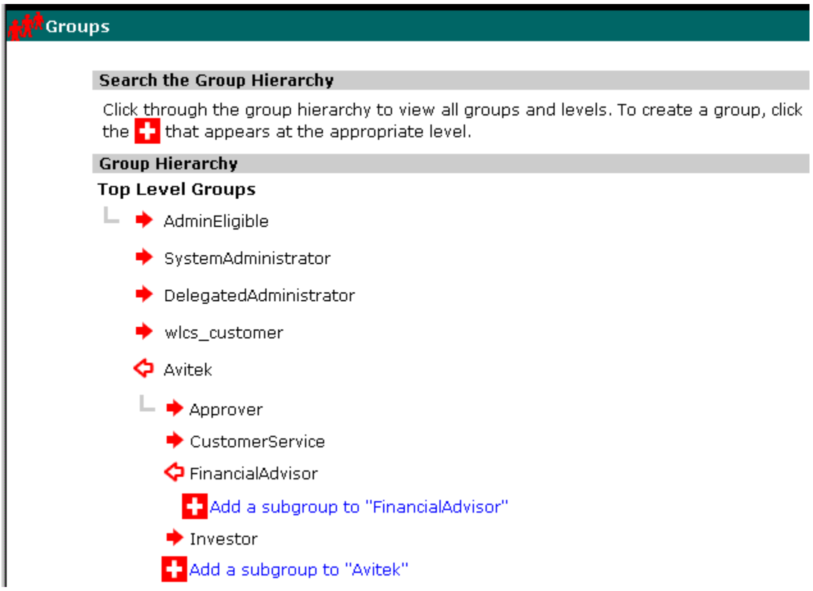
Figure 8-5 The Create a Group Page



Click through the group hierarchy to view all groups and levels. To create a group, click the that appears at the appropriate level.

4. Within the Group Hierarchy tree view, expand the hierarchy as needed to display the add icon (+) at the level you wish to add the group. (Figure 8-6)

Figure 8-6 Expand the Group Hierarchy To Display the Add Icon (+)



Groups

Search the Group Hierarchy

Click through the group hierarchy to view all groups and levels. To create a group, click the ➕ that appears at the appropriate level.

Group Hierarchy

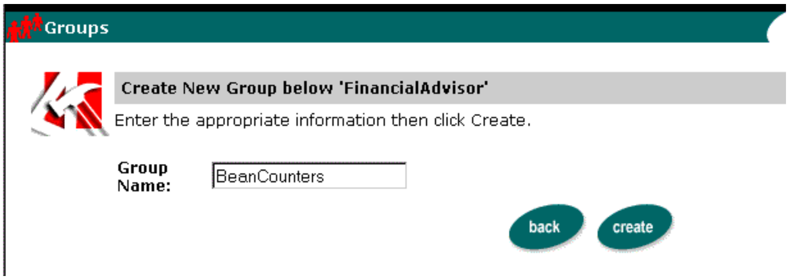
Top Level Groups

- AdminEligible
- SystemAdministrator
- DelegatedAdministrator
- wlcs_customer
- Avitek
  - Approver
  - CustomerService
  - FinancialAdvisor
    - Add a subgroup to "FinancialAdvisor"
  - Investor
  - Add a subgroup to "Avitek"

5. Click on the plus sign. The Create a Group page appears.

6. Enter the name of the new group in the Group Name field. (Figure 8-7)

Figure 8-7 Enter the Name of the New Group



Groups

Create New Group below 'FinancialAdvisor'

Enter the appropriate information then click Create.

Group Name: BeanCounters

7. Click Create. A success or failure message appears.

8. Click Back to return to the Create a Group page, or click Create to enter another new group name at the same level (step 6.)

9. On the Create a Group page, click Finished to return to the User Management Home Page.

After you create a group, you can then personalize it by overriding the default property values. (The default property values are set up in the E-Business Control Center. For more information, see Chapter 7, “Creating and Managing Property Sets,” in this guide.)

Adding Users to Groups

To add users to groups:

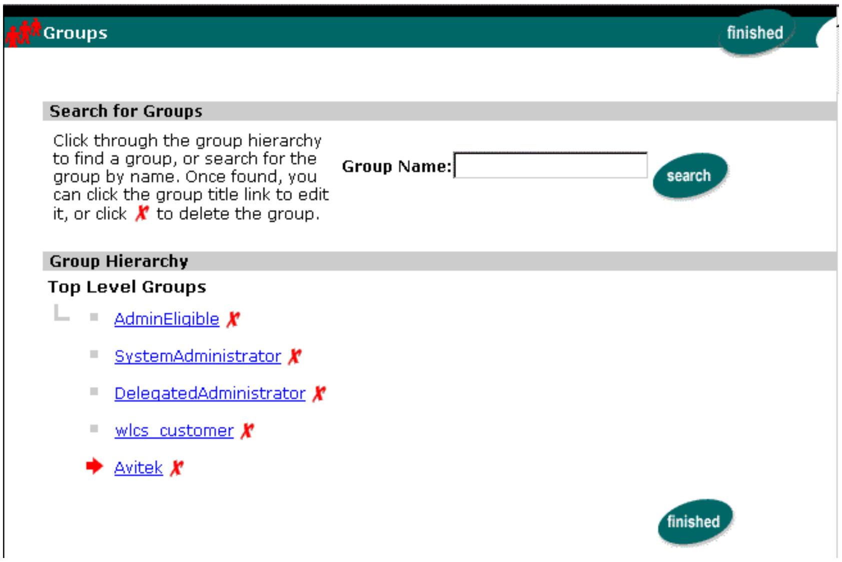
1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon.

   For more information, see “WebLogic Portal Administration Tools” in the *WebLogic Portal Architectural Overview*.

2. On the WebLogic Portal Administration Tools page, click the User Management icon. The User Management Home page appears. (See Figure 8-4.)

3. On the User Management Home page, click Groups in the Groups banner. The Search for a Group page appears. (Figure 8-8)

Figure 8-8 Search for a Group Page



Search for Groups

Click through the group hierarchy to find a group, or search for the group by name. Once found, you can click the group title link to edit it, or click ✗ to delete the group.

Group Name:

Group Hierarchy

Top Level Groups

- AdminEligible ✗
- SystemAdministrator ✗
- DelegatedAdministrator ✗
- wlcs_customer ✗
- Avitek ✗

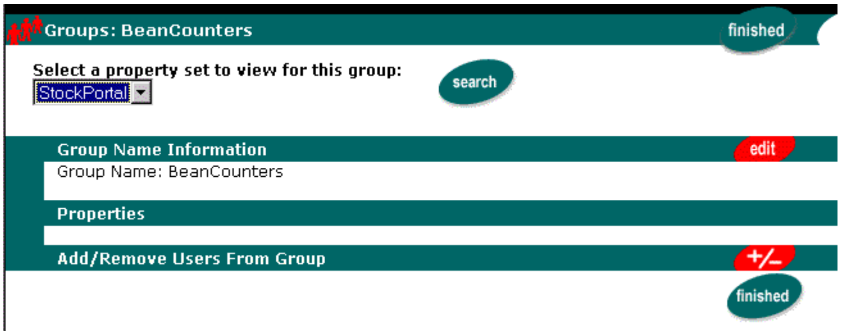
4. To locate the appropriate group, do one of the following:

   a. To locate the group by name, enter the group name in the Group Name field, then click Search.

   b. To locate the group within the Group Hierarchy, navigate the Group Hierarchy tree view.

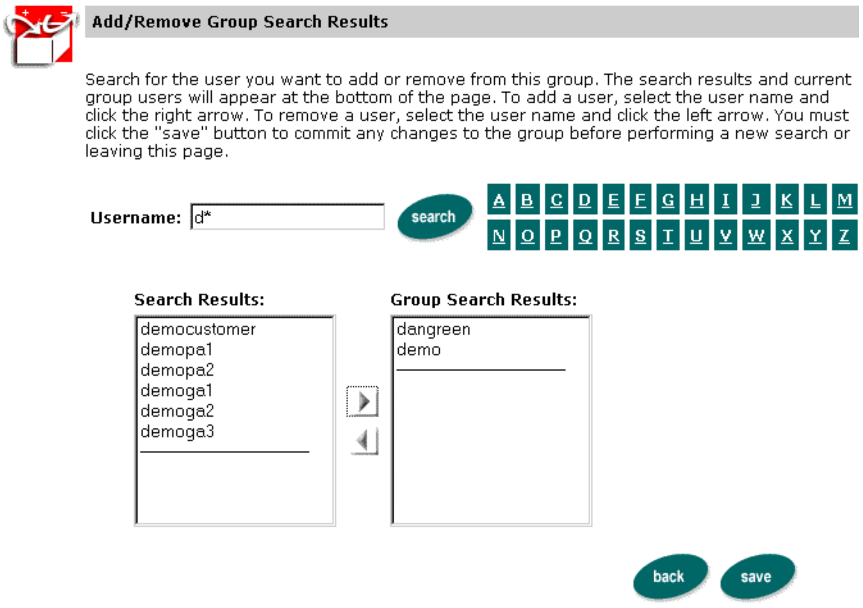
5. Select the group. The Group Properties view appears. (Figure 8-9)

Figure 8-9 Group Properties View



6. Click the add/remove icon (+/-) at the bottom of the page. The Add/Remove Users tool appears. (Figure 8-10)

Figure 8-10 Add/Remove Users Page



Search for the user you want to add or remove from this group. The search results and current group users will appear at the bottom of the page. To add a user, select the user name and click the right arrow. To remove a user, select the user name and click the left arrow. You must click the "save" button to commit any changes to the group before performing a new search or leaving this page.

Username: d*

Search Results:
democustomer
demopa1
demopa2
demoga1
demoga2
demoga3

Group Search Results:
dangreen
demo

To locate a user, do one of the following:

a. To locate the user by name, enter the username in the Username field, then click Search. The search results appear at the bottom of the page.

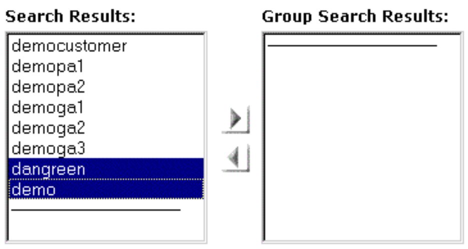
b. To see a list of all users within an alphabetized category, click the appropriate letter corresponding to the first letter of the username. A list of users appear at the bottom of the page.

c. To see a list of all users in the database, use the wildcard feature. Enter a partial username immediately followed by an asterisk (*). The asterisk is a search return variable.

Note: Only users already entered in the database are available in these lists. To add new users to the database, see "Creating Users" on page 8-24.

7. Select the username, or a group of names, from the Search Results field. (Figure 8-11)

Figure 8-11 Use Left-Arrow to Move User Names into a Group



democustomer
demopa1
demopa2
demoga1
demoga2
demoga3
dangreen
demo

8. Click the left-to-right directional arrow. The username(s) appears in the Group Search Results field.

9. Click Save to commit the selected name(s) to the Group. You must click Save before beginning a new search.

10. Click Back to return to the Group Properties view.

Note: The search applies to both list boxes.

Removing Users from Groups

To remove users from groups:

1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon.

   For more information, see "WebLogic Portal Administration Tools" in the *WebLogic Portal Architectural Overview*.

2. On the WebLogic Portal Administration Tools page, click the User Management icon. The User Management Home page appears. (See Figure 8-4.)

3. On the User Management Home page, click Groups in the Groups banner. The Search for a Group page appears. (Figure 8-8)

   To locate the appropriate group, do one of the following:

   a. To locate the group by name, enter the group name in the Group Name field, then click Search.

   b. To locate the group within the Group Hierarchy, navigate the Group Hierarchy tree view.

4. Select the group. The Group Properties view appears. (Figure 8-9)

5. Click the add/remove icon (+/-) at the bottom of the page. The Add/Remove Users tool appears. (Figure 8-10)

   To locate a user, do one of the following:

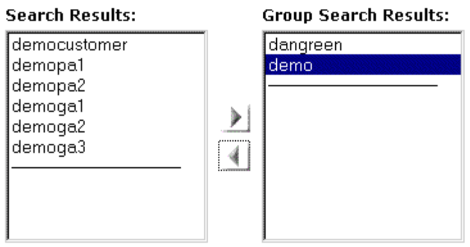
   a. To locate the user by name, enter the username in the Username field, then click Search. The search results appear at the bottom of the page.

   b. To see a list of all users within an alphabetized category, click the appropriate letter corresponding to the first letter of the username. A list of users appear at the bottom of the page.

   c. To see a list of all users in the database, use the wildcard feature. Enter a partial username immediately followed by an asterisk (*). The asterisk is a search return variable.

6. Select the username, or a group of usernames, from the Group Search Results field. (Figure 8-12)

Figure 8-12 Use Right-Arrow to Move Usernames Out of a Group



7. Click the right-to-left directional arrow. The username(s) is removed from the Group Users field and appears in Search Results.
8. Click Save to remove the username(s) from the Group.
9. Click Back to return to the Group Properties view.

Editing Group Property Values

To edit group property values:

1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon.

   For more information, see “WebLogic Portal Administration Tools” in the *WebLogic Portal Architectural Overview*.

2. On the WebLogic Portal Administration Tools page, click the User Management icon. The User Management Home page appears. (See Figure 8-4.)

3. On the User Management Home page, click Groups in the Groups banner. The Search for a Group page appears. (Figure 8-8)

   To locate the appropriate group, do one of the following:

   a. To locate the group by name, enter the group name in the Group Name field, then click Search.

   b. To locate the group within the Group Hierarchy, navigate the Group Hierarchy tree view.

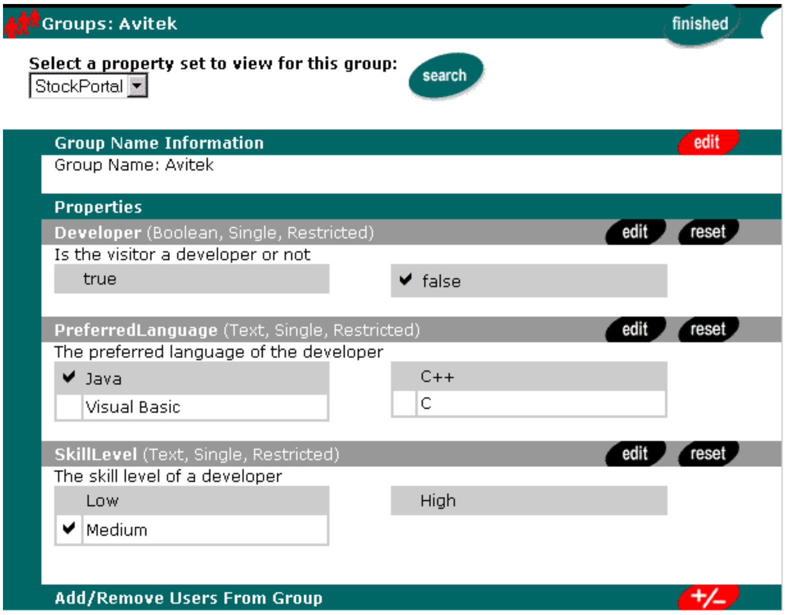
4. Select the group. The Group Properties view appears. (Figure 8-9)

5. Select or search for a property set to view for this group. The group's default property values appear if no other property set has been accessed during the tools session.

Note: Default property values are created using the E-Business Control Center. For specific instructions on property set management, see Chapter 7, "Creating and Managing Property Sets."

6. Click Search. The Edit Default Properties page appears. (Figure 8-13)

Figure 8-13 Edit Default Properties Page



Select a property set to view for this group: StockPortal

Group Name Information
Group Name: Avitek

Properties

Developer (Boolean, Single, Restricted)
Is the visitor a developer or not
true
[✓ false]

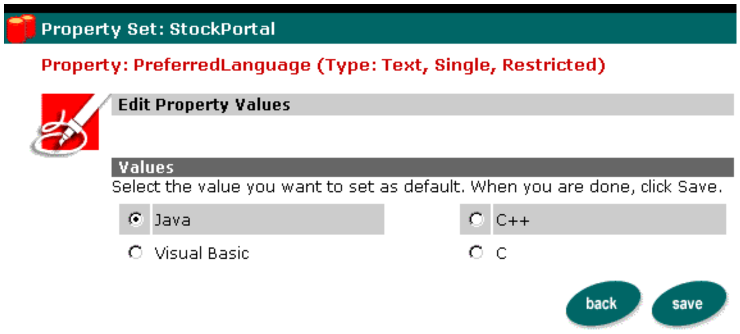
PreferredLanguage (Text, Single, Restricted)
The preferred language of the developer
[✓ Java]
C++
Visual Basic
C

SkillLevel (Text, Single, Restricted)
The skill level of a developer
Low
High
[✓ Medium]

Add/Remove Users From Group

7. Click Edit on the appropriate Property bar. The associated Edit Property Values page appears. (Figure 8-14)

Figure 8-14 Edit Property Values Page



Property Set: StockPortal

Property: PreferredLanguage (Type: Text, Single, Restricted)

Edit Property Values

Values

Select the value you want to set as default. When you are done, click Save.

[✓ Java] C++

Visual Basic C

8. Change the values on the Edit Property Values page.

Note: Non-default property sets and properties that were not configured through the Site Infrastructure tab in the E-Business Control Center are not editable here.

9. Click Save.

10. Click Back to return to the Group Properties view.

11. To edit other properties, return to step 5.

12. If you click the Reset button on the Property bar (instead of Edit as we did in step 7.), the property is set to null for that user. This will have one of three results:

    - First, if the property has a default value, the group will have that default value. Note that the default value is not copied into the group's settings. The group's value is just set to null so that the default value will be returned when `getProperty()` is called for that property. If the default value changes, calling `getProperty()` will return the new default value.
    - Second, if the property is defined in a Property Set but does not have a default value, the user will have a null for that property.
    - Third, if the property was dynamically defined (that is, it does not belong to a Property Set), resetting causes that property to be deleted.

Deleting Groups

To delete groups:

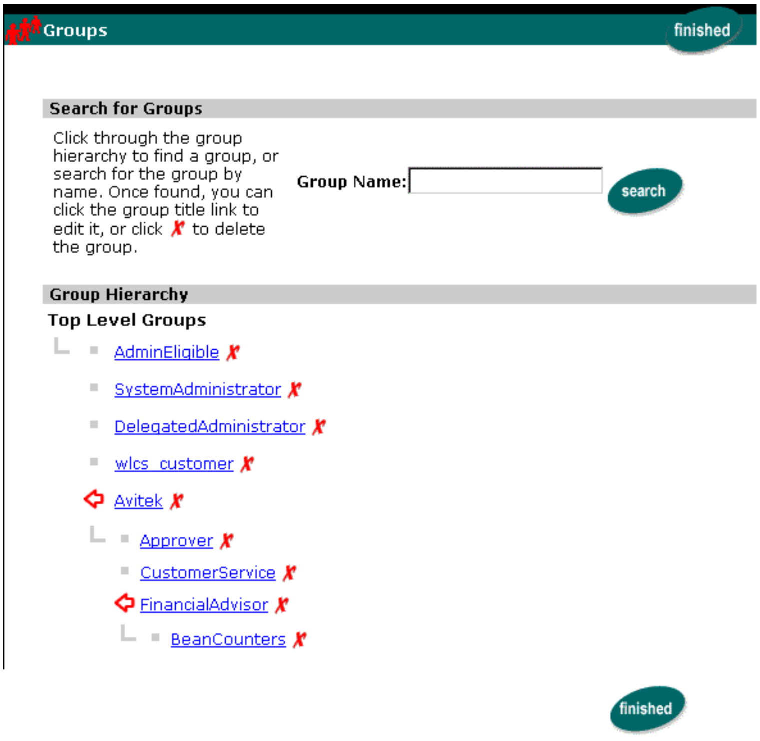
1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon.

   For more information, see "WebLogic Portal Administration Tools" in the *WebLogic Portal Architectural Overview*.

2. On the WebLogic Portal Administration Tools page, click the User Management icon. The User Management Home page appears. (See Figure 8-4.)

3. On the User Management Home page, click Groups in the Groups banner. The Search for a Group tool appears. (Figure 8-15)

Figure 8-15 Search for a Group To Delete



Click through the group hierarchy to find a group, or search for the group by name. Once found, you can click the group title link to edit it, or click ✗ to delete the group.

Group Name:

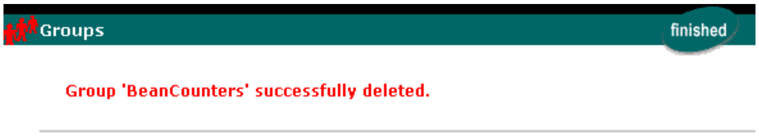
a. To locate the group to delete by name, enter the group name in the Group Name field, then click Search. The group name must be entered exactly.

b. To locate the group to delete within the Group Hierarchy, navigate the Group Hierarchy tree view.

4. Click the X to the right of the group name.

5. A confirmation box appears. (Figure 8-16)

Figure 8-16 Confirmation Box: Group Successfully Deleted



Group 'BeanCounters' successfully deleted.

6. Click Finished to return to the User Management Home Page.

Creating and Modifying Users

This section describes using the WebLogic Portal Administration Tools to complete the following tasks:

- Creating Users
- Editing User Property Values
- Deleting Users
- Deleting User Records That Do Not Exist in the Realm from the Personalization Database

For information on using JSP tags or APIs to create and modify users, refer to Chapter 13, "Personalization Server JSP Tag Library Reference," and to the *Javadoc* API documentation for the `UserManager` and `GroupManager` EJBs.

For information on how customers can create and modify their own user profiles, refer to the *Guide to Registering Customers and Managing Customer Services*.

Creating Users

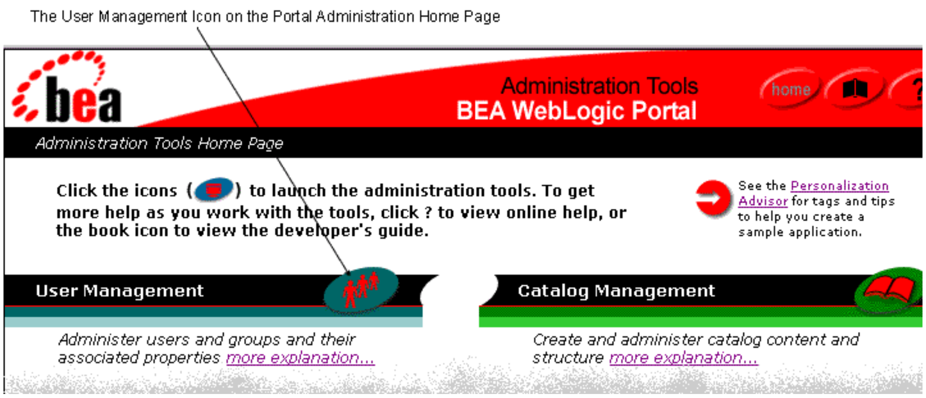
Note: The administration tools do not allow the creation of a user with username "system" or "guest", as these are reserved WebLogic Server terms.

To create users:

1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon. (See Figure 8-17.)

   For more information, see "WebLogic Portal Administration Tools" in the *WebLogic Portal Architectural Overview*.

Figure 8-17 User Management Icon on the WebLogic Portal Administration Tools


The User Management Icon on the Portal Administration Home Page
bea
Administration Tools
BEA WebLogic Portal
home
?
Administration Tools Home Page

Click the icons ( ) to launch the administration tools. To get more help as you work with the tools, click ? to view online help, or the book icon to view the developer's guide.

See the Personalization Advisor for tags and tips to help you create a sample application.

User Management

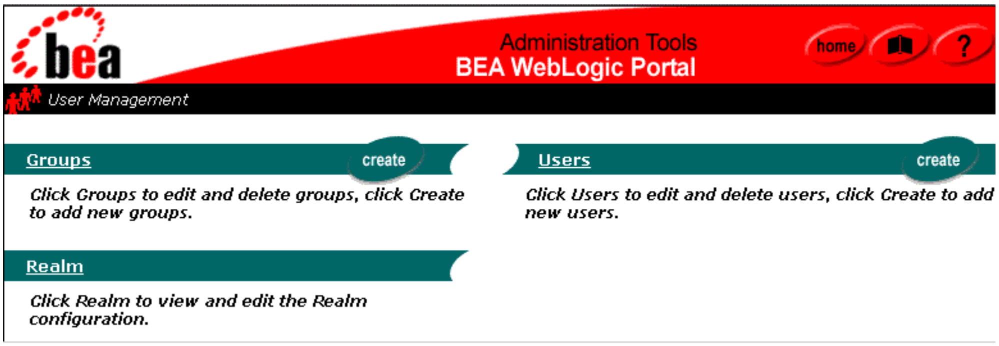
Administer users and groups and their associated properties more explanation...

Catalog Management

Create and administer catalog content and structure more explanation...

2. The User Management Home page appears. (Figure 8-18)

Figure 8-18 User Management Home Page



Groups create

Click Groups to edit and delete groups, click Create to add new groups.

Users create

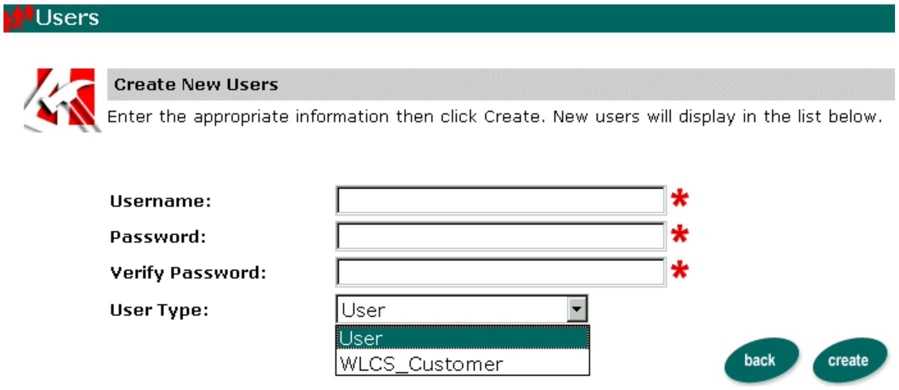
Click Users to edit and delete users, click Create to add new users.

Realm

Click Realm to view and edit the Realm configuration.

3. On the User Management Home page, click Create in the Users banner. The Create New Users page appears. (Figure 8-19)

Figure 8-19 Create New Users Page



Create New Users

Enter the appropriate information then click Create. New users will display in the list below.

Username: *

Password: *

Verify Password: *

User Type: User

User

WLCS_Customer

4. Enter the username in the Username field.

   Note: Limit username to 25 characters.

5. Enter the password associated with the Username in the Password field.

6. In the Verify Password field, re-enter the password provided in step 5.

   Note: Characters in password fields appear as asterisks.

7. From the User Type list, select a profile type. The user will be an instance of this profile type. This allows the system to access explicit properties in a Unified Profile type, and ensures proper data cleanup when the user is removed.

8. Click Create. The new user appears at the bottom of the page. Alternatively, click Back to return to the User Management Home page without creating the new user.

Note: The WLCS RDBMSrealm allows mixed case user creation. (For example: User, user.)

After you create a user, you can then personalize it by overriding the default property values. (The default property values are set up in the E-Business Control Center. For more information, see Chapter 7, "Creating and Managing Property Sets," in this guide.)

Editing User Property Values

Note: Explicit properties of UUP are only editable from the administration tools if a property set is created that mirrors those properties.

To edit user property values:

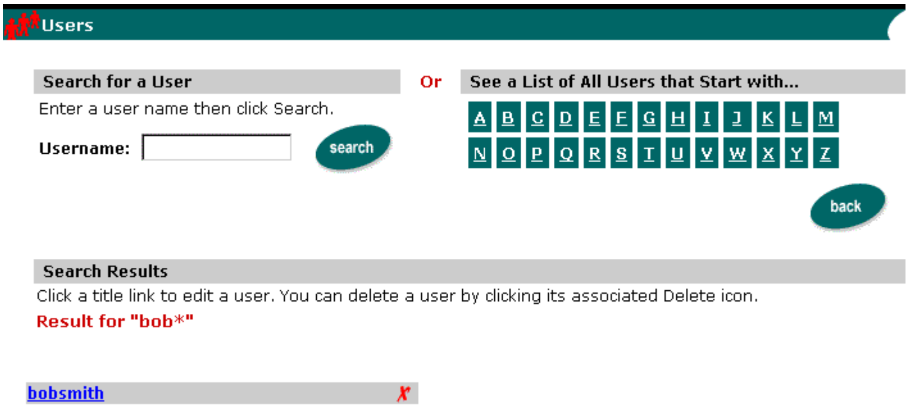
1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon. (See Figure 8-17.)

   For more information, see "WebLogic Portal Administration Tools" in the *WebLogic Portal Architectural Overview*.

2. The User Management Home page appears. (Figure 8-18)

3. On the User Management Home page, click Users in the Users banner. The Search for a User tool appears. (Figure 8-20)

Figure 8-20 Search for a User


Users
Search for a User
Or
See a List of All Users that Start with...
Enter a user name then click Search.
Username:
search
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
back
Search Results
Click a title link to edit a user. You can delete a user by clicking its associated Delete icon.
Result for "bob*"
bobsmith

To locate a user, do one of the following:

a. To locate the user by name, enter the username in the Username field, then click Search. The search results appear at the bottom of the page.

b. To see a list of all users within an alphabetized category, click the appropriate letter corresponding to the first letter of the username. A list of users appear at the bottom of the page.

c. To see a list of all users in the database, use the wildcard feature. Enter a partial username immediately followed by an asterisk (*). The asterisk is a search return variable.

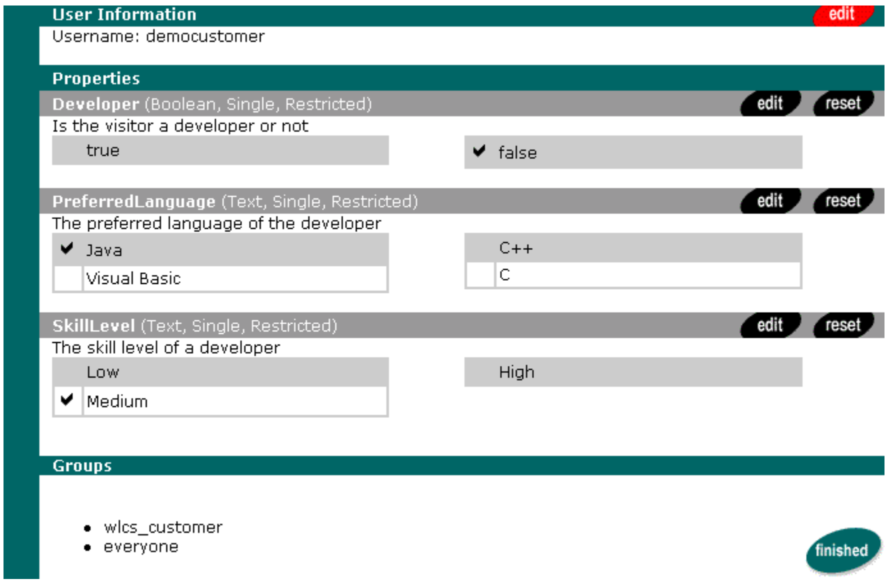
4. Select the user. The User Property view appears.

5. In the drop-down list, select a property set to view for this user.

Note: Default property values are created using the E-Business Control Center. For specific instructions on property set management, see Chapter 7, "Creating and Managing Property Sets."

6. Click Search. The User Properties view appears. (Figure 8-21)

Figure 8-21 User Properties View



User Information
Username: democustomer

Properties

Developer (Boolean, Single, Restricted)
Is the visitor a developer or not
true
✓ false

PreferredLanguage (Text, Single, Restricted)
The preferred language of the developer
✓ Java
C++
Visual Basic
C

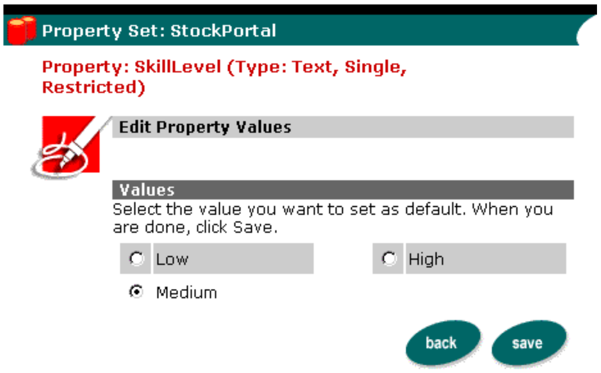
SkillLevel (Text, Single, Restricted)
The skill level of a developer
Low
High
✓ Medium

Groups

- wlcs_customer
- everyone

7. Click Edit on the appropriate Property bar. The associated Edit Property Values page appears. (Figure 8-22)

Figure 8-22 Edit Property Values



Property Set: StockPortal

Property: SkillLevel (Type: Text, Single, Restricted)

Edit Property Values

Values
Select the value you want to set as default. When you are done, click Save.

○ Low
○ High
● Medium

8. Change the user’s values at the Edit Property Values page.

9. Click Save. A message appears indicating whether or not the edit was successful. Alternatively, click Back to return to the User Properties view without saving your changes.
10. Click Back to return to the User Properties view.
11. Return to step 4 and edit other properties as necessary.

Note: If you click the Reset button on the Property bar (instead of Edit as we did in step 6), the property is set to null for that user. This will have one of three results:

- First, if the property has a default value, the user will have that default value. Note that the default value is not copied into the user's settings. The user's value is just set to null so that the default value will be returned when getProperty() is called for that property. If the default value changes, calling getProperty() will return the new default value.
- Second, if the property is defined in a Property Set but does not have a default value, the user will have a null for that property.
- Third, if the property was dynamically defined (that is, it does not belong to a Property Set), resetting causes that property to be deleted.

Deleting Users

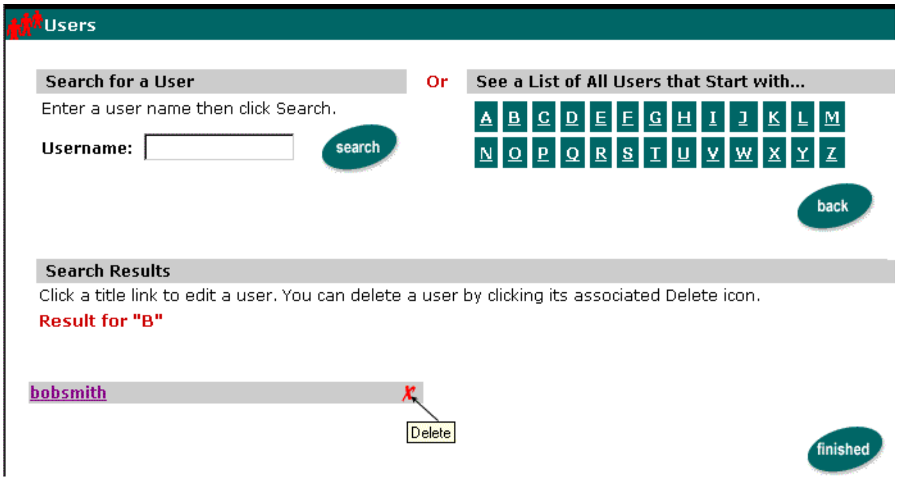
To delete users:

1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon. (See Figure 8-17.)

   For more information, see “WebLogic Portal Administration Tools” in the *WebLogic Portal Architectural Overview*.
2. The User Management Home page appears. (Figure 8-18)

3. On the User Management Home page, click Users in the Users banner. The Search for a User tool appears. (Figure 8-23)

Figure 8-23 Search for a User to Delete


Users
Search for a User
Enter a user name then click Search.
Username:
search
Or
See a List of All Users that Start with...
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
back
Search Results
Click a title link to edit a user. You can delete a user by clicking its associated Delete icon.
Result for "B"
bobsmith
Delete
finished

To locate a user, do one of the following:

a. To locate the user by name, enter the username in the Username field, then click Search. The search results appear at the bottom of the page.

b. To see a list of all users within an alphabetized category, click the appropriate letter corresponding to the first letter of the username. A list of users appear at the bottom of the page.

c. To see a list of all users in the database, use the wildcard feature. Enter a partial username immediately followed by an asterisk (*). The asterisk is a search return variable.

4. Click the X to right of the username to delete the user. A confirmation dialog box appears.

5. Click OK to confirm the deletion and return to the Search for Users page.

Deleting User Records That Do Not Exist in the Realm from the Personalization Database

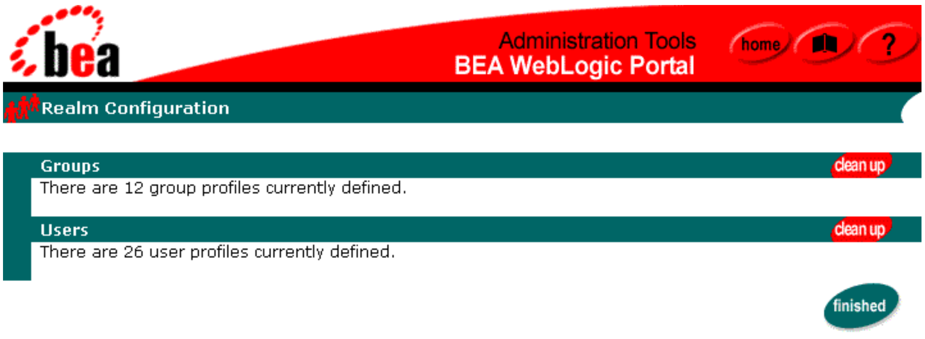
When users are deleted from the realm, they might leave behind orphan profile records in your personalization database. This will only happen if the users are deleted directly from the realm, and not through the `UserManager` session bean. For example, a user might be deleted from an LDAP realm through the LDAP server's admin tool. To delete user profile records that no longer exist in the realm from the Personalization database:

1. Start the WebLogic Portal Administration Tools for your application. On the Administration Tools home page, click the User Management icon. (See Figure 8-17.)

   For more information, see "WebLogic Portal Administration Tools" in the *WebLogic Portal Architectural Overview*.

2. n the User Management Home page, click Realm in the Realm banner. The Realm Configuration window appears. (Figure 8-24)

Figure 8-24 The Realm Configuration Window



There are 12 group profiles currently defined.

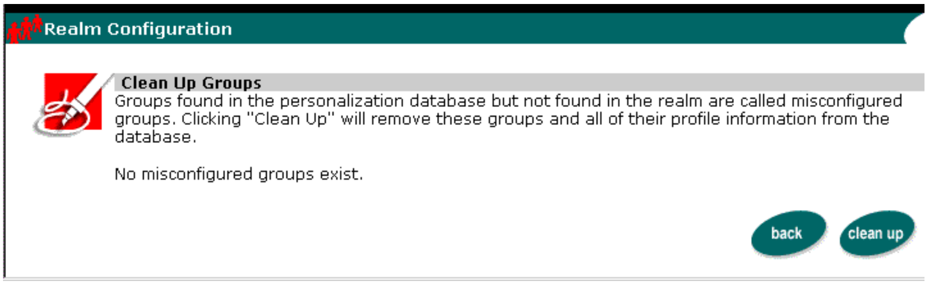
There are 26 user profiles currently defined.

3. Two banners, Groups and Users, are displayed.

4. Click the Cleanup button on the Users banner for a count of orphaned user profile records and an opportunity to delete them.

5. Click the Cleanup button on the Groups banner for a count of orphaned group profile records and an opportunity to delete them. (Figure 8-25)

Figure 8-25 The Realm Window Offers An Opportunity to Clean Up Orphans



Clean Up Groups
Groups found in the personalization database but not found in the realm are called misconfigured groups. Clicking "Clean Up" will remove these groups and all of their profile information from the database.

No misconfigured groups exist.

6. When finished, click Back to return to the Realm Configuration window.
7. Click Finished, to return to the User Management Administration Tools Home page.

Accessing User and Group Data

Some WebLogic Personalization Server services, such as content selectors, access user and group data without requiring your intervention. If your business needs require additional use of your user and group data, you can do the following:

- Use JSP Tags to Access User and Group Data
- Use APIs to Access User and Group Data

Use JSP Tags to Access User and Group Data

WebLogic Personalization Server provides JSP tag libraries that you can use to access user and group profile information, as well as create and delete users and groups, and manage user-group relationships.

For a complete list of user and group management JSP tags, refer to Chapter 13, “Personalization Server JSP Tag Library Reference.”

Use APIs to Access User and Group Data

The `UserManager` and `GroupManager` session EJBs provide user management functionality in a WebLogic Personalization Server-specific context. Services provided by the EJBs include:

- Creating/removing users
- Creating/removing groups
- Adding users to groups/removing users from groups
- Adding groups to groups/removing groups from groups
- Retrieving usernames corresponding to a group
- Retrieving group names corresponding to a user

For a complete list of `UserManager` and `GroupManager` services, see the UserManager and GroupManager *Javadoc* API documentation.

As discussed in “Alternate Security Realms and User Profiles” on page 8-6, the `UserManager` and `GroupManager` are intended as a single entry point to both the security realm and the personalization database. All methods on these session beans delegate to either the realm, the profile components, or both when appropriate. This enables the developer to use a custom security realm implementation and have it integrate seamlessly with the WebLogic Personalization Server.

In addition to `UserManager` and `GroupManager`, you can instantiate and use the lightweight `ProfileWrapper` object to access a user or group profile. A `ProfileWrapper` can access the correct ProfileManager session bean(s) based on the identity it was initialized with. If it was initialized with both a user and group name, the group will be used as an explicit successor for all getProperty methods that do not take an explicit successor. For more information about `ProfileWrapper`, refer to the *Javadoc* API documentation.

Setting Global Values for a Profile

Profile properties can also belong to the “default”, or null, property set, by passing in a null value when setting the property. Doing this will mean that the property cannot be validated or recognized by the E-Business Control Center, but it provides a way to set global values for a profile. When a property is requested, the profile first looks for a value with that name in the given property set, and if it is not found, it looks in the default property set. For example, a user profile can have the property `Age` defined in the default property set, and as long as there is no property called `Age` in another property set, calling `getProperty(x, "Age")` will always return that value for any property set `x`.

Accessing Properties from an LDAP Server

The WebLogic Portal sample applications include a default property set named “ldap,” which can retrieve properties from an LDAP server. Before you use this property set with the sample applications or with your own application, you must do the following:

1. If you have already deployed the application on a WebLogic Portal instance, stop the server.
2. Deploy the `ldapprofile.jar` component within your application as described in “Add JAR Files” under “Assembling and Deploying Enterprise Applications” in the *Deployment Guide*.
3. Start the server and deploy the application.

4. Start the WebLogic Server Administration Console for the active domain, and do the following:

   a. In the left pane, click Deployments → EJB → ldapprofile.

   b. Right-click this node.

   c. Select Edit Descriptor.

   d. When the descriptor comes up in a new window, navigate to `ldapprofile.jar` → EJB Jar → Enterprise Beans → Sessions → LdapPropertyManager → Env Entries.

5. This will list the connection variables, and their default values as shown in Table 8-15. You will need to edit these values to match your LDAP server's configuration.

Table 8-15 Edit These Values to Match Your LDAP Server's Configuration

Connection Variables	Description
`config/serverURL`	The URL of your LDAP server.
`config/principal`	The name of the principal to use when connecting to your LDAP server, usually "admin" or something similar.
`config/principalCredential`	The Password of the principal defined above.
`config/userDN`	The location of users in your LDAP directory.
`config/groupDN`	The location of groups in your LDAP directory
`config/usernameAttribute`	The attribute used to describe usernames in your LDAP directory, usually "uid". This will be used to map WLS usernames to LDAP usernames.
`config/groupnameAttribute`	The attribute used to describe group names in your LDAP directory, usually "cn". This will be used to map WLS group names to LDAP group names.

6. After editing these values, persist them by navigating to the root `ldapprofile.jar` node and clicking Persist.

7. Restart the server.

Incorporating Data from Other External Sources

This section describes how to create a Unified User Profile that incorporates user data from external data sources in addition to or instead of LDAP servers, such as a legacy system or another database (See Figure 8-3).

This section includes the following subsections:

- How WebLogic Portal Retrieves User Data from External Sources
- Configuring WebLogic Portal To Retrieve User Data from External Sources

Note: If you want to incorporate user data only from an LDAP server, you do not need to complete the steps in this section. Instead, refer to "Accessing Properties from an LDAP Server" on page 8-34.

Unified User Profile Security

A Unifed User Profile is not a custom WebLogic security realm. User passwords are set in the WebLogic security realm, not in the user profile with other user properties. When `UserManager.setPassword()` is called, it delegates to the WebLogic security realm, not to a `ProfileManager`. However, the Unified User Profile and the WebLogic security realm are closely associated, because the `UserManager` is the synchronization point between the Unified User Profile and the WebLogic security realm. For example, the `UserManager.createUser()` method delegates to the security realm to create the user there, and then to a `ProfileManager` to create the user profile using the `createUniqueId()` method of one or more `EntityPropertyManagers`. All user management operations should be performed through the `UserManager` instead of directly through the security realm in order to keep the security realm and the user profile in sync.

See `UserManager` in the WebLogic Portal Javadoc at http://e-docs.bea.com/wlp/docs40/interm/javadoc.htm.

For information on writing a custom security realm, go to http://e-docs.bea.com/wls/docs61/security/prog.html#1041025.

How WebLogic Portal Retrieves User Data from External Sources

This section provides background information for completing the tasks that are described in “Configuring WebLogic Portal To Retrieve User Data from External Sources” on page 8-40.

To retrieve user data from a non-LDAP, external source, WebLogic Portal does the following (See Figure 8-26):

1. A JSP tag, API, or WebLogic Portal service invokes the `ProfileFactory` and passes a user ID as a parameter. The `ProfileFactory` returns a `ProfileWrapper` object for the user.

   `ProfileWrapper` is a lightweight object that is used to access a user or group profile. For more information, refer to the *Javadoc* API documentation.

   For example, you use the `PostLoginProcessor` Webflow component to retrieve user information on a specific JSP. When a customer accesses the JSP, `PostLoginProcessor` invokes the `ProfileFactory`.

   See “Initializing the Customer Profile” in the *Guide to Developing Campaign Infrastructure*.

2. The JSP tag, API, or WebLogic Portal service calls the `getProperty("propertySet", "propertyName")` method of the `ProfileWrapper`.

3. `ProfileWrapper` calls the `getProfileManager` method of the `UserManager`.

   `UserManager` is the synchronization point between user profile support and the WebLogic Server security realm. The `UserManager` handles user creation and deletion by delegating to various `ProfileManagers`. The `UserManager` has a registry of `ProfileTypes` that map a `ProfileType` name to a particular deployment of a `ProfileManager`. For more information about the Java objects, refer to the *Javadoc* API documentation.

   The `getProfileManager` method does the following:

   a. Retrieves the user's type from the WebLogic Portal RDBMS repository.

      You can specify a user’s type when you create a user, (the `createUser()` method takes a user type as a parameter), and WebLogic Portal stores the type designation in its RDBMS repository. After you create a user, you cannot change the type.

b. Looks in the `UserManager` deployment descriptor to find a `ProfileManager` that corresponds to the user type. If the user does not have a type, `UserManager` returns the default `ProfileManager`.

c. Returns a `ProfileManager`.

`ProfileManager` is a stateless session bean used to access profile values. It coordinates property inheritance and the mapping of properties to different data sources.

`UserManager` and `ProfileManager` are both configurable through deployment descriptors: you should never have to modify/extend them by writing code.

4. `ProfileWrapper` calls the `getProperty("propertySet", "propertyName", "username")` method of the `ProfileManager` from the previous step.

   The `getProperty` method does the following:

   a. Looks in the `ProfileManager` deployment descriptor for an `EntityPropertyManager` mapping that matches the propertySet/propertyName. If there is no such mapping, it looks for a mapping that matches the propertySet. If there is no such mapping, it finds the default `EntityPropertyManager`.

   b. Returns an `EntityPropertyManager`.

   `EntityPropertyManager` is responsible for reading/writing properties to/from a datasource.

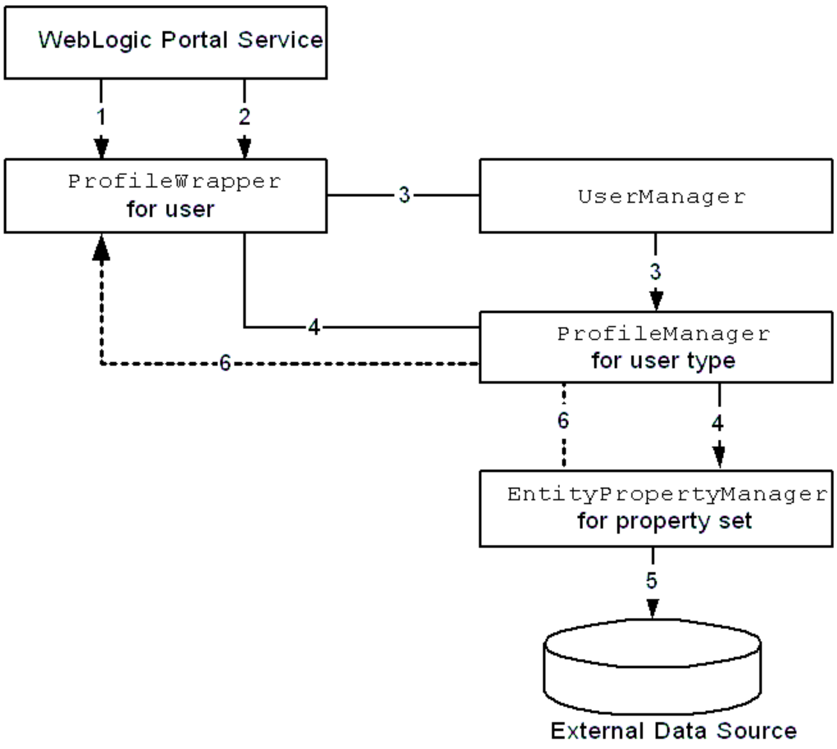
5. The `EntityPropertyManager` retrieves the property from its data source.

6. `EntityPropertyManager` returns the value to `ProfileManager`, which returns it to `ProfileWrapper`, which returns it to the client. If the `EntityPropertyManager` returns null, the `ProfileManager` invokes successor searches (described in “Property Inheritance” on page 8-2).

7. If the successor searches return null, the `PropertySetManager` is used to get the default value for the property for the given property set. The `PropertySetManager` returns the value to the `ProfileManager`, which returns it to the `ProfileWrapper`, which returns it to the client.

Note: You should use the WebLogic Portal Administration Tools to create a property set definition containing your UUP properties. This UUP property set can be used to provide default property values—default properties that are also required by the Portal personalization tools. For example, the tool for creating customer segments needs to know which properties (and which types of properties) are available for defining a customer segment.

Figure 8-26 How WebLogic Portal Retrieves Data From an External Source



Note: The retrieval of default property values from the property set metadata in the portal schema is not shown in this figure.

A Unified User Profile is not a custom WebLogic security realm. For information on Unified User Profile security, see "Unified User Profile Security" on page 8-36.

Configuring WebLogic Portal To Retrieve User Data from External Sources

To retrieve user data from external sources, complete the following tasks:

1. Create an EntityPropertyManager EJB to Represent External Data
2. Deploy a ProfileManager That Can Use the New EntityPropertyManager

Create an EntityPropertyManager EJB to Represent External Data

To incorporate data from an external source, you must first create a stateless session bean that implements the methods of the `com.bea.p13n.property.EntityPropertyManager` remote interface. `EntityPropertyManager` is the remote interface for a session bean that handles the persistence of property data and the creation and deletion of profile records if appropriate.

In addition, the stateless session bean should include a home interface and an implementation class. For example:

```
MyEntityPropertyManager
    extends com.bea.p13n.property.EntityPropertyManager
```

```
MyEntityPropertyManagerHome
    extends com.bea.p13n.property.EJBHome
```

Your implementation class can extend the `EntityPropertyManagerImpl` class. However the only requirement is that your implementation class is a valid implementation of the `MyEntityPropertyManager` remote interface. For example:

```
MyEntityPropertyManagerImpl extends
com.bea.p13n.property.internal.EntityPropertyManagerImpl
```

or

```
MyEntityPropertyManagerImpl extends
javax.ejb.SessionBean
```

We recommend the following guidelines for your new EJB:

- Your custom `EntityPropertyManager` is not a default `EntityPropertyManager`. A default `EntityPropertyManager` is used to get/set/remove properties in the Portal schema. Your custom `EntityPropertyManager` does not have to support the following methods. It can throw `java.lang.UsupportedOperationException` instead:

`getDynamicProperties`

`getEntityNames`

`getHomeName`

`getPropertyLocator`

`getUniqueId`

- If you want to be able to use the Portal framework and tools to create and remove users in your external data store then you must support the `createUniqueId()` and `removeEntity()` methods. However, your custom `EntityPropertyManager` is not the default `EntityPropertyManager` so your createUniqueId method does not have to return a unique number. It must create the user entity in your external data store and then it can return any number, such as -1.

- The following recommendations apply to the EntityPropertyManager methods that you must support:

  `getProperty()` – Use caching. You should support the `getProperties` method to retrieve all properties for a user at once, caching them at the same time. Your `getProperty` method should use `getProperties`.

  `setProperty()` – Use caching.

  `removeProperties()`, `removeProperty()` – After these methods are called then a call to getProperty should return null for the property. Remove properties from the cache too.

- Your implementations of the `getProperty()`, `setProperty()`, `removeProperty()`, and `removeProperties()` methods must include any logic necessary to connect to the external system.

- If you want to cache property data, the methods must be able to cache profile data appropriately for that system. (See the `com.bea.p13n.cache` package.)

- If the external system contains read-only data, any methods that modify profile data must throw a `java.lang.UnsupportedOperationException`. Additionally, if the external data source contains users that are created and deleted by something other than WebLogic Personalization Server, your `createUniqueId` and `removeEntity` methods can simply throw an `UnsupportedOperationException`.

- To avoid class loader dependency issues, make sure that your EJB resides in its own package.
- For ease of maintenance, place the compiled classes of your custom `EntityPropertyManager` bean in your own JAR file (instead of modifying an existing WebLogic Portal JAR file).

Before you deploy your JAR file, follow the steps in the next section.

Deploy a ProfileManager That Can Use the New EntityPropertyManager

A “user type” is a mapping of a `ProfileType` name to a particular `ProfileManager`. This mapping is done in the `UserManager` EJB deployment descriptor.

To access the data in your new `EntityPropertyManager` EJB, you must do **one** of the following:

- In most cases you will be able to use the default deployment of `ProfileManager`, the `UserProfileManager`. You will modify the `UserProfileManager`’s deployment descriptor to map a property set and/or properties to your custom `EntityPropertyManager`. If you support the `createUniqueId()` and `removeEntity()` methods in your custom `EntityPropertyManager`, you can use WebLogic Portal Administration Tools to create a user of type “User” with a profile that can get/set properties using your custom `EntityPropertyManager`. For more information, refer to “Modifying the Existing ProfileManager Deployment Configuration” on page 8-43.
- In some cases you may want to deploy a newly configured `ProfileManager` that will be used instead of the `UserProfileManager`. This new `ProfileManager` is mapped to a `ProfileType` in the deployment descriptor for the `UserManager`. If you support the `createUniqueId()` and `removeEntity()` methods in your custom `EntityPropertyManager`, you can use the WebLogic Portal Administration Tools (or API) to create a user of type “MyUser” (or whatever you name it) that can get/set properties using the customized deployment of the `ProfileManager` that is, in turn, configured to use your custom `EntityPropertyManager`. For more information, refer to “Configuring and Deploying a New ProfileManager” on page 8-47.

ProfileManager is a stateless session bean that manages access to the profile values that the EntityPropertyManager EJB retrieves. It relies on a set of mapping statements in its deployment descriptor to find data. For example, the ProfileManager receives a request for the value of the DateOfBirth property, which is located in the PersonalData property set. ProfileManager uses the mapping statements in its deployment descriptor to determine which EntityPropertyManager EJB contains the data.

Modifying the Existing ProfileManager Deployment Configuration

If you use the existing UserProfileManager deployment to manage your user profiles, perform the following steps to modify the deployment configuration.

Under most circumstances, this is the method you should use to deploy your UUP. An example of this method is the deployment of the custom EntityPropertyManager for LDAP property retrieval, the LdapPropertyManager. The classes for the LdapPropertyManager are packaged in ldapprofile.jar. (See “Accessing Properties from an LDAP Server” on page 8-34.) The deployment descriptor for the UserProfileManager EJB is configured to map the “ldap” property set to the LdapPropertyManager. The UserProfileManager is deployed in usermgmt.jar.

1. Back up the usermgmt.jar file in your enterprise application root directory.
2. From usermgmt.jar, extract META-INF/ejb-jar.xml and open it for editing.
3. In ejb-jar.xml, find the following element:

```
<!-- map all properties in property set ldap to ldap server -->
<env-entry>
  <env-entry-name>PropertyMapping/ldap</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>LdapPropertyManager</env-entry-value>
</env-entry>
```

   and add an <env-entry> element after this to map a property set to your custom EntityPropertyManager like this:

```
<!-- map all properties in UUPExample property set to
MyEntityPropertyManager -->
<env-entry>
  <env-entry-name>PropertyMapping/UUPExample</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
```

4. In `ejb-jar.xml`, find the following element:

```
<!-- an ldap property manager -->
<ejb-ref>
  <ejb-ref-name>ejb/LdapPropertyManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.bea.p13n.property.EntityPropertyManagerHome</home>
  <remote>com.bea.p13n.property.EntityPropertyManager</remote>
</ejb-ref>
```

and add a `<ejb-ref>` element after this to map a reference to an EJB that matches the name from the previous step with `ejb/` prepended like this:

```
<!-- an example property manager -->
<ejb-ref>
  <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>examples.usermgmt.MyEntityPropertyManagerHome</home>
  <remote>examples.usermgmt.MyEntityPropertyManager</remote>
</ejb-ref>
```

The home and remote class names match the classes from your EJB JAR file for your custom `EntityPropertyManager`.

5. If your `EntityPropertyManager` implementation handles creating and removing profile records, you must also add Creator and Remover entries. For example:

```
<env-entry>
  <env-entry-name>Creator/Creator1</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
<env-entry>
```

```
  <env-entry-name>Remover/Remover1</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
```

This instructs the `UserProfileManager` to call your custom `EntityPropertyManager` when creating or deleting user profile records. The names “Creator1” and “Remover1” are arbitrary. All Creators and Removers will be iterated through when the `UserProfileManager` creates or removes a user profile. The value for the Creator and Remover matches the `ejb-ref-name` for your custom `EntityPropertyManager` without the `ejb/` prefix.

6. From usermgmt.jar, extract `META-INF/weblogic-ejb-jar.xml` and open it for editing.

7. In `weblogic-ejb-jar.xml`, find the following elements:

```
<weblogic-enterprise-bean>
  <ejb-name>UserProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
      <jndi-name>${APPNAME}.BEA_personalization.
      EntityPropertyManager</jndi-name>
    </ejb-reference-description>
```

and add an ejb-reference-description to map the `ejb-ref` for your custom `EntityPropertyManager` to the JNDI name. This JNDI name must match the name you assigned in `weblogic-ejb-jar.xml` in the JAR file for your customer `EntityPropertyManager`. It should look like this:

```
<weblogic-enterprise-bean>
  <ejb-name>UserProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
      <jndi-name>${APPNAME}.BEA_personalization.
      EntityPropertyManager</jndi-name>
    </ejb-reference-description>
    <ejb-reference-description>

      <ejb-ref-name>ejb/MyEntityPropertyManager
      </ejb-ref-name>

      <jndi-name>${APPNAME}.BEA_personalization.
      MyEntityPropertyManager</jndi-name>

    </ejb-reference-description>
```

Note the `${APPNAME}` string substitution variable. The WebLogic EJB container automatically substitutes the enterprise application name to scope the JNDI name to the application.

8. Update usermgmt.jar for your new deployment descriptors. You can use the `jar uf` command to update the modified `META-INF/` deployment descriptors.

9. Edit `META-INF/application.xml` for your enterprise application to add an entry for your custom `EntityPropertyManager` EJB module like this:

   ```
   <module>

     <ejb>UUPExample.jar</ejb>

   </module>
   ```

10. If you are using an application-wide cache, you can manage it from the WebLogic Server Administration Console if you add a `<Cache>` tag for your cache to the `META-INF/application-config.xml` deployment descriptor for your enterprise application like this:

    ```
    <Cache

      Name="UUPExampleCache"

      TimeToLive="60000"

    />
    ```

    For information on using `com.bea.p13n.cache.Cache`, see “Using Caches” in the *Performance Tuning Guide*.

11. Verify the modified `usermgmt.jar` and your custom `EntityPropertyManager` EJB JAR archive are in the root directory of your enterprise application and start your server.

12. Use the WebLogic Server Administration Console to verify your EJB module is deployed to the enterprise application and then use the console to add your server as a target for the EJB module. You need to select a target to have your domain's `config.xml` file updated to deploy your EJB module to the server.

13. Use the E-Business Control Center to create a User Profile (property set) that matches the name of the property set that you mapped to your custom `EntityPropertyManager` in `ejb-jar.xml` for the `UserProfileManager` (in `usermgmt.jar`). You could also map specific property names in a property set to your custom `EntityPropertyManager`.

    Note: Be sure to synchronize the new data to your server after the property set is created.

14. Your new Unified User Profile type is ready to use. You can use the WebLogic Portal Administration Tools to create a user of type "User," and it will use your UUP implementation when the "UUPExample" property set is being modified. When you call `createUser("bob", "password")` or `createUser("bob", "password", null)` on the `UserManager`, several things will happen:

    - A user named "bob" is created in the security realm.
    - A WebLogic Portal Server profile record is created for Bob in the WebLogic Portal RDBMS repository.
    - If you set up the Creator mapping, the `UserManager` will call the default `ProfileManager` deployment (`UserProfileManager`) which will call your custom `EntityPropertyManager` to create a record for Bob in your data source.
    - Retrieving Bob's profile will use the default `ProfileManager` deployment (`UserProfileManager`), and when you request a property belonging to the "UUPExample" property set, the request will be routed to your custom `EntityPropertyManager` implementation.

Configuring and Deploying a New ProfileManager

If you are going to deploy a newly configured `ProfileManager` instead of using the default `ProfileManager` (`UserProfileManager`) to manage your user profiles, perform the following steps to modify the deployment configuration. In most cases, you will not have to use this method of deployment. Use this method only if you need to support multiple types of users that require different `ProfileManager` deployments—deployments that allow a property set to be mapped to different custom `EntityPropertyManagers` based on `ProfileType`.

An example of this method is the deployment of the custom `CustomerProfileManager` in `customer.jar`. The `CustomerProfileManager` is configured to use the custom `EntityPropertyManager` (`CustomerPropertyManager`) for properties in the "CustomerProperties" property set. The `UserManager` EJB in `usermgmt.jar` is configured to map the "WLCS_Customer" ProfileType to the custom deployment of the `ProfileManager`, `CustomerProfileManager`.

1. Back up the `usermgmt.jar` file in your enterprise application root directory.
2. From `usermgmt.jar`, extract `META-INF/ejb-jar.xml` and open it for editing.
3. In `ejb-jar.xml`, copy the entire `<session>` tag for the `UserProfileManager` and configure it to use your custom implementation class for your new deployment of `ProfileManager`.

   In addition, you could extend the `UserProfileManager home` and `remote` interfaces with your own interfaces if you want to repackage them to correspond to your packaging (for example., `examples.usermgmt.MyProfileManagerHome`, `examples.usermgmt.MyProfileManager`). However, it is sufficient to replace the bean implementation class:

```
<session>

  <ejb-name>MyProfileManager</ejb-name>

  <home>com.bea.p13n.usermgmt.profile.ProfileManagerHome
  </home>

  <remote>com.bea.p13n.usermgmt.profile.ProfileManager
  </remote>

  <ejb-class>examples.usermgmt.MyProfileManagerImpl
  </ejb-class>

  <session-type>Stateless</session-type>

  <transaction-type>Container</transaction-type>


  <!-- map all properties in UUPExample property
  set to MyEntityPropertyManager -->

  <env-entry>

    <env-entry-name>PropertyMapping/UUPExample</env-entry-name>

    <env-entry-type>java.lang.String</env-entry-type>

    <env-entry-value>MyEntityPropertyManager</env-entry-value>

  </env-entry>
```

```
<!-- register a Creator for MyEntityPropertyManager -->
<env-entry>
  <env-entry-name>Creator/Creator1</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
<!-- register a Remover for MyEntityPropertyManager -->
<env-entry>
  <env-entry-name>Remover/Remover1</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>

<!-- the default property manager -->
<ejb-ref>
  <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.bea.p13n.property.EntityPropertyManagerHome
  </home>
  <remote>com.bea.p13n.property.EntityPropertyManager
  </remote>
</ejb-ref>
<!-- my custom property manager -->
<ejb-ref>
  <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>examples.usermgmt.EntityPropertyManagerHome</home>
  <remote>examples.usermgmt.EntityPropertyManager</remote>
</ejb-ref>
<!-- property set manager -->
<ejb-ref>
```

```
    <ejb-ref-name>ejb/PropertySetManager</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.property.PropertySetManagerHome</home>
    <remote>com.bea.p13n.property.PropertySetManager</remote>
  </ejb-ref>
  <!-- group profile manager, which is the user's successor -->
  <ejb-ref>
    <ejb-ref-name>ejb/GroupProfileManager</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.usermgmt.profile.ProfileManagerHome
    </home>
    <remote>com.bea.p13n.usermgmt.profile.ProfileManager
    </remote>
    <ejb-link>GroupProfileManager</ejb-link>
  </ejb-ref>


  <security-role-ref>
    <description>This ref declares the Administrative role
    for this bean</description>
    <role-name>SystemAdminRole</role-name>
    <role-link>SystemAdminRole</role-link>
  </security-role-ref>
  <security-role-ref>
    <description>This ref declares the Administrative role
    for this bean</description>
    <role-name>DelegatedAdminRole</role-name>
    <role-link>DelegatedAdminRole</role-link>
  </security-role-ref>
</session>
```

You must create an `<env-entry>` element to map a property set to your custom `EntityPropertyManager`. You must also create a `<ejb-ref>` element to map a reference to an EJB that matches the name from the `PropertyMapping` with `ejb/` prepended. The `home` and `remote` class names for your custom

`EntityPropertyManager` match the classes from your EJB JAR file for your custom `EntityPropertyManager`.

Also, if your `EntityPropertyManager` implementation handles creating and removing profile records, you must also add Creator and Remover entries. This instructs your new `ProfileManager` to call your custom `EntityPropertyManager` when creating or deleting user profile records. The name suffixes for the Creator and Remover, "Creator1" and "Remover1", are arbitrary. All Creators and Removers will be iterated through when your `ProfileManager` creates or removes a user profile. The value for the Creator and Remover matches the `<ejb-ref-name>` for your custom `EntityPropertyManager` without the `ejb/` prefix.

4. In `ejb-jar.xml`, you must add an `<ejb-ref>` to the `UserManager` EJB section to map your `ProfileType` to your new deployment of the ProfileManager like this:

```
<ejb-ref>
  <ejb-ref-name>ejb/ProfileType/UUPExampleUser</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.bea.p13n.usermgmt.profile.ProfileManagerHome</home>
  <remote>com.bea.p13n.usermgmt.profile.ProfileManager</remote>
</ejb-ref>
```

   The `<ejb-ref-name>` must start with `ejb/ProfileType/` and must end with the name that you want to use as the profile type as an argument in the `createUser()` method of `UserManager`.

5. From `usermgmt.jar`, extract `META-INF/weblogic-ejb-jar.xml` and open it for editing.

6. In `weblogic-ejb-jar.xml`, copy the weblogic-enterprise-bean tag for the `UserProfileManager` and configure it for your new `ProfileManager` deployment:

```
<weblogic-enterprise-bean>
  <ejb-name>MyProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
```

```
          <jndi-name>${APPNAME}.BEA_personalization.
          EntityPropertyManager</jndi-name>

        </ejb-reference-description>

        <ejb-reference-description>

          <ejb-ref-name>ejb/PropertySetManager</ejb-ref-name>

          <jndi-name>${APPNAME}.BEA_personalization.
          PropertySetManager</jndi-name>

        </ejb-reference-description>

        <ejb-reference-description>

          <ejb-ref-name>ejb/MyEntityPropertyManager
          </ejb-ref-name>

          <jndi-name>${APPNAME}.BEA_personalization.
          MyEnitityPropertyManager</jndi-name>

        </ejb-reference-description>

    </reference-descriptor>

    <jndi-name>${APPNAME}.BEA_personalization.
    MyProfileManager</jndi-name>

  </weblogic-enterprise-bean>
```

You must create an `<ejb-reference-description>` to map the `<ejb-ref>` for your custom `EntityPropertyManager` to the JNDI name. This JNDI name must match the name you assigned in `weblogic-ejb-jar.xml` in the JAR file for your custom `EntityPropertyManager`.

Note the `${APPNAME}` string substitution variable. The WebLogic Server EJB container automatically substitutes the enterprise application name to scope the JNDI name to the application.

7. In `weblogic-ejb-jar.xml`, copy the `<transaction-isolation>` tag for the `UserProfileManager` and configure it for your new `ProfileManager` deployment:

```
<transaction-isolation>

  <isolation-level>TRANSACTION_READ_COMMITTED
  </isolation-level>

  <method>

    <ejb-name>MyProfileManager</ejb-name>

    <method-name>*</method-name>
  </method>
</transaction-isolation>
```

8. Create a temporary `usermgmt.jar` for your new deployment descriptors and your new `ProfileManager` bean implementation class (and `remote` and `home` interfaces if you are replacing those). This temporary EJB JAR archive should not have any container classes in it. Run `ejbc` to generate new container classes.
9. Edit `META-INF/application.xml` for your enterprise application to add an entry for your custom `EntityPropertyManager` EJB module like this:

```
<module>
  <ejb>UUPExample.jar</ejb>
</module>
```

10. If you are using an application-wide cache, you can manage it from the WebLogic Server Administration Console if you add a `<Cache>` tag for your cache to the `META-INF/application-config.xml` deployment descriptor for your enterprise application like this:

```
<Cache
  Name="UUPExampleCache"
  TimeToLive="60000"
/>
```

    For information on using `com.bea.p13n.cache.Cache`, see "Using Caches" in the *Performance Tuning Guide*.

11. Verify the modified `usermgmt.jar` and your custom `EntityPropertyManager` EJB JAR archive are in the root directory of your enterprise application and start your server.
12. Use the WebLogic Server Administration Console to verify your EJB module is deployed to the enterprise application, then use the WebLogic Server Administration Console to add your server as a target for the EJB module. You must select a target to have your domain's `config.xml` file updated to deploy your EJB module to the server.
13. Use the E-Business Control Center to create a User Profile (property set) that matches the name of the property set that you mapped to your custom `EntityPropertyManager` in `ejb-jar.xml` for the `UserProfileManager` (in `usermgmt.jar`). You could also map specific property names in a property set to your custom `EntityPropertyManager`.

Note: Be sure to synchronize the new data to your server after the property set is created.

14. Your new Unified User Profile type is ready to use. You can use the WebLogic Portal Administration Tools to create a user of type "UUPExampleUser," and it will use your UUP implementation when the "UUPExample" property set is being modified. That is because you mapped the `ProfileType` using an `<ejb-ref>` in your `UserManager` deployment descriptor, `ejb/ProfileType/UUPExampleUser`. Now, when you call `createUser("bob", "password", "UUPExampleUser")` on the `UserManager`, several things will happen:

    - A user named "bob" is created in the security realm.
    - A WebLogic Portal Server profile record is created for Bob in the WebLogic Portal RDBMS repository.
    - If you set up the Creator mapping, the `UserManager` will call your new `ProfileManager` deployment, which will call your custom `EntityPropertyManager` to create a record for Bob in your data source.
    - Retrieving Bob's profile will use your new `ProfileManager` deployment, and when you request a property belonging to the "UUPExample" property set, the request will be routed to your custom `EntityPropertyManager` implementation.

CHAPTER

9 Creating and Managing Content

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

This topic includes the following sections:

- Configuring DocumentManager MBeans
- Setting Up Document Connection Pools
- Setting up WebLogic Connection Pools
- Web Application Configuration

- Using the BulkLoader to Load File-based Content
  - Command-Line Usage
  - How the BulkLoader Finds Files
  - How the BulkLoader Finds Metadata Properties
  - Cleaning Up the Database
  - Loading Internationalized Documents
  - Generating Schema Files

What Is the Content Manager?

The Content Manager run-time subsystem provides access to content through tags and EJBs. The Content Management tags allow a JSP developer to receive an enumeration of Content objects by querying the content database directly using a search expression syntax. The Content Manager component works alongside the other components to deliver personalized content, but does not have a GUI-based tool for edit-time customization.

Choosing a Content Engine

The content engine behind the `ContentManager` can be set up to be the reference implementation that BEA provides out-of-the-box, or a third-party content engine.

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

For customers who have larger amounts of content and want more control over the publishing and tagging of content, BEA partners with third-party vendors to add flexibility to the WebLogic Personalization Server. Third-party content engines provide robust, content-creation management solutions while the Content Manager personalizes and serves the content to the end user.

Running Queries Against the Content Repository

The Content Management component supports querying that returns content from a content repository using several methods:

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

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

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

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

Note: See “Querying the Content” on page 9-10 for more information about queries.

Methods for Retrieving and Displaying Documents

WebLogic Personalization Server provides several methods for retrieving documents from a content management system and displaying them on your Web site.

A *document* is a graphic, a segment of HTML or plain text, or a file that must be viewed with a plug-in. We recommend that you store most of your web site's dynamic documents in a content management system because it offers an effective way to store and manage information.

Note: Campaigns cannot be used with anonymous users. Campaigns require a user ID that has two characteristics: the ID must be associated with a user profile, and that user profile must be saved (persisted). However, the anoymous profile for a user who is not logged in is a runtime profile (not saved), and not associated with a user ID.

Personalization features such as <pz:div> and <pz:contentSelector> JSP tags do work for anonymous users. This is because these features can use a runtime profile without a user ID,

Table 9-16 compares the methods of content retrieval that WebLogic Personalization Server provides.

Table 9-16 Methods for Retrieving and Displaying Documents

Use This Method...	When You Want To...
Content selectors and `<pz:contentSelector>` tags	■ Use a centrally maintained infrastructure for matching Web site content with events, customer profiles, or customer segments. Business Engineers develop the infrastructure, then Business Analysts use the E-Business Control Center to define and modify conditions under which content selectors query the content management system for documents. ■ Retrieve any type of content that your content management system contains (and that a browser supports). ■ Display each document that a content-management query returns. Content selectors store the results of a query in an array. You can use other JSP tags to display some or all of the documents that are in the array. ■ Place the results of the query in a cache. Content selectors require you to determine the MIME-type of the documents and to supply the appropriate HTML that the browser requires to display them.
`<pz:contentQuery>` tag	■ Run a static, narrowly-defined query to display a document only in a specific JSP. You must modify each occurrence of this tag if you want to modify its query. If you want this tag to display contents for specific customers or in response to an event, you must surround it with additional tags that evaluate the display condition.

Table 9-16 Methods for Retrieving and Displaying Documents (Continued)

Use This Method...	When You Want To...
Ad placeholders and `<ph:placeholder>` tags	■ Use a centrally maintained infrastructure for matching advertising documents with events, customer profiles, or customer segments. Business Engineers develop the infrastructure, then Business Analysts use the E-Business Control Center to define and modify the queries that each placeholder can run. ■ Run queries as part of a scenario action in a campaign (available only with Campaign services). ■ Use a single infrastructure to support multiple, concurrent advertising agenda. Ad placeholders use an Ad Conflict Resolver to select a single query if multiple agenda request to run multiple queries in the same location at the same time. ■ Automatically generate the HTML that the browser requires to display the query results. Without customization, ad placeholders support only HTML, image, and Shockwave documents.
`<ad:adTarget>` tag	■ Make sure that a specific ad query runs in a specific location. ■ Automatically generate the HTML that the browser requires to display the query results. The `<ad:adTarget>` tag is not part of the infrastructure for supporting multiple advertising agenda. It cannot run a query as part of a scenario action. You must modify each occurrence of this tag if you want to modify its query. If you want this tag to display contents for specific customers or in response to an event, you must surround it with additional tags that evaluate the display condition. Without customization, the `<ad:adTarget>` tag supports only HTML, image, and Shockwave documents.

`<cm:printDoc>` tag,
or any of these methods:
`Document.getContent()`
`DocumentManager.getContentBlock()`
	■ Use the content management system's document ID to include non-personalized content in a HTML-based page.
	The tag does not generate HTML to support the content it retrieves; it inserts the document into the JSP page exactly as it is stored in the content management system. Business Engineers must modify each occurrence of this tag if you want to change the document that it retrieves.

Table 9-16 Methods for Retrieving and Displaying Documents (Continued)

Use This Method...	When You Want To...
`<cm:getProperty>` tag, or the method `Content.getProperty()`	■ Retrieves the value of the specified content metadata property into a variable specified by `resultId`. If `resultId` is not specified, the value will be inlined into the page, similar to the `<cm:printProperty>` tag. This tag operates on any ConfigurableEntity, not just the Content object. However, it does not support ConfigurableEntity successors.
`<cm:printProperty>` tag or the method `Content.getProperty()`	■ Display the value of a document attribute as a string. You can use this tag to display the value of any content object's attribute, not just document-type objects in a content management system.
`<cm:select>` tag, or the method `ContentManager.getContent()`	■ Use a query to include non-personalized content in a HTML-based page. ■ Place the results of the query in a cache. The tag does not generate HTML to support the content it retrieves; it inserts the document into the JSP page exactly as it is stored in the content management system. Business Engineers must modify each occurrence of this tag if you want to change the document that it retrieves.
`<cm:selectById>` tag, or the method `ContentManager.getContent()`	■ Use the content management system's document ID to include non-personalized content in a HTML-based page. ■ Place the document in a cache. The tag does not generate HTML to support the content it retrieves; it inserts the document into the JSP page exactly as it is stored in the content management system. Business Engineers must modify each occurrence of this tag if you want to change the document that it retrieves.

Differences Between Content Management and Document Management

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

`Documents` are a specialized type of `Content` that provide two methods for retrieval: a metadata-searching mechanism and retrieval of the pure bytes of the document's file. `Documents` should include additional explicit metadata properties related to the file and its versioning, including its size, name, path, author, and version. A document management system usually provides document-based services for documents that reside in the system’s repository.

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

Querying the Content

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

- JSP tags
- ContentHelper
- ContentManager
- ContentHome

For more information, see the *Javadoc* API documentation

Structuring a Query

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

```
attribute_name comparison_operator literal_value
```

> **Note:** For more information about the query syntax, see the *Javadoc* API documentation for `com.bea.p13n.content.expression.ExpressionHelper`.

Several constraints apply to queries constructed using this syntax:

- String literals must be enclosed in single quotes.
  - `‘WebLogic Server’`
  - `‘football’`
- Date literals can be created via a simplistic `toDate` method that takes one or two `String` arguments (enclosed in single quotes). The first, if two arguments are supplied, is the `SimpleDateFormat` format string; the second argument is the

date string. If only one argument is supplied, it should include the date string in 'MM/dd/yyyy HH:mm:ss z' format.

- `toDate('EE dd MMM yyyy HH:mm:ss z', 'Thr 08 Nov 2001 16:56:00 MDT')`
- `toDate('02/23/2005 13:57:43 MST')`

■ Use the `toProperty` method to compare properties whose names include spaces or other special characters. In general, use `toProperty` when the property name does not comply with the Java variable-naming convention that uses alphanumeric characters.

- `toProperty ('My Property') = 'Content'`

■ To include a scope into the property name, use either `scope.propertyName` or the `toProperty` method with two arguments.

- `toProperty ('myScope', 'myProperty')`

Note: The reference document management system ignores property scopes.

■ Use \ along with the appropriate character(s) to create an escape sequence that includes special characters in string literals.

- `toProperty ('My Property\'s Contents') = 'Content'`

■ Additionally, use Java-style Unicode escape sequences to embed non-ASCII characters in string literals.

- Description like `'*\u65e5\u672c\u8a9e*'`

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

Note: Use `ExpressionHelper.toStringLiteral` to convert an arbitrary string to a fully quoted and escaped string literal which can be put in a query.

■ The *`now`* keyword—only used on the literal value side of the expression—refers to the current date and time.

■ Boolean literals are either `true` or `false`.

■ Numeric literals consist of the numbers themselves without any text decoration (like quotation marks). The system supports scientific notation in the forms (for example, *`1.24e4`* and *`1.24E-4`*).

- An exclamation mark (!) can be placed at an opening parenthesis to negate an expression.
  - `!(keywords contains ‘football’) || (size >= 256)`
- The Boolean `and` operator is represented by the literal `&&`.
  - `author == ‘james’ && age < 55`
- The Boolean `or` operator is represented by the literal `||`.
  - `creationDate > now || expireDate < now`

The following examples illustrate full expressions:

Example 1:

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

Example 2:

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

Using Comparison Operators to Construct Queries

To support advanced searching, the system allows construction of nested Boolean queries incorporating comparison operators. Table 9-17 summarizes the comparison operators available for each metadata type. (For more information about the native types supported in WebLogic Personalization Server, see "Support for Native Types" on page 1-10.)

Table 9-17 Comparison Operators Available for Each Metadata Type

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

Table 9-17 Comparison Operators Available for Each Metadata Type (Continued)

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

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

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

Constructing Queries Using Java

To construct queries using Java syntax instead of using the query language supplied with the Content Management component, see the *Javadoc* API documentation for `com.bea.p13n.content.expression.ExpressionHelper`.

The `ContentManager` session bean is the primary interface to the functionality of the Content Management component. Using a `ContentManager` instance, content is returned based on a `com.bea.p13n.content.expression.Search` object with an embedded `com.bea.p13n.expression.Expression`, which represents the expression tree.

In the expression tree, the following caveats apply for it to be valid for the ContentManager:

- Each branch node can only be of the type:

```
com.bea.p13n.expression.operator.logical.LogicalAnd,
com.bea.p13n.expression.operator.logical.LogicalOr,
com.bea.p13n.expression.operator.logical.LogicalMulitAnd, or
com.bea.p13n.expression.operator.logical.LogicalMultiOr.
```

  Any other branch node type is invalid.

- Each leaf node can only be of the type:

```
com.bea.p13n.expression.operator.comparative.Equals,
com.bea.p13n.expression.operator.comparative.GreaterOrEquals,
com.bea.p13n.expression.operator.comparative.GreaterThan,
com.bea.p13n.expression.operator.comparative.LessOrEquals,
```

`com.bea.p13n.expression.operator.comparative.LessThan`,
`com.bea.p13n.expression.operator.comparative.NotEquals`,
`com.bea.p13n.expression.operator.string.StringLike`,
`com.bea.p13n.expression.operator.collection.CollectionContains`, or
`com.bea.p13n.expression.operator.collection.CollectionsContainsAll`

Any other branch node type is invalid.

- Any valid branch or leaf node may be contained in a `com.bea.p13n.expression.operator.logical.LogicalNot` node.
- In each leaf node, the left-hand-side will always be a `com.bea.p13n.content.expression.PropertyRef` node, which must contain `Strings` for `getPropertySet()` and `getPropertyName()`.
- The right-hand-side of these leaf nodes can be a `java.util.Collection`, `Long, Double, String`, or `java.sql.Timestamp`:
  `com.bea.p13n.expression.operator.comparative.Equals`,
  `com.bea.p13n.expression.operator.comparative.NotEquals`,
  `com.bea.p13n.expression.operator.comparative.GreaterOrEquals`,
  `com.bea.p13n.expression.operator.comparative.GreaterThan`,
  `com.bea.p13n.expression.operator.comparative.LessOrEquals`,
  `com.bea.p13n.expression.operator.comparative.LessThan, or`
  `com.bea.p13n.expression.operator.collection.CollectionContains`
- The right-hand-side of `com.bea.p13n.expression.operator.string.StringLike` leaf nodes can be a `String`. Anything else is invalid.
- The right-hand-side of `com.bea.p13n.expression.operator.collection.CollectionsContainsAll` leaf nodes can be a `java.util.Collection`. Anything else is invalid.

JSP Tags

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

- The `<cm:select>` tag uses only the search expression query syntax to select content.

- The `<cm:selectById>` tag retrieves content using the content's unique identifier.
- The `<cm:printProperty>` tag inlines the value of the specified Content metadata property as a string.
- The `<cm:printDoc>` tag inlines the raw bytes of a Document object into the JSP output stream.

See Chapter 13, "Personalization Server JSP Tag Library Reference," for more information on any of these tags.

Using the Document Servlet

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

Table 9-18 Request Parameters Supported by the Document Servlet

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

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

Example 1: Usage in a JSP

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

```
<cm:select sortBy="creationDate ASC, title ASC"

query=" type = ‘News’ && timeOfDay = ‘Evening’ && mimeType like
‘text/*’ "id="newsList"/>

<ul>

<es:forEachInArray array="<%=newsList%>" id="newsItem"
type="com.bea.p13n.content.Content">

     <li><a href="ShowDoc/<cm:printProperty id="newsItem"
     name="identifier" encode="url"/>"><cm:printProperty
     id="newsItem" name="title" encode="html"/></a>

  </es:forEachInArray>

</ul>
```

Example 2: Usage in a JSP

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

```
<cm:select max="5" sortBy="name" id="list"

query=" KeyWords like ‘*birds*’ && mimeType like ‘image/*’ "

contentHome="java:comp/env/ejb/MyDocumentManager"/>

<ul>

<es:forEachInArray array="<%=list%>" id="img"
type=”com.bea.p13n.content.Content”>

   <li><img src="/ShowDoc/<cm:printProperty id="img"
   name="identifier"
   encode="url"/>?contentHome=<es:convertSpecialChars
   string="java:comp/env/ejb/MyDocumentManager"/>">

<es:forEachInArray>

</ul>
```

Configuring the Content Manager

The DocumentManager EJB deployment descriptor handles the EJB portion of the Content Management component configuration. The DocumentManager also needs to be integrated into the PropertySetManager EJB deployment descriptor so that content property sets are exposed to the system. The DocumentManager EJB accesses a document connection pool, which is defined in an application's `META-INF/application-config.xml` file. Optionally, the `DocumentManager` EJB can access a document connection pool configured via the WLS console.

For Web Applications to correctly access the Content Management Component, some additional configuration is required in the Web Application deployment descriptor.

For more information, see the *Deployment Guide*.

Configuring the DocumentManager EJB Deployment Descriptor

The DocumentManager EJB understands the following environment settings in its deployment descriptor:

- `DocumentManagerMBeanName`—specifies the Name of the `DocumentManager` MBean to use to configure this `DocumentManager`.
  - In the application's `appliation-config.xml` file, a `<DocumentManager>` entry must exist with a `Name` attribute equal to the value specified in the deployment descriptor. If this is not specified in the deployment descriptor, it defaults to "default".
- `DocumentConnectionPoolName`—specifies the Name of the `DocumentConnectionPool` MBean for the document connection pool this `DocumentManager` should use.
  - In the application's `application-config.xml` file, a `<DocumentConnectionPool>` entry must exist with a `Name` attribute equal to the value specified in the deployment descriptor.

  - If the `DocumentConnectionPoolName` attribute of the `DocumentManager` MBean this is configured to use is set, then the value in the deployment descriptor is ignored.

- `jdbc/docPool`—specifies the J2EE resource reference to the `javax.sql.DataSource` that this `DocumentManager` should use to access a document connection pool.

  If `jdbc/docPool` is specified in the deployment descriptor, then:

  - `DocumentConnectionPoolName` is ignored, and
  - Any `DocumentConnectionPool` MBeans in the application's `application-config.xml` file are also ignored.

- `PropertyCase`—specifies how the `DocumentManager` modifies the incoming property name.

  - If the `PropertyCase` attribute of the `DocumentManager` MBean this is using is set, the value in the deployment descriptor is ignored.
  - If this is *lower*, all property names are converted to lowercase.
  - If this is *upper*, all property names are converted to uppercase.
  - If this is anything else or not specified, property names are not modified.

  Use *lower* or *upper* depending upon the document connection pool implementation being used. For the document reference implemenation, do not specify the `PropertyCase`.

Configuring the PropertySetManager EJB Deployment Descriptor for Content Management

In the `PropertySetManager` EJB deployment descriptor, add the following environment settings:

- `repository/CONTENT`—specifies the fully qualified class name of `com.bea.p13n.property.PropertySetRepository` implementation. Use `com.bea.p13n.content.PropertySetRepositoryImpl` to integrate with the Content Management component.

- ejb/ContentManagers/[type]— the com.bea.p13n.content.PropertySetRepositoryImpl looks for all environment entries starting with ejb/ContentManagers. It expects these to be J2EE EJB references to ContentManagers (or subclasses).

  To integrate a ContentManager or DocumentManager with the PropertySetManager, add an EJB reference here. For example, ejb/ContentManagers/Document is mapped to the standard DocumentManager.

Alternatively, you can set the JNDIName attribute the DocumentManager MBean to the JNDI Home name of the DocumentManager (see page 22 for a definition of this attribute). The ${APPNAME} construct can be used in the value; it will be replaced by the current J2EE application name. The com.bea.p13n.content.PropertySetRepositoryImpl will automatically pick up those DocumentManagers and the J2EE EJB reference is not required.

Configuring DocumentManager MBeans

The DocumentManager implementation uses DocumentManager MBeans to maintain the configuration for the DocumentManager. A deployed DocumentManager finds which DocumentManager MBean to use from the DocumentManagerMBeanName EJB deployment descriptor setting. That value must correspond to the Name attribute of a DocumentManager MBean in the application.

To configure a DocumentManager MBean, you can modify the application's META-INF/application-config.xml file to add or change the following XML:

```
<DocumentManager
  Name="default"
  DocumentConnectionPoolName="default"
  PropertyCase="none"
  MetadataCaching="true"
  MetadataCacheName="documentMetadataCache"
  UserIdInCacheKey="false"
  ContentCaching="true"
  ContentCacheName="documentContentCache"
  MaxCachedContentSize="32768"
>
</DocumentManager>
```

Attributes of the DocumentManager MBean

The attributes are as follows:

- `DocumentConnectionPoolName`—specifies the name of the `DocumentConnectionPool` MBean the `DocumentManager` should use. (See the section, “Setting Up Document Connection Pools” on page 9-23.)
- `PropertyCase`—specifies how the `DocumentManager` modifies the incoming property name.
  - If this is *lower*, all property names are converted to lowercase.
  - If this is *upper*, all property names are converted to uppercase.
  - If this is anything else or not specified, property names are not modified.

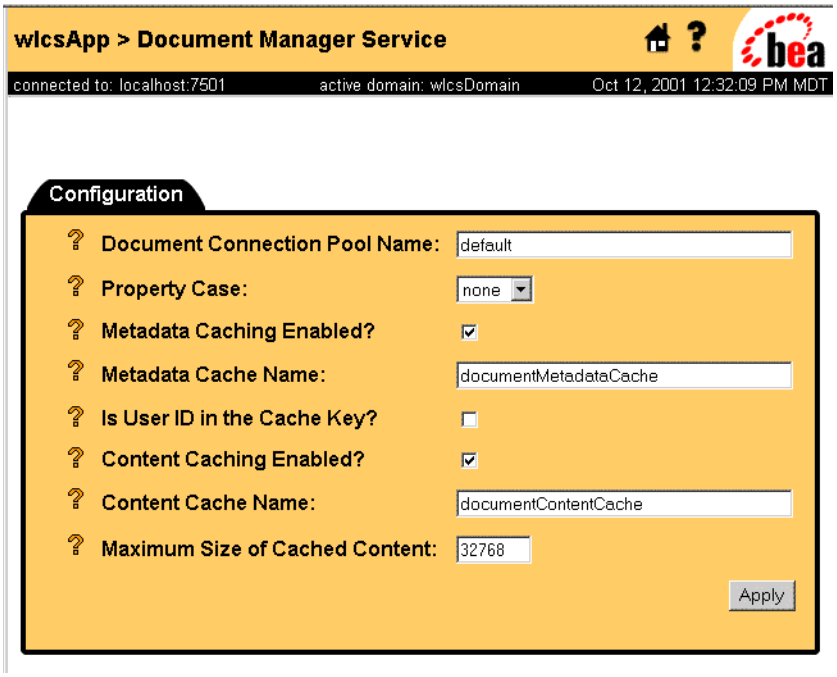
  Use *lower* or *upper* depending upon the document connection pool implementation being used. For the document reference implementation, do not specify the `PropertyCase`.
- `MetadataCaching`—specifies whether the `DocumentManager` should cache Document metadata from searches. Use `true` to have the DocumentManager cache search results in the `com.bea.p13n.cache.Cache` specified by `MetadataCacheName`; otherwise, use `false`. This defaults to `true`.
- `MetadataCacheName`—specifies the name of the `com.bea.p13n.cache.Cache` to use if `MetadataCaching` is set to `true`. This defaults to `documentMetadataCache`.
- `UserIdInCacheKey`—specifies whether the user’s identifier should be used as part of the key when caching document metadata or content. This defaults to `true`. If using the WebLogic Personalization Server reference implementation document management system, set this to `false`.
- `ContentCaching`—specifies whether the `DocumentManager` should cache document content (that is, the bytes of the document). Use `true` to have the `DocumentManager` caches document content bytes; otherwise use `false`. This defaults to `true`.
- `ContentCacheName`—specifies the name of the `com.bea.p13n.cache.Cache` to use if `ContentCaching` is set to `true`. This defaults to `documentContentCache`.

- `MaxCachedContentSize`—specifies the maximum size of a document's content bytes that the `DocumentManager` will cache, if ContentCaching is `true`. This defaults to 32768 (32K).
- `JNDIName` – Specifies the JNDI home name of the `DocumentManager` EJB that is connected to this MBean. The `${APPNAME}` construct can be used in the value; it will be replaced by the current J2EE application name. This is used by the `com.bea.p13n.content.PropertySetRepositoryImpl` to tie document property set information into the `PropertySetManager`.

Editing the DocumentManager MBean in the WebLogic Console

Once a `DocumentManager` MBean has been initially configured in the `application-config.xml` file, it can be edited via the WebLogic Server Administration Console, as show in Figure 9-1 below.

Figure 9-1 Using the WLS Console to Edit the Document Manager MBean



connected to: localhost:7501 active domain: wlcsDomain Oct 12, 2001 12:32:09 PM MDT

Document Connection Pool Name:	default
Property Case:	none
Metadata Caching Enabled?	✓
Metadata Cache Name:	documentMetadataCache
Is User ID in the Cache Key?	☐
Content Caching Enabled?	✓
Content Cache Name:	documentContentCache
Maximum Size of Cached Content:	32768

Setting Up Document Connection Pools

The `DocumentManager` implementation uses connection pools to a specialized JDBC driver to handle searches. A deployed `DocumentManager` finds the document connection pool to use via either the `DocumentConnectionPoolName` attribute of its `DocumentManager` MBean or the `DocumentConnectionPoolName` EJB deployment descriptor setting. That value must correspond to a `DocumentConnectionPool` MBean.

To configure a `DocumentConnectionPool` MBean, modify the application's `META-INF/application-config.xml` file to add or change the following XML:

```
<DocumentConnectionPoolName="default"
  DriverName="com.bea.p13n.content.document.jdbc.Driver"
  URL="jdbc:beasys:docmgmt:com.bea.p13n.content.document.ref.
  RefDocumentProvider"
  Properties="jdbc.dataSource=weblogic.jdbc.pool.commercePool;
  schemaXML=D:/bea/wlportal4.0/dmsBase/doc-schemas;
  docBase=D:/bea/wlportal4.0/dmsBase"
  InitialCapacity="20"
  MaxCapacity="20"
  CapacityIncrement="0"
/>
```

Attributes for the DocumentConnectionPool MBean

The attributes are as follows:

- `DriverName`—specifies the JDBC driver class name to use. This should be set to `com.bea.p13n.content.document.jdbc.Driver`.
- `URL`—specifies the JDBC URL to use.
  - For the WebLogic Personalization Server's reference implementation document management system, this should be set to: `jdbc:beasys:docmgmt:com.bea.p13n.content.document.ref.RefDocumentProvider`.
  - For a different Document Provider, use: `jdbc:beasys:docmgmt:<classname>`, where `<classname>` is the fully qualified class name of the implementation of `com.bea.p13n.content.document.spi.DocumentProvider`.

- `Properties`—This is the semi-colon separated list of `name=value` pairs which will be passed to the `DocumentProvider` specified in the `URL`. See the “Properties” section below for a list of properties the reference implementation understands.
- `InitialCapacity`—specifies the initial number of connections to create when the document connection pool is started.
- `MaxCapcity`—specifies the maximum number of connections this pool will ever create and maintain.
- `CapacityIncrement`—specifies the number of connections the pool will create whenever it needs to create an available connection.
- `LoginTimeout`—specifies the amount of time to wait for a connection: after this time expires, an exception is thrown. Use 0 or less to have the pool not timeout, which is the default.
- `ClassPath`—Specifies the semicolon -separated list of additional directories and JARs the connection pool should use use when attempting to load the `Driver` and the `DocumentProvider` classes. All paths are assumed to be relative to the application directory.

Properties

The WebLogic Personalization Server reference implementation `DocumentProvider` understands the following `Properties`:

- `jdbc.dataSource`—specifies the JNDI name of the `javax.sql.DataSource` to use to get database connections. This datasource should be connected to the database that contains the `DOCUMENT` and `DOCUMENT_METADATA` tables.
- `jdbc.url`—specifies the JDBC URL to connect to.
  If `jdbc.dataSource` is specified, this is ignored.
- `jdbc.driver`—specifies the JDBC driver class to load.
  If `jdbc.dataSource` is specified, this is ignored.
- `jdbc.isPooled`—If `true`, or if `jdbc.url` starts with `jdbc:weblogic:pool` or `jdbc:weblogic:jts`, or if `jdbc.dataSource` is specified, then assumes the connection is pooled and won't cache it. If anything else, assumes the connection is not pooled and will maintain one connection.

- `jdbc.supportsLikeEscapeClause`—specifies whether the underlying database supports the `SQL LIKE ESCAPE` clause. If this is not specified, the connection will be queried.
- `jdbc.docBase`—specifies under which base directory the documents are stored. Assumes all paths coming from the database are relative to this directory.
- `jdbc.schemaXML`—specifies the path to the directory containing XML files following the doc-schemas DTD which contain the property set information. The system will recurse through the directory, loading all files ending in `.xml`.
- `jdbc.isolationLevel`:—configures the transaction isolation level to set on the database connections. This can be one of the following:

  `READ_COMMITTED,`

  `READ_UNCOMMITTED,`

  `SERIALIZABLE,`

  `REPEATABLE_READ,` or

  `NONE.`

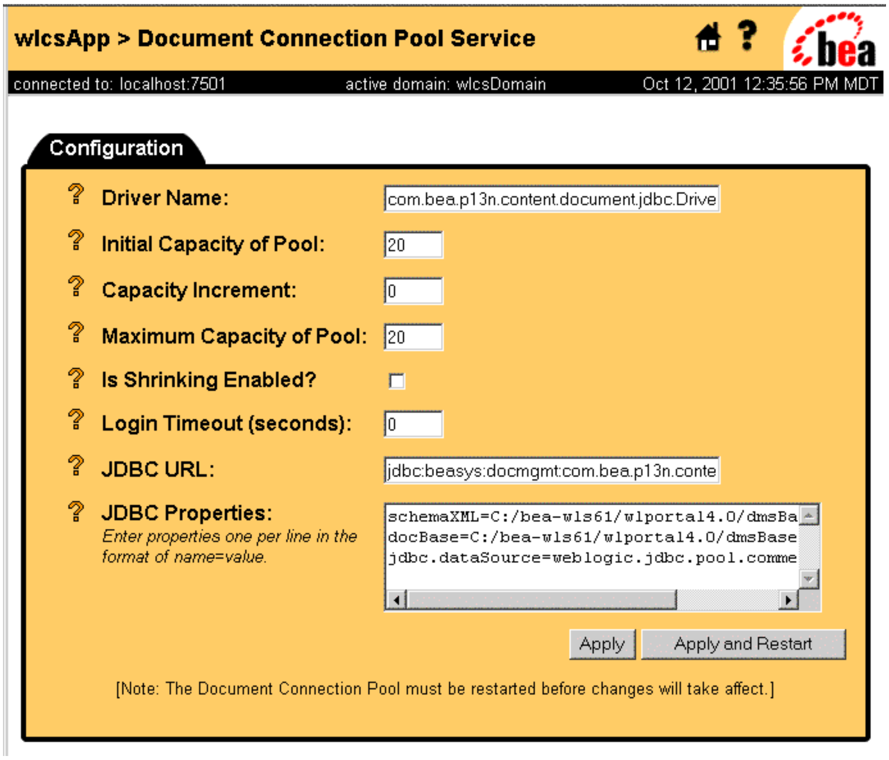
  If not specified, it defaults to `SERIALIZABLE`.

  For further details, see the *Javadoc* API documentation for `java.sql.Connection`.
- `jdbc.column.<colName>`: —Specifies an additional column to the DOCUMENT table. The value is the comma-separated list of property names that map onto that column. This can be specified multiple times. This should be used in conjunction with the the `-columnMap` and/or `-column` arguments to the BulkLoader. If the same property is mapped to more than one column, the result is indeterminate.

Editing a DocumentConnectionPool MBean in the WebLogic Console

Once a DocumentConnectionPool MBean has been initially configured in the application-config.xml, it can be edited via the WebLogic Server Administration Console, as shown in Figure 9-2.

Figure 9-2 Using the WLS Console to Edit a DocumentConnectionPool MBean


wlcsApp > Document Connection Pool Service

connected to: localhost:7501 active domain: wlcsDomain Oct 12, 2001 12:35:56 PM MDT

Configuration

Driver Name:	com.bea.p13n.content.document.jdbc.Drive
Initial Capacity of Pool:	20
Capacity Increment:	0
Maximum Capacity of Pool:	20
Is Shrinking Enabled?	☐
Login Timeout (seconds):	0
JDBC URL:	jdbc:beasys:docmgmt:com.bea.p13n.conte

JDBC Properties:
Enter properties one per line in the format of name=value.

```
schemaXML=C:/bea-wls61/wlportal4.0/dmsBa
docBase=C:/bea-wls61/wlportal4.0/dmsBase
jdbc.dataSource=weblogic.jdbc.pool.comme
```

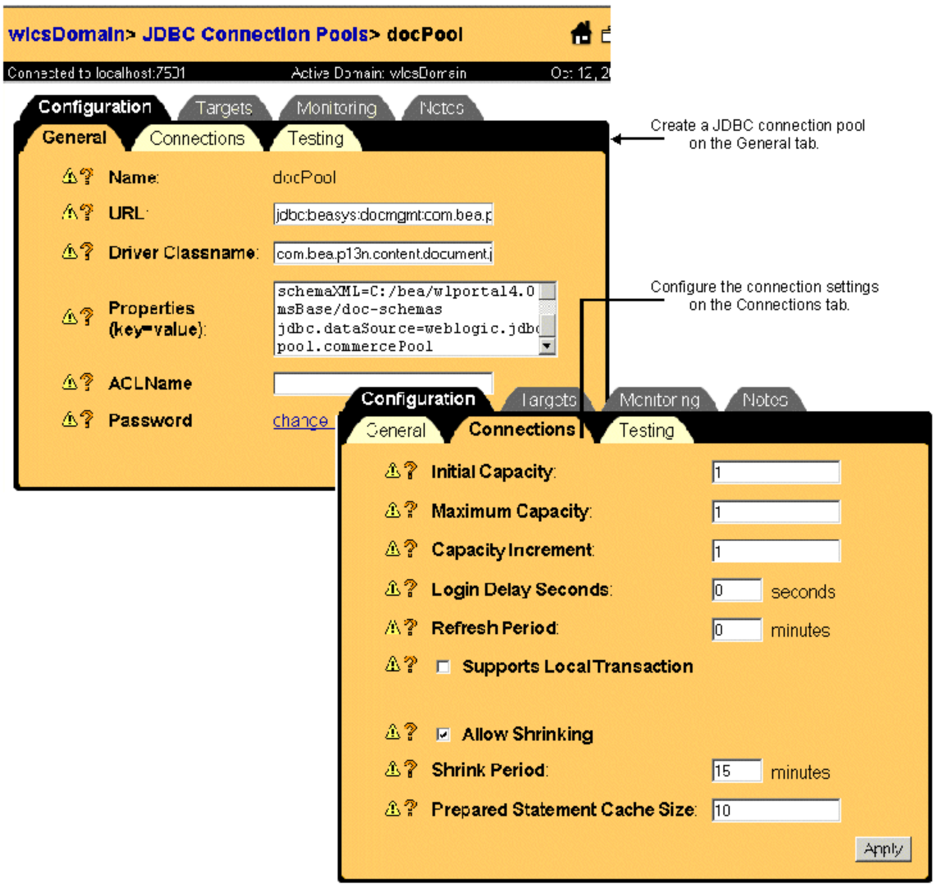
[Note: The Document Connection Pool must be restarted before changes will take affect.]

Setting up WebLogic Connection Pools

If you map jdbc/docPool in your `DocumentManager` EJB deployment descriptor, you will need to configure the WebLogic JDBC connection pool and data source.

Figure 9-3 shows how you can create a JDBC connection pool and configure the connection settings through the WebLogic Server Administration Console. The URL field is the same as the URL field in the `DocumentConnectionPool` MBean above. The `Driver Classname` is the same as the Driver field above. The Properties field is the same as the Properties field above.

Figure 9-3 Creating and Configuring a JDBC Connection Pool


wlcsDomain> JDBC Connection Pools> docPool
Connected to localhost:7501
Active Domain: wlcsDomain
Configuration
Targets
Monitoring
Notes
General
Connections
Testing
Create a JDBC connection pool on the General tab.
Configure the connection settings on the Connections tab.
Apply

General tab:

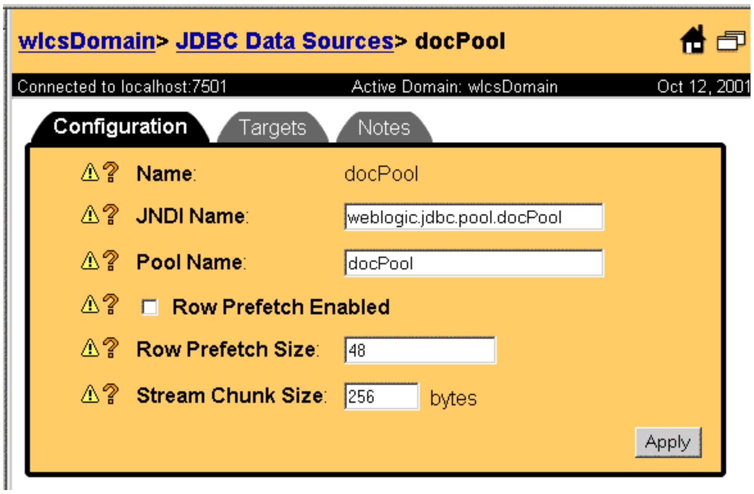
Name:	docPool
URL:	jdbc:beasys:docmgmt:com.bea.p
Driver Classname:	com.bea.p13n.content.document.j
Properties (key=value):	schemaXML=C:/bea/wlportal4.0 msBase/doc-schemas jdbc.dataSource=weblogic.jdbc pool.commercePool
ACLName	
Password	change

Connections tab:

Initial Capacity:	1
Maximum Capacity:	1
Capacity Increment:	1
Login Delay Seconds:	0 seconds
Refresh Period:	0 minutes
☐ Supports Local Transaction	
☑ Allow Shrinking	
Shrink Period:	15 minutes
Prepared Statement Cache Size:	10

Then, you can configure the data source connected to the connection pool, as show in Figure 9-4.

Figure 9-4 Configuring the Data Source


wlcsDomain> JDBC Data Sources> docPool

Connected to localhost:7501 Active Domain: wlcsDomain Oct 12, 2001

Configuration | Targets | Notes

Name: docPool

JNDI Name: weblogic.jdbc.pool.docPool

Pool Name: docPool

☐ Row Prefetch Enabled

Row Prefetch Size: 48

Stream Chunk Size: 256 bytes

Apply

The JNDI name selected here will be used in the `jdbc/docPool` resource reference in the `DocumentManager` EJB deployment descriptor.

For more information about using the WebLogic Server Administration Console for configuring and managing JDBC connection pools, see the topic “JDBC Connection Pool” in the WebLogic Server documentation.

You do not need to do this if you configure the `DocumentConnectionPool` MBean. If you choose to use a WLS connection pool, you will need be certain that your `DocumentProvider` implementation and all classes that it references are available in the system CLASSPATH of your server. Otherwise, you will most likely receive errors on startup. For more information about the CLASSPATH environment variable, see “Setting Environment Variables” under “Starting and Shutting Down the Server” in the *Deployment Guide*.

Web Application Configuration

To correctly access the various pieces of the Content Management component, you will need to configure EJB references to `ejb/ContentManager` and `ejb/DocumentManager`. Additionally, you need to have the com.bea.p13n.content.servlets.ShowDocServlet mapped into your Web Application. It is suggested to map it to the /ShowDoc/* URL in your Web Application. In your Web Application's WEB-INF/web.xml, you can add:

```
<servlet>
  <servlet-name>ShowDocServlet</servlet-name>
  <servlet-class> com.bea.p13n.content.servlets.ShowDocServlet
  </servlet-class>

  <!-- Make showdoc always use the local ejb-ref DocumentMnager -->

  <init-param>
    <param-name>contentHome</param-name>
    <param-value>java:comp/env/ejb/DocumentManager</param-value>
  </init-param>

</servlet>

...

<servlet-mapping>
  <servlet-name>ShowDocServlet</servlet-name>
  <url-pattern>/ShowDoc/*</url-pattern>
</servlet-mapping>
```

This will allow the ShowDoc/ URI under your Web Application's context root (for example, /wlcs/ShowDoc) to be sent to the ShowDocServlet. The contentHome <init-param> will cause that ShowDocServlet to always use the ejb/DocumentManager EJB reference; you can take this out to allow ShowDocServlet to obey any contentHome request parameters.

To access the Content Management tag libraries, you will need to:

- Copy the cm_taglib.jar file in the Web Application's WEB-INF/lib directory. (It can be copied from WL_PORTAL_HOME/lib/p13n/web.)
- Make sure that cm.tld is mapped to /WEB-INF/lib/cm_taglib.jar in a <taglib> entry in your Web Application's WEB-INF/web.xml file.

For more information, see the *Deployment Guide* and the web.xml and weblogic.xml files in WL_PORTAL_HOME/applications.

Using the BulkLoader to Load File-based Content

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

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

Command-Line Usage

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

```
java com.bea.p13n.content.document.ref.loader.BulkLoader
     [-/+verbose] [-/+recurse] [-/+delete] [-/+metaparse] [-/+cleanup]
     [-/+hidden] [-/+inheritProps] [-/+truncate] [-/+ignoreErrors]
     [-schemaName <name>] [-encoding <encoding>] [-commitAfter <num docs>]
     [-properties <name>] -conPool <name> [-schema <name>] [+schema]
     [-match <pattern>] [-ignore <pattern>] [-htmlPat <pattern>]
     [-d <dir>] [-mdext <ext>] [--]
     [files... directories...] [-filter <filter class>] [+filters]
     [-columnMap <file.properties>]
     [-column <columnName>=<propName,...>][+columns]
```

Table 9-19 The BulkLoader's Command-line Switches

`-verbose`	Emits verbose messages.
`+verbose`	Runs quietly [default].
`-recurse`	Recurses into directories [default].

Table 9-19 The BulkLoader's Command-line Switches

`+recurse`	Does not recurse into directories.
`-delete`	Removes document from database.
`+delete`	Inserts documents into database [default].
`-metaparse`	Parses HTML files for `<meta>` tags [default].
`+metaparse`	Does not parse HTML files for `<meta>` tags.
`-cleanup`	If specified, this only performs a table cleanup using the `-d` argument as the document base. (All files will need to be under that directory.)
`+cleanup`	Turns off table cleanup (do a document load) [default].
`-hidden`	Specifies to ignore hidden files and directories [default].
`+hidden`	Specifies to include hidden files and directories.
`-inheritProps`	Specifies to have metadata properties be inherited when recursing [default].
`+inheritProps`	Specifies to have metadata properties not be inherited when recursing.
`-truncate`	Attempts to truncate data values if they are too large for the database (controlled via `loader.properties`).
`+truncate`	Does not attempt to truncate data values [default].
`-ignoreErrors`	Ignores any errors while loading a document (errors will still be reported).
`+ignoreErrors`	Stops processing on any error [default].
`-htmlPat <pattern>`	Specifies a pattern for determining which files are HTML files when determining whether to do the `<meta>` tag parse. This can be specified multiple times. If none are specified, `*.htm` and `*.html` are used.
`-properties <name>`	Specifies the location of the loaddocs.properties file that should contain the `connectionPool` definition. This file may contain `jdbc.column.<columnName>=<propname>` entries similar to the `-columnMap` argument.
`-conPool <name>`	Specifies the `connectionPool` name from the properties file from which the BulkLoader should get the connection information.
`-schema <name>`	Specifies the path to the schema file the BulkLoader will generate (defaults to `document-schema.xml`).
`+schema`	If specified, then no schema file will be created.

Table 9-19 The BulkLoader's Command-line Switches

`-schemaName <name>`	Specifies the name of the schema generated by the BulkLoader. Defaults to "LoadedData".
`-encoding <name>`	Specifies the file encoding to use. Defaults to your system's default encoding. (See your JDK documentation for the valid encoding names.)
`-commitAfter <num>`	Commits the JDBC transaction after this many documents are loaded. Defaults to: only at the end of the full load.
`-match <pattern>`	Specifies a file pattern the BulkLoader should include. This can be specified multiple times. If none are specified, all files and directories are included.
`-ignore <pattern>`	Specifies a file pattern the BulkLoader should not include. This can be specified multiple times.
`-d <dir>`	Specifies the docBase that non-absolute paths will be relative to. If not specified, `"."` (current directory) is used.
`-mdext <ext>`	Specifies the filename extension for metadata property files. The value should starts with a `"."` (defaults to `.md.properties)`.
`-filter <filter class>`	Specifies the class name of a `LoaderFilter` to run files through. This can be specified multiple times to add to the list of Loader Filters.
`+filters`	Clears the current list of Loader Filters. (This will clear the default filters as well.)
`--`	Everything after this is considered a file or directory.
`-columnMap <file.properties>`	Specifies a properties file containing the `jdbc.column.<columnName>=<propname,...>` list of additional columns to the DOCUMENT table (see `-column`). This cannot be used to override behavior for standard columns.
`-column <columnName>=<propName, ...>`	Specifies an additional column to the DOCUMENT table and the property names that map onto the column. This cannot be used to override behavior for standard columns.
`+columns`	Clears any configured additional columns.

How the BulkLoader Finds Files

The following sequence describes how the BulkLoader locates files:

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

   Note: If the file or directory is not an absolute path, then it is assumed to be relative to the `docBase` specified by the `-d` option.
2. To determine if the BulkLoader should process a file or directory, it checks to see if the file is marked as a hidden file.

   Note: If it is a hidden file (or directory) and the `+hidden` option was not specified, then the file or directory is ignored.
3. If the file or directory does not exist or is not readable by the user executing the BulkLoader, a warning is displayed and the file or directory is ignored.
4. If the file or directory is a file, then it is loaded.
5. If the loaded object is a directory and recursion is enabled, then the files and directories under the directory are retrieved by filtering against the `-match` and `-ignore` options.

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

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

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

How the BulkLoader Finds Metadata Properties

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

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

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

Finally, the BulkLoader will pass the file to the `loadProperties` method of each registered LoaderFilter (the `-filter` option). The LoaderFilter may assign additional metadata to the file. When the BulkLoader starts up, it looks for a `com/bea/p13n/content/document/ref/loader/loader.properties` file in the classpath. From that, it looks for a `loader.defFilters` property. This is the colon-separated list of `LoaderFilter` class names the BulkLoader should always load. Unless that file is modified, the BulkLoader will load an `ImageLoaderFilter`, which will pull the width and height from `*.gif`, `*.jpg`, `*.png`, and `*.xbm` image files.

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

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

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

Cleaning Up the Database

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

Loading Internationalized Documents

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

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

```
java com.bea.p13n.content.document.ref.loader.BulkLoader -verbose
-properties loaddocs.properties -conPool commercePool -schema
dmsBase\schemas\unicode-files.xml -d dmsBase unicode-files
-encoding Unicode
```

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

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

The list of available encoding names is contained in the documentation for your JDK, or the documentation for the tool which created the file. If you are not creating files containing non-ASCII characters, this should not affect you. If you want to check if the BulkLoader is correctly parsing your HTML file, you can use the `com.bea.p13n.content.document.ref.loader.MetaParser` class.

For example:

```
java com.bea.p13n.content.document.ref.loader.MetaParser
unicode.htm unicode
```

would print out the `<meta>` tags found in the `unicode.htm` file, assumed to be Unicode encoded. Of course, any non-ASCII character probably will not print correctly to your console window, but you can tell what it thinks it found.

Generating Schema Files

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

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

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

CHAPTER

10 Working with Ad Placeholders

An ad placeholder is one of several mechanisms that WebLogic Portal provides for retrieving documents from a content management system. A *document* is a graphic, a segment of HTML or plain text, or a file that must be viewed with a plug-in. (We recommend that you store most of your Web site's dynamic content as documents in a content management system because it offers an effective way to store and manage information.)

Ad placeholders are intended to display documents that advertise products or services (ads) and to record customer reactions to them. You can use a single set of ad placeholders to support multiple advertising projects that change over time. If you use WebLogic Portal, you can use ad placeholders to display ads for campaigns.

A Business Analyst (BA) uses the BEA E-Business Control Center to define the behavior of an ad placeholder. Then, a Business Engineer (BE) creates ad placeholder JSP tags in JSPs.

Similar to ad placeholders, the `<ad:adTarget>` JSP tag also provides services for displaying ads. However, as described later in this topic, the `<ad:adTarget>` JSP tag provides a subset of the ad placeholder services.

This topic includes the following sections:

- What Are Ad Placeholders, Ad Attributes, and Placeholder Tags?
- Resolving Ad Query Conflicts
- Creating Ad Placeholder Tags
- Supporting Additional MIME Types
- How Placeholders Select and Display Ads
- How to Configure Ad Placeholders in an Application

To learn more about using a content management system with WebLogic Portal, refer to Chapter 9, “Creating and Managing Content,” in this guide. For a comparison of content retrieval methods available with WebLogic Portal, refer to “Methods for Retrieving and Displaying Documents” on page 9-5.

What Are Ad Placeholders, Ad Attributes, and Placeholder Tags?

This section describes the following items:

- Ad Placeholders
- Ad Attributes in the Content Management System
- Ad Placeholder JSP Tags
- The <ad:adTarget> JSP Tag

Ad Placeholders

An ad placeholder is a named entity that contains one or more queries. When a customer requests a JSP that contains an ad placeholder tag, the placeholder selects a single ad query to run and generates the HTML that the browser requires to display the results of the query.

For example, you want to display ads in the top banner of your Web site’s home page. You define an ad placeholder and create ad queries for the placeholder. Then you create an ad placeholder JSP tag in the top banner of the home page. When a customer requests the home page, the placeholder selects a query, runs the query, and displays the results in the banner.

This section includes the following subsections:

- Types of Queries That Ad Placeholders Run
- Types of Documents That Ad Placeholders Display

Types of Queries That Ad Placeholders Run

Ad placeholders can run a default query or a query that is associated with a specific scenario in a campaign.

You create default ad queries when you define the ad placeholder in the E-Business Control Center. A placeholder runs a default query each time a customer loads a page that includes the placeholder. For example, you define a default query for a top banner placeholder and the placeholder runs the query each time a customer loads a page with the top banner.

You create scenario queries when you define scenario actions in the E-Business Control Center. (Scenario actions specify a list of actions to take in response to a chain of events.) A placeholder contains a scenario query only if a customer or an event triggers the scenario action. For example, you create a scenario that does the following:

When a customer places a handsaw product in the shopping cart, the scenario places an ad for miter boxes in the ad placeholder on the shopping cart page. When the customer requests the shopping cart page, the shopping cart ad placeholder runs the query for miter box ads and displays the results.

You can prevent a placeholder from running default queries if any scenario actions have specified a query for the placeholder, or you can allow the Ad Conflict Resolver to choose a default query or a scenario query. For more information, refer to "Resolving Ad Query Conflicts" on page 10-10 in this guide.

Types of Documents That Ad Placeholders Display

Placeholders use a document's MIME-type attribute to generate the appropriate HTML tags that the browser requires. By default, ad placeholders generate the appropriate HTML tags only for the following MIME types:

- XHTML (a fragment or an entire document). For this type of document, a placeholder passes the text directly to the JSP.
- Images. For this type of document, a placeholder generates an `<img>` tag with attributes that the browser needs to display the image. If you want images to be clickable, you must specify the target URL and other link-related information as ad attributes in your content management system.
- Shockwave files. For this type of document, a placeholder generates the `<OBJECT>` tag, which Microsoft Internet Explorer on Windows uses to display the file, and the `<EMBED>` tag, which browsers that support the Netscape-compatible plug-in used to display the file. In your content

management system, you can specify attributes for the `<OBJECT>` and `<EMBED>` tags.

For information on setting up placeholders to support additional MIME types, refer to "Supporting Additional MIME Types" on page 10-18 in this guide.

Ad Attributes in the Content Management System

Ad placeholders use a set of document attributes that you define in your content management system to support the following features:

- Choosing a single document if a query returns multiple documents
- Making an image ad clickable
- Supplying movie preferences for a Shockwave file

For information about associating attributes with documents, refer to the documentation for your content management system. If you use the reference BulkLoader, refer to Chapter 9, "Creating and Managing Content," in this guide.

Table 10-20 describes the `adWeight` attribute, which you can associate with XHTML, image, and Shockwave documents.

Table 10-20 Attributes for All Document Types

Attribute Name	Value Type	Description and Recommendations
adWeight	Integer	Provides an integer that is used to select a document if a query returns multiple documents. Assign a high number to ads that you want to have a greater chance of being selected. For more information, refer to "How an Ad Placeholder Chooses from Ad Query Results" on page 10-13 in this guide. The default value for this attribute is 1. **Note:** In the E-Business Control Center, you can assign a priority to a query for a scenario action. The priority, which bears no relation to the `adWeight` attribute, gives a greater or lesser chance that a placeholder runs a query. The `adWeight` attribute is used to choose an ad after a query has run. For more information, refer to "How the Ad Conflict Resolver Chooses a Query" on page 10-12 in this guide.

Table 10-21 describes attributes in addition to the `adWeight` attribute that you can associate with image files.

Table 10-21 Attributes for Image Files

Attribute Name	Value Type	Description and Recommendations
adTargetUrl	String	Makes an image clickable and provides a target for the clickthrough, expressed as a URL. The Events Service records the clickthrough. Use either `adTargetUrl`, `adTargetContent`, or `adMapName`, depending on how you want to identify the destination of the ad clickthrough.
adTargetContent	String	Makes an image clickable and provides a target for the clickthrough, expressed as the content management system's content ID. The Events Service records the clickthrough. Use either `adTargetUrl`, `adTargetContent`, or `adMapName`, depending on how you want to identify the destination of the ad clickthrough.

Table 10-21 Attributes for Image Files (Continued)

Attribute Name	Value Type	Description and Recommendations
adMapName	String	Makes an image clickable, using an image map to specify one or more targets. The value for this attribute is used in two locations: ■ In the anchor tag that makes the image clickable, `<a href=`*`value`*`> <img> </a>` ■ In the map definition, `<map name=`*`value`*`>` Use either `adTargetUrl`, `adTargetContent`, or `adMapName`, depending on how you want to identify the destination of the ad clickthrough. If you specify a value for `adMapName`, you must also specify a value for `adMap`.
adMap	String	Supplies the XHTML definition of an image map. If you specify a value for `adMap`, you must also specify a value for `adMapName`.
adWinTarget	String	Displays the target in a new pop-up window, using JavaScript to define the pop-up window. The only value supported for this attribute is `newwindow`.
adWinClose	String	Specifies the name of a link that closes a pop-up window. The link appears at the end of the window content. For example, if you provide “Close this window” as the value for this attribute, then “Close this window” appears as a hyperlink in the last line of the pop-up window. If a customer clicks the link, the window closes.
adAltText	String	Specifies a text string for the `alt` attribute of the `<img>` tag. If you do not include this attribute, the `<img>` tag does not specify an `alt` attribute.
adBorder	Integer	Specifies the value for the `border` attribute of the `<img>` tag. If you do not include this attribute, the `border` attribute is given a value of `"0"`.

Table 10-22 describes attributes in addition to the `adWeight` attribute that you can associate with Shockwave files. Ad placeholders and the `<ad:adTarget>` tag format these values as attributes of the `<OBJECT>` tag, which Microsoft Internet Explorer on Windows uses to display the file, and the `<EMBED>` tag, which browsers that support the Netscape-compatible plug-in used to display the file.

For more information about these attributes, refer to your Shockwave developer documentation.

Table 10-22 Attributes for Shockwave Files

Attribute Name	Value Type	Description and Recommendations
swfLoop	String	Specifies whether the movie repeats indefinitely (`true`) or stops when it reaches the last frame (`false`). Valid values are `true` or `false`. If you do not define this attribute, the default value is `true`.
swfQuality	String	Determines the quality of visual image. Lower qualities can result in faster playback times, depending on the client's Internet connection. Valid values are `low`, `high`, `autolow`, `autohigh`, `best`.
swfPlay	String	Specifies whether the movie begins playing immediately on loading in the browser. Valid values are `true` or `false`. If you do not define this attribute, the default value is `true`.
swfBGColor	String	Specifies the background color of the movie. This attribute does not affect the background color of the HTML page. Valid value syntax is #*`RRGGBB`*.
swfScale	String	Determines the dimensions of the movie in relation to the area that the HTML page defines for the movie. Valid values are `showall`, `noborder`, `exact fit`.
swfAlign	String	Determines whether the movie aligns with the center, left, top, right, or bottom of the browser window. If you do not specify a value, the movie is aligned in the center of the browser. Valid values are `l`, `t`, r, `b`.

Table 10-22 Attributes for Shockwave Files (Continued)

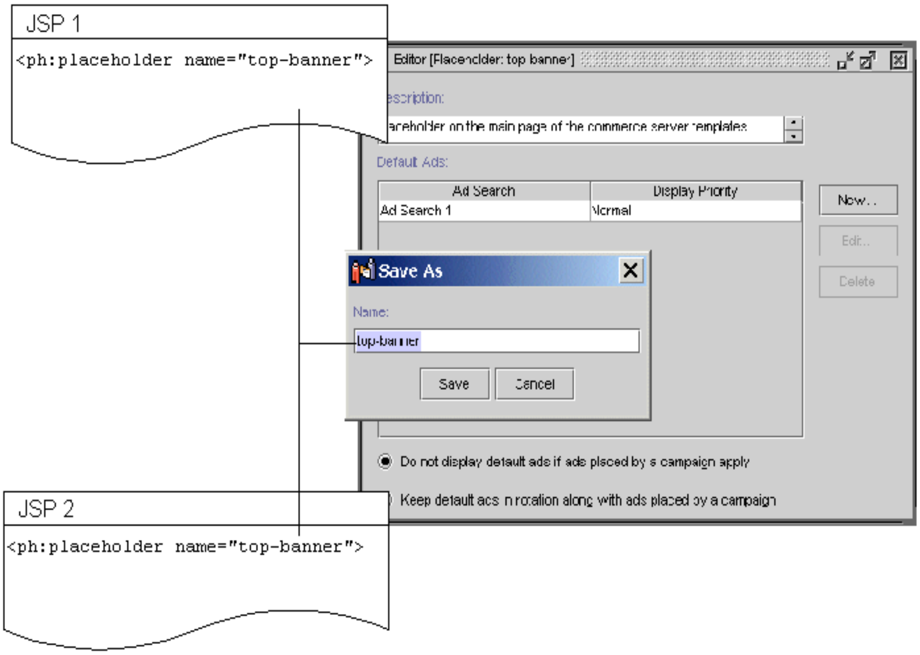
Attribute Name	Value Type	Description and Recommendations
swfSAlign	String	Determines the movie's alignment in relation to the browser window. Valid values are `l`, `t`, `r`, `b`, `tl`, `tr`, `bl`, `br`.
swfBase	String	Specifies the directory or URL used to resolve relative pathnames in the movie. Valid values are `.(period)`, *`directory-name`*, *`URL`*.
swfMenu	String	Determines whether the movie player displays the full menu. Valid values are `true` or `false`.

Ad Placeholder JSP Tags

An ad placeholder JSP tag refers to the placeholder definition that you create in the E-Business Control Center. Then it displays the results of the query that the placeholder runs. You can create multiple placeholder tags that refer to a single placeholder definition. (See Figure 10-1.)

For more information about placeholder tags, refer to <ph:placeholder> in Chapter 13, "Personalization Server JSP Tag Library Reference," in this guide.

Figure 10-1 Multiple Tags Using a Single Definition



```
<ph:placeholder name="top-banner">
```

acehol der on the main page of the commerce server templates

Ad Search	Display Priority
Ad Search 1	Normal

Name: top-banner

● Do not display default ads if ads placed by a campaign apply

Keep default acs n rotation along with ads placed by a campaign

```
<ph:placeholder name="top-banner">
```

The <ad:adTarget> JSP Tag

The `<ad:adTarget>` JSP tag is an additional mechanism for selecting and displaying ads. Use `<ad:adTarget>` if it is essential that a specific query run in a specific location.

Like an ad placeholder, `<ad:adTarget>` can do the following:

- Generate the HTML that a browser requires to display the types of documents that are described in “Types of Documents That Ad Placeholders Display” on page 10-3.
- Use the document attributes that are described in “Ad Attributes in the Content Management System” on page 10-4.

- Use the Ad Service to choose an ad if a query returns multiple documents, as described in "How an Ad Placeholder Chooses from Ad Query Results" on page 10-13.

However, the `<ad:adTarget>` is **unlike** ad placeholders in the following ways:

- It contains its own query; it does not refer to a definition that a BA creates in the E-Business Control Center. If you want to change the query, you modify the tag in the JSP.
- A campaign scenario cannot specify a query to run in an `<ad:adTarget>` tag. Scenarios can only use ad placeholders to run queries.
- Because it contains only a single query, it does not need to use the Ad Conflict Resolver as described in "How the Ad Conflict Resolver Chooses a Query" on page 10-12.

For a more information about `<ad:adTarget>`, refer to Chapter 13, "Personalization Server JSP Tag Library Reference," in this guide.

Resolving Ad Query Conflicts

A placeholder can contain many ad queries: you can define multiple default queries and multiple scenarios can send queries to a placeholder. To determine which ad query to run, a placeholder uses the Ad Conflict Resolver.

In addition, an ad query can return multiple documents. To determine which ad to display, a placeholder uses the `adWeight` document attribute.

This section includes the following subsections:

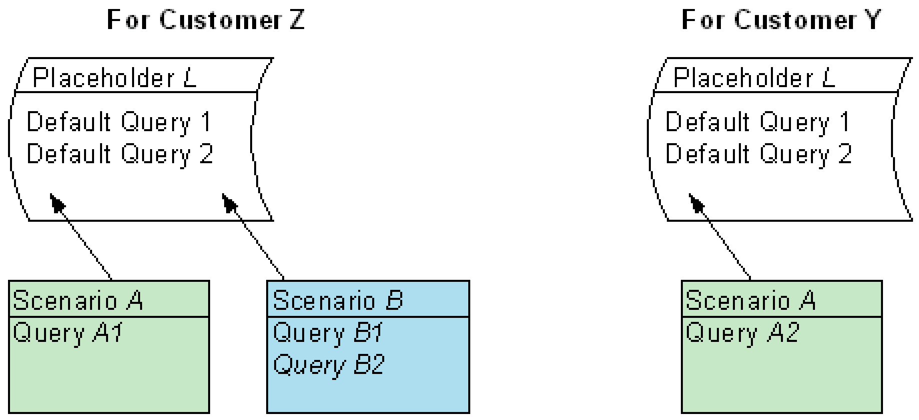
- How Ad Placeholders Contain Multiple Queries
- How the Ad Conflict Resolver Chooses a Query
- How an Ad Placeholder Chooses from Ad Query Results

If you need to make sure that a given ad query runs in a specific location, use an `<ad:adTarget>` tag, which can contain only a single query. For more information, refer to "The <ad:adTarget> JSP Tag" on page 10-9 in this guide.

How Ad Placeholders Contain Multiple Queries

In addition to containing default queries, an ad placeholder can contain queries that scenarios define. Depending on customers' profiles and the events that customers trigger, a placeholder can contain different queries for different customers. (See Figure 10-2.)

Figure 10-2 Different Ad Queries for Different Customers



For example, you create placeholder `L` at the top of a portlet to display ads for any of the following products:

- Handsaws and miter boxes. You want ads for handsaws and miter boxes to display for any customer, anonymous or authenticated. When you define placeholder *L*, you include default queries for ads about handsaws and miter boxes.
- Electric drills. You want ads for electric drills, which are part of the Hardware 2001 campaign, to display when a Bronze Customer or Gold Customer logs in. When you define the Hardware 2001 campaign, you include a scenario that places ad queries for electric drills in placeholder *L* when a Bronze Customer or Gold Customer logs in.
- Circular saws. You want ads for circular saws, which are part of the Hardware 2001 campaign, to display when a Gold Customer logs in. When you define the Hardware 2001 campaign, you define a scenario that recognizes when a Gold

Customer logs in. For that scenario, you specify an action that places ad queries for pneumatic hammers in placeholder *L*.

When the Bronze Customer Pat Gomes logs in and accesses the portlet, WebLogic Portal adds queries for handsaws (which applies to all customers) and electric drills (which applies to Bronze Customers) to ad placeholder *L*. Then it uses the Ad Conflict Resolver to determine which ad query to run.

How the Ad Conflict Resolver Chooses a Query

When you define an ad placeholder in the E-Business Control Center, you can assign a priority to the default ad queries; when you define scenario actions that specify ad queries, you can assign a priority to the scenario's ad query. The priority affects the probability that an ad query will run relative to other ad queries in the placeholder.

For example, ad placeholder *L* contains three ad queries:

- Campaign Ad query X, which has a medium priority. The Ad Conflict Resolver gives all medium-priority ads 2 points
- Default Ad Y, which has a low priority and receives 1 point
- Default Ad Z, which also has a low priority and receives 1 point

The total number of points in ad placeholder *L* is 4. To determine which of the three ad queries to run, the Ad Conflict Resolver does the following:

1. It creates 4 slots in the ad placeholder. The number of slots corresponds to the total number of points currently in the ad placeholder.
2. It places campaign ad query X, which has 2 points into 2 slots. Each of the other ad queries, with 1 point, gets a single slot:
   a. Slot 1 = campaign ad query X
   b. Slot 2 = campaign ad query X
   c. Slot 3 = default ad query Y
   d. Slot 4 = default ad query Z
3. It generates a random number between 1 and 4, which is equal to the number of slots in the ad placeholder.

4. It matches the generated number with a slot in the placeholder. Because campaign ad query X occupies two of four slots, it has a 50% chance of being run. Default ad queries Y and Z each have a 25% chance of being run.
5. If a query does not find any documents, the placeholder chooses another query and runs it.

If the campaign associated with ad query X ends, then the total number of points in ad placeholder *L* is reduced to 2. To determine which ad query to run, the Ad Conflict Resolver does the following:

1. It creates two slots in the ad placeholder and assigns ad query Y and ad query Z each to a single slot.
2. It generates a random number between 1 and 2.
3. It matches the generated number with a slot in the placeholder. Now, each ad query has a 50% chance of running.

How an Ad Placeholder Chooses from Ad Query Results

Depending on how broadly you define an ad query and on the number of documents in your content management system, an ad query could return multiple documents. In your content management system, you can add the `adWeight` attribute to documents that display as ads.

If a placeholder or `<ad:adTarget>` query returns multiple documents, the ad placeholder or the `<ad:adTarget>` tag does the following:

1. It determines the `adWeight` values for all documents that the query returns and adds them together.

   For example, an ad query returns the following three ads:

   - Ad X, with an `adWeight` value of 2
   - Ad Y, with an `adWeight` value of 1
   - Ad Z, with an `adWeight` value of 1

   The total weight for the documents that the query returns is 4.
2. It creates 4 slots, corresponding to the total weight in the query.

3. It places ad X, with a weight of 2 into 2 slots. Each of the other ads, with weights of 1, gets a single slot:

   a. Slot 1 = ad X

   b. Slot 2 = ad X

   c. Slot 3 = ad Y

   d. Slot 4 = ad Z

4. It generates a random number between 1 and 4, which is equal to the total weight in the query.

5. It matches the generated number with a slot. Because ad X occupies two of four slots, it has a 50% chance of being displayed. Ads Y and Z each have a 25% chance of being displayed.

Creating Ad Placeholder Tags

After a BA uses the E-Business Control Center to create ad placeholders, a BE creates ad placeholder tags in the Web site's JSPs. The placeholder definition determines the behavior of the placeholder tag.

You can create placeholders in JSPs that directly display content to a customer (for example, `index.jsp`) or in JSPs that are included in other JSPs (for example, `heading.jsp`).

To Create an Ad Placeholder Tag

1. In a text editor, open a JSP.

2. Import the tag library by adding the following tag near the top of the JSP:

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

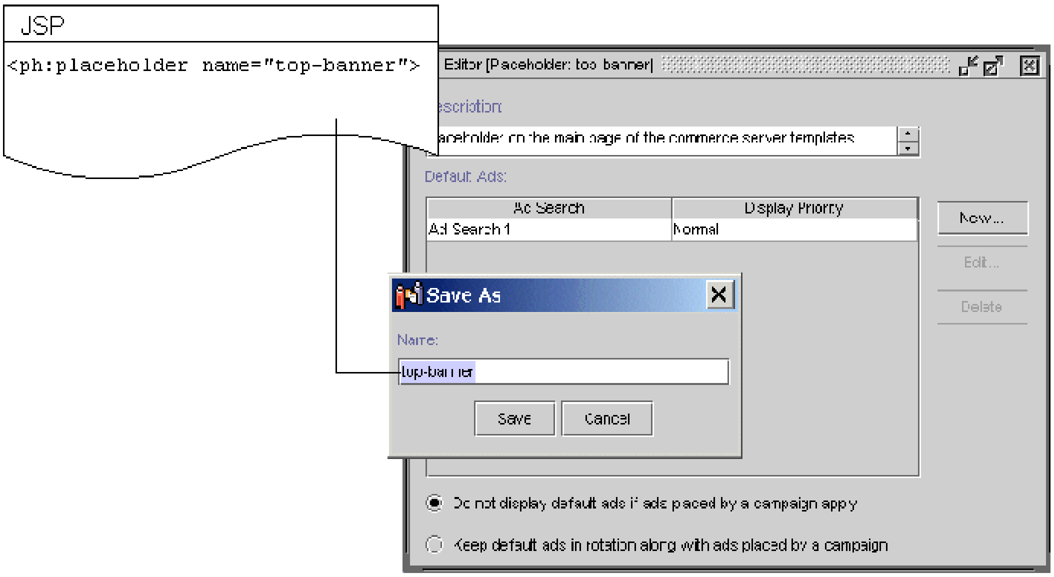
3. Find the location in which the Business Analyst wants to display the ad.

4. Use the following syntax to create the placeholder tag:

```
<ph: placeholder= “{ placeholder-name | scriptlet }” >
```

where *`placeholder-name`* refers to the name of an existing placeholder definition (see Figure 10-3) or where *`scriptlet`* returns the name of an existing placeholder. The name can be either the URI of the placeholder definition file (for example, "`/placeholders/top-banner.pla`") or just the placeholder filename without the `.pla` extension ("`top-banner`" in this example).

Figure 10-3 Placeholder Names Must Match



```
<ph:placeholder name="top-banner">
```

Editor [Placeholder: top banner]

Description: laceholder on the main page of the commerce server templates

Default Ads:

Ad Search	Display Priority
Ad Search 1	Normal

Save As

Name: top-banner

[✓ Do not display default ads if ads placed by a campaign apply]

Keep default ads in rotation along with ads placed by a campaign

Listing 10-1 shows an example from the heading include file of the wlcsApp reference application (`PORTAL_HOME\applications\wlcsApp\wlcs\commerce\includes\heading.inc`).

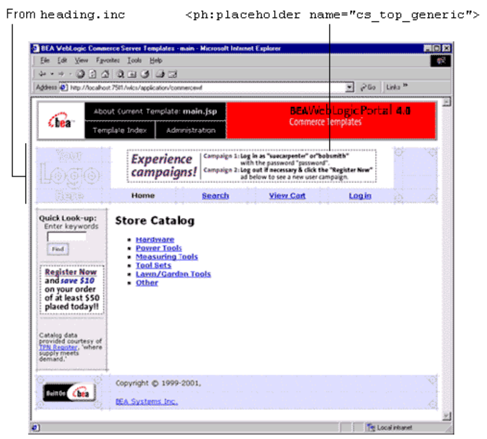
All JSP files in the wlcs reference Web application include `heading.inc` to create consistency in the top banner. Instead of requiring that the banner on each page use the same placeholder, the placeholder in `heading.inc` uses a scriptlet to determine the value of the `name` attribute. A JSP can use the default value for the `name` attribute (which is `cs_top_generic`), or it can define a variable named `banner` and specify a placeholder name as the value for the variable.

Listing 10-1 Using a Scriptlet for the Placeholder Name

```
<%

   String banner = (String)pageContext.getAttribute("bannerPh");
   banner = (banner == null) ? "/placeholders/cs_top_generic.pla" : banner;

%>

<!-- ------------------------------------------------------------ -->

<table width="100%" border="0" cellspacing="0" cellpadding="0" height="108">

  <tr>
     <td rowspan="2" width="147" height="108">
     <img src="<webflow:createResourceURL
resource="/commerce/images/header_logo.gif"/>" width="147" height="108">
    </td>

  <td colspan="7" height="75" align="center" valign="middle">


<ph:placeholder name="<%= banner %>" />

</td>
```

Figure 10-4 illustrates how WebLogic Portal renders the placeholder in the `main.jsp` file, which is the home page for the wlcs reference Web application.

Figure 10-4 Placeholder in the wlcs Web Application



Campaign 1: Log in as "suecarpenter" or "bobsmith" with the password "password".
Campaign 2: Log out if necessary & click the "Register Now" ad below to see a new user campaign.

Register Now and save $10 on your order of at least $50 placed today!!

Catalog data provided courtesy of TPN Register, "where supply meets demand."

For more information about the `<ph:placeholder>` tag, refer to Chapter 13, "Personalization Server JSP Tag Library Reference," in this guide.

Supporting Additional MIME Types

To display an ad, placeholders refer to a document's MIME type and then generate the HTML tags that a browser requires for the specific document type. For example, to display an image-type document, an ad placeholder must generate the `<img>` tag that a browser requires for images. By default, ad placeholders can generate the appropriate HTML only for the following MIME types:

- XHTML (a fragment or an entire document). For this type of document, a placeholder passes the text directly to the JSP.
- Images. For this type of document, a placeholder generates an `<img>` tag with attributes that the browser needs to display the image. If you want images to be clickable, you must specify the target URL and other link-related information as ad attributes in your content management system.
- Shockwave files. For this type of document, a placeholder generates the `<OBJECT>` tag, which Microsoft Internet Explorer on Windows uses to display the file, and the `<EMBED>` tag, which browsers that support the Netscape-compatible plug-in use to display the file. In your content management system, you can specify attributes for the `<OBJECT>` and `<EMBED>` tags.

If you are familiar with basic Java programming, you can write classes that enable placeholders to generate HTML for additional MIME types. To support additional MIME types, you must complete the following tasks:

- Create and Compile a Java Class to Generate HTML
- Register the New Class

Create and Compile a Java Class to Generate HTML

To generate the HTML that the browser requires to display the MIME type, create and compile a Java class that implements the `com.bea.p13n.ad.AdContentProvider` interface. For information on this interface, refer to WebLogic Portal *Javadoc*.

After you compile the class, you must make sure the class is available to the application. One way to do this is to add the class appropriately to one of the deployed jar files, such as `placeholder.jar` or your own jar file. Another way to make the class available to the applicaticatoin is to save it under a directory that is specified in the system's `CLASSPATH` environment variable. For example, create a `WL_PORTAL_HOME/classes` directory and add it to the set-environmment script. For more information about the `CLASSPATH` environment variable, refer to "Setting Environment Variables," under "Starting and Shutting Down the Server" in the *Deployment Guide*.

Register the New Class

After you save the class in a directory that is in your classpath, you must notify WebLogic Portal of its existence:

1. Stop the WebLogic Portal instance that is running your application. For information on stopping a server, refer to "Starting and Shutting Down a Server" in the *Deployment Guide*.
2. Create a backup copy of `PORTAL_HOME/application/`*`your-application`*`/META-INF/application-config.xml`
3. Open `application-config.xml` in a text editor and find the `<AdService>` element.
4. Add the following as a subelement of `<AdService>`:

   ```
   <AdContentProvider
           Name="MIME-type"
           Provider="name-of-your-class"
           Properties="optional-properties-for-your-class"
         >
   </AdContentProvider>
   ```

   Provide the following values for the attributes of the `AdContentProvider` element:

   - `Name`. The name of the MIME type that you want to support.
   - `Provider`. The name of the compiled Java file. If you saved the file below a directory that your `CLASSPATH` environment variable names, you must include the file's pathname, starting one directory level below the directory in classpath.

- `Properties`. Any additional properties or parameters want to pass to your object.

For example, if you added `WL_PORTAL_HOME/classes` to the system classpath, save your class to support AVI files as `WL_PORTAL_HOME/classes/myclasses/MimeAvi.class`.

Alternately, if you had an already deployed JAR of your own EJBs called `myServices.jar` (which is listed in the application's `META-INF/application.xml` file), add `myclasses/MimeAvi.class` to that JAR file.

To register your classname, add a subelement to the `AdService` element as illustrated in Listing 10-2.

Listing 10-2   Add An AdContentProvider Element

```
<AdService
      Name="wlcsApp"
      DisplayFlushSize="10"
      Rendering="com.bea.p13n.ad.AdContentProviderBase"
      EventTracker="com.bea.campaign.AdTracking"
      AdClickThruURI="AdClickThru"
      ShowDocURI="ShowDoc"
    >

      <AdContentProvider
        Name="text"
        Provider="com.bea.p13n.ad.render.TextContentProvider"
        Properties=""
      >
      </AdContentProvider>

      <AdContentProvider
        Name="image"
        Provider="com.bea.p13n.ad.render.ImageContentProvider"
        Properties="AdClickThruURI=AdClickThru;ShowDocURI=ShowDoc"
      >
      </AdContentProvider>

      <AdContentProvider
        Name="application/x-shockwave-flash"
        Provider="com.bea.p13n.ad.render.ShockwaveContentProvider"
        Properties="ShowDocURI=ShowDoc"
      >
      </AdContentProvider>
      <AdContentProvider
        Name="video/x-msvideo"
        Provider="myclasses.MimeAvi"
        Properties=""
      >
      </AdContentProvider>

  </AdService>
```

5. Save your modifications to `application-config.xml`.
6. Restart WebLogic Portal.

How Placeholders Select and Display Ads

Placeholders use the following process to select and display ads in a given JSP (see Figure 10-5):

1. Any of the following activities place ad queries in an ad placeholder:
   - You use the E-Business Control Center to define default queries for a placeholder.
   - As part of carrying out a campaign action, the Campaign Service adds queries to the placeholder.
2. When a user requests a JSP that contains a placeholder, if the ad placeholder contains more than one ad query, the Ad Service calls the Ad Conflict Resolver to select an ad query.

   For more information, refer to “How the Ad Conflict Resolver Chooses a Query” on page 10-12 in this guide.
3. The Ad Service does the following:

   a. It forwards the query to the content management system. If the query returns more than one ad, the ad placeholder uses the `adWeight` attribute of each ad to determine which one to retrieve.

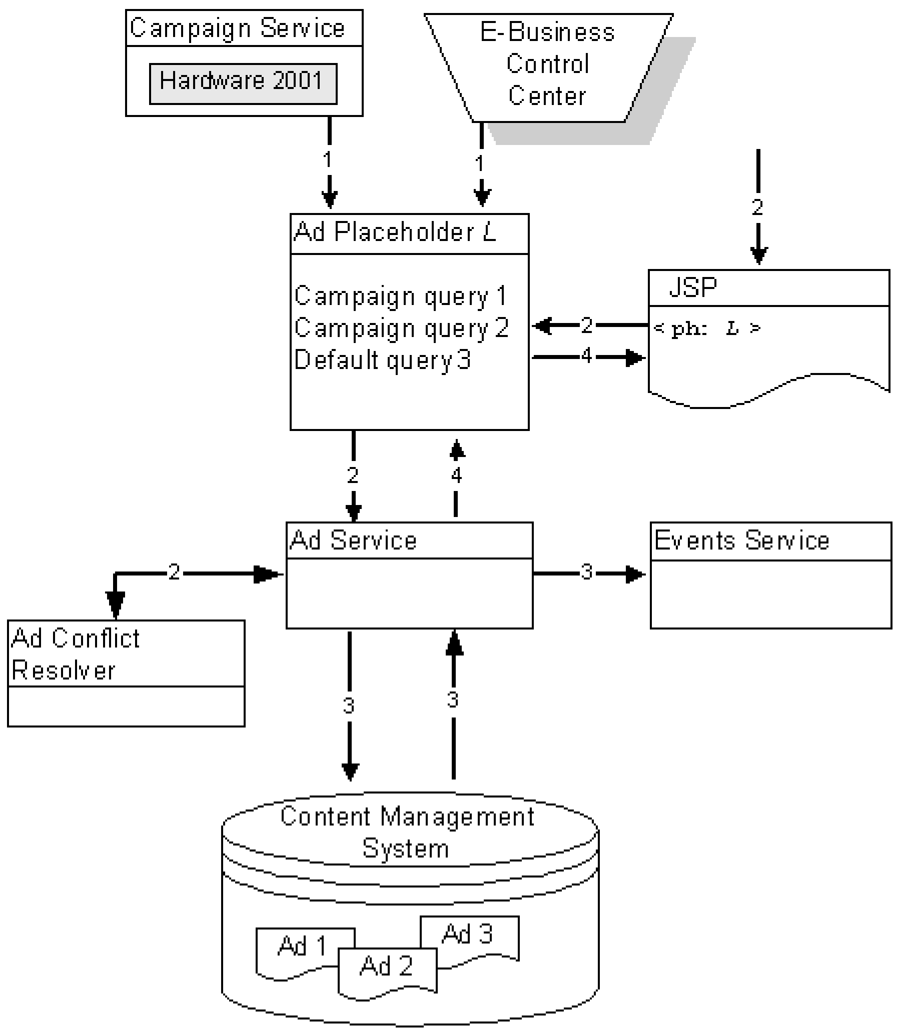
b. If the ad is associated with an active campaign, it determines whether the campaign has fulfilled its goal of displaying the ad a specific number of times. If the ad has already been displayed the specified number of times, the Ad Service selects another ad.

c. It sends data to the Events Service indicating that the placeholder has displayed the ad.

For more information, refer to "How an Ad Placeholder Chooses from Ad Query Results" on page 10-13 in this guide, and "Campaign Service Properties" under "The Server Configuration" in the *Deployment Guide*.

4. The ad placeholder renders the ad content and places it in the JSP at the location of the placeholder tag.
5. If a customer clicks on the ad, the Ad Service redirects the URL and notifies the Event Service that a customer clicked the ad.

Figure 10-5 How Placeholders Display Ads



How to Configure Ad Placeholders in an Application

For the `<ph:placeholder>` and `<ad:adTarget>` tags to work correctly, you will need the following configuration:

- EJB references to `ejb/PlaceholderService`, `ejb/AdBucketService`, and `ejb/DocumentManager` must be specified in the Web Application's `web.xml` and `weblogic.xml` files.
- The `com.bea.p13n.ad.servlets.AdClickThruServlet` must be mapped to `/AdClickThru/*` in the Web Application's `web.xml` file.
- The `com.bea.p13n.content.servlets.ShowDocServlet` must be mapped to `/ShowDoc/*` in the Web Application's `web.xml` file.

For more information, see the *Deployment Guide*.
Also, refer to the `web.xml` and `weblogic.xml` files in `WL_PORTAL_HOME/applications/portal/stockportal/WEB-INF`.

CHAPTER

11 Creating Localized Applications with the Internationalization Tags

This topic includes the following sections:

What Is the I18N Framework?

WebLogic Personalization Server, WebLogic Portal, Commerce services and Campaign services use the Internationalization Framework provided in WebLogic Server. Developers who are building server-side components, such as EJBs, pipeline components, or inline processors, will find the information about logging and text formatting in the WebLogic Server documentation WebLogic Server particularly useful.

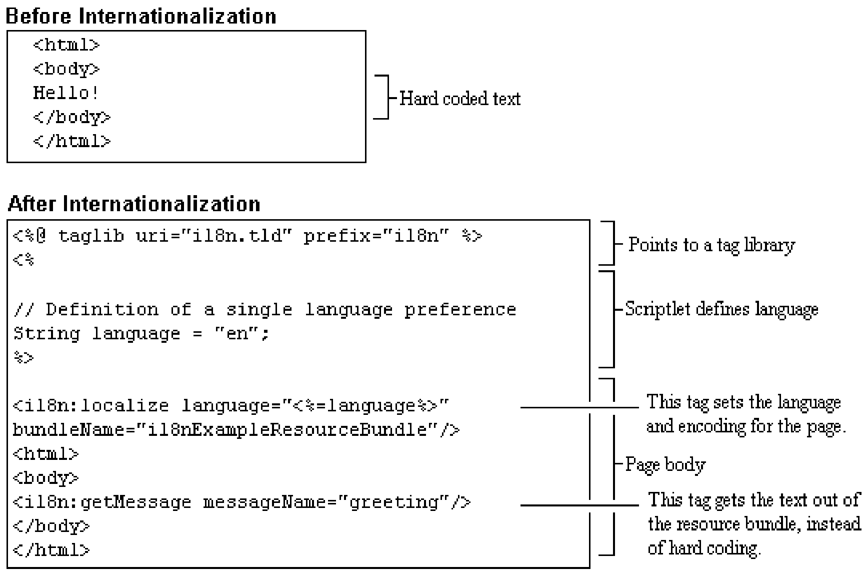
Note: For detailed information about the WebLogic Server Internationalization Framework, see the topic "Using the BEA WebLogic Server Internationalization Tools and Utilities" in the BEA WebLogic Server product documentation.

For localizing JavaServer Pages (JSPs), the WebLogic Personalization Server provides a simple framework that allows access to localized text labels and messages. The WebLogic Personalization Server's internationalization (I18N) framework is accessible from JSPs through a small I18N tag library. An example is shown in Figure 11-1. The JSP extension tag library provides the following services:

1. Retrieves a static text label from a resource bundle (implemented as a properties file).
2. Retrieves a message from a resource bundle (implemented as a properties file).
3. Initializes a page context with a particular language, country, and variant for label and message retrieval throughout a page.
4. Properly sets the content type (text/HTML) and character encoding for a page.

The Internationalization Framework makes it possible to dynamically retrieve all the strings that the user sees from the `<i18n:getMessage>` tag, and avoid embedding strings statically (that is, avoid hard-coding them) in your JSP page.

Figure 11-1 An Example of Internationalization Code


Before Internationalization
Hard coded text
After Internationalization
Points to a tag library
Scriptlet defines language
This tag sets the language and encoding for the page.
Page body
This tag gets the text out of the resource bundle, instead of hard coding.

```
<html>
<body>
Hello!
</body>
</html>
```

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%

// Definition of a single language preference
String language = "en";
%>

<i18n:localize language="<%=language%>"
bundleName="i18nExampleResourceBundle"/>
<html>
<body>
<i18n:getMessage messageName="greeting"/>
</body>
</html>
```

Localizing Your JSP

The conventions used in the I18N tag library are based on the more general conventions used to internationalize Java applications. To understand the conceptual foundations for the `<i18n:getMessage>`tag, see the *Javadoc* for `java.text.MessageFormat` in the Sun Microsystems, Inc. *Java 2 SDK, Standard Edition* documentation. To better understand the ideas that served as the foundation for these tags, study the *Javadoc* for `java.util.ResourceBundle` and `java.util.Locale`.

The following tags are included in the I18N framework:

`<i18n:getMessage>`

`<i18n:localize>`

<i18n:getMessage>

This tag retrieves a localized label or message (based on the absence/presence of an `args` attribute). The tag optionally takes a bundle name, language, country, and variant to aid in locating the appropriate properties file for resource bundle loading.

This tag is used in the localization of JSP pages. All pages that have an internationalization requirement should use this tag.

For more information about the `<i18n:getMessage>` tag, see Chapter 13, "Personalization Server JSP Tag Library Reference."

<i18n:localize>

This tag allows you to specify a language, country, variant, and resource bundle name to use throughout a page when accessing resource bundles via the `<i18n:getMessage>` tag. This is a convenient way to specify these attributes once, so that you do not have to specify them again each time you use `<i18n:getMessage>` to retrieve localized static text or messages.

Note: Changes to the resource bundles will not be recognized until the server is restarted.

The `<i18n:localize>` tag also specifies a character encoding and content type to be specified for a JSP page. Because of this, the tag should be used as early in the page as possible—before anything is written to the output stream—so that the bytes are properly encoded. If you intend to display text in more than one language, pick a character set that encompasses all the languages on the page.

When an HTML page is included in an enclosing page (for example, as portlets are included in portal pages), only the outermost page can use the `<i18n:localize>` tag. This is because the `<i18n:localize>` tag sets the encoding for the page, and the encoding must be set in the parent (outermost) page before any bytes are written to the response's output stream. Therefore, be careful that the encoding for the parent page is sufficient for all the content on that page as well as any included pages. The child (included) pages may continue to use the `<i18n:getMessage>` tag. However, if the included pages are using text from their own bundle, they must provide the `bundleName` parameter to the `<i18n:getMessage>` tag.

Note: If your page contains only dynamic strings (strings retrieved using the `<i18n:getMessage tag>`), then do not use the `<i18n:localize>` tag in conjunction with the `<%@ page contentType="<something>" >` page directive defined in the JSP specification. The directive is unnecessary if you are using the `<i18n:localize>` tag, and can result in inconsistent or wrong `contentType` declarations.

For more information about the `<i18n:localize>` tag, see Chapter 13, "Personalization Server JSP Tag Library Reference."

The JspMessageBundle

The `<i18n:getMessage>` tag uses an implementation similar to that of java.util.ResourceBundle, but it is slightly modified. Unlike a ResourceBundle, the `<i18n:getMessage>` tag looks only for properties files (like the PropertyResourceBundle) within the ServletContext (on the doc path). This means that you can keep properties files containing localized text relative to the associated JSP page, instead of having to have them on the `CLASSPATH`.

Another difference is that the resource bundles (properties files) used by `<i18n:getMessage>` are specified using the `"/"` character instead of the `"."`. For instance, the path to a JspMessageBundle might look like this: `/jsp/ordersystem/placeOrder`.

If a bundle name is specified, then it can be specified *absolutely* or *relatively*. Absolute paths are treated as such if they begin with a `"/"`. Paths not beginning with `"/"` are searched for relative to the JSP page's location.

If no bundle name is specified, then bundle name defaults to the name of the JSP page. For instance, if you have a JSP page called placeOrder.jsp, then `<i18n:getMessage>` would look in the same directory for a `placeOrder.properties` file to serve as the resource bundle for the placeOrder.jsp page.

When `<i18n:getMessage>`is searching for a resource bundle, it uses `pageContext.getServletContext().getResourceAsStream()` to load the resource bundle. Therefore, `<i18n:getMessage>`searches the web application for the property file rather than searching CLASSPATH. If no message bundle can be found, a MissingResourceException occurs.

How the Localization Tag Works

The `<i18n:localize>` tag first examines all provided attributes and default attributes, and then performs the following three steps:

1. **Determines the base bundle name.**

   If a base bundle name is not provided, the bundle name defaults to the name of the JSP page.

   For example, if the name of the JSP page is `placeOrder.jsp`, then the default bundle name would be `placeOrder`. On the file system, it would look for `placeOrder.properties`.

2. **Determines the language to use.**

   The tag will first look for resource bundles that correspond to the language parameter passed in to the tag.

   If no match between bundle and language is found, then the tag will try to find a match between resource bundles and languages defined in the request header.

   If a match can be made, the first language that matches is the language that is used.

   If no language is specified, the default is U.S. English (en_US).

   If no message bundle can be found, then language is set to nothing ("") and "UTF-8" encoding will be used unless otherwise specified.

3. **Determines which character encoding (charset) to use.**

   If character encoding is not specified, a charset appropriate for the language determined in step 2 is chosen.

   If a character encoding is specified, then that will be the charset used by the page, regardless of what language was chosen in step 2.

   Once the charset is determined, it is specified for the page by calling the `setContentType()` method on the servlet response. A call to `setContentType()` might look like this:

   ```
   response.setContentType("text/html; charset=ISO-8859-1");
   ```

Character Encoding

When specifying the encoding, it is important to note that some encodings may not be supported for your particular operating system, virtual machine, or client browsers. To see what Sun Microsystems, Inc. supports in the J2SE package, see http://www.java.sun.com.

If for any reason an encoding for a language cannot be determined and none is specified, UTF-8 encoding is used.

Displaying More Than One Character Set on a Page

In general, it is best is to leave the charset parameters unspecified since this is more flexible and fault tolerant. An exception might be when two languages (such as Greek and Japanese) need to be displayed in the same page. In that case, you can set the charset to "UTF-8".

For a page with multiple charsets to display correctly, the end users must have the appropriate fonts installed on their machines. If a font cannot be found, non-printable characters will typically display in place of the missing characters. (Non-printable characters often look like rows of empty boxes.)

Default Character Encodings

Table 11-23 shows how the `<i18n:localize>` tag maps languages to character encodings. These are the default settings.

You can override these defaults by providing any charset tag parameter you choose. For example, in the table below, the default charset for Japanese is Shift_JIS, but you could pass in x-sjis, EUC_JP, or iso-2022-jp instead. Or, as another example, to use Chinese Taiwan locale in place of Chinese, override GB2312 with Big5.

Table 11-23 Default Character Encodings

Language Code	Language Name	Character Encoding
ar	Arabic	ISO-8859-6
be	Byelorussian	ISO-8859-5

bg	Bulgarian	ISO-8859-5
ca	Catalan	ISO-8859-1
cs	Czech	ISO-8859-2
da	Danish	ISO-8859-1
de	German	ISO-8859-1
el	Greek	ISO-8859-7
en	English	ISO-8859-1
es	Spanish	ISO-8859-1
et	Estonian	ISO-8859-1
fi	Finnish	ISO-8859-1
fr	French	ISO-8859-1
hr	Croatian	ISO-8859-2
hu	Hungarian	ISO-8859-2
is	Icelandic	ISO-8859-1
it	Italian	ISO-8859-1
iw	Hebrew	ISO-8859-8
ja	Japanese	Shift_JIS
ko	Korean	EUC_KR
lt	Lithuanian	ISO-8859-2
lv	Latvian (Lettish)	ISO-8859-2
mk	Macedonian	ISO-8859-5
nl	Dutch	ISO-8859-1
no	Norwegian	ISO-8859-1
pl	Polish	ISO-8859-2

pt	Portuguese	ISO-8859-1
ro	Romanian	ISO-8859-2
ru	Russian	ISO-8859-5
sh	Serbo-Croatian	ISO-8859-5
sk	Slovak	ISO-8859-2
sl	Slovenian	ISO-8859-2
sq	Albanian	ISO-8859-2
sr	Serbian	ISO-8859-5
sv	Swedish	ISO-8859-1
th	Thai	TIS620
tr	Turkish	ISO-8859-9
uk	Ukrainian	ISO-8859-5
zh	Chinese	GB2312
other		UTF-8

Double-byte character encoding

The Internationalization property files in the WebLogic Personalization Server are standard Java property files, and as such, they do not accept non-ASCII characters. Non-ASCII characters must be converted to Unicode escapes before being embedded into these files. The native2ascii tool can be used to convert property files to and from other character encodings.

On Windows, the native2ascii tool is found in the bin directory of the JDK.

For more information about Java property files and the native2ascii tool, see http://java.sun.com/j2se/1.3/docs/api/java/util/Properties.html

For more information about Unicode escapes, see:
http://java.sun.com/docs/books/jls/second_edition/html/lexical.doc.html#100850

Example

To write “Hello World!” in English, the contents of the property file `helloWorld_en.properties` would look like this:

```
helloWorld=Hello World!
```

To write “Hello World!” in Japanese, the contents of the property file `helloWorld_ja.properties` would look like this:

```
helloWorld=\u3053\u3093\u306b\u3061\u306f\u3001\u4e16\u754c\u306e
\u4eba\u3005\uff01
```

This Unicode will render “Hello World!” in Japanese characters like this:

こんにちは、世界の人々！

Note: Japanese consists of Kanji and other character sets called Hiragana and Katakana. In this example, "Hello" is written in Hiragana and "World" is written in Kanji.

Steps for Localizing Your Application

1. Familiarize yourself with the documentation for the Internationalization `<i18n:*>` tags in Chapter 13, “Personalization Server JSP Tag Library Reference.”. For sample code, see Figure 11-1 “An Example of Internationalization Code” on page 11-3.

2. Include the `<i18n:localize>` tag in all outermost pages with an internationalization requirement. The tag should be used as early in the page as possible—before anything is written to the output stream—so that the bytes are properly encoded.

   For example:

   ```
   <%@ taglib uri="i18n.tld" prefix="i18n" %>
   <i18n:localize language="<%=language%>"
   ```

Note: When JSP pages are being included inside an enclosing page, only the enclosing page can use the `<i18n:localize>` tag.

3. Move all text that must be localized (including image URLs that must be localized) to property files that serve as resource bundles. Provide a resource bundle (property file) for each language you plan to support. One resource bundle per JSP page per language is the recommended approach.

Note: Changes to the property files will not be recognized until the server is restarted.

   For example: Use `<i18n:getMessaage messageName="greeting"/>` instead of hardcoding “Welcome!”

4. (Optional.) Specify a directory path for the property files (resource bundles). The bundle location must be specified *relative* to the JSP location, or *absolutely*, under the document root. This step is optional; if nothing is provided,the tag will look for the properties file where the JSP exists.

5. Refer to all localized text in a JSP page by using the `<i18n:getMessage>` tag. If a bundle name is specified, make sure that the `<i18n:getMessage>` tag is referring to the correct resource bundle location (relative or absolute path).

   For example:
   If the JSP is in `public_html\mypage.jsp`, then the bundle location could be (absolute) `"/mypage/text_us.properties" or`
   (relative) `"text_us.properties"`.

6. Test the page for all languages that you support. Make sure that the localized text and images display correctly and that the page layout is correct.

Code Examples

The following examples show how to use the JSP internationalization framework with JavaScript and Java scriptlets.

Using the JSP Internationalization Framework with JavaScript

This example displays a JavaScript dialog with a localized message in it.

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%
String language="en";
%>
<i18n:localize language="<%=language%>"
bundleName="i18nJavaScriptExampleResourceBundle"/>

<script language="JavaScript">
function popDialog() {
alert("<i18n:getMessage messageName="greeting"/>")
}
</script>

<html>
<body>
<a href="javascript:popDialog();">Click here to see localized
text!</a>
</body>
</html>
```

Using JSP Internationalization Framework with Java Scriptlets

This example gets a localized message, and uses that message in two Java scriptlets. One scriptlet prints to system out, the other inlines it into the page.

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%
String language="en";
%>
<i18n:localize language="<%=language%>"
bundleName="i18nJavaScriptExampleResourceBundle"/>

<html>
<body>
<i18n:getMessage messageName="greeting" id="theGreeting"/>
<p>
<%="Localized text for 'greeting': " + theGreeting%>
<p>
<%
System.out.println("Localized text for 'greeting': " +
theGreeting);
%>

</body>
</html>
```

CHAPTER

12 The WebLogic Personalization Server Database Schema

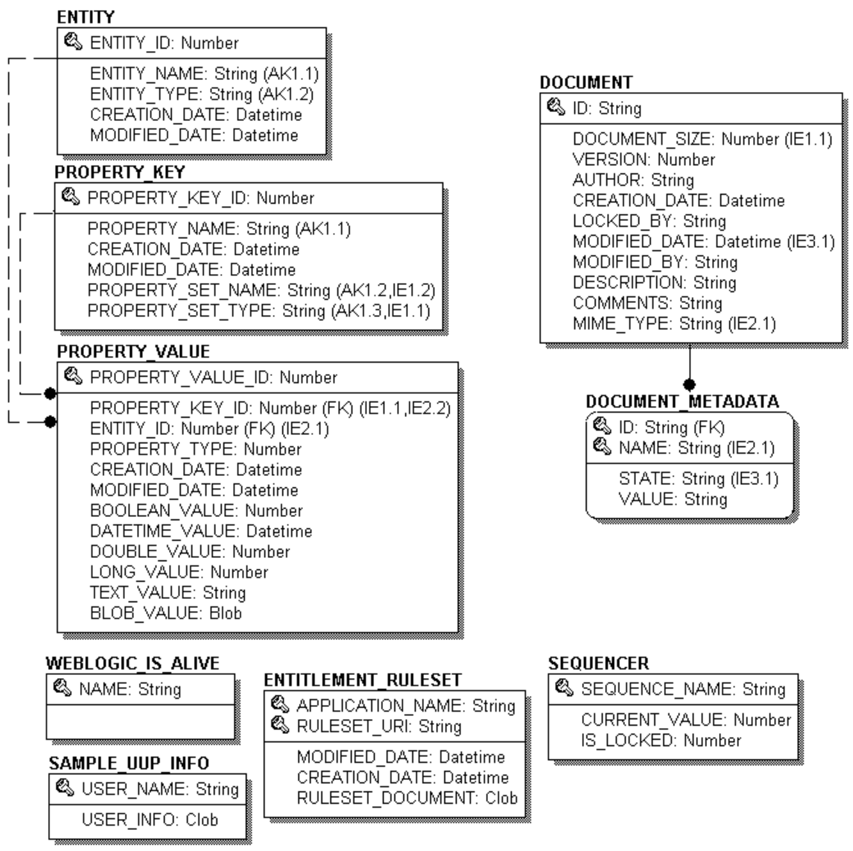
This topic documents the database schema for the WebLogic Personalization Server. This topic includes the following sections:

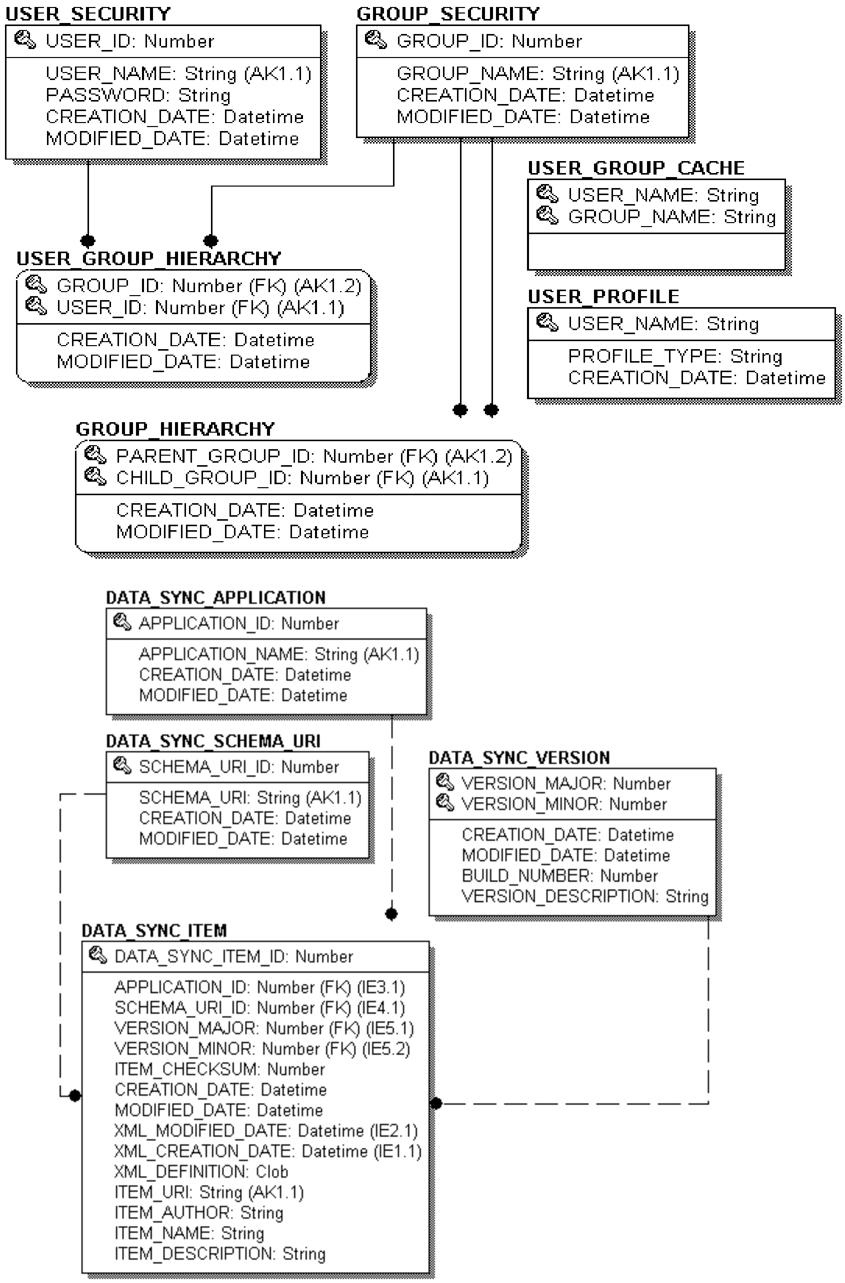
- The Entity-Relation Diagram
- List of Tables Comprising the WebLogic Personalization Server
- The Personalization Server Data Dictionary
- The SQL Scripts Used to Create the Database
- Defined Constraints

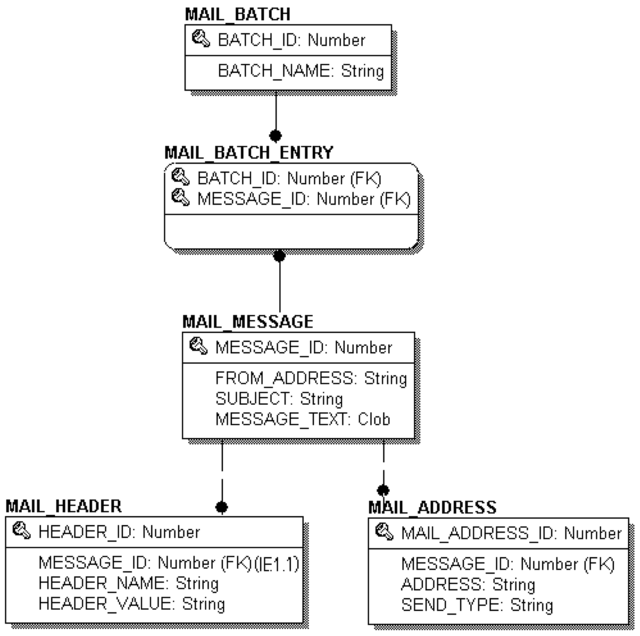
The Entity-Relation Diagram

Figure 12-1 shows the logical Entity-Relation diagram for the WebLogic Personalization Server database. See the subsequent sections in this topic for information about the data type syntax.

Figure 12-1 Entity-Relation Diagram for the WebLogic Personalization Server


ENTITY
ENTITY_ID: Number
ENTITY_NAME: String (AK1.1)
ENTITY_TYPE: String (AK1.2)
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
PROPERTY_KEY
PROPERTY_KEY_ID: Number
PROPERTY_NAME: String (AK1.1)
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
PROPERTY_SET_NAME: String (AK1.2,IE1.2)
PROPERTY_SET_TYPE: String (AK1.3,IE1.1)
PROPERTY_VALUE
PROPERTY_VALUE_ID: Number
PROPERTY_KEY_ID: Number (FK) (IE1.1,IE2.2)
ENTITY_ID: Number (FK) (IE2.1)
PROPERTY_TYPE: Number
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
BOOLEAN_VALUE: Number
DATETIME_VALUE: Datetime
DOUBLE_VALUE: Number
LONG_VALUE: Number
TEXT_VALUE: String
BLOB_VALUE: Blob
DOCUMENT
ID: String
DOCUMENT_SIZE: Number (IE1.1)
VERSION: Number
AUTHOR: String
CREATION_DATE: Datetime
LOCKED_BY: String
MODIFIED_DATE: Datetime (IE3.1)
MODIFIED_BY: String
DESCRIPTION: String
COMMENTS: String
MIME_TYPE: String (IE2.1)
DOCUMENT_METADATA
ID: String (FK)
NAME: String (IE2.1)
STATE: String (IE3.1)
VALUE: String
WEBLOGIC_IS_ALIVE
NAME: String
ENTITLEMENT_RULESET
APPLICATION_NAME: String
RULESET_URI: String
MODIFIED_DATE: Datetime
CREATION_DATE: Datetime
RULESET_DOCUMENT: Clob
SEQUENCER
SEQUENCE_NAME: String
CURRENT_VALUE: Number
IS_LOCKED: Number
SAMPLE_UUP_INFO
USER_NAME: String
USER_INFO: Clob


USER_SECURITY
USER_ID: Number
USER_NAME: String (AK1.1)
PASSWORD: String
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
GROUP_SECURITY
GROUP_ID: Number
GROUP_NAME: String (AK1.1)
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
USER_GROUP_CACHE
USER_NAME: String
GROUP_NAME: String
USER_GROUP_HIERARCHY
GROUP_ID: Number (FK) (AK1.2)
USER_ID: Number (FK) (AK1.1)
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
USER_PROFILE
USER_NAME: String
PROFILE_TYPE: String
CREATION_DATE: Datetime
GROUP_HIERARCHY
PARENT_GROUP_ID: Number (FK) (AK1.2)
CHILD_GROUP_ID: Number (FK) (AK1.1)
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
DATA_SYNC_APPLICATION
APPLICATION_ID: Number
APPLICATION_NAME: String (AK1.1)
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
DATA_SYNC_SCHEMA_URI
SCHEMA_URI_ID: Number
SCHEMA_URI: String (AK1.1)
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
DATA_SYNC_VERSION
VERSION_MAJOR: Number
VERSION_MINOR: Number
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
BUILD_NUMBER: Number
VERSION_DESCRIPTION: String
DATA_SYNC_ITEM
DATA_SYNC_ITEM_ID: Number
APPLICATION_ID: Number (FK) (IE3.1)
SCHEMA_URI_ID: Number (FK) (IE4.1)
VERSION_MAJOR: Number (FK) (IE5.1)
VERSION_MINOR: Number (FK) (IE5.2)
ITEM_CHECKSUM: Number
CREATION_DATE: Datetime
MODIFIED_DATE: Datetime
XML_MODIFIED_DATE: Datetime (IE2.1)
XML_CREATION_DATE: Datetime (IE1.1)
XML_DEFINITION: Clob
ITEM_URI: String (AK1.1)
ITEM_AUTHOR: String
ITEM_NAME: String
ITEM_DESCRIPTION: String


MAIL_BATCH
BATCH_ID: Number
BATCH_NAME: String
MAIL_BATCH_ENTRY
BATCH_ID: Number (FK)
MESSAGE_ID: Number (FK)
MAIL_MESSAGE
MESSAGE_ID: Number
FROM_ADDRESS: String
SUBJECT: String
MESSAGE_TEXT: Clob
MAIL_HEADER
HEADER_ID: Number
MESSAGE_ID: Number (FK)(IE1.1)
HEADER_NAME: String
HEADER_VALUE: String
MAIL_ADDRESS
MAIL_ADDRESS_ID: Number
MESSAGE_ID: Number (FK)
ADDRESS: String
SEND_TYPE: String

List of Tables Comprising the WebLogic Personalization Server

The WebLogic Personalization Server is comprised of the following tables. In this list, the tables are sorted by functionality:

Ads and Placeholders tables
- The AD_BUCKET Database Table
- The AD_COUNT Database Table
- The PLACEHOLDER_PREVIEW Database Table

Data Synchronization tables
- The DATA_SYNC_APPLICATION Database Table
- The DATA_SYNC_ITEM Database Table
- The DATA_SYNC_SCHEMA_URI Database Table
- The DATA_SYNC_VERSION Database Table

Documentation Management tables
- The DOCUMENT Database Table
- The DOCUMENT_METADATA Database Table

Mail tables
- The MAIL_ADDRESS Database Table
- The MAIL_BATCH Database Table
- The MAIL_BATCH_ENTRY Database Table
- The MAIL_HEADER Database Table
- The MAIL_MESSAGE Database Table

User Management tables
- The GROUP_HIERARCHY Database Table
- The GROUP_SECURITY Database Table
- The USER_GROUP_CACHE Database Table
- The USER_GROUP_HIERARCHY Database Table
- The USER_PROFILE Database Table
- The USER_SECURITY Database Table

Common tables used by both WebLogic Personalization Server and WebLogic Portal

The ENTITLEMENT_RULESET Database Table Database Table
The ENTITY Database Table
The PROPERTY_KEY Database Table
The PROPERTY_VALUE Database Table
The SAMPLE_UUP_INFO Database Tablee
The SEQUENCER Database Table
The WEBLOGIC_IS_ALIVE Database Table

The Personalization Server Data Dictionary

In this section, the WebLogic Personalization Server schema tables are arranged alphabetically as a data dictionary.

Note: Even though the following documentation references "foreign keys" to various tables, these constraints do not currently exist in this release of WebLogic Personalization Server. However, they will be (available in future releases) in place in future versions of WebLogic Personalization Server and we want you to be aware of these relationships now.

The AD_BUCKET Database Table

Table 12-1 describes the AD_BUCKET table. This table maintains content queries for ads.

The Primary Key is `AD_BUCKET_ID`.

Table 12-24 AD_BUCKET Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`AD_BUCKET_ID`	`NUMBER(15)`	`NOT NULL`	PK—a unique, system-generated number used as the record identifier.
`USER_NAME`	`VARCHAR (200)`	`NOT NULL`	The user's name associated with the ad.

Table 12-24 AD_BUCKET Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
PLACEHOLDER_XML_REF	VARCHAR(254)	NOT NULL	The location identifier of the XML-based placeholder definition file.
APPLICATION_NAME	VARCHAR(100)	NOT NULL	The name of the application for which the ad has been scoped.
CONTEXT_REF	VARCHAR(254)	NULL	The scenario unique identifier.
CONTAINER_REF	VARCHAR(254)	NULL	The campaign unique identifier.
CONTAINER_TYPE	VARCHAR(50)	NULL	Identifies the service associated with the CONTAINER_REF.
WEIGHT	NUMBER(15)	NULL	A weighted scheme used in prioritizing one placeholder over another.
VIEW_COUNT	NUMBER(15)	NULL	*Disabled. Reserved for future use.*
EXPIRATION_DATE	DATE	NULL	The date and time the ad expires or becomes invalid.
CREATION_DATE	DATE	NOT NULL	The date and time this record was created.
MODIFIED_DATE	DATE	NOT NULL	The date and time this record was last modified.
AD_QUERY	CLOB	NULL	The actual content query.

The AD_COUNT Database Table

Table 12-2 describes the AD_COUNT table. This table tracks the number of times the ads are displayed and clicked though.

The Primary Key is comprised of AD_ID, CONTAINER_REF, and APPLICATION_NAME.

Table 12-25 AD_COUNT Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`AD_ID`	`VARCHAR(254)`	`NOT NULL`	A unique, system-generated number used as the record identifier.
`CONTAINER_REF`	`VARCHAR(254)`	`NOT NULL`	The campaign unique identifier.
`APPLICATION_NAME`	`VARCHAR(100)`	`NOT NULL`	The name of the application for which the ad clicks or views were scoped
`DISPLAY_COUNT`	`NUMBER(15)`	`NOT NULL`	The number of times the ad has been displayed.
`CLICK_THROUGH_COUNT`	`NUMBER(15)`	`NOT NULL`	The number of times the ad has been clicked on.

The DATA_SYNC_APPLICATION Database Table

Table 12-3 describes the DATA_SYNC_APPLICATION table. This table holds the various applications available for the data synchronization process..

The Primary Key is `APPLICATION_ID`.

Table 12-26 DATA_SYNC_APPLICATION Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`APPLICATION_ID`	`NUMBER(15)`	`NOT NULL`	PK - A unique, system-generated number used as the record identifier.
`APPLICATION_NAME`	`VARCHAR(100)`	`NOT NULL`	The deployed J2EE application name. (This should match the name in the WebLogic Server console.)
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time this record was created.
`MODIFIED_DATE`	`DATE`	`NOT NULL`	The date and time this record was last modified.

The DATA_SYNC_ITEM Database Table

Table 12-4 describes the DATA_SYNC_ITEM table. This table stores all the data items to be synchronized.

For information on defined constraints in this table, see “Defined Constraints” on page 12-30.

The Primary Key is `DATA_SYNC_ITEM_ID`.

Table 12-27 DATA_SYNC_ITEM Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`DATA_SYNC_ITEM_ID`	`NUMBER(15)`	`NOT NULL`	PK - A unique, system-generated number used as the record identifier.
`APPLICATION_ID`	`NUMBER(15)`	`NOT NULL`	FK – to `DATA_SYNC_APPLICATON.APPLICATION_ID`
`SCHEMA_URI_ID`	`NUMBER(15)`	`NOT NULL`	FK – to `DATA_SYNC_SCHEMA_URI.SCHEMA_URI_ID`
`VERSION_MAJOR`	`NUMBER(15)`	`NOT NULL`	FK – to `DATA_SYNC_VERSION.VERSION_MAJOR`
`VERSION_MINOR`	`NUMBER(15)`	`NOT NULL`	FK – to `DATA_SYNC_VERSION.VERSION_MINOR`
`ITEM_CHECKSUM`	`NUMBER(15)`	`NOT NULL`	A generated number representing the contents of the `XML_DEFINITION` column.
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time this record was created.
`MODIFIED_DATE`	`DATE`	`NOT NULL`	The date and time this record was last modified.
`XML_MODIFIED_DATE`	`DATE`	`NOT NULL`	The date and time the XML file was last modified.
`XML_CREATION_DATE`	`DATE`	`NOT NULL`	The date and time the XML file was created.

Table 12-27 DATA_SYNC_ITEM Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
XML_DEFINITION	CLOB	NOT NULL	The XML representation of the data item to be synchronized.
ITEM_URI	VARCHAR(254)	NOT NULL	The path on the file system of the data item to be synchronized.
ITEM_AUTHOR	VARCHAR(200)	NULL	Metadata info—the o/s login.
ITEM_NAME	VARCHAR(100)	NULL	Metadata info—the full path to the item.
ITEM_DESCRIPTION	VARCHAR(254)	NULL	Metadata info—a general description of the item to be synchronized.

The DATA_SYNC_SCHEMA_URI Database Table

Table 12-5 describes the DATA_SYNC_SCHEMA_URI table. This table holds information pertaining to each of the governing schemas used by various documents.

The Primary Key is `SCHEMA_URI_ID`.

Table 12-28 DATA_SYNC_SCHEMA_URI Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
SCHEMA_URI_ID	NUMBER(15)	NOT NULL	PK - A unique, system-generated number used as the record identifier.
SCHEMA_URI	VARCHAR(254)	NOT NULL	The governing schema of the document.
CREATION_DATE	DATE	NOT NULL	The date and time this record was created.
MODIFIED_DATE	DATE	NOT NULL	The date and time this record was last modified.

The DATA_SYNC_VERSION Database Table

Table 12-6 describes the DATA_SYNC_VERSION table. This table is not being used currently. It is reserved for future use and is expected to accommodate data synchronization versioning. As a result, this table only holds one record.

The Primary Key is comprised of both `VERSION_MAJOR` and `VERSION_MINOR`.

Table 12-29 DATA_SYNC_VERSION Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
VERSION_MAJOR	NUMBER(15)	NOT NULL	The current record has a value of zero.
VERSION_MINOR	NUMBER(15)	NOT NULL	The current record has a value of zero.
CREATION_DATE	DATE	NOT NULL	The date and time the record was created.
MODIFIED_DATE	DATE	NOT NULL	The date and time the record was last modified.
BUILD_NUMBER	NUMBER(15)	NULL	The build number associated with the version.
VERSION_DESCRIPTION	VARCHAR(30)	NULL	A description of the particular sync version.

The DOCUMENT Database Table

Table 12-7 describes the DOCUMENT table. This table is used to store information pertinent to each document used within the WebLogic Personalization Server.

The Primary Key is `ID`.

Table 12-30 DOCUMENT Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
ID	VARCHAR(254)	NOT NULL	The identifier of the document. This specifies the relative path (case sensitive using forward slashes) to the actual file.

Table 12-30 DOCUMENT Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
`DOCUMENT_SIZE`	`NUMBER(15)`	`NOT NULL`	The size of the document in bytes.
`VERSION`	`NUMBER(15)`	`NULL`	The version of the document.
`AUTHOR`	`VARCHAR(50)`	`NULL`	The author's name of this document.
`CREATION_DATE`	`DATE`	`NULL`	The date this document was created in the system.
`LOCKED_BY`	`VARCHAR(50)`	`NULL`	This column identifies who has this document locked for edits or updates.
`MODIFIED_DATE`	`DATE`	`NULL`	The date and time this record was last modified.
`MODIFIED_BY`	`VARCHAR(50)`	`NULL`	This column stores the name of the individual who last modified the document record.
`DESCRIPTION`	`VARCHAR(2000)`	`NULL`	A description of the document.
`COMMENTS`	`VARCHAR(2000)`	`NULL`	An area to store miscellaneous notes about the document.
`MIME_TYPE`	`VARCHAR(100)`	`NOT NULL`	This column identifies which MIME type (or file type) is associated with this document. This is supposed to be MIME 1.0.

The DOCUMENT_METADATA Database Table

Table 12-8 describes the DOCUMENT_METADATA table. This table is used to store user-defined properties associated with each document.

For information on defined constraints in this table, see "Defined Constraints" on page 12-30.

The Primary Key is comprised of both `ID` and `NAME`.

Table 12-31 DOCUMENT_METADATA Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`ID`	`VARCHAR(254)`	`NOT NULL`	The document identifier. This is a foreign key to the `ID` column of the `DOCUMENT` table.
`NAME`	`VARCHAR(240)`	`NOT NULL`	The metadata name for the document.
`STATE`	`VARCHAR(50)`	`NULL`	The current state of this metadata property. This is used by Interwoven and can be set to null.
`VALUE`	`VARCHAR(2000)`	`NULL`	The value to be associated with the metadata name (`NAME`).

The ENTITLEMENT_RULESET Database Table

Table 12-9 describes the ENTITLEMENT_RULESET table. This table stores the access decision rules used by the Entitlements Engine.

The Primary Key is comprised of both `APPLICATION_NAME` and `RULESET_URI`.

Table 12-32 ENTITLEMENT_RULESET Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`APPLICATION_NAME`	`VARCHAR(100)`	`NOT NULL`	PK – A unique application name within a J2EE server.

Table 12-32 ENTITLEMENT_RULESET Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
RULESET_URI	VARCHAR(254)	NOT NULL	The URI used to identify an entitlement access decision rule.
CREATION_DATE	DATE	NOT NULL	The date and time this record was created.
MODIFIED_DATE	DATE	NOT NULL	The date and time this record was last modified.
RULESET_DOCUMENT	CLOB	NULL	The XML document describing an access decision rule.

The ENTITY Database Table

Table 12-10 describes the ENTITY table. Any ConfigurableEntity within the system will have an entry in this table.

The Primary Key is `ENTITY_ID`.

Table 12-33 ENTITY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
ENTITY_ID	NUMBER(15)	NOT NULL	PK - A unique, sequence-generated number used as the record identifier.
ENTITY_NAME	VARCHAR(200)	NOT NULL	The name of the ConfigurableEntity.
ENTITY_TYPE	VARCHAR(100)	NOT NULL	Defines what type of ConfigurableEntity this is.
CREATION_DATE	DATE	NOT NULL	The date and time this record was created.
MODIFIED_DATE	DATE	NOT NULL	The date and time this record was last modified.

The GROUP_HIERARCHY Database Table

Table 12-11 describes the PARENT_CHILD_GROUP table. This table stores relationship information between groups.

For information on defined constraints in this table, see "Defined Constraints" on page 12-30.

The Primary Key is comprised of both `PARENT_GROUP_ID` and `CHILD_GROUP_ID`.

Table 12-34 GROUP_HIERARCHY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`PARENT_GROUP_ID`	`NUMBER(15)`	`NOT NULL`	The parent group identifier. This column is a foreign key to the `ENTITY_ID` column in the `ENTITY` table.
`CHILD_GROUP_ID`	`NUMBER(15)`	`NOT NULL`	The child group identifier. This column is a foreign key to the `ENTITY_ID` column in the `ENTITY` table.
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time this record was created.
`MODIFIED_DATE`	`DATE`	`NOT NULL`	The date and time this record was last modified.

The GROUP_SECURITY Database Table

Table 12-12 describes the GROUP_SECURITY table. This table stores relationship information between groups.

The Primary Key is `GROUP_ID`.

Table 12-35 GROUP_SECURITY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`GROUP_ID`	`NUMBER(15)`	`NOT NULL`	PK – a unique, system-generated number used as the record identifier.

Table 12-35 GROUP_SECURITY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`GROUP_NAME`	`VARCHAR(200)`	`NOT NULL`	The name of the group.
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time this record was created.
`MODIFIED_DATE`	`DATE`	`NOT NULL`	The date and time this record was last modified.

The MAIL_ADDRESS Database Table

Table 12-13 describes the metadata for the E-Business Control Center MAIL_ADDRESS table. This table stores all of the address info for e-mail purposes.

For information on defined constraints in this table, see "Defined Constraints" on page 12-30.

The Primary Key is `MAIL_ADDRESS_ID`.

Table 12-36 MAIL_ADDRESS Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`MAIL_ADDRESS_ID`	`NUMBER(15)`	`NOT NULL`	PK—a unique, system-generated number to be used as the record `ID`.
`MESSAGE_ID`	`NUMBER(15)`	`NOT NULL`	FK—foreign key to the `MAIL_MESSAGE` table.
`ADDRESS`	`VARCHAR(254)`	`NOT NULL`	Stores the various e-mail addresses on the distribution list.
`SEND_TYPE`	`VARCHAR(4)`	`NOT NULL`	Determines how the `ADDRESS` should be included on the distribution. Possible values are `TO, CC,` or `BCC`.

The MAIL_BATCH Database Table

Table 12-14 describes the metadata for the E-Business Control Center MAIL_BATCH table. This table establishes a batch for each mailing.

The Primary Key is `BATCH_ID`.

Table 12-37 MAIL_BATCH Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`BATCH_ID`	`NUMBER(15)`	`NOT NULL`	PK—a unique, system-generated number to be used as the record `ID`.
`BATCH_NAME`	`VARCHAR(254)`	`NOT NULL`	The name of the mail message batch.

The MAIL_BATCH_ENTRY Database Table

Table 12-15 describes the metadata for the E-Business Control Center MAIL_BATCH_ENTRY table. This table is used to correlate the mail batch with the specific mail message.

For information on defined constraints in this table, see “Defined Constraints” on page 12-30.

The Primary Keys are `BATCH_ID` and `MESSAGE_ID`.

Table 12-38 MAIL_BATCH_ENTRY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`BATCH_ID`	`NUMBER(15)`	`NOT NULL`	PK and FK—a unique, system-generated number to be used as the record `ID`.
`MESSAGE_ID`	`NUMBER(15)`	`NOT NULL`	PK and FK—foreign key to the `MAIL_MESSAGE` table.

The MAIL_HEADER Database Table

Table 12-16 describes the metadata for the E-Business Control Center MAIL_HEADER table. This table contains all of the header information specific to the e-mail message.

For information on defined constraints in this table, see "Defined Constraints" on page 12-30.

The Primary Key is `HEADER_ID`.

Table 12-39 MAIL_HEADER Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`HEADER_ID`	`NUMBER(15)`	`NOT NULL`	PK—a unique, system-generated number to be used as the record `ID`.
`MESSAGE_ID`	`NUMBER(15)`	`NOT NULL`	FK—foreign key to the `MAIL_MESSAGE` table.
`HEADER_NAME`	`VARCHAR(50)`	`NULL`	The name of the mail message header.
`HEADER_VALUE`	`VARCHAR(254)`	`NULL`	The value of the mail message header.

The MAIL_MESSAGE Database Table

Table 12-17 describes the metadata for the E-Business Control Center MAIL_MESSAGE table. This table contains the specifics of the mail message (e.g., the subject line, text, etc.).

The Primary Key is `MESSAGE_ID`.

Table 12-40 MAIL_MESSAGE Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`MESSAGE_ID`	`NUMBER(15)`	`NOT NULL`	PK—a unique, system-generated number to be used as the record `ID`.
`FROM_ADDRESS`	`VARCHAR(254)`	`NULL`	Identifies who is sending the message.

Table 12-40 MAIL_MESSAGE Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
SUBJECT	VARCHAR(128)	NULL	Stores the mail message subject.
MESSAGE_TEXT	CLOB	NULL	Holds the content of the mail message.

The PLACEHOLDER_PREVIEW Database Table

Table 12-18 describes the PLACEHOLDER_PREVIEW table. This table is used as a mechanism to hold the placeholder for previewing purposes only.

The Primary Key is `PREVIEW_ID`.

Table 12-41 PLACEHOLDER_PREVIEW Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
PREVIEW_ID	NUMBER	NOT NULL	PK—a unique, system generated number used as the record identifier.
XML_DEFINITION	CLOB	NULL	The representation of the expression to be previewed.

The PROPERTY_KEY Database Table

Table 12-19 describes the PROPERTY_KEY table. Any property assigned to a ConfigurableEntity has a unique PROPERTY_ID. This identifier and associated information is stored here.

The Primary Key is `PROPERTY_KEY_ID`.

Table 12-42 PROPERTY_KEY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
PROPERTY_KEY_ID	NUMBER(15)	NOT NULL	PK—a unique, system-generated number used as the record identifier.

Table 12-42 PROPERTY_KEY Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
`PROPERTY_NAME`	`VARCHAR(100)`	`NOT NULL`	The name of the property.
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time this record was created.
`MODIFIED_DATE`	`DATE`	`NOT NULL`	The date and time this record was last modified.
`PROPERTY_SET_NAME`	`VARCHAR(100)`	`NULL`	The name of the property set.
`PROPERTY_SET_TYPE`	`VARCHAR(100)`	`NULL`	The type the property set.

The PROPERTY_VALUE Database Table

Table 12-20 describes the PROPERTY_VALUE table. This table stores property values for boolean, datetime, float, integer, text, and user-defined properties.

For information on defined constraints in this table, see “Defined Constraints” on page 12-30.

The Primary Key is `PROPERTY_VALUE_ID`.

Table 12-43 PROPERTY_VALUE Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`PROPERTY_VALUE_ID`	`NUMBER(15)`	`NOT NULL`	PK – a unique, system-generated number used as the record identifier.
`PROPERTY_KEY_ID`	`NUMBER(15)`	`NOT NULL`	FK - to `PROPERTY_KEY.PROPERTY_KEY_ID`
`ENTITY_ID`	`NUMBER(15)`	`NOT NULL`	FK – to `ENTITY.ENTITY_ID`
`PROPERTY_TYPE`	`NUMBER(1)`	`NOT NULL`	Valid entries are: 0=Boolean, 1=Integer, 2=Float, 3=Text, 4=Date and Time, 5=User-Defined (BLOB)
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time this record was created.

Table 12-43 PROPERTY_VALUE Table Metadata (Continued)

Column Name	Data Type	Null Value	Description and Recommendations
`MODIFIED_DATE`	`DATE`	`NOT NULL`	The date and time this record was last modified.
`BOOLEAN_VALUE`	`NUMBER(1)`	`NULL`	The value for each boolean property identifier.
`DATETIME_VALUE`	`DATE`	`NULL`	The value for each date and time property identifier.
`DOUBLE_VALUE`	`NUMBER`	`NULL`	The value associated with each float property identifier.
`LONG_VALUE`	`NUMBER(20)`	`NULL`	The value associated with the integer property.
`TEXT_VALUE`	`VARCHAR(254)`	`NULL`	The value associated with the text property.
`BLOB_VALUE`	`BLOB`	`NULL`	The value associated with the user-defined property.

The SAMPLE_UUP_INFO Database Table

Table 12-21 describes the SAMPLE_UUP_INFO table. This is an example of how to use the Unified Profile Types.

The Primary Key is `USER_NAME`.

Table 12-44 SAMPLE_UUP_INFO Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`USER_NAME`	`VARCHAR(100)`	`NOT NULL`	A username.
`USER_INFO`	`CLOB`	`NOT NULL`	User data stored in XML representation.

The SEQUENCER Database Table

Table 12-22 describes the SEQUENCER table. The SEQUENCER table is used to maintain all of the sequence identifiers (for example, property_meta_data_id_sequence, and so on) used in the application.

For information on defined constraints in this table, see "Defined Constraints" on page 12-30.

The Primary Key is `SEQUENCE_NAME`.

Table 12-45 SEQUENCER Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`SEQUENCE_NAME`	`VARCHAR(50)`	`NOT NULL`	PK – A unique name used to identify the sequence.
`CURRENT_VALUE`	`NUMBER(15)`	`NOT NULL`	The current value of the sequence.
`IS_LOCKED`	`NUMBER(1)`	`NOT NULL`	This flag identifies whether or not the particular `SEQUENCE_ID` has been locked for update. This column is being used as a generic locking mechanism that can be used for multiple database environments.

The USER_GROUP_CACHE Database Table

Table 12-23 describes the USER_GROUP_CACHE table. In the event of a deep group hierarchy, this table will flatten the group hierarchy and enables quick group membership searches.

Note: The startup process GroupCache is disabled by default. This table will only be used if enabled.

The Primary Key is comprised of both USER_NAME and GROUP_NAME.

Table 12-46 USER_GROUP_CACHE Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
USER_NAME	VARCHAR(200)	NOT NULL	A user's name.
GROUP_NAME	VARCHAR(200)	NOT NULL	A group name.

The USER_GROUP_HIERARCHY Database Table

Table 12-24 describes the USER_GROUP_HIERARCHY table. This table allows you to store associated users and groups.

For information on defined constraints in this table, see "Defined Constraints" on page 12-30.

The Primary Key is comprised of both GROUP_ID and USER_ID.

Table 12-47 USER_GROUP_HIERARCHY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
GROUP_ID	NUMBER(15)	NOT NULL	FK – to USER_SECURITY.USER_ID
USER_ID	NUMBER(15)	NOT NULL	FK – to GROUP_SECURITY.GROUP_ID
CREATION_DATE	DATE	NOT NULL	The date and time this record was created.
MODIFIED_DATE	DATE	NOT NULL	The date and time this record was last modified.

The USER_PROFILE Database Table

Table 12-25 describes the USER_PROFILE table. This table stores all user login/password combinations.

The Primary Key is `USER_NAME`.

Table 12-48 USER_PROFILE Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`USER_NAME`	`VARCHAR(200)`	`NOT NULL`	PK - The name of the user.
`PROFILE_TYPE`	`VARCHAR(100)`	`NOT NULL`	A type of profile associated with the user (such as `WLCS_Customer`).
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time this record was created.

The USER_SECURITY Database Table

Table 12-26 describes the USER_SECURITY table. This table holds all the user records for security authentication of the rdbms realm.

The Primary Key is `USER_ID`.

Table 12-49 USER_SECURITY Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`USER_ID`	`NUMBER(15)`	`NOT NULL`	PK—a unique, system-generated number used as the record identifier.
`USER_NAME`	`VARCHAR(200)`	`NOT NULL`	The user's name.
`PASSWORD`	`VARCHAR(50)`	`NULL`	The user's password.
`CREATION_DATE`	`DATE`	`NOT NULL`	The date and time this record was created.
`MODIFIED_DATE`	`DATE`	`NOT NULL`	The date and time this record was last modified.

The WEBLOGIC_IS_ALIVE Database Table

Table 12-27 describes the WEBLOGIC_IS_ALIVE table. This table is used by the JDBC connection pools to insure the connection to the database is still alive.

The Primary Key is `NAME`.

Table 12-50 WEBLOGIC_IS_ALIVE Table Metadata

Column Name	Data Type	Null Value	Description and Recommendations
`NAME`	`VARCHAR(100)`	`NOT NULL`	Used by the JDBC connection pools to insure the connection to the database is still alive.

The SQL Scripts Used to Create the Database

The database schemas for WebLogic Portal and WebLogic Personalization Server are all created by executing the `create_all` script for the target database environment.

Scripts

Regardless of your database, execute one of the following to generate the necessary database objects for the modules desired (WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal):

- `P13N_HOME\db\create_all.bat` (Windows)
- `P13N_HOME/db/create_all.sh` (UNIX)

The following are the various directories underneath
`WL_COMMERCE_HOME/db`
(as seen in a UNIX environment):
`P13N_HOME/db/cloudscape/351`
`P13N_HOME/db/oracle/817`

Note: In this documentation,`P13N_HOME` is used to designate the directory where the WebLogic Personalization Server product is installed.

Each of the databases supported have the same number of scripts in each of their subdirectories. The scripts are listed and described in Table 12-51 below.

Table 12-51 The Scripts Supporting the Databases

Script Name	Description
`create_all.bat`	Windows script used to connect to the database and create the necessary database objects for the modules desired (e.g., WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal)

Table 12-51 The Scripts Supporting the Databases (Continued)

Script Name	Description
`create_all.sh`	Unix script used to connect to the database and create the necessary database objects for the modules desired (e.g., WebLogic Portal, WebLogic Personalization Server, Commerce services, Campaign services and Sample Portal)
`campaign_create_fkeys.sql`	SQL script used to create all foreign keys associated with the Campaign services.
`campaign_create_indexes.sql`	SQL script used to create all indexes associated with the Campaign services.
`campaign_create_tables.sql`	SQL script used to create all tables associated with the Campaign services.
`campaign_create_triggers.sql`	SQL script used to create all database triggers associated with the Campaign services.
`campaign_create_views.sql`	SQL script used to create all views associated with the Campaign services.
`campaign_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the Campaign services.
`campaign_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the Campaign services.
`campaign_drop_indexes.sql`	SQL script used to drop all indexes associated with the Campaign services.
`campaign_drop_tables.sql`	SQL script used to drop all tables associated with the Campaign services.
`campaign_drop_views.sql`	SQL script used to drop all views associated with the Campaign services.
`p13n_create_fkeys.sql`	SQL script used to create all foreign keys associated with the WebLogic Personalization Server.
`p13n_create_indexes.sql`	SQL script used to create all indexes associated with the WebLogic Personalization Server.
`p13n_create_tables.sql`	SQL script used to create all tables associated with the WebLogic Personalization Server.
`p13n_create_triggers.sql`	SQL script used to create all database triggers associated with the WebLogic Personalization Server.

Table 12-51 The Scripts Supporting the Databases (Continued)

Script Name	Description
`p13n_create_views.sql`	SQL script used to create all views associated with the WebLogic Personalization Server.
`p13n_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the WebLogic Personalization Server.
`p13n_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the WebLogic Personalization Server.
`p13n_drop_indexes.sql`	SQL script used to drop all indexes associated with the WebLogic Personalization Server.
`p13n_drop_tables.sql`	SQL script used to drop all tables associated with the WebLogic Personalization Server.
`p13n_drop_views.sql`	SQL script used to drop all views associated with the WebLogic Personalization Server.
`portal_create_fkeys.sql`	SQL script used to create all foreign keys associated with the WebLogic Portal.
`portal_create_indexes.sql`	SQL script used to create all indexes associated with the WebLogic Portal.
`portal_create_tables.sql`	SQL script used to create all tables associated with the WebLogic Portal.
`portal_create_triggers.sql`	SQL script used to create all database triggers associated with the WebLogic Portal.
`portal_create_views.sql`	SQL script used to create all views associated with the WebLogic Portal.
`portal_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the WebLogic Portal.
`portal_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the WebLogic Portal.
`portal_drop_indexes.sql`	SQL script used to drop all indexes associated with the WebLogic Portal.
`portal_drop_tables.sql`	SQL script used to drop all tables associated with the WebLogic Portal.
`portal_drop_views.sql`	SQL script used to drop all views associated with the WebLogic Portal.

Table 12-51 The Scripts Supporting the Databases (Continued)

Script Name	Description
`sample_portal_create_fkeys.sql`	SQL script used to create all foreign keys associated with the Sample Portal.
`sample_portal_create_indexes.sql`	SQL script used to create all indexes associated with the Sample Portal.
`sample_portal_create_tables.sql`	SQL script used to create all tables associated with the Sample Portal.
`sample_portal_create_triggers.sql`	SQL script used to create all database triggers associated with the Sample Portal.
`sample_portal_create_views.sql`	SQL script used to create all views associated with the Sample Portal.
`sample_portal_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the Sample Portal.
`sample_portal_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the Sample Portal.
`sample_portal_drop_indexes.sql`	SQL script used to drop all indexes associated with the Sample Portal.
`sample_portal_drop_tables.sql`	SQL script used to drop all tables associated with the Sample Portal.
`sample_portal_drop_views.sql`	SQL script used to drop all views associated with the Sample Portal.
`wlcs_create_fkeys.sql`	SQL script used to create all foreign keys associated with the Commerce services.
`wlcs_create_indexes.sql`	SQL script used to create all indexes associated with the Commerce services.
`wlcs_create_tables.sql`	SQL script used to create all tables associated with the Commerce services.
`wlcs_create_triggers.sql`	SQL script used to create all database triggers associated with the Commerce services.
`wlcs_create_views.sql`	SQL script used to create all views associated with the Commerce services.
`wlcs_drop_constraints.sql`	SQL script used to drop all constraints (other than foreign keys) associated with the Commerce services.

Table 12-51 The Scripts Supporting the Databases (Continued)

Script Name	Description
`wlcs_drop_fkeys.sql`	SQL script used to drop all foreign key constraints associated with the Commerce services.
`wlcs_drop_indexes.sql`	SQL script used to drop all indexes associated with the Commerce services.
`wlcs_drop_tables.sql`	SQL script used to drop all tables associated with the Commerce services.
`wlcs_drop_views.sql`	SQL script used to drop all views associated with the Commerce services.

Defined Constraints

Various constraints are defined and used in the WebLogic Personalization Server database schema. These constraints can be found in the following scripts:

`p13n_create_fkeys.sql`—contains the Foreign Keys

`p13n_create_tables.sql`—contains the Check Constraints

Table 12-52 Constraints Defined on WebLogic Personalization Server Database Tables

Table Name	Constraints
DATA_SYNC_ITEM	**Column**—APPLICATION_ID **Constraint—**FK1_SYNC_ITEM **Constraint Type—**FOREIGN KEY Ensures that each DATA_SYNC_ITEM references an existing DATA_SYNC_APPLICATION via the APPLICATION_ID column. **Column**—SCHEMA_URI_ID **Constraint—**FK2_SYNC_ITEM **Constraint Type—**FOREIGN KEY Ensures that each DATA_SYNC_ITEM references an existing DATA_SYNC_SCHEMA_URI via the SCHEMA_URI_ID column. **Columns**—VERSION_MAJOR and VERSION_MINOR **Constraint—**FK3_SYNC_ITEM **Constraint Type—**FOREIGN KEY Ensures that each DATA_SYNC_ITEM references an existing DATA_SYNC_VERSION via the VERSION_MAJOR, VERSION_MINOR columns.
DOCUMENT_METADATA	**Column**—ID **Constraint—** FK1_DOCUMENT_MD **Constraint Type—**FOREIGN KEY Ensures that each DOCUMENT_METADATA references an existing DOCUMENT via the ID column.
GROUP_HIERARCHY	**Column**—PARENT_GROUP_ID **Constraint—**FK1_GROUP_HRCHY **Constraint Type—**FOREIGN KEY Ensures that each PARENT_GROUP_HIERARCHY references an existing GROUP_SECURITY via the GROUP_ID column. **Column**—CHILD_GROUP_ID **Constraint—**FK2_GROUP_HRCHY **Constraint Type—**FOREIGN KEY Ensures that each CHILD_GROUP_HIERARCHY references an existing GROUP_SECURITY via the GROUP_ID column.
MAIL_ADDRESS	**Column**—MESSAGE_ID **Constraint—**FK1_MAIL_ADDRESS **Constraint Type—**FOREIGN KEY Ensures that each MAIL_ADDRESS references an existing MAIL_MESSAGE via the MESSAGE_ID column.

Table 12-52 Constraints Defined on WebLogic Personalization Server Database Tables

Table Name	Constraints
`MAIL_BATCH_ENTRY`	**Column**—BATCH_ID **Constraint—**FK1_MB_ENTRY **Constraint Type—**FOREIGN KEY Ensures that each `MAIL_BATCH_ENTRY` references an existing `MAIL_BATCH` via the `BATCH_ID` column. **Column**—MESSAGE_ID **Constraint—**FK2_MB_ENTRY **Constraint Type—**FOREIGN KEY Ensures that each `MAIL_BATCH_ENTRY` references an existing `MAIL_MESSAGE` via the `MESSAGE_ID` column.
`MAIL_HEADER`	**Column**—FK1_MAIL_HEADER **Constraint—**FK1_MAIL_HEADER **Constraint Type—**FOREIGN KEY Ensures that each `MAIL_HEADER` references an existing `MAIL_MESSAGE` via the `FK1_MAIL_HEADER` column.
`USER_GROUP_HIERARCHY`	**Column**—USER_ID **Constraint—**FK1_USER_G_HRCHY **Constraint Type—**FOREIGN KEY Ensures that each `USER_GROUP_HIERARCHY` references an existing `USER_SECURITY` via the `USER_ID` column. **Column**—GROUP_ID **Constraint—**FK2_USER_G_HRCHY **Constraint Type—**FOREIGN KEY Ensures that each `USER_GROUP_HIERARCHY` references an existing `GROUP_SECURITY` via the `GROUP_ID` column.

Table 12-52 Constraints Defined on WebLogic Personalization Server Database Tables

Table Name	Constraints
`PROPERTY_VALUE`	**Column**—ENTITY_ID **Constraint—**FK1_PROP_VALUE **Constraint Type—**FOREIGN KEY Ensures that each `PROPERTY_VALUE` references an existing `ENTITY` via the `ENTITY_ID` column. **Column**—PROPERTY_KEY_ID **Constraint—**FK2_PROP_VALUE **Constraint Type—**FOREIGN KEY Ensures that each `PROPERTY_VALUE` references an existing `PROPERTY_KEY` via the `PROPERTY_KEY_ID` column. **Column**—BOOLEAN_VALUE **Constraint—**CC1_PROP_VALUE **Constraint Type—**CHECK Ensures the value of the `BOOLEAN_VALUE` column is either 0 (false) or 1 (true).
`SEQUENCER`	**Column**—IS_LOCKED **Constraint—**CC1_SEQUENCER **Constraint Type—**CHECK Ensures the value of the `IS_LOCKED` column is either 0 (false) or 1 (true).

CHAPTER

13 Personalization Server JSP Tag Library Reference

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

Note: The `es:` prefix stands for e-services. The `pz:` prefix stands for personalization.

This topic includes the following sections:

- Ads
  <ad:adTarget>
- Content Management
  <cm:getProperty>
  <cm:printDoc>
  <cm:printProperty>
  <cm:select>
  <cm:selectById>
- Internationalization
  <i18n:localize>
  <i18n:getMessage>
- Personalization Tags
  pz Tags and the Internal Cache
  <pz:contentQuery>
  <pz:contentSelector>
  <pz:div>

- Placeholders
  <ph:placeholder>
- Property Sets
  <ps:getPropertyNames>
  <ps:getPropertySetNames>
  <ps:getRestrictedPropertyValues>
- User Management: Profile Management Tags
  <um:getProfile>
  <um:getProperty>
  <um:getPropertyAsString>
  <um:removeProperty>
  <um:setProperty>
- User Management: Group-User Management Tags
  <um:addGroupToGroup>
  <um:addUserToGroup>
  <um:createGroup>
  <um:createUser>
  <um:getChildGroupNames>
  <um:getGroupNamesForUser>
  <um:getParentGroupName>
  <um:getTopLevelGroups>
  <um:getUsernames>
  <um:getUsernamesForGroup>
  <um:removeGroup>
  <um:removeGroupFromGroup>
  <um:removeUser>
  <um:removeUserFromGroup>
- User Management: Security Tags
  <um:login>
  <um:logout>
  <um:setPassword>
- Utility Tags: Personalization Utilities
  <es:convertSpecialChars>
  <es:counter>
  <es:date>
  <es:forEachInArray>
  <es:isNull>

<es:notNull>
<es:transposeArray>
<es:uriContent>

- Utility Tags: WebLogic Utilities
  <wl:cache>
  <wl:process>
  <wl:repeat>

Ads

The Ad tag queries the content management system and displays ads.

Use the following code to import the utility tag library:
`<%@ taglib uri="ad.tld" prefix="ad" %>`

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<ad:adTarget>

The `<ad:adTarget>` (Table 13-53) uses the Ad Service to send an ad query to the content management system. Unlike the <ph:placeholder> tag, the query in the `<ad:adTarget>` tag does not compete with other queries in an ad placeholder.

Use this tag if you need to make sure that a given ad displays to customers in a specific location. Depending on how narrowly you construct the query, you might have to remove or modify this tag when you want to display a different ad.

If the ad query returns more than one ad, the Ad Service uses the `adWeight` attribute of each ad to determine which ad to display.

Table 13-53 <ad:adTarget>

Tag Attribute	Req'd	Type	Description	R/C
query	Yes	String	Contains a query that the Ad Service uses to find content. Use the query syntax described in the *Javadoc* API documentation for `com.beasys.commerce.util.ExpressionHelper`	R

Table 13-53 <ad:adTarget> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
height	No	int	Specifies the height (in pixels) that the placeholder uses when generating the HTML that the browser requires to display a document. The placeholder uses this value only for content types to which display dimensions apply and only if other attributes have not already defined dimensions for a given document. If you do not specify this value and other attributes have not already been defined, the browser behavior determines the height of the document.	R
width	No	int	Specifies the width (in pixels) that the placeholder uses when generating the HTML that the browser requires to display a document. The placeholder uses this value only for content types to which display dimensions apply and only if other attributes have not already defined dimensions for a given document. If you do not specify this value and other attributes have not already been defined, the browser behavior determines the height of the document.	R

Example

This example picks one of the ads in the ad group "Car" and renders it in a space measuring 200 x 400 pixels.

```
<%@ taglib uri="ad.tld" prefix="ad" %>
.
.
.
<ad:adTarget query="group == 'ads'" />
```

Content Management

The Content Management component includes four JSP tags. These tags allow a JSP developer to include non-personalized content in a HTML-based page. The `cm:select` and `cm:selectbyid` tags support content caching for content searches. Note that none of the tags support or use a body.

To import the Content Management JSP tags, use the following code:
`<%@ taglib uri="cm.tld" prefix="cm" %>`

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<cm:getProperty>

The `<cm:getProperty>` tag (Table 13-54) retrieves the value of the specified content metadata property into a variable specified by `resultId`. It does not have a body. If `resultId` is not specified, the value will be inlined into the page, similar to the `<cm:printProperty>` tag. This tag operates on any ConfigurableEntity, not just the Content object. However, it does not support ConfigurableEntity successors.

Table 13-54 <cm:getProperty>

Tag Attribute	Required	Type	Description	R/C
id	No	String	The JSP script variable name which contains the Content instance from which to get the properties.	R
entity	No	ConfigurableEntity	Specifies the com.beasys.commerce.foundation. ConfigurableEntity object from which to get the property. If this is specified and non-null, `id` is ignored. Otherwise, `id` will be used.	R

Table 13-54 <cm:getProperty> (Continued)

Tag Attribute	Required	Type	Description	R/C
name	Yes	String	The name of the property to print.	R
scope	No	String	The scope name for the property to get. If not specified, null is passed in, which is what Document objects expect.	R
resultId	no	String	The name of the JSP script variable which will be populated with the value of the property. If this is not specified, then the value of the property will be inlined into the body of the JSP. If this is specified, then `encode`, `default`, `maxLength`, `dateFormat`, and `numFormat` are ignored.	C
resultType	no	String	The Java type of the property. If this is not specified, then `java.lang.Object` is used.	C
encode	No	String	Either html, url, or none: ■ If html, then the value will be html encoded so that it appears in HTML as expected (& becomes &amp;, < becomes &lt;, > becomes &gt;, and ” becomes &quot;). ■ If url, then it is encoded to x-www-form-urlencoded format via the java.net.URLEncoder. ■ If none or unspecified, no encoding is performed.	R
default	No	String	The value to print if the property is not found or has a null value. If this is not specified and the property value is null, nothing is printed.	R
maxLength	No	String, int	The maximum length of the property’s value to print. If specified, values longer than this will be truncated.	R

Table 13-54 <cm:getProperty> (Continued)

Tag Attribute	Required	Type	Description	R/C
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and prints nothing if an error occurs. `True`: Throws an exception. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
dateFormat	No	String	The java.text.SimpleDateFormat string to use to print the property, if it is a java.util.Date. If the property is not a Date, this is ignored. If this is not set, the Date's default `toString` method is used.	R
numFormat	No	String	The java.text.DecimalFormat string to use to print the property, if it is a java.lang.Number. If the property is not a Number, this is ignored. If this is not set, the Number's default `toString` method is used.	R

Example

Get the String value of the `name` property from the Content object stored at `doc` and place it in the `contentName` variable:

```
<%@ taglib uri="cm.tld" prefix="cm" %>
.
.
.
<cm:getProperty resultId="contentName" resultType="String"
 id="content" name="name" />
<es:notNull item="<%=contentName%>">
The name is not null.
</es:notNull>
```

<cm:printDoc>

The `<cm:printDoc>` tag (Table 13-55) inlines the raw bytes of a `Document` object into the JSP output stream. This tag does not support a body and only supports `Document` objects. It does not differentiate between text and binary data.

Table 13-55 <cm:printDoc>

Tag Attribute	Required	Type	Description	R/C
id	No	String	The JSP script variable name which contains the Content instance from which to get the properties.	R
blockSize	No	String, int	The size of the blocks of data to read. The default is 8K. Use 0 or less to read the entire block of bytes in one operation.	R
start	No	String, int	Specifies the index in the bytes where to start reading. Defaults to 0.	R
end	No	String, int	Specifies the index in the bytes where to stop reading. The default is to read to the end of the bytes.	R
encode	No	String	Either html, url, or none: ■ If html, then the value will be html encoded so that it appears in HTML as expected (& becomes &amp;, < becomes &lt;, > becomes &gt;, and ” becomes &quot;). ■ If url, then it is encoded to x-www-form-urlencoded format via the java.net.URLEncoder. ■ If none or unspecified, no encoding is performed.	R
document	No	Document	Specifies the com.bea.p13n.content.document.Document to use. If this is specified and non-null, `id` will be ignored. Otherwise, `id` will be used.	R

Table 13-55 <cm:printDoc> (Continued)

Tag Attribute	Required	Type	Description	R/C
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and prints nothing if an error occurs. `True`: Throws an exception. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
baseHref	No	String	The URL of the document's BASE HREF. This can be either an absolute URL or a relative URL.	R

Note: If `baseHref` is provided, then the `<cm:printDoc>` tag will output a starting <BASE HREF> using the value of the `baseHref` parameter. If `baseHref` is not a fully complete URL, the missing parts will be filled in based upon the URL of the outermost page.

Additionally, if `baseHref` is provided, then, after printing the document, the `<cm:printDoc>` tag will output a <BASE HREF> based upon the URL of the outermost page.

Example

To get a Document object from an `id` in the `request` attributes and inline the Document's text (which might contain relative links):

```
<%@ taglib uri="cm.tld" prefix="cm" %>
.
.
.
.<% String contentId = request.getParameter("contentId"); %>
<cm:selectById contentId="<%=contentId%>" id="doc" />
<cm:printDoc id="doc" blockSize="1000" baseHref="/ShowDocServlet"
/>
```

<cm:printProperty>

The `<cm:printProperty>` tag (Table 13-56) inlines the value of the specified content metadata property as a string. It does not have a body. This tag operates on any `ConfigurableEntity`, not just the `Content` object. However, it does not support `ConfigurableEntity` successors.

Table 13-56 <cm:printProperty>

Tag Attribute	Required	Type	Description	R/C
id	No	String	The JSP script variable name which contains the Content instance from which to get the properties.	R
name	Yes	String	The name of the property to print.	R
entity	No	ConfigurableEntity	Specifies the com.beasys.commerce.foundation. ConfigurableEntity object from which to get the property. If this is specified and non-null, `id` is ignored. Otherwise, `id` will be used.	R
scope	No	String	The scope name for the property to get. If not specified, null is passed in, which is what Document objects expect.	R
encode	No	String	Either html, url, or none: ■ If html, then the value will be html encoded so that it appears in HTML as expected (& becomes &amp;, < becomes &lt;, > becomes &gt;, and ” becomes &quot;). ■ If url, then it is encoded to x-www-form-url encoded format via the java.net.URLEncoder. ■ If none or unspecified, no encoding is performed.	R
default	No	String	The value to print if the property is not found or has a null value. If this is not specified and the property value is null, nothing is printed.	R

Table 13-56 <cm:printProperty> (Continued)

Tag Attribute	Required	Type	Description	R/C
maxLength	No	String, int	The maximum length of the property's value to print. If specified, values longer than this will be truncated.	R
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and prints nothing if an error occurs. `True`: Throws an exception. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
dateFormat	No	String	The java.text.SimpleDateFormat string to use to print the property, if it is a java.util.Date. If the property is not a Date, this is ignored. If this is not set, the Date's default `toString` method is used.	R
numFormat	No	String	The java.text.DecimalFormat string to use to print the property, if it is a java.lang.Number. If the property is not a Number, this is ignored. If this is not set, the Number's default `toString` method is used.	R

Example

To have a text input field's default value be the first 75 characters of the subject of a `Content` object stored at `doc`:

```
<%@ taglib uri="cm.tld" prefix="cm" %>
.
.
.
<form action=”javascript:void(0)”>
   Subject: <input type=”text” size=”75” name=”subject”
   value=”<cm:printProperty id=”doc” name=”Subject” maxLength=”75”
   encode=”html”/>” >
</form>
```

<cm:select>

This tag uses only the search expression query syntax to select content. It does not support or use a body. After this tag has returned the `<es:forEachInArray>` tag (see "<es:forEachInArray>" on page 13-71,) zero can be used to iterate over the array of `Content` objects. This tag (Table 13-57) supports generic `Content` via a `ContentManager` interface.

Table 13-57 <cm:select>

Tag Attribute	Required	Type	Description	R/C
contentHome	No	String	The JNDI name of the ContentManager EJB Home to use to find content. The object in JNDI at this name must implement a `create` method which returns an object which implements the ContentManager interface. If not specified, the system searches the default content home.	R
max	No	String, long	Limits the maximum number of content items returned. If not present, or zero or less, it returns all of the content items found.	R

sortBy	No	String	A list of document attributes by which to sort the content. The syntax follows the SQL *order by* clause. The sort specification is limited to a list of the metadata attribute names and the keywords `ASC` and `DESC`.
			Examples:
			sortBy="creationDate"
			sortBy="creationDate ASC, title DESC"
				R

Table 13-57 <cm:select> (Continued)

Tag Attribute	Required	Type	Description	R/C
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and returns an empty array if an error occurs. `True`: Throws an exception that causes the JSP page to stop. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
id	Yes	String	The JSP script variable name that will contain the array of Content objects after this tag finishes.	C
query	No	String	A content query string used to search for content. Example: query="mimetype contains 'text' && author='Proulx'"	R
expression	No	Expression	The com.beasys.commerce.foundation.expression.Expression object to use to search for content. If this is null or not specified, then `query` must be specified. Otherwise, `query` is ignored.	R
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the `cacheId`, `cacheScope` and `cacheTimeout` settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R

Table 13-57 <cm:select> (Continued)

Tag Attribute	Required	Type	Description	R/C
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application` ■ `session` (the default) ■ `page` ■ `request`	R
contextParams	No	String or java.util.Map	Additional search parameters to pass to the ContentManager. Some ContentManager implementations may support this.	R
readOnly	Ignored		This attribute is deprecated and no longer used. When found, it is ignored.	

Example

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

```
<%@ taglib uri="cm.tld" prefix="cm" %>
.
.
.
<cm:select
contentHome="<%=ContentHelper.DEF_CONTENT_MANAGER_HOME%>" max="5"
useCache="true" cacheTimeout="300000" cacheId="Evening News"
```

```
sortBy="creationDate ASC, title ASC" query="
    type = ‘News’ && timeOfDay = ‘Evening’ && mimetype like
    ‘text/*’ " id="newsList"/>

<ul>
   <es:forEachInArray array="<%=newsList%>" id="newsItem"
   type="com.bea.p13n.content.Content">
       <li><cm:printProperty id="newsItem" name="Title"
       encode="html" />
   </es:forEachInArray>
</ul>
```

<cm:selectById>

The `<cm:selectById>` tag (Table 13-58) retrieves content using the `Content’s` unique identifier. This tag does not have a body. This tag is basically a wrapper around the `select` tag. It works against any `Content` object which has a string-capable primary key.

Table 13-58 <cm:selectById>

Tag Attribute	Required	Type	Description	R/C
contentHome	No	String	The JNDI name of the ContentManager EJB Home to use to find content. The object in JNDI at this name must implement a `create` method which returns an object that implements the ContentManager interface. If not specified, the system searches the default content home.	R
contentId	Yes	String	The string identifier of the piece of content.	R
onNotFound	No	String	If the content object specified by `contentId` cannot be found, this controls the behavior. If this is set, then an Exception will be thrown with the value as the message; if this is not set, the tag will return `null`.	R

Table 13-58 <cm:selectById> (Continued)

Tag Attribute	Required	Type	Description	R/C
failOnError	No	String, Boolean	This attribute can have one of two values: `False` (default value): Handles JSP processing errors gracefully and returns null if an error occurs. `True`: Throws an exception that causes the JSP page to stop. You can handle the exception in the code, let the page proceed to the normal error page, or let the application server handle it less gracefully.	R
id	Yes	String	The JSP script variable name that contains the Content object after this tag finishes. If the Content object with the specified identifier does not exist, it contains null.	C
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the `cacheId`, `cacheScope` and `cacheTimeout` settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R

Table 13-58 <cm:selectById> (Continued)

Tag Attribute	Required	Type	Description	R/C
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application` ■ `session` (the default) ■ `page` ■ `request`	R
contextParams	No	String or java.util.Map	Additional search parameters to pass to the ContentManager. Some ContentManager implementations may support this.	R
readOnly	Ignored		This attribute is deprecated and no longer used. When found, it is ignored.	

Example

To fetch the `Document` (using ContentCaching) with an identifier of `1234` and inline its content:

```
<%@ taglib uri="cm.tld" prefix="cm" %>
.
.
.
<cm:selectById
contentHome="<%=ContentHelper.DEF_CONTENT_MANAGER_HOME%>"
contentId="contentportlet/sports1.htm"
id="doc" useCache="true" cacheTimeout="300000" cacheId="1234" />
<cm:printDoc id="doc" />
```

Internationalization

These tags are used in the localization of JSP pages that have an internationalization requirement.

Use the following code to import the utility tag library:
`<%@ taglib uri="i18n.tld" prefix="i18n" %>`

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<i18n:localize>

This tag allows you to define the language, country, variant, and base bundle name to be used throughout a page when accessing resource bundles via the `<i18n:getmessage>` tag.

This tag (Table 13-59) also specifies a character encoding and content type to be specified for a JSP page. Because of this, the tag should be used as early in the page as possible—before anything is written to the output stream—so that the bytes are properly encoded.

Note: When an HTML page is included in a larger page, only the larger page can use the `<i18n:localize>` tag. This is because the `<i18n:localize>` tag sets the encoding for the page, and the encoding must be set in the parent (including) page before any bytes are written to the response's output stream. The parent page must set an encoding that is sufficient for all the content on that page as well as any included pages.

Note: The preferred approach is to retrieve all strings dynamically from the `<i18n:getMessage>` tag, and avoid embedding strings statically (that is, avoid hard-coding them) in your JSP page.

If your page contains only dynamic strings (strings retrieved using the `<i18n:getMessage tag>`), then do not use the `<i18n:localize>` tag in conjunction with the `<%@ page contentType="<something>" >` page directive defined in the JSP specification. The directive is unnecessary if you are using the `<i18n:localize>` tag, and can result in inconsistent or wrong `contentType` declarations.

If you must mix static strings and dynamic strings on the same page, then you will need to use the page directive. Ensure that the character set specified by the `<i18n:localize>` tag is compatible with the character set specified in the page directive.

Table 13-59 <i18n:localize>

Tag Attribute	Required	Type	Description	R/C
bundleName	No	String	The base name of the MessageBundle is used to retrieve localized text for a JSP page.	R
language	No	String or String`[]`	A String—two character ISO Language Code—denoting the user's preferred language, or a String`[]` containing a list of preferred language codes for a user, with stronger preferences indexed lower (earlier) in the array.	R
country	No	String	The two character ISO Country Code for a country. For example, this code would be used to look for a MessageBundle containing text localized to English speaking users in the U.S. as opposed to English speaking users in the U.K.	R
variant	No	String	A String representing a locale's variant. The variant is used when localization demands a more specific locale than can be denoted by having just language and a country.	R
locale	No	java.util.Locale	Instead of specifying language, country, and variant as Strings, a `java.util.Locale` object can be provided. If provided, the values in the Locale's language, country, and variant fields will negate any of the other language, country, and variant values passed to the tag as Strings.	R

Table 13-59 <i18n:localize> (Continued)

Tag Attribute	Required	Type	Description	R/C
charset	No	String	The name of the character encoding set to use for this page. Defaults to "UTF-8" if no encoding can be determined for the chosen language, otherwise an encoding appropriate for the chosen language is used.	R
contentType	No	String	The type of content contained in the page, defaults to "text/html".	R

Example 1

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
.
.
.
<%
// Definition of a single language preference
String language = "en";
%>

<i18n:localize language="<%=language%>"
bundleName="i18nExampleResourceBundle"/>
<html>
<body>
<i18n:getMessage messageName="greeting"/>
</body>
</html>
```

Example 2

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
.
.
.
<%
// Array that defines two languages preferences - English and
// Spanish in that order of preference.
```

```
String[] languages = new String[] { "en", "es" };

%>

<i18n:localize language="<%=languages%>"
bundleName="i18nExampleResourceBundle"/>
<html>
<body>
<i18n:getMessage messageName="greeting"/>
</body>
</html>
```

<i18n:getMessage>

This tag (Table 13-60) is used in conjunction with the `<i18:localize>` tag to retrieve localized static text or messages from a JspMessageBundle.

Table 13-60 <i18n:getMessage>

Tag Attribute	Required	Type	Description	R/C
id	No	String	Holds the value of the label (or message) in the JSP page.	C
messageName	Yes	String	The key for the message bundle.	R
messageArgs	No	Object `[]`	The arguments to the message bundle. If no args are provided, it is assumed that static text (not a message) is to be returned. For example, {"Wednesday", "78"}; might be used to construct the message “Today is Wednesday, and the temperature is 78 degrees Fahrenheit.”	R
bundleName	No	String	If properly initialized in the `<i18n:localize>` tag, there is no need to pass this tag attribute unless it is desired to use a different bundle for a particular tag invocation	R

Table 13-60 <i18n:getMessage> (Continued)

Tag Attribute	Required	Type	Description	R/C
language	No	String	If properly initialized in the `<i18n:localize>` tag, there is no need to pass this tag attribute, unless it is desired to use a different language for a particular tag invocation.	R
country	No	String	If properly initialized in the `<i18n:localize>` tag, there is no need to pass this tag attribute, unless it is desired to use a different country for a particular tag invocation.	R
variant	No	String	If properly initialized in the `<i18n:localize>` tag, there is no need to pass this tag attribute, unless it is desired to use a different variant for a particular tag invocation.	R
locale	No	java.util.Locale	If properly initialized in the `<i18n:localize>` tag, there is no need to pass this tag attribute, unless it is desired to use a different locale (language, country, and variant) for a particular tag invocation.	R

Example

JSP File

This code produces this output:
Welcome To This Page! 14 out of 100 files have been saved.

```
<%@ taglib uri="i18n.tld" prefix="i18n" %>
.
.
.
<%
// Definition of a single language preference
String language = "en";
```

```
// Creation of message arguments
Object[] args = new Object[]
{
    new Integer(14),
    new Integer(100)
};
%>

<i18n:localize language="<%=language%>"
bundleName="i18nExampleResourceBundle"/>
<html>
<body>
<i18n:getMessage messageName="greeting"/>
<i18n:getMessage messageName="message" messageArgs="<%=args%>"/>
</body>
</html>
```

Property file

Here are the entries in the property file named "`i18nExampleResourceBundle.properties`":

```
greeting=Welcome To This Page!
message={0} out of {1} files have been saved.
```

Personalization Tags

The <pz:div> tag, <pz:contentSelector> tag, and <pz:contentQuery> tag use the Advisor to classify the user, select content, and retrieve content, respectively.

To import the Personalization JSP tags, use the following code:
```
<%@ taglib uri="pz.tld" prefix="pz" %>
```

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

This section contains the following:

- pz Tags and the Internal Cache
- <pz:contentQuery>
- <pz:contentSelector>
- <pz:div>

pz Tags and the Internal Cache

Content search `contextParams` were added in WebLogic Portal 4.0 to support per-search configuration attributes. These included the ability to determine whether to use the internal cache. The <`cm:select`> and <`cm:selectById`> tags were updated to support setting the contextParams, but the `pz` tags do not have this capability. In order to control whether a <`pz:contentSelector`> uses the internal cache, use the following approach.

Add the following to a <`pz:content*`> tag:
```
contextParams="someName=someValue"
```

A runtime expression like the following should be used:
```
contextParams='<%="aName=aValue bName=bValue cName=cValue"%>'
```

<pz:contentQuery>

The `<pz:contentQuery>` tag (Table 13-61) performs a content attribute search for content in a content manager. If the `useCache` attribute is set to `true`, the results of a content management query will be cached. The tag only has a begin tag and does not have a body or end tag. It returns an array of `Content` objects returned from the content manager as the result of executing the content query.

Personalization content tags required for JSP developers to access the `Content` object returned might include:

An object array iterator tag. This tag provides a way to iterate over the `Content` objects in the array. Use the `<es:forEachInArray>` tag to iterate over an array of `Objects`. (See "<es:forEachInArray>" on page 13-71 for more information.)

- Content access tags. Content tags access metadata attributes in the content, retrieve content, and put it into the HTTP response output stream. (See the section "Content Management" on page 13-6 for more information.)

Table 13-61 <pz:contentQuery>

Tag Attribute	Required	Type	Description	R/C
max	No	String, long	Limits the maximum number of content items returned. If not present, it returns all of the content items found.	R

sortBy	No	String	A list of document attributes by which to sort the content. The syntax follows the SQL *order by* clause. The sort specification is limited to a list of the metadata attribute names and the keywords `ASC` and `DESC`.
			Examples:
			sortBy="creationDate"
			sortBy="creationDate ASC, title DESC"
				R

query	Yes	String	A content query string used to search for content. Example: query= "mimetype contains 'text' && author='Proulx'"	R

Table 13-61 <pz:contentQuery> (Continued)

Tag Attribute	Required	Type	Description	R/C
contentHome	Yes	String	The JNDI name of the ContentManager EJB Home. The object in the JNDI at this name must implement a `create` method which returns an object which implements the ContentManager interface. For more information, see the section "Specify a Value for contentHome" on page 13-32.	R
id	Yes	String	The array variable name that contains the content objects found. If no content is found, the variable is assigned an empty array (not null) of Content objects.	C
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): ContentCache is not used. If `false` (not specified), the `cacheId`, `cacheScope` and `cacheTimeout` settings are ignored. `True`: ContentCache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R

Table 13-61 <pz:contentQuery> (Continued)

Tag Attribute	Required	Type	Description	R/C
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application`. Any JSP page in the web application that any customer requests can access the cache. ■ `session` (the default). Any JSP in the web application that the current customer requests can access the cache. ■ `page`. Only the current JSP that any customer requests can access the cache. ■ `request`. Only the current user request can access the cache. If a customer re-requests the page, the content selector re-runs the query and recreates the cache.	R

Example

```
<%@ page import="bea.p13n.content.ContentHelper" %>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
.
.
.
<pz:contentQuery id="docs"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME%>"
   query="author = 'Hemingway'" />

<ul>


 <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
  </es:forEachInArray>
</ul>
```

<pz:contentSelector>

The `<pz:contentSelector>` tag (Table 13-62) allows arbitrary personalized content to be recommended based on a content selector rule.

A content selector rule first evaluates a set of conditions that you define in the E-Business Control Center (for example, whether or not a user fits a specified classification). If the conditions evaluate to true, content is retrieved from the Content Manager based on a content query defined in the E-Business Control Center.

Note: Rules are created in the E-Business Control Center. This GUI tool is designed to allow Business Analysts to develop their own customer segments. Because the Business Analysts are not exposed to the concept of rules, you will see content selector rules called simply "content selectors" and classifier rules referred to as "customer segmentation."

To cache the results of the content selector rule, set the `useCache` attribute to `true`. If the cache has not timed out, subsequent calls to the contentSelector tag will return the cached results without re-evaluating the content query.

The `<pz:contentSelector>` tag only has a begin tag and does not have a body or end tag. It returns an array of `Content` objects returned from the Content Manager as a result of executing the content query.

Tags possibly required for JSP developers to access the `Content` objects returned might include:

- An object array iterator tag. This tag provides a way to iterate over the `Content` objects in the array. Use the `<es:forEachInArray>` tag to iterate over an array of `Objects`.
- Content access tags. Content tags access metadata attributes in the content and retrieve content and put it into the HTTP response output stream. (See the section "Content Management" on page 13-6 for more information.)

Table 13-62 <pz:contentSelector>

Tag Attribute	Req'd	Type	Description	R/C
rule	Yes	String	The name of the content selector in the content selector definitions created in the E-Business Control Center.	R

Table 13-62 <pz:contentSelector> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
contentHome	Yes	String	The JNDI name of the ContentManager EJB Home. The object in the JNDI at this name must implement a `create` method which returns an object which implements the ContentManager interface. For more information, see the section "Specify a Value for contentHome" on page 13-32.	R
max	No	String, long	Limits the maximum number of content items returned. If not present, or if equal to -1L, it returns all of the content items found.	R

sortBy	No	String	A list of document attributes by which to sort the content. The syntax follows the SQL *order by* clause. The sort specification is limited to a list of the metadata attribute names and the keywords `ASC` and `DESC`.
			Examples:
			sortBy="creationDate"
			sortBy="creationDate ASC, title DESC"
				R

query	No	String	A content query string used to search for content. This query overrides any query that a Business Analyst creates in the E-Business Control Center. Example: query="mimetype contains 'text' && author='Salinger'"	R
id	Yes	String	The array variable name that contains the content objects found. If no content is found, the variable is assigned an empty array (not null) of Content objects.	C

Table 13-62 <pz:contentSelector> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
useCache	No	String, Boolean	Determines whether Content is cached. This attribute can have one of two values: `False` (default value): The Content cache is not used. If `false` (not specified), the cacheId, cacheScope and cacheTimeout settings are ignored. `True`: Content cache is used.	R
cacheId	No	String	The identifier name used to cache the Content. Internally, the cache is implemented as a Map; this will become the key. If not specified, the `id` attribute of the tag is used.	R
cacheTimeout	No	String, long	The time, in milliseconds, for which the cached Content is valid. If more than this amount of time has passed since the Content was cached, the cached Content will be cleared, retrieved, and placed back into the cache. Use -1 for no-timeout (always use the cached Content). Default = -1.	R

Table 13-62 <pz:contentSelector> (Continued)

Tag Attribute	Req'd	Type	Description	R/C
cacheScope	No	String	Specifies the lifecycle scope of the content cache. Similar to `<jsp:useBean>`. Possible values: ■ `application`. Any JSP page in the web application that any customer requests can access the cache. ■ `session` (the default). Any JSP in the web application that the current customer requests can access the cache. ■ `page`. Only the current JSP that any customer requests can access the cache. ■ `request`. Only the current user request can access the cache. If a customer re-requests the page, the content selector re-runs the query and recreates the cache.	R

Specify a Value for contentHome

The content selector tag must use the `contentHome` attribute to specify the JNDI home of the content management system. If you use the reference content management system or a third-party integration, you can use a scriptlet to refer to the default content home. Because the scriptlet uses the `ContentHelper` class, you must first use the following tag to import the class into the JSP:

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
```

Then, when you use the content selector tag, specify the `contentHome` as follows:

```
<pz:contentSelector
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
... />
```

If you create your own content management system, you must specify the JNDI home for your system instead of using the ContentHelper scriptlet. In addition, if your content management system provides a JNDI home, you can specify that one instead of using the ContentHelper scriptlet.

Example

```
<%@ page import="com.bea.p13n.content.ContentHelper" %>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getProfile profileKey="bob"
profileId="myProfile" scope="session"/>
<pz:contentSelector rule="PremierCustomerSpotlight"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="docs" />
<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
      <li>The document title is: <cm:printproperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

Note: The `sortBy` attribute, when used in conjunction with the `max` attribute, works differently for explicit (system-defined) and implicit (user-defined) attributes. If you sort on explicit attributes (`identifier, mimeType, size, version, author, creationDate, modifiedBy, modifiedDate, lockedBy, description, or comments`) the sort is done on the database; therefore if you combine `max="10"` and `sortBy`, the system will perform the sort and then get the first 10 items. If you sort on implicit attributes, the sort is done *after* the max have been selected.

For more information about using this tag, see the section “Using Content-Selector Tags and Associated JSP Tags” in Chapter 4, “Working with Content Selectors,” in this guide.

<pz:div>

The `<pz:div>` tag (Table 13-63) allows a piece of content to be conditionally included as a result of a classifier rule being executed by the rules engine. If the user's profile matches the classification specified in the E-Business Control Center, then the conditional content is included. This tag has a begin tag, a body, and an end tag. The tag returns a list of `Classification` objects that the user belongs to.

Table 13-63 <pz:div>

Tag Attribute	Required	Type	Description	R/C
rule	Yes	String	The name of the classifier rule in the customer segment definitions created in the E-Business Control Center.	R
id	No	String	A collection that contains the Classification objects that apply to the user for the given classifier rule.	C

Example

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

<um:getProfile profileKey="bob"
profileId="myProfile" scope="session"/>
<pz:div  id=”classifications” rule="PremierCustomer">


<%
//if the user is classified as a Premier Customer, a list with one
entry should be returned//
   java.util.Iterator iterator=classifications.iterator();
   while (iterator.hasNext())
   {
   com.bea.p13n.user. Classification
classification=(Classification) iterator.next();
  out.println (classification.getName());
   }
%>

   <p>Please check out our new Premier Customer bonus program.<p>
</pz:div>
```

Placeholders

The placeholder tag is a named location on a JSP. You use the E-Business Control Center to define the behavior of a placeholder.

Use the following code to import the utility tag library:
`<%@ taglib uri="ph.tld" prefix="ph" %>`

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<ph:placeholder>

The `<ph:placeholder>` tag (Table 13-64) implements a placeholder, which describes the behavior for a location on a JSP page.

You use the E-Business Control Center to define a placeholder. For more information, see the topic "Displaying Ads" in the *Guide to Using the E-Business Control Center*.

Multiple placeholder tags can refer to the same placeholder. Each instance of a placeholder tag invokes its placeholder definition separately. If the placeholder definition specifies multiple queries, each placeholder tag instance can display different ads, even though each instance shares the same definition.

When WebLogic Personalization Server receives a request for a JSP that contains an ad placeholder, the placeholder tag contacts the Ad Service, a session EJB that invokes business logic to determine which ad to display. For more information, see the section "How Placeholders Select and Display Ads" in Chapter 4, "Working with Content Selectors," in this guide.

For information on a related tag, see `<ad:adTarget>`.

Table 13-64 <ph:placeholder>

Tag Attribute	Req'd	Type	Description	R/C
name	Yes	String	A string that refers to a placeholder definition.	R
height	No	int	Specifies the height (in pixels) that the placeholder uses when generating the HTML that the browser requires to display a document. The placeholder uses this value only for content types to which display dimensions apply and only if other attributes have not already defined dimensions for a given document. If you do not specify this value and other attributes have not already been defined, the browser behavior determines the height of the document.	R
width	No	int	Specifies the width (in pixels) that the placeholder uses when generating the HTML that the browser requires to display a document. The placeholder uses this value only for content types to which display dimensions apply and only if other attributes have not already defined dimensions for a given document. If you do not specify this value and other attributes have not already been defined, the browser behavior determines the height of the document.	R

Example

This example displays the ad specified in the `MainPageBanner` placeholder.

```
<%@ taglib uri="ph.tld" prefix="ph" %>
. . .
<ph:placeholder name="/placeholders/MainPageBanner.pla"/>
```

Property Sets

The Property Set tags allow access to the list of available properties and property sets. Property sets are manipulated through the E-Business Control Center.

Use the following code to import the utility tag library:
`<%@ taglib uri="ps.tld" prefix="ps" %>`

All Property Sets tags send results to the same file. If you are checking for results, include this import directive at the top of the page:

```
<%@ page
import="com.bea.p13n.property.servlets.jsp.taglib.PropertySetTagC
onstants" %>
```

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<ps:getPropertyNames>

The `<ps:getPropertyNames>` tag (Table 13-65) is used to get a list of property names given a property set.

Table 13-65 <ps:getPropertyNames>

Tag Attribute	Required	Type	Description	R/C
propertySetName	Yes	String	The name of the property set to add the new search.	R
propertySetType	Yes	String	Type of property set to search.	R
id	Yes	String	The `id` of the variable to hold the list of property names, as a String array.	C

Table 13-65 <ps:getPropertyNames> (Continued)

Tag Attribute	Required	Type	Description	R/C
result	no	String	The identifier of an Integer variable that will be created and initialized with the result of the operation. Possible values: *Query is successful*: `PropertySetTagConstants.PROPERTY_SEARCH_OK` *Problem getting the list of property names*: `PropertySetTagConstants.PROPERTY_SEARCH_FAILED` *Property set named by* `propertySetName` *and* `propertySetType` *could not be found*: `PropertySetTagConstants.INVALID_PROPERTY_SET`	C

Example

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

<%@ page import=
"com.bea.p13n.property.servlets.jsp.taglib.PropertySetTagConstant
s"
%>

<% String myPropertySet="Demographics"; %>

<p> ------- <b>ps:getPropertyNames</b> -------------

<br>

<ps:getPropertyNames propertySetName="<%= myPropertySet %>"

propertySetType="USER" id="propertyNames" result="myResult"/>

<% for (int i=0; i<propertyNames.length; i++)

out.println(propertyNames[i] + "<br>");

%>
```

<ps:getPropertySetNames>

The `<ps:getPropertySetNames>` tag (Table 13-66) is used to get a list of property sets given a property set type.

Table 13-66 <ps:getPropertySetNames>

Tag Attribute	Required	Type	Description	R/C
propertySetType	Yes	String	Type of property set to search.	R
id	Yes	String	The identifier of the variable to hold the list of property names, as a String array.	C
result	No	String	The identifier of an Integer variable that will be created and initialized with the result of the operation. Possible values: *Query is successful*: `PropertySetTagConstants.PROPERTY_SET_SEARCH_OK` *Problem getting the list of property names*: `PropertySetTagConstants.PROPERTY_SET_SEARCH_FAILED` *Property set named by* `propertySetName` *and* `propertySetType` *could not be found*: `PropertySetTagConstants.INVALID_PROPERTY_SET`	C

Example

```
<%@ taglib uri="ps.tld" prefix="ps" %>
.
.
.
<ps:getPropertySetNames propertySetType="USER"
id="userPropertySets" result="myResult" />
```

<ps:getRestrictedPropertyValues>

The `<ps:getRestrictedPropertyValues>` tag (Table 13-67) returns a list of restricted values for a specific property definition, converted into Strings. These values will be returned as an array of Strings.

Table 13-67 <ps:getRestrictedPropertyValues>

Tag Attribute	Required	Type	Description	R/C
propertySetName	Yes	String	The name of the property set containing the property.	R
propertySetType	Yes	String	Type of property set containing the property.	R
propertyName	Yes	String	The name of the property to inspect.	R
id	Yes	String	The identifier of the variable to hold the list of property names, as a String array.	C

result	No	String	The identifier of an Integer variable that will be created and initialized with the result of the operation.
			Possible values:
			Query is successful:
			`PropertySetTagConstants.PROPERTY_SEARCH_OK`
			Problem accessing the property:
			`PropertySetTagConstants.PROPERTY_SEARCH_FAILED`
			Property set named by `propertySetName` *and* `propertySetType` *could not be found*:
			`PropertySetTagConstants.INVALID_PROPERTY_SET`
			The requested property is not restricted:
			`PropertySetTagConstants.PROPERTY_NOT_RESTRICTED`
				C

Example

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

<%@ page import=
"com.bea.p13n.property.servlets.jsp.taglib.PropertySetTagConstant
s"
%>

<p> ---- <b>ps:getRestrictedPropertyValues</b> -----
<br>Possible values for PreferredLanguage:

<ps:getRestrictedPropertyValues propertySetName="Demographics"
    propertySetType="USER" propertyName="PreferredLanguage"
    id="values" result="myResult"/>

<ul>
<% if (myResult.intValue() ==
PropertySetTagConstants.PROPERTY_SEARCH_OK)
  {
       for ( int i=0; i<values.length; i++ ) {
              %><li><%=values[i]%>
<% }
  }
%>
</ul>
```

User Management: Profile Management Tags

User Management tags allow access to user and group profile information, as well as operations such as creating and deleting users and groups, and managing user-group relationships.

To import the User Management JSP tags, use the following code:
```
<%@ taglib uri="um.tld" prefix="um" %>
```

All User Management tags send results to the same file. If you are checking for results, include this import directive at the top of the page:
```
<%@ page
import="com.bea.p13n.usermgmt.servlets.jsp.taglib.UserManagementTa
gConstants" %>
```

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<um:getProfile>

The `<um:getProfile>` tag (Table 13-68) retrieves the profile corresponding to the provided profile key and profile type. The tag has no enclosed body. The retrieved profile can be treated as a `com.bea.p13n.usermgmt.profile.ProfileWrapper`. Along with the profile key and profile, an explicit successor key and successor type can be specified, as specified by the `profileType` attribute. This successor will then be used, along with the retrieved profile, in subsequent invocations of the `<um:getProperty>` tag to ensure property inheritance from the successor. If no successor is retrieved, standard `ConfigurableEntity` successor search patterns will apply to retrieved properties.

Table 13-68 <um:getProfile>

Tag Attribute	Required	Type	Description	R/C
profileKey	Yes	String	A unique identifier that can be used to retrieve the profile which is sought. Example: "<%=username%>"	R
successorKey	No	String	A unique identifier that can be used to retrieve the profile successor. Example: "<%=defaultGroup%>"	R
scope	No	String	The HTTP scope of the retrieved profile. Pass `"request"` or `"session"` as the values. Defaults to `session`.	C
groupOnly	No	String	Specifies to retrieve a group profile named by the `profileKey`, rather than a user profile. No successor will be retrieved when this value is `true`. Defaults to `false`.	C
profileId	No	String	A variable name from which the retrieved profile is available for the duration of the JSP's page scope.	C
successorId	No	String	A variable name from which the retrieved successor is available for the duration of the JSP's page scope.	C

Table 13-68 <um:getProfile> (Continued)

Tag Attribute	Required	Type	Description	R/C

result	No	String	A variable name from which the result of the operation is available.
			Possible values:
			Success:
			`UserManagementTagConstants.GET_PROFILE_OK`
			Error encountered:
			`UserManagementTagConstants.GET_PROFILE_FAILED`
			`UserManagementTagConstants.NO_SUCH_PROFILE`
			`UserManagementTagConstants.NO_SUCH_SUCCESSOR`
				C

Example 1

This example shows a profile being retrieved with no successor specified, and an explicitly-supplied `session` scope.

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getProfile profileKey="bob" profileType="AcmeUser"
profileId="myProfile" scope="session"/>
```

Example 2

This example shows a default user profile type being retrieved with a default successor type, and an explicitly-supplied `request` scope.

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getProfile profileKey="bob" successorKey="engineering"
scope="request"/>
```

<um:getProperty>

The `<um:getProperty>` tag (Table 13-69) 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. For more information, see Chapter 8, "Creating and Managing Users," in this guide.

Table 13-69 <um:getProperty>

Tag Attribute	Required	Type	Description	R/C
propertySet	No	String	The Property Set from which the property's value is to be retrieved. Example: "Demographics" **Note:** If no property set is provided, the property is retrieved from the profile's default (unscoped) properties.	R
propertyName	Yes	String	The name of the property to be retrieved. Example: "Date_of_Birth"	R
id	No	String	If the `id` attribute is supplied, the value of the retrieved property will be available in the variable name to which `id` is assigned. Otherwise, the value of the property is inlined.	C

Example 1

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getProperty id="myBirthDate" propertySet="Demographics"
propertyName="Date_of_Birth"/>
My birthday is <%=myBirthDate%>.
```

Example 2

```
My birthday is <um:getProperty propertySet="Demographics"
propertyName="Date_of_Birth"/>.
```

<um:getPropertyAsString>

The `<um:getPropertyAsString>` tag (Table 13-70) works exactly like the `<um:getProperty>` tag above, but ensures that the retrieved property value is a `String`. The following example shows a multi-valued property which returns a `Collection`, but presents a list of favorite colors.

Table 13-70 <um:getPropertyAsString>

Tag Attribute	Required	Type	Description	R/C
propertySet	No	String	The Property Set from which the property's value is to be retrieved. Example: "Demographics" **Note:** If no property set is provided, the property is retrieved from the profile's default (unscoped) properties.	R
propertyName	Yes	String	The name of the property to be retrieved. Example: "Date_of_Birth"	R
id	No	String	If the `id` attribute is supplied, the value of the retrieved property will be available in the variable name to which `id` is assigned. Otherwise, the value of the property is inlined.	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getPropertyAsString id="myBirthDate"
```

```
propertySet=”Demographics” propertyName=”Date_of_Birth”/>
My birthday is <%=myBirthDate%>.
```

<um:removeProperty>

The `<um:removeProperty>` tag (Table 13-71) removes the specified property from the current session’s profile or from the Anonymous User Profile. The tag has no enclosed body. Subsequent calls to `<um:getProperty>` for a removed property would result in the default value for the property as prescribed by the property set, or from the Profile’s successor.

Table 13-71 <um:removeProperty>

Tag Attribute	Required	Type	Description	R/C
propertySet	No	String	The Property Set from which the property's value is to be retrieved. Example: “Demographics” **Note:** The property is removed from the profile's default (unscoped) properties if no property set is provided.	R
propertyName	Yes	String	The name of the property to be removed. Example: “Income_Range”	R

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:removeProperty propertySet="<%=thePropertySet%>"
propertyName="<%=thePropertyName%>"/>
```

<um:setProperty>

The `<um:setProperty>` tag (Table 13-72) updates a property value for either the session's current profile, or for the Anonymous User Profile. This tag has no enclosed body.

Table 13-72 <um:setProperty>

Tag Attribute	Required	Type	Description	R/C
propertySet	No	String	The Property Set in which the property's value is to be set. Example: "Demographics" **Note:** The property is set for the profile's default (unscoped) properties if no property set is provided.	R
propertyName	Yes	String	The name of the property to be set. Example: "Gender"	R
value	Yes	Object	The new property value.	R

result	No	String	The name of an Integer object to which the result of the set property operation is assigned.
			Success:
			`UserManagementTagConstants.SET_PROPERTY_OK`
			Error encountered:
			`UserManagementTagConstants.SET_PROPERTY_FAILED`
				C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<% String myGender = request.getParameter("gender"); %>
<um:setProperty propertySet="Demographics" propertyName="Gender"
value="<%=myGender%>"/>
```

User Management: Group-User Management Tags

User Management tags allow access to user and group profile information, as well as operations such as creating and deleting users and groups, and managing user-group relationships.

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

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

All User Management tags send results to the same file. If you are checking for results, include this import directive at the top of the page:

```
<%@ page
import="com.bea.p13n.usermgmt.servlets.jsp.tags.UserManagementTagC
onstants" %>
```

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<um:addGroupToGroup>

The `<um:addGroupToGroup>` tag (Table 13-73) adds the group corresponding to the provided `childGroupName` to the group corresponding to the provided `groupName`. Since a group can only have one parent, any previous database records which reflect the group belonging to another parent will be destroyed. Both the parent group and the child group must previously exist for proper tag behavior. The tag has no enclosed body.

Note: This tag should only be invoked when the current realm is an implementation of `weblogic.security.acl.ManageableRealm`. This interface is implemented by the default WebLogic Personalization Server realm (`com.bea.p13n.security.realm.RDBMSRealm`).

Table 13-73 <um:addGroupToGroup>

Tag Attribute	Required	Type	Description	R/C
childGroupName	Yes	String	The name of the child group. Example: “<%=childGroupName%>”	R
parentGroupName	No	String	The name of the parent group. Example: “<%=parentGroupName%>”	R
result	Yes	String	The name of an Integer variable to which the result of the add group to group operation is assigned. Possible values: *Success*: `UserManagementTagConstants.ADD_GROUP_OK` *Error encountered*: `UserManagementTagConstants.ADD_GROUP_FAILED`	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:addGroupToGroup childGroupName=“<%=childGroupName%>”
parentGroupName=”<%=parentGroupName%>” result=”result”/>
```

<um:addUserToGroup>

The `<um:addUserToGroup>` tag (Table 13-74) adds the user corresponding to the provided `username` to the group corresponding to the provided `groupName`. Both the specified user and the specified group must previously exist for proper tag behavior. The tag has no enclosed body.

Note: This tag should only be invoked when the current realm is an implementation of `weblogic.security.acl.ManageableRealm`. This interface is implemented by the default WebLogic Personalization Server realm (`com.bea.p13n.security.realm.RDBMSRealm`).

Table 13-74 <um:addUserToGroup>

Tag Attribute	Required	Type	Description	R/C
username	Yes	String	The name of the user to be added to the group. Example: "<%=username%>"	R
groupName	Yes	String	The name of the group to which the user is being added. Example: "<%=groupName%>"	R
result	Yes	String	The name of an Integer variable to which the result of the add user to group operation is assigned. Possible values: *Success:* `UserManagementTagConstants.ADD_USER_OK` *Error encountered:* `UserManagementTagConstants.ADD_USER_FAILED`	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:addUserToGroup userName="<%=userName%>"
groupName="<%=groupName%>" result="result"/>
```

<um:createGroup>

The `<um:createGroup>` tag (Table 13-75) creates a new group in the realm, and a corresponding group profile in the personalization database. This tag has no enclosed body.

Note: This tag should only be invoked when the current realm is an implementation of `weblogic.security.acl.ManageableRealm`. This interface is implemented by the default WebLogic Personalization Server realm (`com.bea.p13n.security.realm.RDBMSRealm`).

Table 13-75 <um:createGroup>

Tag Attribute	Required	Type	Description	R/C
groupName	Yes	String	The name of the new group. Example: "<%=groupName%>"	R
id	No	String	A variable name to which the created Group object is available for the duration of the page's scope.	C
parentName	No	String	The name of the group to set as the parent of the new group.	R
result	Yes	String	The name of an Integer variable to which the result of the create group operation is assigned. Possible Values: *Success:* `UserManagementTagConstants.CREATE_GROUP_OK` *Error encountered:* `UserManagementTagConstants.CREATE_GROUP_FAILED` *A group with the specified group name already exists*: `UserManagementTagConstants.GROUP_EXISTS`	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:creategroup groupName="<%=groupName%>" result="result"/>
```

<um:createUser>

The `<um:createUser>` tag (Table 13-76) creates a new user profile. This tag has no enclosed body. Although classified as a Group-User management tag, this tag can be used in conjunction with run-time activities, in that it will persist any properties associated with a current Anonymous User Profile if specified.

Note: This tag should only be invoked when the current realm is an implementation of `weblogic.security.acl.ManageableRealm`. This interface is implemented by the default WebLogic Personalization Server realm (`com.bea.p13n.security.realm.RDBMSRealm`).

Table 13-76 <um:createUser>

Tag Attribute	Required	Type	Description	R/C
username	Yes	String	The name of the new user. Example: "<%=username%>"	R
password	Yes	String	The password for the new user. Example: "<%=password%>"	R
profileType	No	String	Specifies the extended type of user (for example, WLCS_Customer) to create a user of that type.	R
saveAnonymous	No	String	Whether to persist current anonymous user profile attributes in the newly-created user's profile. Defaults to `false`. Example: "`false`"	R

Table 13-76 <um:createUser> (Continued)

Tag Attribute	Required	Type	Description	R/C
id	No	String	A variable name to which the created User object is available for the duration of the page's scope.	C
result	Yes	String	The name of an Integer variable to which the result of the create user operation is assigned. Possible values: *Success:* `UserManagementTagConstants.CREATE_USER_OK` *Error encountered*: `UserManagementTagConstants.CREATE_USER_FAILED` *A user with the specified username already exists*: `UserManagementTagConstants.USER_EXISTS`	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:createUser userName="<%=username%>" password="<%=password"%>
result="result"/>
```

<um:getChildGroupNames>

The `<um:getChildGroupNames>` tag (Table 13-77) returns the names of any groups that are children of the given group.

Table 13-77 <um:getChildGroupNames>

Tag Attribute	Required	Type	Description	R/C
groupName	Yes	String	The name of the group to search for child groups.	R
id	Yes	String	The name of the identifier which will be assigned the String array of child group names.	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getChildGroupNames groupName="SomeGroup"
id="childrenOfSomeGroup" />
```

<um:getGroupNamesForUser>

The `<um:getGroupNamesForUser>` tag (Table 13-78) retrieves a `String` array that contains the group names corresponding to groups to which the provided user immediately belongs. This tag has no enclosed body.

Table 13-78 <um:getGroupNamesForUser>

Tag Attribute	Required	Type	Description	R/C
username	Yes	String	The name of the user whose matching groups are sought. Example: “<%=username%>”	R
id	Yes	String	A variable name to which the resultant group names are assigned. Example: “myGroups”	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getGroupNamesForUser userName=”<%=username%>” id=”myGroups”/>
```

<um:getParentGroupName>

The `<um:getParentGroupName>` tag (Table 13-79) retrieves the name of the parent of the group associated with the provided `groupName`. The information is taken from the realm. This tag has no enclosed body.

Table 13-79 <um:getParentGroupName>

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

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getParentGroupName groupName="<%=groupName%>"
id="parentGroupName"/>
```

<um:getTopLevelGroups>

The `<um:getTopLevelGroups>` tag (Table 13-80) retrieves an array of group names, each of which has no parent group. The information is taken from the realm. This tag has no enclosed body.

Table 13-80 <um:getTopLevelGroups>

Tag Attribute	Required	Type	Description	R/C
id	Yes	String	A variable name to which the top-level `Group` objects are available for the duration of the page's scope. Example: "topLevelGroups"	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getTopLevelGroups id="topLevelGroups"/>
```

<um:getUsernames>

The `<um:getUsernames>` tag (Table 13-81) retrieves a `String` array that contains the usernames matching the provided search expression. The search expression supports only the asterisk (*) wildcard character, and is case insensitive. As many asterisks as desired may be used in the search expression. This tag has no enclosed body.

Table 13-81 <um:getUsernames>

Tag Attribute	Required	Type	Description	R/C
searchExp	No	String	The search expression to apply to the user name search. Defaults to '*' Example: "t*"	R

Table 13-81 <um:getUsernames> (Continued)

Tag Attribute	Required	Type	Description	R/C
userLimit	No	String (representing an Integer)	The maximum number of users to be returned from the search. (String which has a particular `Integer.valueOf.`) Defaults to 100. If user count exceeds userLimit, the length of the array in `id` is truncated to the length of userLimit. Example: "500"	R
id	Yes	String	A variable name to which the resultant user names are assigned. Example: "myUsers"	C
result	No	String	The name of an Integer variable to which the result of the `getUsernames` operation is assigned. Possible values: *Success:* `UserManagementTagConstants.USER_SEARCH_OK` *General error*: `UserManagementTagConstants.USER_SEARCH_FAILED`	C

Note: The `USER_SEARCH_FAILED` value is returned only when a general error occurs while searching for the user, such as a database connection failure. If no user matches the search criteria, the result will not be equal to `UserManagementTagConstants.USER_SEARCH_FAILED`, but the length returned by the array in `id` will be zero.

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getUsernames userLimit="500" searchExp="t*" id="myUsers"/>
<%System.out.println("I found " + myUsers.length + " users.");%>
```

<um:getUsernamesForGroup>

The `<um:getUsernamesForGroup>` tag (Table 13-82) retrieves a `String` array that contains the usernames matching the provided search expression and correspond to members of the provided group. The search expression supports only the asterisk (*) wildcard character, and is case insensitive. As many asterisks as desired may be used in the search expression. This tag has no enclosed body.

Table 13-82 <um:getUsernamesForGroup>

Tag Attribute	Required	Type	Description	R/C
searchExp	No	String	The search expression to apply to the user name search. Defaults to " *". Example: `"t*"`	R
groupName	Yes	String	The name of the group whose matching members are sought. Example: “engineering”	R
userLimit	No	String (representing an Integer)	The maximum number of users to be returned from the search. (String which has a particular `Integer.valueOf.`) Defaults to 100. If user count exceeds userLimit, the length of the array in id is truncated to the length of userLimit. Example: “500”	R
id	Yes	String	A variable name to which the resultant user names are assigned. Example: “myUsers”	C

Table 13-82 <um:getUsernamesForGroup> (Continued)

Tag Attribute	Required	Type	Description	R/C
result	No	String	The name of an Integer variable to which the result of the get usernames for group operation is assigned. Possible values: *Success:* `UserManagementTagConstants.USER_SEARCH_OK` *General error*: `UserManagementTagConstants.USER_SEARCH_FAILED`	C

Note: The `USER_SEARCH_FAILED` value is returned only when a general error occurs while searching for the user, such as a database connection failure. If no user matches the search criteria, the result will not be equal to `UserManagementTagConstants.USER_SEARCH_FAILED`, but the length returned by the array in `id` will be zero.

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:getUsernamesForGroup groupName="engineering" userLimit="500"
searchExp="t*" id="myUsers"/>
<%System.out.println("I found " + myUsers.length + " users in my
group.");%>
```

<um:removeGroup>

The `<um:removeGroup>` tag (Table 13-83) removes the group corresponding to the provided `groupName`. This tag has no enclosed body.

Note: This tag should only be invoked when the current realm is an implementation of `weblogic.security.acl.ManageableRealm`. This interface is implemented by the default WebLogic Personalization Server realm (`com.bea.p13n.security.realm.RDBMSRealm`).

Table 13-83 <um:removeGroup>

Tag Attribute	Required	Type	Description	R/C
groupName	Yes	String	The name of the group to be removed. Example: "<%=groupName%>"	R
result	Yes	String	The name of an Integer variable to which the result of the remove group operation is assigned. Possible Values: *Success:* `UserManagementTagConstants.REMOVE_GROUP_OK` *Error encountered:* `UserManagementTagConstants.REMOVE_GROUP_FAILED`	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:removeGroup groupName=”<%=groupName%>” result=”result”/>
```

<um:removeGroupFromGroup>

The `<um:removeGroupFromGroup>` tag (Table 13-84) removes a child group from a parent group.

Table 13-84 <um:removeGroupFromGroup>

Tag Attribute	Required	Type	Description	R/C
childGroupName	Yes	String	The name of the child group to remove from its parent.	R
parentGroupName	Yes	String	The name of the parent group from which the child group will be removed.	R
result	Yes	String	The name of an Integer variable to which the result of the remove group from group operation is assigned. Possible values: *Success:* `UserManagementTagConstants.REMOVE_GROUP_OK` *Failure:* `UserManagementTagConstants.REMOVE_GROUP_FAILED`	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:removeGroupFromGroup parentGroupName="SomeGroup"
childGroupName="ChildGroupToRemove" result="myResult" />
```

<um:removeUser>

The `<um:removeUser>` tag (Table 13-85) removes the user corresponding to the provided `username`. It can remove any type of extended user that has its profileType set in the database. This tag has no enclosed body.

Note: This tag should only be invoked when the current realm is an implementation of `weblogic.security.acl.ManageableRealm`. This interface is implemented by the default WebLogic Personalization Server realm (`com.bea.p13n.security.realm.RDBMSRealm`).

Table 13-85 <um:removeUser>

Tag Attribute	Required	Type	Description	R/C
username	Yes	String	The username of the user to be removed. Example: "<%=username%>"	R
result	Yes	String	The name of an Integer variable to which the result of the remove user operation is assigned. Possible values: *Success:* `UserManagementTagConstants.REMOVE_USER_OK` *Error encountered*: `UserManagementTagConstants.REMOVE_USER_FAILED`	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:removeUser userName=”<%=username%>” result=”result”/>
```

<um:removeUserFromGroup>

The `<um:removeUserFromGroup>` tag (Table 13-86) removes a user from a group.

Note: This tag should only be invoked when the current realm is an implementation of `weblogic.security.acl.ManageableRealm`. This interface is implemented by the default WebLogic Personalization Server realm (`com.bea.p13n.security.realm.RDBMSRealm`).

Table 13-86 <um:removeUserFromGroup>

Tag Attribute	Required	Type	Description	R/C
username	Yes	String	The username of the user to remove from the given group.	R
groupName	Yes	String	The name of the group from which the given user will be removed.	R
result	Yes	String	The name of an Integer variable to which the result of the remove user from group operation is assigned. Possible values: *Success:* `UserManagementTagConstants.REMOVE_USER_OK` *Failure:* `UserManagementTagConstants.REMOVE_USER_FAILED`	C

Example

```
<%@ taglib uri="um.tld" prefix="um" %>
.
.
.
<um:removeUserFromGroup username="UserToRemove"
groupName="SomeGroup" result="myResult" />
```

User Management: Security Tags

User Management tags allow access to user and group profile information, as well as operations such as creating and deleting users and groups, and managing user-group relationships.

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

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

All User Management tags send results to the same file. If you are checking for results, include this import directive at the top of the page:

```
<%@ page
import="com.bea.p13n.usermgmt.servlets.jsp.taglib.UserManagementTa
gConstants" %>
```

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<um:login>

The `<um:login>` tag (Table 13-87) provides weak authentication (username, password) against the current security realm, and sets the authenticated user as the current WebLogic user. This tag has no enclosed body.

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

Table 13-87 <um:login>

Tag Attribute	Required	Type	Description	R/C
result	Yes	String	The name of an Integer variable to which the result of the login operation is assigned. Possible values: *Success:* `UserManagementTagConstants.LOGIN_OK` *General error when performing authentication*: `UserManagementTagConstants.LOGIN_ERROR` *Authentication failed because of invalid username/password combination*: `UserManagementTagConstants.LOGIN_FAILED`	C

<um:logout>

The `<um:logout>` tag (Table 13-88) ends the current user's WebLogic Server session. This tag should be used in combination with the `<um:login>` tag.

Table 13-88 <um:logout>

Tag Attribute	Required	Type	Description	R/C
No attributes				

<um:setPassword>

The `<um:setPassword>` tag (Table 13-89) updates the password for the user corresponding to the provided username.

Note: This tag should only be invoked when the current realm is an implementation of `weblogic.security.acl.ManageableRealm`. This interface is implemented by the default WebLogic Personalization Server realm (`com.bea.p13n.security.realm.RDBMSRealm`). In addition, the user object used by the current realm must implement `weblogic.security.acl.CredentialChanger`.

Table 13-89 <um:setPassword>

Tag Attribute	Required	Type	Description	R/C
username	Yes	String	The username of the user whose password is to be changed.	R
password	Yes	String	The new user password.	R
result	Yes	String	The name of an Integer variable to which the result of the set password operation is assigned. Possible values: *Success*: `UserManagementTagConstants.SET_PASSWORD_OK` *Failure*: `UserManagementTagConstants.SET_PASSWORD_FAILED`	C

Personalization Utilities

The `<es:jsptaglib>` tag contains generic tags you can use to create JSP pages.

Use the following code to import the utility tag library:
`<%@ taglib uri="es.tld" prefix="es" %>`

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

<es:convertSpecialChars>

The `<es:convertSpecialChars> (Table 13-90)` tag converts characters which would normally signify special meaning to an HTML browser into characters which can be displayed as intended.

For example, the following sentence must be converted because it uses the '<' and '>' characters, which signify tag opening and closing to the browser:
Enter <filename> here:

Table 13-90 <es:convertSpecialChars>

Tag Attribute	Required	Type	Description	R/C
string	Yes	String	The string to be converted.	R

Example

This example allows a string containing a less-than sign to be rendered in HTML.

```
<%@ taglib uri="es.tld" prefix="es" %>
.
.
.
<es:convertSpecialChars string="<thisString>"/>
```

<es:counter>

The `<es:counter>` tag (Table 13-91) is used to create a `for` loop.

Table 13-91 <es:counter>

Tag Attribute	Required	Type	Description	R/C
type	No	String	The type of the counter. Possible values are `int` or `long`. Default is `int`.	R
id	Yes	String	A unique name for the variable.	R
minCount	Yes	Int	The start position for the loop.	R
maxCount	Yes	Int	The end position for the loop.	R

Example

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

<es:date>

The `<es:date>` tag (Table 13-92) is used to get a date- and time-formatted String based on the user's time zone preference.

Table 13-92 <es:date>

Tag Attribute	Required	Type	Description	R/C
timeZoneId	No	String	Defaults to the time zone on the server.	R

Table 13-92 <es:date>

Tag Attribute	Required	Type	Description	R/C
formatStr	No	String	A date and time format string that adheres to the java.text.SimpleDateFormat. The default value is `MM/dd/yyyy HH:mmss:z`.	R

Example

```
<%@ taglib uri="es.tld" prefix="es" %>
.
.
.
<es:date formatStr="MMMM dd yyyy" timeZoneId="MST" />
```

<es:forEachInArray>

The `<es:forEachInArray>` tag (Table 13-93) is used to iterate over an array.

Table 13-93 <es:forEachInArray>

Tag Attribute	Required	Type	Description	R/C
id	Yes	String	The variable for each value in the array.	R
type	Yes	String	The type of each value in the array.	R
array	Yes	Object `[]`	The array to iterate over.	R
counterId	No	String	The position in the array.	R

Example

```
<es:forEachInArray id="item" array="<%=items%>" type="String"
counterId="i">
     <% System.out.println("items[" + i + "]: " + item);%>
</es:forEachInArray>
```

<es:isNull>

The `<es:isNull>` tag (Table 13-94) is used to check if a value is null. In the case of a `String`, the `<es:isNull>` tag is used to check if the `String` is null or has a value. An empty string will cause `isNull` to be `false`. (An empty string is not null.)

Table 13-94 <es:isNull>

Tag Attribute	Required	Type	Description	R/C
item	Yes	Object	The variable to evaluate.	R

Example

```
<%@ taglib uri="es.tld" prefix="es" %>
.
.
.
<es:isNull item="<%=value%>">
     Error: the value is null.
</es:isNull>
```

<es:notNull>

The `<es:notNull>` tag (Table 13-95) is used to check if a value is not null. In the case of a `String`, the `<es:notNull>` tag is used to check if the `String` is not null or has a value. An empty string will cause `notNull` to be `true`. (An empty string is treated as a value.)

Table 13-95 <es:notNull>

Tag Attribute	Required	Type	Description	R/C
item	Yes	Object	The variable to evaluate.	R

Example

```
<%@ taglib uri="es.tld" prefix="es" %>
.
.
.
<es:notNull item="<%=value%>">
     The value is not null.
</es:notNull>
```

<es:transposeArray>

The `<es:transposeArray>` tag (Table 13-96) is used to transpose a standard [row][column] array to a [column][row] array.

Table 13-96 <es:transposeArray>

Tag Attribute	Required	Type	Description	R/C
id	Yes	String	The variable that holds the [c][r] array.	R
type	Yes	String	The type of variable in the [r][c] array, such as String.	R
array	Yes	Object`[][]`	The variable that holds the [r][c] array.	R

Example

```
<%@ taglib uri="es.tld" prefix="es" %>
.
.
.
<es:transposeArray id="byColumnRow" array="<%=byRowColumn%>"
type="String">
   ...
</es:transposeArray>
```

<es:uriContent>

The `<es:uriContent>` tag (Table 13-97) is used to pull content from a URL. It is best used for grabbing text-heavy pages.

Table 13-97 <es:uriContent>

Tag Attribute	Required	Type	Description	R/C
id	Yes	String	The variable that holds the downloaded content of the URI.	R
uri	Yes	String	The fully qualified URI from which to get the content.	R

Example

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

Note: If you combine HTML pages with relative URL's, you must fully qualify them to the correct host in each URL, or else images (on other resources) may not be retrieved properly by the browser.

WebLogic Utilities

The `<wl:jsptaglib>` tag library contains custom JSP extension tags which are supplied as a part of the WebLogic Server platform.

To import the WebLogic Utilities JSP tags, use the following code:
`<%@ taglib uri="weblogic.tld" prefix="wl" %>`

Note: In the following tables, the Required column specifies if the attribute is required (yes) or optional (no). In the R/C column, C means that the attribute is a Compile time expression, and R means that the attribute can be either a Request time expression or a Compile time expression.

Note: See *Javadoc* API documentation for further descriptions of the `<wl:>` tags.

<wl:cache>

The `<wl:cache>` tag specifies that its contents do not necessarily need to be updated every time it is displayed.

Table 13-98 <wl:cache>

Tag Attribute	Required	Type	Description	R/C
timeout	No	Integer	Controls the time-to-live of the data, or how often the data must be updated independent of all other controls. This value is in seconds.	R
scope	No	String	Controls the time-to-live of the data, or how often the data must be updated independent of all other controls. This value is in seconds	C
name	No	String	Uniquely identifies this cache. If you do not specify a name a random name will be generated.	C

Table 13-98 <wl:cache>

Tag Attribute	Required	Type	Description	R/C
size	No	Integer	The maximum number of entries that can be in the cache. It defaults to an unlimited cache. It is only relevant for when there is an associated key.	R
vars	No	String	In addition to caching the transformed output of the cache, you can also cache calculated values within the block. These variables are specified exactly the same way as the cache keys. This type of caching is called Input caching.	C
key	No	String	Specifies a comma separated list of values accessible from the current page that the data depends on. These values act as additional keys into the cache.	C
async	No	String	If the async parameter is set to `true`, the cache will be updated asynchronously, if possible. The user that initiates the cache hit sees the old data.	C

<wl:process>

The `<wl:process>` tag (Table 13-99) is used for query attribute-based flow control. By using a combination of the four attributes, you can selectively execute the statements between the `<wl:process>` and `</wl:process>` tags.

Table 13-99 <wl:process>

Tag Attribute	Required	Type	Description	R/C
name	No	String	The name of a query attribute.	R
notName	No	String	The name of a query attribute.	R
value	No	String	The value of a query attribute.	R
notValue	No	String	The value of a query attribute.	R

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

Attribute	Value	notValue	Neither "value" nor "notValue"
name	Named attribute is equal to the value.	Named attribute does not equal the value.	Name attribute's value is not null.
not Name			notName attribute's value is null.

Example

```
<%@ taglib uri="weblogic.tld" prefix="wl" %>
.
.
.
<wl:process name="lastBookRead" value="A Man in Full">
<!-- This section of code will be executed
   if lastBookRead exists and the value of lastBookRead is
   "A Man in Full" -->
</wl:process>
```

<wl:repeat>

The `<wl:repeat>` tag (Table 13-100) is used to iterate over a variety of Java objects, as specified in the set attribute.

Table 13-100 <wl:repeat>

Tag Attribute	Required	Type	Description	R/C
set	No	Object	The set of objects that includes: ■ Enumerations ■ Iterators ■ Collections ■ Arrays ■ Vectors ■ Result Sets ■ Result Set MetaData ■ Hashtable keys	R
count	No	Int	Iterate over first "count" entries in the set.	R
id	No	String	Variable to contain current object being iterated over.	C
type	No	String	Type of object that results from iterating over the set you passed in. Defaults to Object. This type must be fully qualified.	C

Index